open source PRESS

Die IBM Mainframe Architektur

z/OS, z/VM und Linux

Wolfram Greis

Wolfram Greis: Die IBM-Mainframe-Architektur

open source PRESS

Wolfram Greis

Die IBM-Mainframe-Architektur

z/OS, z/VM und Linux

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Gesamtlektorat: Dr. Markus Wirtz
Satz: Open Source Press GmbH (LaTeX)
Umschlaggestaltung: Fritz Design GmbH, Erlangen
Gesamtherstellung: Kösel, Krugzell

ISBN 3-937514-05-8 http://www.opensourcepress.de

Grußwort

Liebe Leserinnen und Leser,

der Mainframe ist der Urahn der heutigen Server und ist doch selbst immer jung geblieben: groß, zuverlässig und leistungsstark. Unsere Entwickler haben das enorme Potential eines Universalrechners bereits in der Geburtsstunde 1960 erkannt, als das damalige System S/360 gerade frisch dem Labor entschlüpfte. Dass die spätere Großrechner-Architektur immer weiter wachsen und gedeihen würde und wir mit ihr eine Grundlage für die heutige On Demand Welt schaffen würden, konnte damals keiner ahnen. Jetzt legen wir mit unserer Mainframe Charta die strategische Grundlage für eine Zukunft, in der unsere Kunden den Großrechner als Rückgrat ihrer Informationstechnik betreiben.

Eine große Rolle in der Zukunft des Mainframes wird Linux spielen. Ich würde sogar sagen, dass Linux eine Art Verjüngungskur für den Großrechner sein wird. Denn: Linux und der Mainframe harmonieren hervorragend miteinander. Als Universalrechner kann sich der Mainframe ganz einfach technologisch anpassen – an Ihren Kapazitätsbedarf und Ihre Geschäftsprozesse. Ganz einfach durch An- und Abschalten von Komponenten. Auf Bedarf. On demand.

Solche und viele andere Themen lassen sich in diesem Buch nachschlagen, lesen und verstehen. Wolfram Greis beschäftigt sich hier mit den Basics des Mainframes und auch mit seiner Zukunft. Der Autor beleuchtet die Mainframe-Architektur von allen Seiten, sein Buch ist übersichtlich strukturiert und erklärt umfangreiche Teilbereiche des Mainframe anschaulich und verständlich. In meiner Bibliothek wird es nicht fehlen und ich werde es auch unseren Kunden empfehlen.

Johann Haala
Direktor Vertrieb IBM eServer zSeries Central Region

Vorwort

Ein Mainframe muss wohl doch etwas Besonderes sein, da immer mehr Hersteller ihre Server als „Mainframes" oder „Mainframe-like" bezeichnen. Aber was zeichnet einen Mainframe aus? Größe? Leistungsfähigkeit? CPU-Power? I/O Durchsatz? Robustheit? Zuverlässigkeit?

Der Begriff „Mainframe" ist weder geschützt noch gibt es eine eindeutige Definition. Er kommt aus der Vergangenheit der Großsysteme, als diese noch ganze Säle ausfüllten. Die elektronischen Komponenten wurden in riesige Stahlrahmen (Steel Frames) eingebaut, die Kernkomponenten in einem Hauptrahmen (Main Frame), der für die Rechenoperationen zuständig war.

Heutzutage sind die Räume in einem Rechenzentrum zwar immer noch gut gefüllt, der eigentliche Mainframe nimmt davon allerdings nur noch einen ganz geringen Teil in Anspruch. Den größten Platzbedarf in einer typischen Mainframe-Umgebung hat heute die Peripherie, vor allem die Plattensubsysteme und Bandroboter.

Ein moderner Mainframe verfügt über eine Kombination der oben erwähnten Eigenschaften, die ihn zum zentralen Server prädestinieren. Es ist nicht ein einzelnes Merkmal, das einen Mainframe kennzeichnet, sondern die Summe der Eigenschaften, die für eine von vielen Benutzern gleichzeitig beanspruchte, leistungsfähige elektronische Datenverarbeitung notwendig sind. Eine robuste und zuverlässige Hardware sowie ein ausgereiftes Betriebssystem mit entsprechenden Subsystemen bilden die Grundlage für Middleware und Anwendungssysteme.

Ein Mainframe ist in der Regel auf kommerzielle Anwendungen ausgerichtet und wird in großen Unternehmen für Online-Anwendungen (Transaktionsverarbeitung) beispielsweise im Finanz- und Dienstleistungsbereich sowie für wissenschaftliche Anwendungen wie zum Beispiel für Wettervorhersagen eingesetzt.

Bei der Transaktionsverarbeitung kommt es eher auf große I/O-Bandbreiten an, während bei wissenschaftlichen Anwendungen meist extrem hohe Rechenleistung im Vordergrund steht. Der IBM Mainframe ist, wie wir noch sehen werden, insbesondere für Anwendungen geschaffen, die auf hohe I/O-Bandbreiten und hohe Netzwerkbandbreiten angewiesen sind.

Der Mainframe steht vorwiegend für zentrale Verarbeitung – im Gegensatz zu verteilter Verarbeitung. In moderneren Umgebungen dient er oft als Transaktions- und Datenbankserver in einem Netzwerk, auf den von verteilten Arbeitsstationen und verteilten Servern zugegriffen wird. Es gibt diverse Analysen und Statistiken, aus denen hervorgeht, dass zwischen 60 und 80 Prozent aller geschäftskritischen Daten auf Mainframes vorgehalten werden. Dies ist in erster Linie auf überdurchschnittliche Performanz, Verfügbarkeit, Skalierbarkeit und Sicherheit zurückzuführen.

Der Mainframe unterstützt heute moderne E-Business-Anwendungen. Die Hardware, das Betriebssystem und die durch WebSphere implementierte J2EE-Architektur ergänzen sich bei der Transaktionsverarbeitung, Security und Message Queuing.

Wer auf dem IBM Mainframe arbeiten will oder muss, der erhält mit diesem Buch einen Einblick in die Systemkomponenten und deren Funktionsweisen. Allein aufgrund der Tatsache, dass die Großrechner in der IT aller Großunternehmen Schlüsselpositionen einnehmen und in den nächsten 10 Jahren auch weiterhin einnehmen werden, lohnt sich die Beschäftigung mit diesem Rechnersystem. Insofern können Sie die Lektüre dieses Buches auch als Investition in die Sicherung Ihrer „Marktfähigkeit" betrachten.

Dieses Buch ist im Verlag Open Source Press erschienen. Was haben der IBM Mainframe und das Betriebssystem mit Open Source zu tun? Auf den ersten Blick wenig: Das z/OS als wichtigstes Betriebssystem basiert zwar inzwischen zum überwiegenden Teil auf gcc (GNU Compiler Collection), ist deshalb aber natürlich noch keine Open Source Software. Allerdings sind auf dem Mainframe nun auch Open-Source-Produkte verfügbar – allen voran „Linux for zSeries" als Betriebssystem –, wodurch mit einem Schlag auch zahlreiche weitere Anwendungen aus dem Open-Source-Bereich zur Verfügung stehen.

Die Zielgruppe, die dieses Buch ansprechen möchte, sind grundsätzlich alle, die sich mit einem Mainframe auseinandersetzen und ein Grundverständnis für diese Architektur benötigen, um sinnvoll mit dem Mainframe arbeiten oder wenigstens mitreden können, wenn es um dessen Einsatz geht. Angesprochen werden somit all jene, die aus der „Open-Source-Ecke" kommen und möglicherweise über Linux for zSeries bereits Berührung mit dem Mainframe hatten, aber auch Personen, die bisher nichts mit Open Source „am Hut" hatten.

Zusammengefasst gibt es nach meiner Auffassung drei Gründe, die dafür sorgen, dass der Mainframe auch in Zukunft eine große Rolle in den Rechenzentren spielen wird:

1. seine „State-of-the-Art-Technik", die den Mainframe in Sachen Robustheit, Skalierbarkeit, Sicherheit und Performanz auszeichnet,

2. die Unterstützung neuer, standardisierter Technologien wie J2EE in Verbindung mit WebSphere und

3. die Einsatzmöglichkeiten von Open Source und insbesondere von Linux auf dem Mainframe im Zusammenspiel mit den klassischen MVS-Systemen und den Legacy-Anwendungen.

Danken möchte ich vor allem Professor Dr. Ing. Wilhelm G. Spruth für die zahlreichen fruchtbaren Diskussionen sowie für seinen Einsatz, die Ausbildung im Bereich Mainframes an Hochschulen voranzutreiben. Über ein gemeinsames Projekt an der IT Akademie Bayern in Augsburg unter der Leitung von Volker Falch durften wir ein Curriculum für angehende „Mainframer" auf die Beine stellen, das aufgrund des großen Erfolgs bereits zum zweiten Mal durchgeführt wird. Viele Anregungen und Ergebnisse dieses Lehrgangs sind in dieses Buch eingeflossen.

Für die Durchsicht des Manuskripts möchte ich mich vor allem bei Ulrich Günschmann bedanken, der mit seiner konstruktiven Kritik dazu beigetragen hat, dass das Buch „runder" wurde und auch einige fachliche Aspekte besser dargestellt werden konnten.

Außerdem gilt der Dank meinen Kollegen von der TPS DATA AG in Zürich, allen voran meinem Kompagnon Max Holliger, für die Toleranz und die Unterstützung, die mir bei der Arbeit an diesem Buch entgegengebracht wurde.

Wolfram Greis Öhningen, im März 2005

Inhaltsverzeichnis

Kapitel 1

Einführung

"It is not the strongest of the species that survive, nor the most intelligent, but the one most responsive to change."
— Charles Darwin

1.1 Vielzweckrechner

Die ersten Rechner, die in den 1950-er und 1960-er Jahren eingesetzt wurden, waren für bestimmte Anwendungsgebiete entwickelt und gebaut worden. So gab es beispielsweise Rechner für militärische Zwecke, für wissenschaftliche Anwendungen oder wiederum solche, die auf die Unternehmensverwaltung ausgerichtet waren. Der Anteil der Hardware an den Gesamtkosten einer IT-Umgebung betrug zur damaligen Zeit meist mehr als 80 Prozent.

Um Ressourcen effizient nutzen zu können, kam die Idee auf, ein Programm innerhalb eines Unternehmens von einem Rechner auf einen anderen Rechner zu

übertragen. Das war jedoch aufgrund der unterschiedlichen Architekturen nicht ohne weiteres möglich. Die Programme mussten meist umgeschrieben und angepasst, in jedem Fall aber neu kompiliert werden, und zwar auch dann, wenn die Kapazität eines Rechners nicht mehr ausreichte und ein größeres und schnelleres Modell installiert wurde.

So entstand das Bedürfnis, Rechner zu entwickeln, die alle Verarbeitungsarten innerhalb eines Unternehmens unterstützten und kompatibel zueinander waren. Diese Vielzweckrechner (*General-Purpose Computer*) waren die Lösung, um effizienter und mit weniger Ressourcen alle DV-spezifischen Verarbeitungen „auf einem Blech" durchführen zu können. Die Bezeichnung „/360" für die erste Generation der IBM Mainframes nimmt entsprechend Bezug auf die Zahl der Kompassgrade als Sinnbild für den Anspruch, alle Anwendungsrichtungen innerhalb eines Unternehmens abzudecken: Kommerzielle und wissenschaftliche Anwendungen – sowohl in Form von Batch-Verarbeitung als auch interaktiv abgearbeitet – auf einem Rechner.

1.2 Historie der IBM-Architekturen

Mit der Ankündigung der /360-Architektur im Jahr 1964 nutzte erstmals ein Hersteller von Computern den Begriff „Architektur" und setzte diesen auch um. Federführend dabei war ein Design- und Entwicklungsteam um Gene Amdahl, der als einer der Väter der /360-Architektur gilt. Zwei weitere Namen in diesem Zusammenhang sind Fred P. Brooks und Gerry A. Blaauw, die die Entwicklung wesentlich mitgeprägt haben. Ein Artikel von H. Lorin macht deutlich, um was es geht: "The output of an architectural effort is a document that imposes requirements on a design. The output of a design effort is the mapping of an architecture into a technology in order to achieve stated price/performance goals for a model of the architecture. Thus – a product-compatible product line (various models each of which respond in the same way to a list of operation codes and addresses) can be defined at different price/performance levels."[1]

Kerngedanke ist demnach eine „Produkt-kompatible Produktlinie", also eine Modellreihe von Rechnern auf *einer* technologischen Grundlage, aber in unterschiedlichen Ausführungen gemäß den jeweiligen Preis/Performance-Anforderungen. Die Architektur definiert also, wie ein System funktioniert. Die Beschreibung dieser Architektur kann in den so genannten *Principles of Operation* nachgelesen werden. Diese PoPs gibt es seit 1964, sie wurden immer wieder erweitert und um jeweils neue Funktionalitäten ergänzt.[2]

In den PoPs steht beispielsweise, dass die Architektur eines Rechners definiert ist als "its attributes as seen by the programmer; that is, the conceptual structure and

[1] H. Lorin: „System Architecture in Transition – an Overview", *IBM Systems Journal*, vol. 25, 1986.
[2] Für die aktuelle z/Architecture beispielsweise unter: http://www-1.ibm.com/servers/eserver/zseries/zos/bkserv/r6pdf/zarchpops.html.

functional behaviour as distinct from the organization of data flow, the logical design, and the performance of any particular implementation. Several dissimilar machine implementations may conform to a single architecture. When programs running on different machine implementations produce the results that are defined by a single architecture, the implementations are considered to be compatible."

Von Anfang an standen verschiedene Modelle mit unterschiedlichen Leistungsmerkmalen und damit auch in unterschiedlichen Preiskategorien zur Verfügung, die – und das ist das Entscheidende – *kompatibel* zueinander waren. Diese Kompatibilität sichert IBM seinen Kunden seit 1964 zu, und zwar konsequent – mit allen Vor- und Nachteilen. Allein für die /360-Architektur standen im Laufe ihres Lebenszyklus insgesamt zwölf unterschiedliche Modelle zur Verfügung, für die „zSeries" derzeit bereits über 100.

Die Architektur wurde in der Folge immer wieder erweitert, während die für eine Architektur entwickelten Programme und Anwendungen immer auch von den Folgearchitekturen unterstützt wurden, ohne dass irgendwelche Änderungen vorgenommen werden mussten.

Auf der Basis einer Architektur können – auch das ist typisch für die Mainframe-Serie der IBM – unterschiedliche Betriebssysteme entwickelt werden. Das bedeutet, dass für die unterschiedlichen Architekturstufen jeweils mehrere Betriebssysteme zur Auswahl stehen. Dass diese auch zusammenarbeiten können, liegt auf der Hand.

Diese Tatsache spielt beispielsweise aktuell im E-Business eine entscheidende Rolle. In vielen Unternehmen werden bereits Linux und z/OS parallel auf einer gemeinsamen Hardware betrieben, mit dem Vorteil, dass die Systeme voneinander abgeschottet und dennoch sehr effizient miteinander zu verbinden sind. Eine typische Konfiguration ist beispielsweise ein Linux-System als Webserver-Frontend mit Zugriff auf Mainframe-Backend-Systeme; der Vorteil ist, dass dazwischen nur ein „virtuelles" Netz geschaltet wird und somit zahlreiche Fehlerquellen und Performance-Engpässe, die durch physische Verkabelungen anfallen, eliminiert werden.

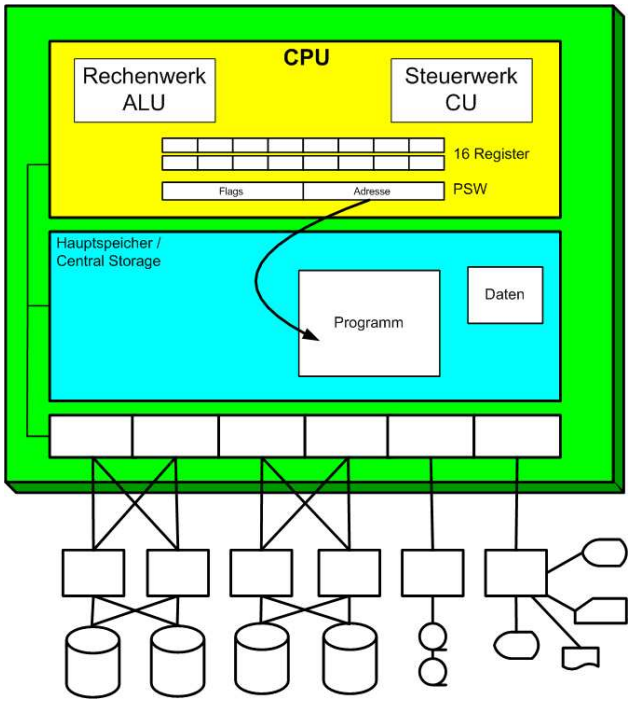
Im Folgenden wird die Entwicklung von der /360- zur aktuellen zSeries-Architektur chronologisch dargestellt; neue Konzepte, die im Laufe der Zeit hinzukamen, wurden aufgrund des bereits erwähnten Kompatibilitäts-Grundsatzes in den Folgearchitekturen beibehalten. So gibt es beispielsweise in der Extended Architecture (XA) immer noch die Kanalschnittstelle für den Anschluss von Peripheriegeräten, allerdings ist diese in ein Kanalsubsystem ausgelagert, das nicht mehr – wie noch bei der Vorgängerarchitektur – fest mit einer CPU verbunden ist. Vielmehr kann nun von mehreren CPUs auf eine gemeinsam genutzte Kanalschnittstelle zugegriffen werden.

1.3 Monoprozessor der /370-Architektur

Ein System setzt sich aus mehreren Hardwarekomponenten zusammen, zu denen an erster Stelle eine CPU gehört. Über Kanäle werden Ein-/Ausgabegeräte mit dem System verbunden. Ein wesentlicher Punkt dabei ist, dass Kanäle nicht nur physische Kabel sind, sondern auch so genannte Kanalprozessoren umfassen, die eine Ausgabe unabhängig von der CPU steuern können. Zwischen Kanälen und Ein-/Ausgabegeräten sind Kontrolleinheiten installiert, welche für die Steuerung und den Betrieb der daran angeschlossenen Geräte verantwortlich sind.

Obwohl die Betriebssystemsoftware Softwarefehler im Allgemeinen handhaben kann, ist in dieser Konfiguration kein Hardware-Backup möglich. Was passiert z. B., wenn der Prozessor ausfällt? Ein einziges Betriebssystem läuft auf einem einzigen Prozessor; es gibt somit in dieser Konfiguration mehrere so genannte *Single Points of Failure*. Ein Single Point of Failure liegt dann vor, wenn in einem System irgendeine kritische Komponente nur einmal vorhanden ist. Dabei spielt es keine Rolle, ob es sich bei der Komponente um Hardware oder Software handelt. Fällt die Komponente aus, ist keine Verarbeitung mehr möglich.

Abbildung 1.1: Monoprozessor



CPU

Die *Central Processing Unit* (CPU) ist das eigentliche Gehirn des Rechners. Hier werden die Befehle interpretiert und entsprechende Aktionen veranlasst. Die CPU setzt sich unter anderem zusammen aus einem *High Speed Buffer* (HSB), der oft auch als „Cache" bezeichnet wird und der auch auf zahlreichen anderen Architekturen implementiert ist.

Eine *Control Unit* (Steuerwerk) stellt die Verbindung zum Hauptspeicher her und kümmert sich um die internen Abläufe innerhalb der CPU.

Eine *Arithmetic and Logical Unit* (ALU, Rechenwerk) führt Rechenoperationen und logische Operationen aus, während die *Register* (16 *General-Purpose Register* sowie Systemregister) Informationen zur Zwischenspeicherung aufnehmen.

Steuerwerk, Rechenwerk und Register sind Komponenten, die in dieser oder ähnlicher Form auf allen Rechnerarchitekturen vorhanden sind. Ein für die IBM-Architektur spezifisches (und entscheidendes) Register ist das *Program Status Word* (PSW). Das PSW besteht aus zwei Teilen, dem Adressteil und den „Flags". Der Adressteil ist der Befehlszähler (*Instruction Counter*), der auf den nächsten auszuführenden Befehl im Hauptspeicher zeigt. Die Flags definieren den Zustand des aktuell ausgeführten Programms. Es sind zumeist einfache Schalter, die entweder ein- oder ausgeschaltet sind. So steuert beispielsweise ein Bit unter diesen Flags, ob sich das Programm im „normalen" oder im privilegierten Zustand befindet. Dies ist ein sehr effizienter und sicherer Mechanismus, um zwischen Anwendungs- und Systemprogrammen unterscheiden zu können.

Ein Maschinenzyklus läuft wie folgt ab:

1. Das Steuerwerk holt sich mit Hilfe des PSW über den Befehlszähler den nächsten auszuführenden Befehl aus dem Hauptspeicher.

2. Bereits bei der Interpretation des Befehls kann das Steuerwerk mit Hilfe des Operationscodes die Länge des aktuellen Befehls ermitteln und zählt die Adresse im PSW um diesen Längenwert hoch. Das PSW zeigt somit bereits auf den nächsten Befehl im Hauptspeicher.

3. Das Steuerwerk übergibt die zu erledigende Arbeit (z. B. die Addition zweier Zahlen oder auch das Verschieben von Daten) an das Rechenwerk.

4. Das Rechenwerk erledigt die ihm übertragene Aufgabe i. d. R. unter Einsatz der Register und gibt die Kontrolle zurück an das Steuerwerk.

5. Das Steuerwerk beginnt mit dem nächsten Zyklus.

Eine einzige CPU kann heute ca. 450 Millionen Operationen pro Sekunde (MIPS) bewältigen.

Hauptspeicher

Der Hauptspeicher der IBM-Architektur ist byte-adressiert. Ursprünglich wurde mit einer 24-Bit-Adresse gearbeitet. Damit war eine Hauptspeichergröße von 16 MB adressierbar. Seit Anfang der 80-er Jahre wurde mit Einführung der *Extended Architecture* (XA) mit 31-Bit-Adressen gearbeitet, wodurch 2 GB Speicher adressierbar wurden.

Mit der zSeries-Architektur wurde die 64-Bit-Adressierung eingeführt, wodurch eine theoretische Adressierung von 16 Exabytes möglich ist.

I/O-Bereich

Im I/O-Bereich wurden für die Verbindung zu den Peripheriegeräten bis zur /370-Architektur parallele Kupferkabel mit einer Kapazität bis zu 4,5 MB/s eingesetzt mit Konsequenzen für die Entfernung zwischen CPU und den Ein-/Ausgabegeräten. Aufgrund von Interferenzen konnten so nur Entfernungen bis ca. 120 m überbrückt werden. Zudem waren die Kabelstränge groß und schwer und ließen sich nicht im laufenden Betrieb anschließen bzw. abhängen. Mit der XA-Architektur wurde die Glasfasertechnologie (ESCON, 17 MB/s – FICON, 100 MB/s) eingeführt. Die Kanäle stellen die Verbindungen zwischen CPU/Hauptspeicher und den Peripheriegeräten dar. Meist geht es dabei um schnelle und effiziente Schnittstellen zur Magnetplatten- und Magnetband-Peripherie. Bei den neueren Rechnerreihen ab der S/390-Architektur können über *Open System Adapter* (OSA) auch LANs (GB-Ethernet), ATM-Ports etc. direkt angeschlossen werden.

Die wichtigsten Merkmale eines Monoprozessors

Im Falle eines Monoprozessors kann jeweils nur ein Programm aktiv sein, da für die Verarbeitung nur ein einziges Program Status Word in der CPU zur Verfügung steht. Die maximale Verarbeitungskapazität entspricht der Kapazität des größten einsetzbaren Prozessors. Es gibt mehrere Single Points of Failure: Im Falle eines Monoprozessors ist es einleuchtend, dass der Ausfall des in diesem Fall einzigen Prozessors eine weitere Verarbeitung unmöglich macht.

1.4 Multiprozessor der Extended Architecture (XA)

Um die Effizienz eines Systems zu erhöhen, wurden Multiprozessoren eingeführt. Ziel dabei ist es, den Ablauf mehrerer Programme gleichzeitig zu ermöglichen.

Diese in Abbildung 1.2 gezeigte XA-Konfiguration hat zwei CPUs. Die Leistung erhöht sich dadurch jedoch nicht auf das Doppelte, der Faktor der Leistungssteige-

rung liegt in etwa bei 1,8 – es müssen Abstriche für die Koordination der Prozessoren untereinander gemacht werden.

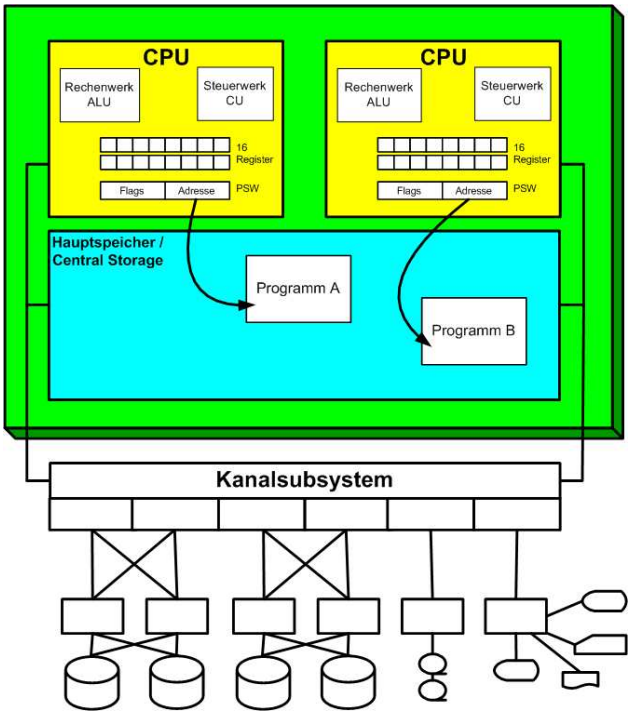


Abbildung 1.2: Multiprozessor

Vor Einführung der XA-Architektur (1981) waren die Kanäle fest mit einer CPU verbunden. Mit der XA-Architektur wurden die Kanäle in das Kanalsubsystem „ausgelagert", das auch als *Channel Subsystem* oder *I/O Subsystem* bezeichnet wird. Für die Steuerung der Ein-/Ausgaben ist der I/O Supervisor zuständig, der dem BIOS eines PC entspricht. Teile des I/O Supervisors wurden ebenfalls in das Kanalsubsystem verlagert, wodurch das Betriebssystem entlastet wird. Auf ein Kanalsubsystem und die entsprechenden Geräte dahinter kann dann ab der XA-Architketur auch von mehreren physischen Rechnern aus zugegriffen werden.

Mit Multiprozessoren wurde nun eine parallele Verarbeitung möglich. In unserer Konfiguration können zwei Tasks (entspricht den Threads in UNIX-Umgebungen) gleichzeitig von jeweils einer CPU abgearbeitet werden. Jede CPU hat einen eigenen Satz von Registern und vor allem auch ein eigenes Program Status Word (PSW).

Merkmale eines Multiprozessors

Die maximale Kapazität wird gegenüber einem Einzelprozessorsystem deutlich erhöht. Sie ist begrenzt durch die maximale Anzahl von CPUs in einem *Central Processor Complex* (CPC). Die Verfügbarkeit wird durch den Einsatz mehrerer CPUs natürlich ebenfalls gesteigert. Wenn eine CPU ausfällt, wird dadurch die Verarbeitungskapazität zwar geschmälert, die funktionale Betriebsfähigkeit ist jedoch nicht beeinträchtigt. Allerdings gibt es immer noch einen Single Point of Failure, nämlich das Betriebssystem. Das Betriebssystem als Single Point of Failure auszuschalten ist mit dem konsequenten Einsatz von Parallel Sysplex möglich.

1.5 Virtualisierung

Der Begriff „Virtualisierung" in diesem Zusammenhang bezeichnet die Möglichkeit, einen physischen Rechner in mehrere virtuelle Systeme aufzuteilen. Es geht darum, dass die Ressourcen eines Rechners (CPUs, Kanalverbindungen, Hauptspeicher etc.) dynamisch und flexibel von mehreren Betriebssystemen gleichzeitig benutzt werden können.

Hierfür gibt es für den IBM Mainframe zwei Alternativen: Zum einen mit der Software *Virtual Machine* (VM bzw. z/VM), zum anderen mit Hilfe einer Hardware/Microcode-Lösung, die bei IBM *Processor Resource/System Manager* (PR/SM) genannt wird.

Lösung mit Virtual Machine (VM)

Die Softwarelösung VM bzw. z/VM hat den Vorteil der Flexibilität und zusätzlicher Funktionen wie beispielsweise des *Conversational Monitoring System* (CMS), mit dem ein Timesharing unter VM realisiert werden kann. Im Gegensatz zur *Time Sharing Option* (TSO) im *Multiple Virtual Storage* (MVS) wird jedoch jedem Benutzer nicht nur ein eigener Adressraum, sondern eine eigene logische Maschine zur Verfügung gestellt.

Dem stehen jedoch auch Nachteile gegenüber, wie größere Komplexität und Overhead: Da es sich um eine Software-Lösung handelt, werden hier natürlich auch Ressourcen (CPU, Storage, I/O) verbraucht. Dieser Overhead kann beachtliche Größenordnungen annehmen. Zusätzlich werden entsprechende Skills für die Systemtechnik (Installation und Pflege des Systems) und für das Operating (Systembetrieb) benötigt.

Da es sich um Software handelt, fallen dafür auch Lizenzkosten an. Wer die Preisbildung im Bereich von IBM Mainframes kennt, weiß, dass diese Preise relativ stark zu Buche schlagen. Allerdings gibt es gerade in Verbindung mit dem Einsatz von

Linux unter VM Spezialkonditionen, die aus Kostengründen den Einsatz von VM interessant machen. Nachdem das VM bereits weitgehend aus den Rechenzentren verschwunden war, erlebt es nun in Verbindung mit Linux eine Renaissance.

Die neueste Version von VM, welche auch die 64-Bit-Adressierung unterstützt, wird von IBM als z/VM vermarktet.

Hardware/Microcode-Lösung

Die Hardware/Microcode-Lösung ist kostengünstiger, braucht weniger Ressourcen und setzt wenig zusätzliches Know-how bezüglich Konfiguration, Systembedienung und Maintenance voraus.

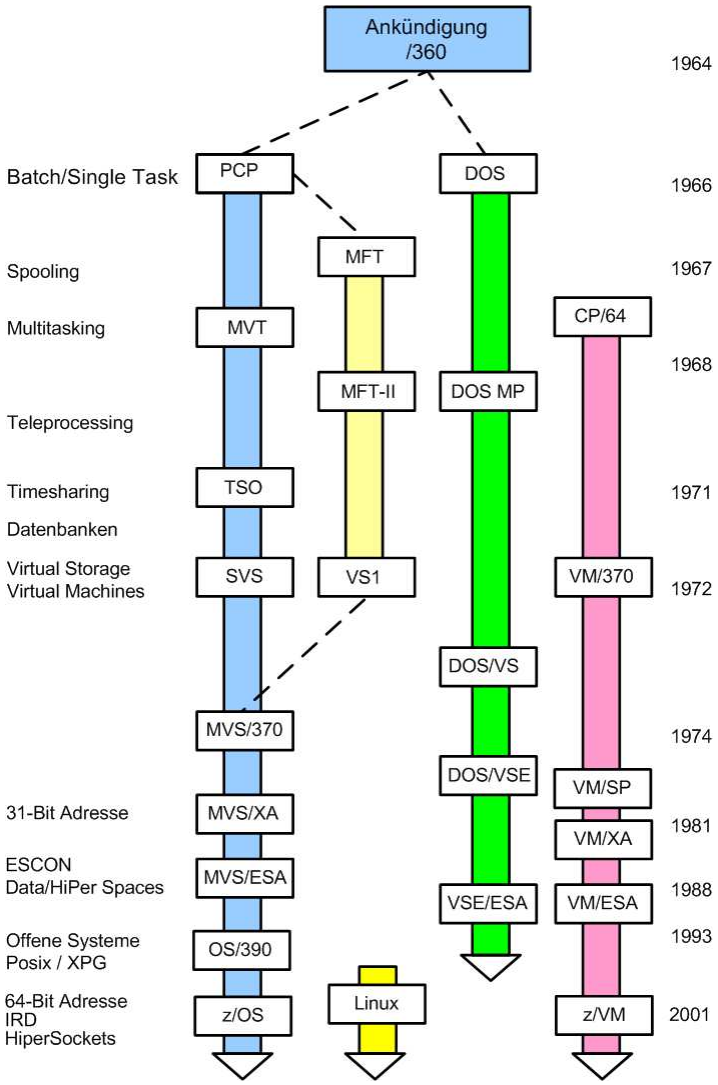
Vorreiter dieser Technologie war die Firma Amdahl, die in der Vergangenheit Stecker-kompatible Mainframes (*Plug-Compatible Mainframes*, PCM) hergestellt hat. Amdahl brachte mit der *Multiple Domain Feature* (MDF) diese Technik auf den Markt und war damit überaus erfolgreich. Hitachi, in der Vergangenheit ebenfalls im PCM-Markt aktiv, hat mit dem *Multiple Logical Partitioning Feature* (MLPF) nachgezogen, und auch IBM musste mit PR/SM eine derartige Lösung bringen, wollte sie nicht Marktanteile verlieren. Ein klassisches Beispiel also, dass der Mitbewerb hier eine Lösung forciert hat, die andernfalls vermutlich nicht zur Verfügung stünde. Leider gibt es den Mitbewerb der Plug-Compatible Mainframes nicht mehr, da sich sowohl Amdahl als auch Hitachi aus diesem Marktsegment zurückgezogen haben und IBM wieder eine gewisse Monopolstellung innehat. Allerdings sind die wichtigsten Mitbewerber von IBM schon seit geraumer Zeit nicht mehr die PCM-Hersteller, sondern vielmehr die großen UNIX-Anbieter wie HP und Sun, die ebenfalls mit Technologien auf den Markt kommen, welche früher den Mainframes vorbehalten waren.

1.6 Architektur und Betriebssysteme

In der IBM-Welt hat es zu einer Architektur immer mehrere Betriebssysteme gegeben. Diese Betriebssysteme waren auf unterschiedliche Kundensegmente ausgerichtet, die MVS-Reihe beispielsweise eher auf Großunternehmen, die VSE-Reihe eher auf mittlere Unternehmen.

Einen Spezialfall bildet die *Transaction Processing Facility* (TPF), ein „abgespecktes" MVS, das speziell auf hohe Transaktionsraten getrimmt wurde. Es ist nur sehr vereinzelt im Einsatz. Es wird deshalb in diesem Buch nicht weiter behandelt.

Abbildung 1.3: IBM-Entwicklung



DOS	= Disk Operating System	ESA	= Enterprise System Architecture
IRD	= Intelligent Resource Director	MFT	= Multiprogramming with a Fixed Number of Tasks
MVS	= Multiple Virtual Storage	MVT	= Multiprogramming with a Variable Number of Tasks
PCP	= Primary Control Program	POSIX	= Portable Operating System Interface
SVS	= Single Virtual Storage	TSO	= Time Sharing Option
VM	= Virtual Machine	VS	= Virtual Storage
VSE	= Virtual Storage Extended	XPG	= X/Open Portability Guide

In der folgenden Tabelle sind die unterschiedlichen Architekturen mit ihren Hauptmerkmalen und den darauf laufenden Betriebssystemen zusammenfassend dargestellt.

Tabelle 1.1: Architekturen im Überblick

Architektur	Hauptmerkmale	Betriebssystem	Bemerkungen
S/360	Hauptspeicher 24-Bit-Adresse	Primary Control Program (PCP)	einfachste Batch-Verarbeitung
		Multiprogramming with a Fixed Number of Tasks (MFT)	Einteilung des Hauptspeichers in feste Partitions, Spooling
		Multiprogramming with a Variable Number of Tasks (MVT)	Belegung des Hauptspeichers variabel und fließend; Beginn der interaktiven Verarbeitung
S/370	Virtueller Speicher 24-Bit Adresse	Single Virtual Storage (SVS)	einzelner virtueller Speicher, eingeteilt in Seiten, die ausgelagert werden
		Virtual Storage 1 (VS1)	begrenzte Anzahl mehrerer virtueller Speicher
		Virtual Machine/370 (VM/370)	Eine physische Maschine kann in mehrere logische Maschinen aufgeteilt werden.
		Disk Operating System/Virtual Storage (DOS/VS)	begrenzte Anzahl mehrerer virtueller Speicher
		Multiple Virtual Storage/370 (MVS/370)	mehrere virtuelle Speicher
		Transaction Processing Facility (TPF)	spezielles Betriebssystem für hohe Transaktionsraten
XA	31-Bit-Adresse I/O Subsystem Expanded Storage	MVS/XA	Teile des I/O Supervisor wurden ausgelagert in das I/O Subsystem. Einführung von System Managed Storage (SMS)
		TPF/XA	Unterstützung der 31-Bit-Adressierung
		VM/XA	Unterstützung der 31-Bit-Adressierung

Fortsetzung:

Architektur	Hauptmerkmale	Betriebssystem	Bemerkungen
ESA	Data/HiPer Spaces ESCON	MVS/ESA	Unterstützung AASF (Advanced Address Space Facility)
		VSE/ESA	Unterstützung AASF
		VM/ESA	Unterstützung AASF
		TPF/ESA	Unterstützung AASF
S/390	Offene Systeme	OS/390	Posix, XPG
		Linux for S/390	Open-Source-Ansatz
zSeries	64-Bit-Adresse	z/OS	64-Bit-Adresse Intelligent Resource Director (IRD) und Hipersockets
		z/VM	VM mit 64-Bit-Unterstützung
		z/VSE	VSE mit 64-Bit-Unterstützung
		Linux for zSeries	64-Bit-Adressierung

Anzumerken ist, dass die Vorgängersysteme zumindest während einer Übergangszeit auch auf den Nachfolgearchitekturen noch funktionieren. So läuft beispielsweise ein OS/390 auch noch auf der zSeries-Architektur.

Außergewöhnlich erfolgreich ist IBM mit der Unterstützung von Linux auf dem Mainframe gestartet. Die derzeitigen Steigerungsraten der von IBM ausgelieferten Rechnerkapazitäten sind zu einem erheblichen Anteil dem Linux-Einsatz auf dem Mainframe zuzuordnen. Dies wird in Kapitel 15 noch Thema sein.

Kapitel 2

Sysplex und GDPS

2.1 Was versteht man unter Sysplex?

Hinter Sysplex (*Sys*tem Com*plex*) steht das Clustering-Konzept der IBM Mainframes, das mit z/OS als Betriebssystem den Aufbau einer ausfallsicheren Umgebung für Anwendungen ermöglicht. Im Gegensatz zu Cluster-Lösungen anderer Hersteller geht es nicht nur darum, Komponenten redundant zu halten bzw. bei Ausfall einer Komponente eine Ersatzkomponente zur Verfügung zu stellen. Vielmehr werden die Informationen einer Anwendung in einem von mehreren Systemen gemeinsam genutzten Speicherbereich gehalten, so dass die Anwendung selbst einen Betriebssystem-Ausfall übersteht, da die relevanten Informationen von einem anderen Betriebssystem übernommen werden können.

Ein Sysplex besteht aus mehreren MVS-Systemen, die mit Hilfe von HW/SW-Produkten gemeinsam und koordiniert arbeiten. Hauptunterschiede zu konventionellen Umgebungen sind das gesteigerte Wachstumspotenzial (Skalierbarkeit) und die höhere Verfügbarkeit einer Sysplex-Umgebung.

Wenn in einer Systemumgebung eine kritische Systemkomponente nur einmal vorhanden ist, egal ob es sich dabei um Hard- oder Software handelt, ist die Gefahr eines Systemausfalls gegeben, wenn diese Komponente (*Single Point of Failure*) – aus welchem Grund auch immer – nicht mehr zur Verfügung steht. Der Einsatz von *Parallel Sysplex* bedeutet somit die konsequente Reduzierung sämtlicher Single Points of Failure.

Eines der großen Ziele von Parallel Sysplex ist es, geplante Ausfälle zu eliminieren und die Auswirkungen ungeplanter Ausfälle auf ein Minimum zu reduzieren, indem die Arbeit von verbleibenden Systemressourcen übernommen wird.

Geplante Ausfälle betreffen beispielsweise die Wartung des Systems oder die Sicherung von Datenbanken. Beides ist heute unter z/OS im laufenden Betrieb möglich. Wenn das System z. B. auf einen neuen Stand gebracht werden soll, kann in einer entsprechend konfigurierten Sysplex-Umgebung mit einem „rollierenden" *Initial Program Load* (IPL, entspricht dem „Booten" des Systems) ein System nach dem anderen aus dem Verbund herausgenommen und ein System auf dem neueren Stand wiederum in den Verbund eingebracht werden.

Hochverfügbarkeit setzt hier nicht voraus, dass Komponenten nicht mehr ausfallen, sondern dass der Ausfall einer Komponente den Systembetrieb insgesamt funktionell nicht beeinträchtigt.

Mit Parallel Sysplex ist es möglich, auf Anwendungsebene eine Verfügbarkeit von 99,999% zu erreichen. IBM spricht hier von den „Five Nines". In Bezug auf Anwendungen bedeutet dies eine Ausfallzeit im unteren Minutenbereich (5-20 Minuten) pro Jahr.

Um diese Hochverfügbarkeit zu erreichen, müssen jedoch zahlreiche Bedingungen erfüllt und entsprechende Voraussetzungen gegeben sein. Eine große Bank in der Schweiz hat beispielsweise festgestellt, dass in ihrer spezifischen Umgebung „nur" eine Verfügbarkeit von 99,8 Prozent erreicht werden kann, und daraufhin eine Arbeitsgruppe eingerichtet, die sich mit der Frage beschäftigt, was alles getan und in die Wege geleitet werden muss, um die Verfügbarkeit zu erhöhen und letztlich die „Five Nines" zu erreichen.

Es ist einleuchtend, dass die Planung für eine Sysplex-Umgebung, wenn es vor allem auf extrem hohe Verfügbarkeit ankommt, weitgehend auf Redundanz der Hardware- und Softwarekomponenten ausgelegt ist. Auch Anwendungen müssen so ausgelegt sein, dass sie über mehrere Systeme hinweg funktionieren und auf die entsprechenden Daten von mindestens zwei Systemen aus zugegriffen werden kann. Da hierfür wichtige Informationen von den betroffenen Systemen gemeinsam genutzt werden, spricht man in diesem Zusammenhang auch von *Data Sharing*. Der Ausfall eines Systems bedingt dann die unterbrechungsfreie Fortführung der Arbeit auf einem anderen.

Während es auf der einen Seite einfach ist, die Vorteile einer systemübergreifenden Zusammenarbeit mit Data Sharing und ständiger Verfügbarkeit herauszustreichen,

ist es oft weniger klar, dass es nicht mit der Installation einiger Softwarekomponenten getan ist. Hier muss ein Konzept ausgearbeitet werden, in dem die Anforderungen des jeweiligen Unternehmens berücksichtigt und mit den technischen Rahmenbedingungen in Einklang gebracht werden.

In der Praxis sind es eine ganze Reihe von Schritten, zum Teil kleine, zum Teil aber auch größere, die notwendig sind, um ein Etappenziel auf dem Weg zu Parallel Sysplex zu erreichen.

Zielsetzungen

Einige Unternehmen gehen den ganzen Weg bis ans Ende und installieren jedes Feature und jede Funktion von Parallel Sysplex. Andere wiederum beschränken sich auf jene Teile, die zur Erreichung eines entsprechend gesetzten Ziels notwendig sind.

Wie viele Schritte zu welchem Zeitpunkt tatsächlich gegangen werden, hängt von eben diesen Zielen ebenso wie der verfügbaren Hard- und Software ab. Das Aufsetzen einer Parallel-Sysplex-Umgebung ist nicht immer eine unbedingte Notwendigkeit. Allerdings können dadurch in jedem Fall zusätzliche Vorteile ausgenutzt werden. Die höchste Verfügbarkeit mit den „Five Nines" ist nur mit Parallel Sysplex und Data Sharing möglich, da ohne Data Sharing ein Betriebssystem weiterhin ein Single Point of Failure ist.

Parallele Verarbeitung

Einfach ausgedrückt erreicht man parallele Verarbeitung dadurch, dass mehrere Verarbeitungselemente zusammenarbeiten und Informationen austauschen, um eine anstehende Arbeit gemeinsam möglichst schnell zu erledigen. Parallelverarbeitung gibt es in mehreren Ausprägungen.

Bereits in den frühen Jahren der Rechnerentwicklung dachte man darüber nach, wie man mehrere Prozessoren zusammenschalten kann, um die Rechnerleistung zu erhöhen. Heute eingesetzte IBM Mainframes ermöglichen die Zusammenarbeit von maximal 48 Processing Units (PUs) in einem so genannten *Central Processor Complex* (CPC). Von diesen können 32 PUs frei konfiguriert werden. Zwischen 4 und 16 sind als *System-Assist-Prozessoren* (SAP) – sie wickeln im Wesentlichen die Ein-/Ausgabeprozesse ab – und Ersatzprozessoren reserviert.

2.2 Basis Sysplex

Um die Schwierigkeiten bei der Verwaltung mehrerer MVS-Systeme zu reduzieren, führte IBM im September 1990 das Sysplex-Konzept ein. Eine Basis-Sysplex-

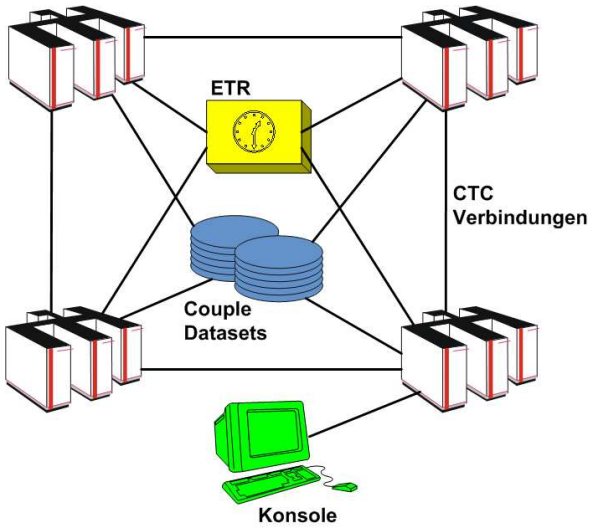
Konfiguration bildet die Grundlage für eine vereinfachte Mehrsystemverwaltung mit Hilfe einer *Cross-System-Coupling-Facility*-Komponente (XCF). XCF-Dienste ermöglichen entsprechend autorisierten Anwendungen, mit anderen Anwendungen auf dem gleichen oder auf anderen Systemen zu kommunizieren.

In einer Basis-Sysplex-Umgebung werden zunächst die physischen Systeme durch Kanalverbindungen (*Channel to Channel*, CTC) zusammengeschlossen. Über eine (oder mehrere) gemeinsame Dateien, einem so genannten *Couple Data Set*, werden Informationen ausgetauscht. Wenn mehr als ein Prozessorkomplex beteiligt ist, muss mit Hilfe eines *Sysplex Timer*, der auch als *External Timer Reference* (ETR) bezeichnet wird, die Uhrzeit auf den beteiligten Systemen synchronisiert werden.

Merkmale eines Basis-Sysplex:

- Die maximale Kapazität ist mit der von LCMP (*Loosely Coupled Multi-Processing*) vergleichbar.
- Die Systemverfügbarkeit ist mit der von LCMP vergleichbar.
- Die Verwaltung der Arbeitslast ist gegenüber LCMP einfacher und kann vor allem weitgehend automatisiert werden.

Abbildung 2.1: Aufbau eines Basis-Sysplex



Von einer Konsole aus, die an einem MVS-System angeschlossen ist, können Befehle mit entsprechenden Präfixen auch für andere in dem Verbund angeschlossene Systeme abgegeben werden.

2.3 Parallel Sysplex

Seit der Einführung von Sysplex wurden die Technologien ständig erweitert. Mit Parallel Sysplex wird eine Möglichkeit zur Verfügung gestellt, Daten auf unterschiedlichen Systemen gemeinsam zu nutzen.

Coupling Facility

Von der Idee her ist eine ***Coupling Facility*** (CF) ein von den beteiligten Systemen genutzter Hauptspeicher, wobei für die Verwaltung und Koordination auch Rechenleistung benötigt wird. Technisch gesehen handelt es sich bei einer Coupling Facility um einen CMOS-Prozessor mit Hauptspeicher, der über Coupling Links mit den an einer Sysplex-Konfiguration beteiligten Prozessorkomplexen verbunden ist.

Eine Coupling Facility ermöglicht somit den Zugriff auf gemeinsame Speicherbereiche, der durch einen ***License Internal Code*** (LIC) verwaltet wird. Dieser LIC wird manchmal auch als ***Coupling Facility Control Code*** (CFCC) bezeichnet.

Die Coupling Facility als Schnittstelle gewährleistet die Integrität und Konsistenz von Daten, die von mehreren MVS-Systemen gleichzeitig verwendet werden. Diese Möglichkeiten machen eine Sysplex-Umgebung tauglich für parallele Verarbeitung, speziell für die Anwendungsbereiche transaktionsorientierter Systeme. Darum stehen hinter einer derartigen Konfiguration mit einer Coupling Facility auch die Begriffe „Data Sharing" oder „Parallel Sysplex".

Mehr als eine CF

In einem Sysplex-Verbund kann mehr als eine Coupling Facility eingesetzt werden. Im Sinne einer höheren Verfügbarkeit ist das gerade in einer Produktionsumgebung auch empfehlenswert. Als Alternative kann für den Backup einer Coupling Facility auch eine ***Internal Coupling Facility*** (ICF) konfiguriert werden, die die Funktionalität einer Coupling Facility in einer LPAR zur Verfügung stellt. Eine ICF ist eine speziell konfigurierte Processing Unit (PU) in einem zSeries-Rechner. Derzeit können maximal 32 MVS-Systeme zusammengeschaltet werden. Diese Anzahl wird künftig bei Bedarf noch erhöht werden. Immerhin sind damit heute bereits Verbünde mit über 30.000 MIPS (Million Instructions Per Second) möglich.

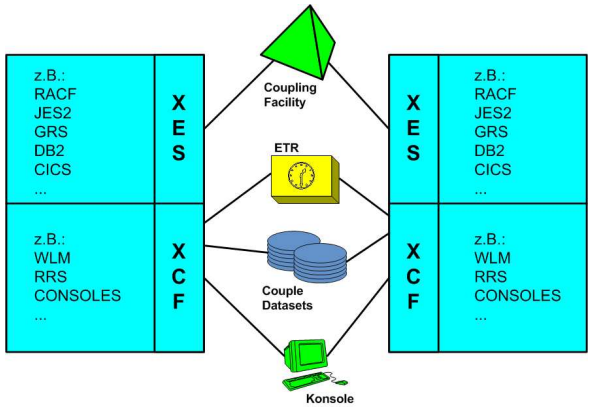
Merkmale eines Parallel Sysplex

- erhöhte maximale Kapazität gegenüber einem Basis-Sysplex
- erhöhte Verfügbarkeit gegenüber einem Basis-Sysplex

- Data Sharing über eine Coupling Facility
- bessere Systemverwaltung und besseres Datenmanagement
- Workload Balancing über die beteiligten Systeme hinweg

Anzumerken ist hier, dass in einer Produktionsumgebung alle kritischen Komponenten redundant vorhanden sein müssen. Das heißt beispielsweise, dass die in Abbildung 2.2 nur einmal vorhandenen Komponenten wie der External Timer und die Coupling Facility mindestens zweifach ausgelegt werden müssen, um den Ansprüchen höchster Verfügbarkeit zu genügen.

Abbildung 2.2: Beispiel einer Parallel-Sysplex-Konfiguration



XCF steht für *Cross Coupling Facility* und ermöglicht die Kommunikation bezüglich Sysplex für Systemkomponenten über so genannte XCF-Signale. Dafür ist (trotz des in diesem Zusammenhang irreführenden Namens) keine Coupling Facility notwendig.

XES steht für *Cross System Extended Services* und ermöglicht in einer Parallel-Sysplex-Konfiguration den Zugriff auf die Coupling Facilities. Hierfür stehen spezielle XES-Services zur Verfügung.

Nur in Verbindung mit einer Coupling Facility ist das Data Sharing möglich, wobei über entsprechend eingerichtete Strukturen Daten zwischen Systemen und Subsystemen mit sehr hohen Zugriffsraten gemeinsam genutzt („gesharet") werden können. Hinter einer Coupling Facility steckt vom Konzept her die Idee eines gemeinsam genutzten Hauptspeichers, wobei die Zugriffsgeschwindigkeiten natürlich langsamer sind als die auf den Hauptspeicher; darüber hinaus wird für die Verwaltung auch Verarbeitungskapazität in Form eines Prozessors benötigt.

Die Softwarekomponenten, die in Verbindung mit Parallel Sysplex eingesetzt werden sollen, müssen Sysplex und Data Sharing unterstützen. Hier gibt es zum einen Systemkomponenten, die das beherrschen, wie *Resource Access Control Facility*

(RACF, vgl. Abschnitt 8.4), *Global Resource Serialization* (GRS) oder das *Job Entry Subsystem* (JES), zum anderen auch Anwendungssysteme wie DB2 und CICS. Auch Fremdanbieter wie beispielsweise die Software AG mit Adabas unterstützen Parallel Sysplex, um ein Datenbanksystem als Beispiel zu erwähnen, das nicht von IBM selbst kommt.

Einige wichtige Begriffe in diesem Zusammenhang:

Sysplex Couple Datasets
: In den Sysplex Couple Datasets auf Datenträgern (Volumes), die zwischen den beteiligten Systemen gemeinsam genutzt werden, liegen Sysplex-bezogene Informationen, die im Hinblick auf Zugriffsgeschwindigkeiten unkritisch sind. Man unterscheidet die Standard Couple Datasets, auf denen die Basisinformationen über eine Sysplex-Konfiguration abgelegt sind, und funktionsbezogene (Functional) Couple Datasets, die beispielsweise für den Workload Manager oder für die Restart Recovery Services benötigt werden.

Sysplex Timer
: Ein Sysplex Timer (HW-Box) wird benötigt, wenn mehrere physische Prozessorkomplexe zusammen eine Sysplex-Umgebung bilden. Dann müssen die Uhrzeiten auf den beteiligten Systemen synchronisiert werden. Dies ist sehr wichtig, da in einer Sysplex-Umgebung mehrere DB-Systeme auf gemeinsam genutzte Daten zugreifen. Logfiles und Zeitstempel (*Timestamps*) spielen eine entscheidende Rolle für die Sicherung der Datenintegrität und das reibungslose Funktionieren von Backup/Recovery-Mechanismen.

Coupling Facility
: Eine Coupling Facility ermöglicht parallele Verarbeitung und verbessertes Data Sharing für autorisierte Anwendungen in einer Sysplex-Umgebung. Es handelt sich dabei um einen CMOS-Prozessor ohne virtuelle Speicherfunktionalitäten, d. h. er ist nur mit Hauptspeicher ausgestattet. In der Coupling Facility werden List-, Lock- und Cache-Strukturen verwaltet.

Coupling Links
: Der Zugriff auf die Coupling Facility wird über schnelle Coupling Links mit hoher Übertragungsrate auf Lichtwellenbasis ermöglicht.

Cross System Coupling Facility
: Bei XCF handelt es sich um eine SW-Komponente des Betriebssystems, die das System Management in Verbindung mit Sysplex-Umgebungen vereinfacht. Wenn ein System ausfällt, ermöglicht XCF beispielsweise den automatischen Restart von Batchjobs und Started Tasks auf einem anderen System in einer Sysplex-Umgebung.

Cross System Extended Services

Die Cross-System-Extended-Services-Komponente (XES) des Betriebssystems ermöglicht es entsprechend autorisierten Anwendungen, die Vorteile einer Sysplex-Umgebung bezüglich Data Sharing zu nutzen. XES ermöglicht letztendlich die Sichtweise eines Single System Image über die Zugriffe auf die Coupling Facilities.

Coupling Facility Structures

Eine Coupling Facility fungiert als High-Speed-Device für den gemeinsamen Zugriff auf Informationen von mehreren Systemen aus. Für die Verwaltung der Informationen stehen drei so genannte „Strukturen" zur Verfügung:

Cache-Strukturen

Cache-Strukturen beinhalten einen Mechanismus, um die Konsistenz der gecacheten Daten zu gewährleisten (*Buffer Invalidation*). Cache-Strukturen können als Hochgeschwindigkeitspuffer für Read/Write-Zugriffe verwendet werden.

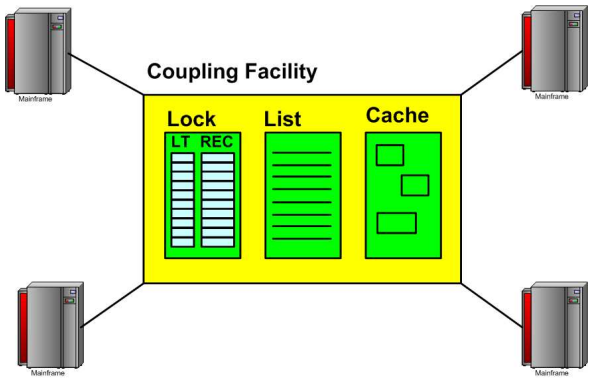
List-Strukturen

Mit List-Strukturen verwalten autorisierte Anwendungen gemeinsame Daten, die in Listen organisiert sind, um gemeinsame Queues und Statusinformationen (z. B. Journaling oder Logstreams) mitzuführen.

Lock-Strukturen

Lock-Strukturen stellen Lock-Mechanismen (*shared* und *exclusive*) für die Serialisierung gemeinsam genutzter Ressourcen zur Verfügung.

Abbildung 2.3: Coupling-Facility-Strukturen



Parallele Verarbeitung

Parallele Verarbeitung ist eine Technik, um die Verfügbarkeit, Performance und Skalierbarkeit von Anwendungen zu erhöhen. Eine bestimmte Workload kann auf meh-

reren *Central Processor Complexes* (CPCs) gleichzeitig abgearbeitet werden. Es können mehrere Server eingesetzt werden, um besseren Durchsatz und schnellere Response-Zeiten zu erzielen. Die Koordination läuft über die Coupling Facilities.

Das Konzept ist grundsätzlich nicht neu. Für wissenschaftliche Anwendungen wird parallele Verarbeitung schon lange eingesetzt. Mit Sysplex lässt sich nun die Parallelisierung auch für kommerzielle Online-Transaktionen realisieren. Anwendungen können in gekoppelten Mehrprozessorsystemen repliziert werden, um dann auch in mehreren Systemen gleichzeitig und nach Bedarf aktiviert zu werden.

GDPS

GDPS steht für *Geographically Dispersed Parallel Sysplex* und ist ein Konzept der IBM, um ein Management für verteilte Sysplex-Umgebungen aufzubauen und damit geplante und ungeplante Ausfälle zu eliminieren bzw. deren Auswirkung zu begrenzen.

Es handelt sich dabei um eine Kombination von Systemprogrammen und Automatisierungsroutinen in Verbindung mit der Parallel-Sysplex-Technologie und der Spiegelung von Daten bzw. Datenbanken.

Die Idee dabei ist, komplette Site-Switches bei geplanten oder ungeplanten Systemausfällen in wenigen Minuten durchführen zu können, ohne dass Datenverluste auftreten.

GDPS ist somit eine Lösung für die Verwaltung einer Remote Copy Configuration in Kombination mit Speichersubsystemen, wobei Automatisierungsroutinen in Verbindung mit Parallel Sysplex von einem Single Point of Control aus für die entsprechende Verfügbarkeit der beteiligten Komponenten sorgen. GDPS unterstützt diverse Transaktionsmanager (z. B. CICS oder IMS/DC) und Datenbankmanager (z. B. DB2, IMS oder VSAM) und wird über die folgenden Schlüsseltechnologien realisiert:

- Parallel Sysplex
- Systems Automation for z/OS
- Enterprise Storage Server (ESS)
- Peer-to-Peer Remote Copy (PPRC)
- Extended Remote Copy (XRC)
- Fiber Saver (2029)

GDPS unterstützt somit sowohl synchrone Formen mit Peer-to-Peer Remote Copy (PPRC) und zum anderen asynchrone Formen mit *Extended Remote Copy* (XRC).

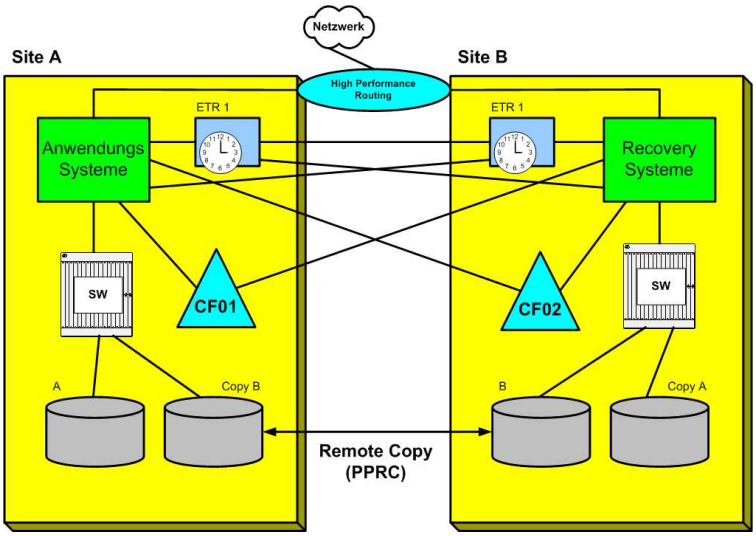
GDPS/PPRC

Mit GDPS/PPRC werden Daten auf so genannten „primary" Disks auf der Anwendungsseite synchron auf „secondary" Disks in einem entfernten Rechenzentrum gespiegelt. Erst wenn das Storage Subsystem auf der Anwendungsseite ein „Write complete" übermittelt bekommt, ist der I/O-Vorgang beendet.

Die Topologie von GDPS/PPRC besteht aus einem Basis- oder Parallel Sysplex Cluster, der auf zwei Rechenzentren aufgeteilt ist, die bis zu 40 Kilometer voneinander entfernt sein können. Der Parallel Sysplex Cluster muss mit redundanter Hardware ausgestattet sein, beispielsweise auf jeder Seite jeweils eine Coupling Facility und jeweils ein Sysplex Timer.

In Abbildung 2.4 werden die kritischen Daten via PPRC von der Site A auf die Site B gespiegelt.

Abbildung 2.4: Datenspiegelung via PPRC



Kapitel 3

Die zSeries-Architektur

Im Oktober 2000 stellte IBM mit der zSeries eine neue Architektur für den High-End-Bereich vor und richtete zugleich ihre Server-Reihen neu aus. Aus den zuvor unter dem Namen „Netfinity" vermarkteten PC-Servern wurde die xSeries, aus den RISC-Systemen mit dem Betriebssystem AIX (ein UNIX-Derivat) die pSeries, aus den AS/400-Systemen die iSeries und aus den Mainframes die zSeries. Das „z" steht dabei für „Zero Outage", also die Möglichkeit, bei richtiger Konzeption und Implementierung eine Systemplattform auf die Beine zu stellen, die rund um die Uhr, sieben Tage pro Woche ohne Unterbrechung betrieben werden kann.

3.1 Hardware-Technologie

Die z900-Server bauen auf den sechs Vorgängergenerationen (G1 bis G6) der S/390 CMOS-Server auf.

z900-Prozessortechnologie

Statt bisher maximal 12 Prozessoren in einem Prozessorverbund sind nun maximal 16 Prozessoren möglich (ohne dass die physischen Kästen größer wurden). Das Herz ist dabei ein *Multi Chip Module* (MCM), auf dem auf den z900-Systemen bis zu 35 CMOS-Chips dicht gepackt untergebracht sind. Auf der neuen z990-Reihe sind es gar 48 CMOS-Chips, die auf vier MCMs verteilt sind. Dadurch wurde sowohl die Geschwindigkeit erhöht (kürzere Pfade) als auch die Verlässlichkeit (weniger Teile pro MCM) weiter gesteigert.

Abbildung 3.1: zSeries 900[1]

Die 35 CMOS Chips der z900 setzen sich zusammen aus 20 so genannten Processor Units (PUs), acht Level-2-Cache Chips, zwei Storage Control Chips, vier Memory Bus Adapter Chips und einem Clock Chip.

Auf einer Fläche von 127 x 127 mm sind auf einem MCM der z900 rund 2,5 Milliarden Transistoren untergebracht. Die Anzahl der Transistoren pro Chip wurde von 25 Millionen (G6-Prozessoren) auf 47 Millionen fast verdoppelt.

Das Wesentliche bei dieser Mainframe-Architektur liegt jedoch nicht allein in der CPU-Leistung, sondern in der Gesamtleistung des Systems. Wenn es um reine CPU-Leistung geht, gibt es zahlreiche andere Systeme, die dem IBM Mainframe ebenbürtig oder sogar überlegen sind. Seine Stärken kann der IBM Mainframe dann ausspielen, wenn es darüber hinaus um Datenverarbeitung im wahrsten Sinne des Wortes geht, wenn Daten in großen Mengen zwischen CPU, Hauptspeicher und externen Geräten hin und her transportiert werden müssen.

[1] Abdruck mit freundlicher Genehmigung der IBM.

Eine große Rolle für die Leistungsfähigkeit spielt auch der in Abbildung 3.2 dargestellte Level 2 Shared Cache, der, wie der Name bereits aussagt, von den beteiligten Processing Units gemeinsam genutzt wird.

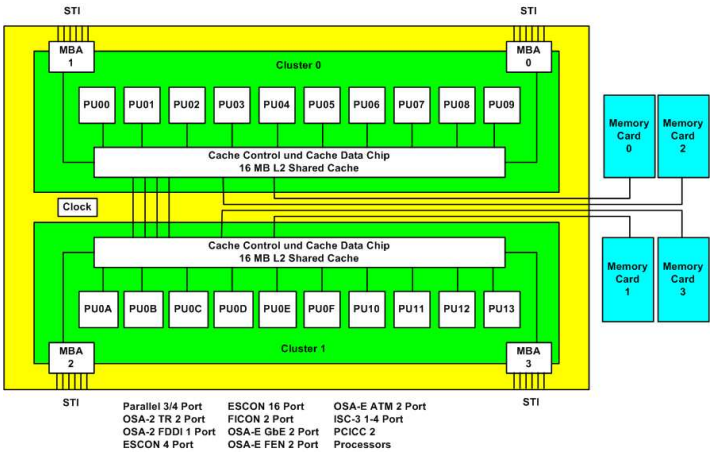


Abbildung 3.2: Layout des Multi-Chip-Moduls (MCM)

I/O-Technologie

Die interne I/O-Bandbreite wurde bei der z900 gegenüber den G5/G6-Rechnern verdreifacht, und zwar durch einen neuen 1 GB Self-Timed Interface (STI) Bus. Es gibt sechs STIs für jeden der vier Memory Bus Adapter. Somit wird eine I/O-Bandbreite von 24GB/s für alle zSeries-900-Modelle erreicht.

Die Anzahl anschließbarer FICON-Verbindungen wurde bei der z900 auf 96 erhöht. Eine FICON- hat die achtfache Kapazität einer ESCON-Verbindung. Bis zu 10 Kilometer können ohne, bis zu 20 Kilometer mit Repeater überbrückt werden. An einem FICON-Kanal können bis zu 16144 Endgeräte (im Vergleich zu 1024 bei ESCON-Kanälen) angeschlossen werden.

Durch das *Dynamic CHPID Management* (DCM) kann nun die volle Bandbreite von 256 Kanälen genutzt werden.

3.2 Speichertechnologie

Eine der wesentlichen Änderungen bezüglich der Architektur ist die 64-Bit-Adressierung. Damit wird der Expanded Storage überflüssig, der im Zuge der XA-Architektur eingeführt wurde, um die Einschränkung der 2-GB-Grenze für die Adressierung des Hauptspeichers erträglicher zu gestalten. Anstatt nicht benötigte Daten

per Paging auf externe DASD-Geräte auszulagern, wurde im Prozessorbereich eine Hauptspeichererweiterung realisiert, damit im so genannten Processor Storage als Kombination aus Hauptspeicher und Expanded Storage ein Page Movement stattfinden konnte. Dieser nicht unerhebliche Overhead ist nun mit der 64-Bit-Architektur überflüssig.

Expanded Storage wird in Verbindung mit der 31-Bit-Architektur weiterhin unterstützt. Für die 64-Bit-Architektur wird der Expanded Storage in Verbindung mit z/VM und darunter liegenden Gastsystemen unterstützt.

Der Hauptspeicher liegt bei der z900-Serie zwischen 5 und 64 GB. Die maximalen 64 GB können von einem einzigen System genutzt oder in einer LPAR-Konfiguration aufgeteilt werden. Da es von der Architektur her in absehbarer Zeit keine Beschränkung nach oben gibt, ist zu erwarten, dass die Größe des Hauptspeichers künftig flexibel mit den Anforderungen der Anwender erweitert werden wird.

3.3 Der Intelligent Resource Director (IRD)

Eine weitere neue und interessante Einrichtung der zSeries-Architektur ist der *Intelligent Resource Director*, der es ermöglicht, Systemressourcen dynamisch zwischen logischen Partitionen zu verteilen. Eine Erweiterung der bestehenden Parallel-Sysplex-Architektur erlaubt somit den dynamischen Einsatz von Systemressourcen in einer Sysplex-Umgebung. Diese Erweiterung steht unter dem Begriff *Multiple Clustered LPARs*. Es handelt sich dabei um eine Integration von PR/SM, WLM und Parallel Sysplex. Der *Workload Manager* (WLM) ist die Komponente im z/OS-Betriebssystem, die für die Verteilung der Ressourcen und die Prioritätensteuerung zuständig ist.

Die *CPU Weights* einer LPAR-Konfiguration können beispielsweise nun dynamisch in Verbindung mit den WLM Goals angepasst werden. Während vor der zSeries-Architektur die Gewichtungsfaktoren für die LPARs fest in der *Hardware Management Console* (HMC) definiert werden mussten, ist der IRD in der Lage, in Abstimmung mit dem WLM diese Gewichtungsfaktoren dynamisch anzupassen.

Der Intelligent Resource Director nutzt das Konzept eines LPAR Cluster, d. h. einen Satz von z/OS Images, die in logischen Partitions auf dem gleichen Prozessorkomplex in einem gemeinsamen Sysplex-Verbund laufen.

LPAR Cluster betrachten die LPARs auf dem gleichen CPC als Pool gemeinsamer Ressourcen. Die momentane Implementation nutzt drei getrennte, aber sich gegenseitig ergänzende Funktionen:

Das *Dynamic CPU Management* ermöglicht es dem WLM im Goal Mode, die CPU-Gewichtungen und die logischen Prozessoren in einem LPAR Cluster dynamisch zu verwalten.

Das *Dynamic Channel-Path Management* (DCM) sorgt für die dynamische Verteilung der Kanalverbindungen zu den logischen Partitionen. Bisher mussten die I/O-Pfade fest zwischen Prozessoren und Control Units verbunden werden. Mit zSeries und z/OS können die Pfade dynamisch den Control Units zugewiesen werden, abhängig von der jeweiligen I/O-Auslastung der Konfiguration.

Und schließlich wird mit dem Intelligent Resource Director das Konzept des *I/O Priority Queuing* auf das Kanalsubsystem ausgeweitet. Die Prioritäten der Workload können nun mit den Prioritäten der LPARs abgestimmt und dynamisch angepasst werden.

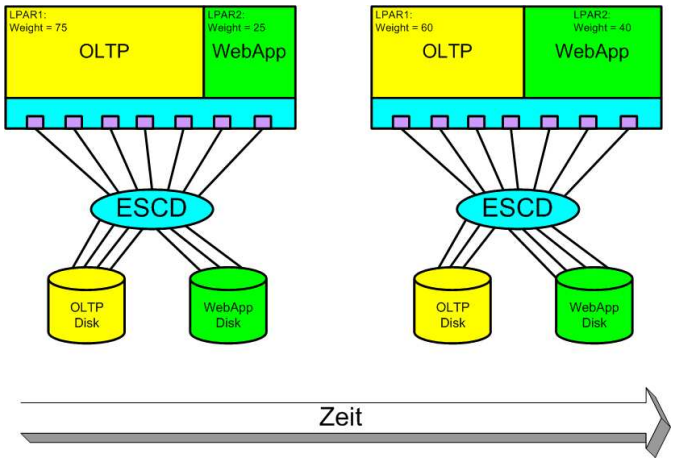


Abbildung 3.3: Wirkung des Intelligent Resource Director

In dem Beispiel von Abbildung 3.3 wird dieselbe Konfiguration zu zwei unterschiedlichen Zeitpunkten dargestellt. Ein *Online Transaction Processing System* (OLTP) läuft in einer LPAR1 und eine E-Business-Anwendung in einer LPAR2. Im Laufe der Zeit stellt der Workload Manager fest, dass die E-Business-Anwendung die definierten Ziele nicht erreicht, die OLTP-Anwendung dagegen die definierten Ziele übererfüllt. Der IRD nimmt deshalb dynamisch eine Anpassung vor, indem er die CPU-Gewichtung für die OLTP-Anwendung von 75 auf 60 reduziert und die CPU-Gewichtung der E-Business Anwendung von 25 auf 40 erhöht. Außerdem wird der OLTP-Anwendung ebenfalls dynamisch ein Kanal weggenommen und der E-Business-Anwendung zur Verfügung gestellt.

Das Entscheidende dabei ist, dass dies alles vom System selbst vorgenommen wird, ohne dass ein Systemtechniker oder ein Operator ein Kommando eingeben muss oder sonst in irgendeiner Weise aktiv wird.

Die Vorteile: Effizientere Nutzung der Hardware-Ressourcen und weniger Kanäle. Die Planung der Konfiguration wird einfacher, und die Ressourcen können dynamisch zugeordnet werden.

3.4 HiperSockets

Ein neues Konzept vor allem im Zuge der Serverkonsolidierung und der Zusammenarbeit der unterschiedlichen Systeme auf einem großen physischen Rechner ist die interne Kommunikationsfähigkeit mit HiperSockets.

Während in der Vergangenheit der allseits bekannte Slogan „Das Netzwerk ist der Computer" für Aufmerksamkeit sorgte, wird dieser im Zusammenhang mit HiperSockets umgekehrt: „Der Computer ist das Netzwerk".

Die bisher übliche Ansammlung von Spezialservern mit Gateway-, Firewall- und sonstigen Diensten verursacht ein erhebliches Maß an Komplexität und Kosten. Performance, Verfügbarkeit, Stabilität sowie die Response-Zeiten werden davon direkt beeinflusst. Je mehr Server involviert sind, desto größer wird logischerweise die Komplexität und damit auch die Anfälligkeit für Störungen und Ausfälle.

Das Konzept der HiperSockets bietet nun die Möglichkeit der Konsolidierung in eine E-Infrastruktur mit einem Server-to-Server-Netzwerk, das ohne Kabel auskommt und viele Netzwerk-Verzögerungen (Latencies) eliminiert.

HiperSockets ermöglichen eine Hochgeschwindigkeits-TCP/IP-Kommunikation zwischen Servern, die in unterschiedlichen LPARs einer zSeries-Hardware laufen. Die Kommunikation wird über den Prozessorspeicher abgewickelt. Die virtuellen Server werden so verbunden, dass sie ein „virtuelles LAN" bilden. Die interne Geschwindigkeit dieser HiperSockets beträgt 500 GB/s.

HiperSockets nutzen ein Konzept, das als *internal Queued Input/Output* (iQDIO) bezeichnet wird. Es handelt sich dabei um eine Microcode-Funktion eines zSeries CEC (Central Electronic Complex). Es wird von folgenden Betriebssystemen unterstützt:

- z/OS ab V1R2
- z/OS.e
- z/VM ab V4R2
- Linux for zSeries (31- und 64-Bit-Mode)

Über HiperSockets können bis zu vier (ab der z/990-Serie bis zu 16) unabhängige virtuelle LANs gebildet werden, die als TCP/IP-Netzwerke in einem zSeries-Prozessorkomplex betrieben werden.

Vorteile der HiperSockets

Mit HiperSockets kann zwischen konsolidierten Servern innerhalb eines physischen Prozessorkomplexes kommuniziert werden. Dahinter steht die Idee, dass zahlreiche Hardware-Boxen einschließlich der sie verbindenden Netzwerkkomponenten

wie Router, Switches, Kabel etc. eliminiert werden. Dies wirkt sich im Hinblick auf Wartung und Kosten äußerst positiv aus.

Konsolidierte Server haben mit Prozessorgeschwindigkeit Zugriff auf die Geschäftsdaten und Transaktionen im klassischen Mainframe-System, ohne dass Netzwerk-Overhead mit entsprechenden Verzögerungen entsteht. Da sich die ganze Server-to-Server-Kommunikation innerhalb eines Prozessorkomplexes abspielt, kann eine deutlich höhere Verfügbarkeit, Performance, Sicherheit und Kosteneffizienz erreicht werden.

Die Transporteigenschaften (z. B. die maximale Framesize) können an die unterschiedlichen Anforderungen angepasst werden. Im Gegensatz dazu haben Ethernet- und Token Ring-LANs eine maximale Framesize, die von der LAN-Architektur abhängt.

Unter Sicherheitsgesichtspunkten und aus anderen Gründen wie beispielsweise Isolation von Anwendungen können Server über unterschiedliche HiperSockets verbunden werden. Alle Security-Features wie beispielsweise auch Firewall-Filtering stehen für HiperSockets Interfaces genauso wie für alle anderen TCP/IP-Schnittstellen zur Verfügung, denn HiperSockets präsentieren sich so wie alle anderen TCP/IP-Schnittstellen. Somit sind sie transparent für Anwendungen und für das Betriebssystem. HiperSockets können darüber hinaus auch die TCP/IP-Kommunikation in einer Sysplex-Umgebung verbessern, wenn die DYNAMICXCF Facility eingesetzt wird.

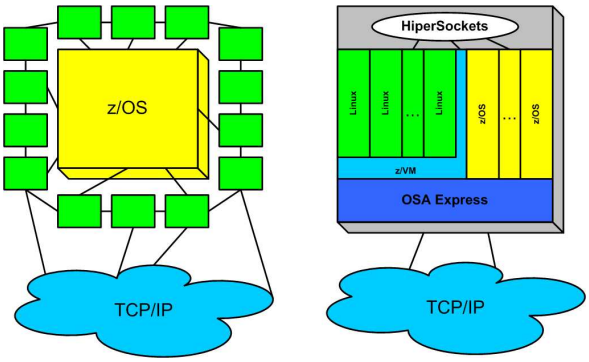


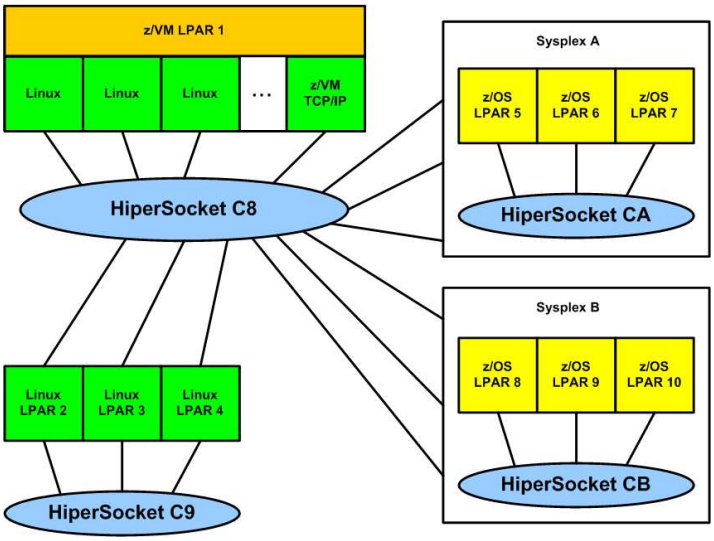
Abbildung 3.4: Server-to-Server-Kommunikation über HiperSockets

Die angeschlossenen physischen Systeme, die in herkömmlichen Umgebungen auch physisch verkabelt sind, werden in einem einzigen physischen Rechner konsolidiert. Ein entscheidender Vorteil liegt in der effizienteren Ausnutzung der Ressourcen, die in dezentralen Umgebungen meist nur wenig ausgelastet sind.

Jeder HiperSocket wird über einen *Channel Path Identifier* (CHPID) identifiziert. Wie bei anderen I/O-Operationen auch, wird eine HiperSocket-Schnittstelle über eine Gerätenummer angesprochen, die über *Hardware Configuration Definition* (HCD) spezifiziert wird.

Es gibt zahlreiche Konfigurationsmöglichkeiten. Die Abbildung 3.5 zeigt zum Beispiel eine zSeries-Konfiguration in einem z900-Prozessorkomplex.

Abbildung 3.5: Beispiel für eine zSeries-Konfiguration in einem z900-Prozessorkomplex



HiperSockets und Microcode

Wie bereits erwähnt, basiert die HiperSocket-Implementation auf Queued Direct Input/Output (QDIO). Der Microcode emuliert dabei den Link Control Layer einer OSA-Express-QDIO-Schnittstelle. Bevor ein Paket über ein herkömmliches LAN transportiert werden kann, muss ein LAN Frame gebildet und die MAC-Adresse des Zielrechners oder eines Routers im LAN in den Frame aufgenommen werden. HiperSockets dagegen benötigen weder LAN Frames, Zielrechner noch Router. TCP/IP Stacks werden über Inbound-Data-Queue-Adressen anstelle von MAC-Adressen angesprochen.

Das z/OS ermöglicht das Einrichten mehrerer TCP/IP Stacks. Die I/O-Steuerung der Geräte wird jedoch nur einmal pro Image benötigt und über VTAM gesteuert. Die *Virtual Telecommunication Access Method* ist der zentrale Adressraum im Betriebssystem für alles, was mit Telekommunikation zu tun hat.

Virtuelle Server und LPARs, in denen z/OS, z/VM, Linux for OS/390 und Linux for zSeries über HiperSockets miteinander verbunden sind, können beliebig kombiniert werden.

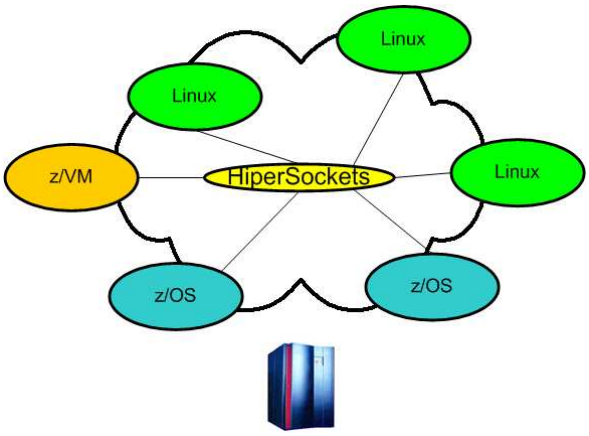


Abbildung 3.6: HiperSockets

3.5 zSeries-Konfigurationen

Mit der z900-Serie stehen 24 neue CPU-Modelle zur Verfügung. In den Modellen sind bis zu 20 Processing Units (PUs) eingebaut. Die Zykluszeit beträgt 1,3 ns.

Eine PU kann eingesetzt werden als:

Central Processor (CP)

Ein Central Processor (CP) ist eine PU mit einem z/Architecture-Instruktionssatz, wie man sie als klassische CPU kennt. Sie kann in unterschiedlichen Modi betrieben werden:

- ESA/390
- z/Architecture
- TPF
- Linux

System Assist Processor (SAP)

Ein System Assist Processor (SAP) ist eine PU, die mit dem Channel Subsystem Microcode betrieben wird, um I/O-Operationen zu steuern. Eine der SAPs in einer Konfiguration ist die Master SAP, die auch für die Kommunikation zwischen dem Multi Chip Module (MCM) und dem Support Element (SE) sorgt.

Internal Coupling Facility (ICF)

Eine Internal Coupling Facility (ICF) ist eine PU, in der der ***Coupling Facility Control Code*** (CFCC) für Parallel-Sysplex-Umgebungen läuft. Sie kann beispielsweise in einer Sysplex-Testumgebung oder als Backup für eine physische Coupling Facility eingesetzt werden.

Integrated Facility for Linux (IFL)

Der Grund dafür, dass es die IFLs gibt, ist die Art und Weise, wie im IBM-Mainframe-Bereich Lizenzkosten für Software verrechnet werden, nämlich – zumindest in der Vergangenheit – auf der Basis der installierten CPU-Kapazität.

Wenn nun jedoch im größeren Stil Linux eingesetzt wird, wobei die unter Linux betriebenen Anwendungen oft Open-Source-Anwendungen sind, würde das heißen, dass die für Linux zusätzlich benötigten CPU-Ressourcen die Lizenzgebühren der Software, die unter z/OS eingesetzt wird, in die Höhe treiben würden.

Eine als IFL konfigurierte Processing Unit ist für die Berechnung der Softwarelizenzen „unschädlich". Eine Integrated Facility for Linux ist eine PU, die für Linux „native" oder für z/VM in Verbindung mit Linux eingesetzt wird. Ein IFL-Prozessor kann dediziert oder von mehreren Linux- oder VM Partitions gemeinsam betrieben werden.

zSeries Application Assist Processor (zAAP)

Im April 2004 wurde eine weitere Möglichkeit angekündigt: Prozessoren können als zSeries Application Assist Processors (zAAPs) konfiguriert werden. Ein zAAP wird auch als Java-Prozessor bezeichnet. Der Hintergrund ist der gleiche wie bei den Linux-Prozessoren: Der Einsatz von Prozessorleistung, die von Java Virtual Machines benötigt wird, soll sich nicht negativ auf die Lizenzkosten für Software auswirken. Die Konfiguration als zAAP wird erst mit der z990-Serie unterstützt.

Nicht zugewiesene PUs werden als „spare PUs" bezeichnet und dynamisch bei Ausfällen hinzugeschaltet. Dadurch erhöht sich natürlich die Verfügbarkeit des Systems. Außerdem können sie im Zuge dynamischer Upgrades verwendet werden: ***Capacity Upgrade on Demand*** (CUoD) oder ***Capacity BackUp*** (CBU). Eine Standardkonfiguration hat mindestens eine spare PU.

LPAR-Konfigurationsbeispiel

Eine Rechnerkonfiguration wird von einem Systemspezialisten über eine ***Hardware Management Console*** (HMC) vorgenommen. Damit werden die logischen Partitionen festgelegt und diesen der jeweils benötigte Hauptspeicher zugeordnet.

Im Beispiel aus Abbildung 3.7 werden sieben Processing Units eingesetzt, davon vier als CPs, eine als ICF und zwei als IFLs.

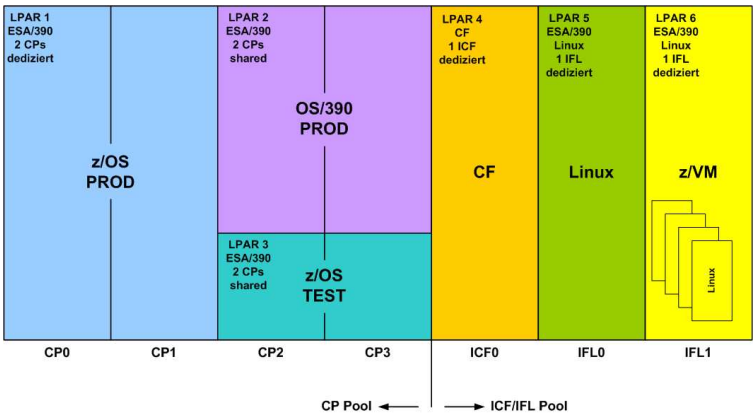


Abbildung 3.7: zSeries-Konfigurationsbeispiel

Der LPAR 1 werden zwei CPs für ein produktives z/OS dediziert zugeordnet. Zwei weitere CPs werden von den LPARs 2 und drei gemeinsam genutzt. Die LPAR 4 wird als Internal Coupling Facility als Backup für eine physische Coupling Facility genutzt. In der LPAR 5 wird ein Linux System „native" gefahren, und in der LPAR 6 ist ein z/VM konfiguriert, unter dem sehr viele Linux-Systeme virtuell betrieben werden können.

Die Vorteile einer derartigen Konfiguration: Das klassische Mainframe-Betriebssystem z/OS bzw. OS/390 wird auf dem gleichen physischen Rechner eingesetzt wie die Linux-Systeme, die entweder nativ in einer LPAR oder unter z/VM eingerichtet werden können. Für die Verbindung zwischen den Systemen werden Hipersockets eingesetzt mit einer Bandbreite von 500 GB/s. Die interne Coupling Facility wird als Backup für eine physische Coupling Facility konfiguriert.

3.6 Die neueste Rechnerserie: z990

Die derzeit leistungsfähigste Rechnerreihe z990 mit dem Codenamen „T-Rex" wurde im Mai 2003 angekündigt und brachte eine bedeutende Leistungssteigerung in allen Bereichen mit sich.

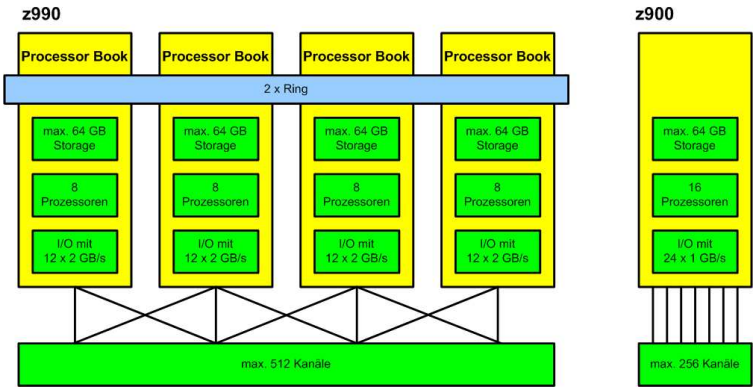
Prozessorleistung

Die Prozessorleistung kann über so genannte *Processor Books* ausgebaut werden, wobei jedes Book prinzipiell ein von der z990-Reihe bekanntes Multi-Chip-Modul (MCM) darstellt. Auf jedem Book befinden sich mehr als drei Milliarden Transistoren.

Die Skalierbarkeit ist durch die Konfiguration über die Processor Books noch größer, da von der Größenordnung her jedes Processor Book einer z900 entspricht.

Zwischen einem und vier Books mit jeweils zwölf Processing Units (PUs) können in einen Central Electronic Complex (CEC) eingeschoben werden. Maximal sind das somit 48 PUs. Von diesen zwölf Processing Units pro Modul sind zwei als System Assist Processor (SAP) konfiguriert, zwei weitere stehen als Ersatz-PUs (spare PU) für den Fall eines Ausfalls einer PU zur Verfügung. Somit sind maximal 32 Processing Units frei konfigurierbar.

Abbildung 3.8: z990



Das Einschieben eines weiteren Processor Book kann im laufenden Betrieb erfolgen. Die Verbindung zwischen den Books wird über einen Shared-Level-2-Cache realisiert. Im Maximalausbau wird eine Leistung von über 9000 MIPS (Million Instructions Per Second) erreicht.

LPARs

Bis zu 30 LPARs können konfiguriert werden (bisher maximal 15). Für die Zukunft ist eine Erweiterung auf bis zu 60 LPARs geplant.

Mit Hilfe des Internal Resource Director (IRD) können den LPARs dynamisch Ressourcen (CPU, Kanäle) zugeordnet werden.

I/O-Konfiguration

Die interne Kommunikation über HiperSockets läuft mit bis zu 500 GB/s, die Kommunikation mit der Außenwelt mit bis zu 96 GB/sec. Maximal 115 FICON/FCP-Verbindungen und maximal 512 ESCON-Verbindungen können konfiguriert werden.

Jedes Processor Book verfügt über zwölf Self Timed Interfaces (STIs), deren Geschwindigkeit im Vergleich zur z900 von 1 auf 2 GB/s erhöht wurde.

Weitere Kennzahlen

Der maximale Hauptspeicherausbau wurde auf 256 GB erhöht. Die Modellreihe umfasst die Modelle A08, B16, C24 und D32, wobei die Nummer der Anzahl konfigurierbarer PUs entspricht. Pro Processor Book werden zwei System-Assist-Prozessoren und zwei Ersatz-PUs zur Verfügung gestellt, die nicht frei konfiguriert werden können.

Kapitel 4

Konzept des virtuellen Speichers

Die Virtualisierung von Ressourcen spielt in heutigen Rechnersystemen eine große Rolle. Es geht darum, durch Virtualisierung mehr Kapazitäten zur Verfügung zu stellen, als real vorhanden sind bzw. die vorhandenen realen Kapazitäten effizienter zu nutzen.

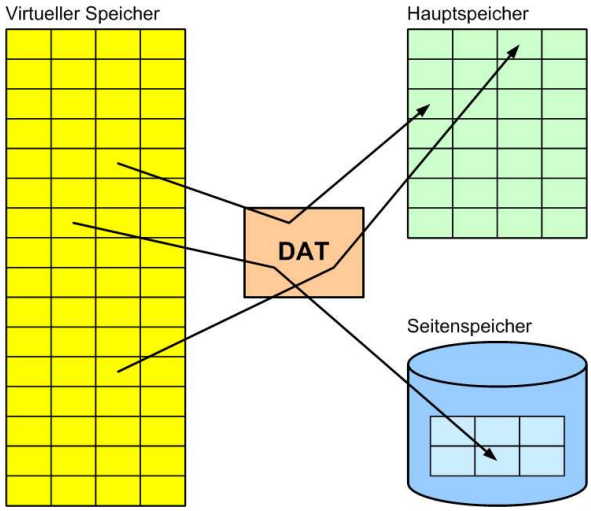
Das Konzept des virtuellen Speichers wurde auf dem IBM-Mainframe im Jahr 1972 eingeführt. Es sorgt für Unabhängigkeit von Programmen jeglicher Größe von dem zur Verfügung stehenden physischen Speicher.

Ziel ist es, Programmteile und Daten, die im Programmablauf gerade benötigt werden, im Hauptspeicher zu halten, und jene, die gerade nicht benötigt werden, auf einen externen Speicher auszulagern. Der virtuelle Speicher ist der über eine virtuelle Adresse ansprechbare Speicherbereich, der physisch entweder im Hauptspeicher oder auf einem externen Speicher, der auch als Seitenspeicher bezeichnet wird, abgebildet wird.

4.1 Dynamic Address Translation (DAT)

Die Umsetzung einer virtuellen Adresse in eine reale Adresse wird über die *Dynamic Address Translation* (DAT) vorgenommen.

Abbildung 4.1: Prinzip des virtuellen Speichers



Die Übertragungseinheit zwischen Hauptspeicher und externem Speicher ist jeweils eine Seite (*Page*) mit 4 KB. Der Platz für eine Seite im Hauptspeicher wird als *Frame* bezeichnet und der Platz für eine Seite auf dem externen Speicher als *Slot*. Den Vorgang des Aus- bzw. Einlagerns nennt man *Paging*.

Die Dynamic Address Translation (DAT) benutzt einen zweiteiligen Tabellenmechanismus. Hierfür wird ein Adressraum (das ist der von einem Benutzer ansprechbare Speicherbereich) in Segmente und Pages eingeteilt. Dementsprechend gibt es eine Segmenttabelle pro Adressraum und zahlreiche Page-Tabellen.

Ursprünglich arbeitete man im Multiple Virtual Storage (MVS) mit einer 24-Bit-Adresse. Dadurch war ein virtueller Speicher von 16 MB adressierbar. Mit der Extended Architecture (XA) wurde die 31-Bit-Adressierung eingeführt und mit der zSeries-Architektur die 64-Bit-Adressierung. Aus Darstellungsgründen und um hier auch die Historie aufzuzeigen, beginnen wir mit der 24-Bit-Adressierung.

Die virtuelle Adresse wird aufgeteilt in drei Bereiche. Bei der 24-Bit-Adresse sind es 8 Bits für den Segmentteil, 4 Bits für den Page-Teil und 12 Bits für das Displacement. Wenn ein Adressraum die Kontrolle zur Verarbeitung bekommt, sorgt das System dafür, dass ein spezielles Systemregister, das *Segment Table Origin Register* (STOR), auf den Beginn der Segmenttabelle zeigt. Damit ist der Start für die Adressierung eines Adressraums gegeben. Mit dem Segmentteil der Adresse wird ein Eintrag in der Segmenttabelle gefunden, der auf den Beginn einer Page-Tabelle

zeigt. Mit dem Page-Teil der Adresse wird wiederum ein Eintrag in der Page-Tabelle gefunden, der auf den Frame im Hauptspeicher zeigt. Mit dem Displacement kann dann innerhalb des Frame im Hauptspeicher das Byte gefunden werden, das mit der 24-Bit-Adresse angesprochen werden soll.

Erwähnenswert ist die Tatsache, dass zwar von der Logik her mit der virtuellen Adresse auf den virtuellen Speicher zugegriffen wird, physisch jedoch der Zugriff über die Adressumsetzung mit den Tabellen dann im Hauptspeicher erfolgt. Sollte die Seite ausgelagert sein, wird bei der Adressumsetzung eine Page-Fault-Bedingung ausgelöst, und es gibt eine Unterbrechung. Die Seite muss dann erst vom Slot auf dem externen Speicher in einen Frame in den Hauptspeicher gebracht werden. Eine Seite kann somit flexibel zwischen Hauptspeicher und externem Speicher hin und her geschoben werden. Die Dynamic Address Translation sorgt dafür, dass jeweils auf die richtigen Informationen zugegriffen wird.

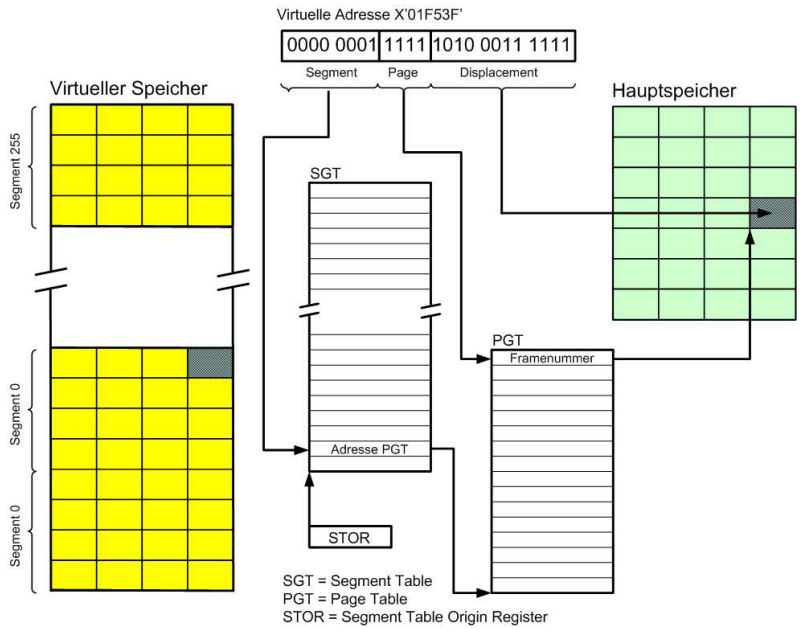


Abbildung 4.2: Tabellenmechanismus der DAT

Wann Seiten ausgelagert werden müssen, wird über einen Zähler mit der Bezeichnung *Available Frame Queue* (AFQ) gesteuert. Wie diese Bezeichnung aussagt, wird damit gekennzeichnet, wie viele Frames im Hauptspeicher verfügbar sind. Fällt der Wert der AFQ unter einen definierten Schwellenwert, werden automatisch Seiten ausgelagert. Für diese Auslagerung kommen die Seiten in Frage, auf die am längsten nicht mehr zugegriffen wurde. Dies wird über einen *Unreferenced Interval Count* (UIC) gesteuert, der pro Frame im Hauptspeicher mitgeführt wird. Somit können die Seiten lokalisiert werden, auf die am längsten nicht mehr zugegrif-

fen wurde. Es werden so lange Seiten ausgelagert, bis die Available Frame Queue wiederum einen definierten Schwellwert erreicht, der dem System signalisiert, dass jetzt genügend Platz im Hauptspeicher zur Verfügung steht.

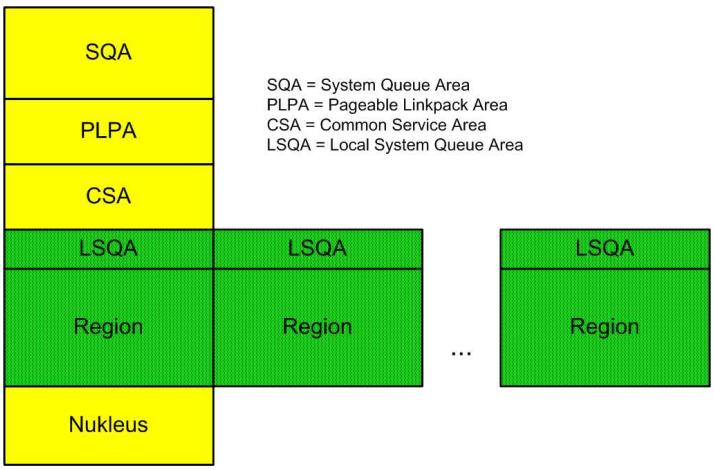
Das Umrechnungsprinzip ist bei einer 31-Bit-Adresse vom Prinzip her das gleiche. Der Unterschied liegt in der Größe der jeweiligen Tabellen und im Speicherlayout, auf das noch eingegangen wird. Die Seitentabelle wird mit 11 Bits adressiert und ist damit ein Megabyte groß, eine Page-Tabelle wird mit 8 Bits adressiert und ist damit 256 Kilobyte groß.

4.2 Layout des virtuellen Speichers

Der virtuelle Speicherbereich eines Adressraums im Multiple Virtual Storage (MVS) ist aufgeteilt in Bereiche, die dem System gehören, und in Bereiche, die jeweils einem Benutzer gehören. Die Benutzerbereiche werden auch als „private Bereiche" bezeichnet. Die privaten Bereiche können vom System praktisch beliebig oft zur Verfügung gestellt werden. Sie haben auch zu dem „Multiple" im MVS geführt, mit dem ausgedrückt wird, dass mehrere virtuelle Adressräume parallel in einem laufenden System aufgebaut werden. Die Systembereiche werden nur einmal aufgebaut und stehen allen Benutzern gemeinsam zur Verfügung. Der gesamte von einem Benutzer adressierbare Bereich wird Adressraum (*Address Space*) genannt.

Es geht hier zunächst um das Layout des virtuellen Speichers der /370-Architektur, um danach aufzuzeigen, was sich mit der Extended Architecture (XA) Anfang der 80-er Jahre mit Einführung der 31-Bit-Adressierung und in der z-Architektur mit der 64-Bit-Adressierung jeweils geändert hat.

Abbildung 4.3: Layout des virtuellen Speichers in der /370-Architektur



Ein Adressraum besteht aus folgenden Bereichen:

System Queue Area (SQA)
: Die SQA enthält Kontrollblöcke und Tabellen, die das gesamte System betreffen. Zum Beispiel befinden sich in der SQA *Address Space Control Blocks* (ASCB), die jeweils einen Adressraum repräsentieren. Auch die Segmenttabellen für die Adressumsetzung werden in diesem Bereich gehalten.

Pageable Link Pack Area (PLPA)
: Die PLPA enthält Programme und Systemroutinen (z. B. SVCs und Zugriffsmethoden), die von allen Benutzern verwendet werden. Über Definitionen in einer Parameter-Bibliothek hat der Systemprogrammierer Einfluss darauf, was in die PLPA kommt. Der Vorteil der PLPA liegt darin, dass häufig benutzte Programme so gut wie resident sind, da sie aufgrund des niedrigen Unreferenced Interval Count (UIC) nicht ausgelagert werden. Selten genutzte Programme hingegen werden auf den externen Speicher ausgelagert und benötigen dann keinen Platz im Hauptspeicher, sondern lediglich auf dem externen Page-Speicher.

Common Service Area (CSA)
: Die CSA ist in erster Linie zuständig für die Kommunikation zwischen verschiedenen Adressräumen. Ursprünglich war dies auch die einzige Möglichkeit für Adressräume, untereinander effizient zu kommunizieren. Inzwischen gibt es mit Cross Memory Services und Data Spaces weitere Möglichkeiten, Daten zwischen mehreren Adressräumen gemeinsam zu nutzen.

Local System Queue Area (LSQA)
: Die LSQA enthält Tabellen und Kontrollblöcke, die nur für den jeweiligen Adressraum relevant sind. Zum Beispiel findet man hier die Page-Tabellen für diesen Adressraum oder die *Task Control Blocks* (TCBs), die von diesem Adressraum verwaltet werden.

Region
: Dies ist der private Bereich, der jeweils einem Benutzer zugeordnet und der pro Adressraum individuell erstellt wird.

Nukleus
: Der Nukleus ist der residente Teil des MVS, der nicht auf den externen Speicher ausgelagert wird. In ihm befinden sich z. B. der I/O Supervisor und Kontrollblöcke wie beispielsweise *Unit Control Blocks* (UCB).

Die Problematik der /370-Architektur war die Adressbreite von 24 Bits und damit verbunden die maximale Größe eines Adressraums von 16 MB. Wenn wir beispielhaft einmal analysieren, wie die Verteilung der 16 MB aussieht, ist erkennbar, dass für den privaten Bereich eines Adressraums nicht mehr allzu viel übrig bleibt.

So ist die Größe des Nukleus unter anderem abhängig von der Anzahl angeschlossener Geräte. Am Ende des Lebenzyklus der /370-Archiktektur war dieser durchschnittlich ca. 4 MB groß.

Die Größe der CSA ist abhängig von der Anzahl im System aktiver Benutzer. Dass diese Anzahl im Laufe der Zeit ständig gewachsen ist, liegt in der Natur der Sache. Gehen wir zum Beispiel von 2 MB aus, ein Wert, der sicher sehr gering bemessen ist.

Die Größe der PLPA ist abhängig von der Anzahl der Module, die zum Zweck der Optimierung in den virtuellen Speicher geladen werden sollen. Hier hat man am Ende des /370-Lebenszyklus „Antituning" betrieben, indem Module, die nicht so häufig gebraucht wurden, aus der PLPA herausgenommen wurden. Gehen wir also bei der PLPA von 2 MB aus.

Bleibt noch die SQA. Deren Größe ist im Wesentlichen wiederum abhängig von der Anzahl der Benutzer im System. In einem relativ kleinen System können wir hier von 3 MB ausgehen. Wenn wir die Zahlen für die Systembereiche nun addieren, kommen wir auf insgesamt 11 MB. Das heißt, für den privaten Bereich bleiben noch ganze 5 MB übrig.

Das Problem waren in diesem Fall weniger die TSO User und die Batchjobs, da diese auch heute noch meist mit 5 MB auskommen, sondern eher die Online-Anwendungen in Verbindung mit Transaktionen und Datenbanken. Bei Online-Anwendungen (wie beispielsweise CICS, vgl. Abschnitt 11.6) bekommt nicht jeder Benutzer seinen eigenen Adressraum, sondern mehrere (und oft sehr viele) Benutzer klinken sich in einen gemeinsamen CICS-Adressraum ein – und dann sind 5 MB eben viel zu wenig.

4.3 Die eXtended Architecture (XA)

Mit der eXtended Architecture (XA) wurde die Adressbreite auf 31 Bit erweitert, wodurch ein virtueller Adressraum von 2 GB angesprochen werden kann. Oft hört man die Frage, warum denn gerade 31 Bits, wo doch für die Adressierung eigentlich 32 Bits zur Verfügung stehen. Die Antwort darauf ist einmal mehr die Kompatibilität. Das 32. Bit wird verwendet, um zu kennzeichnen, ob im „alten" 24-Bit-Modus oder im 31-Bit-Modus gefahren wird. Im 24-Bit-Modus läuft ein Programm weiterhin unterhalb der 16-MB-Grenze, im 31-Bit-Modus oberhalb. Dies hat auch dazu geführt, dass das Speicherlayout neu aufgebaut werden musste. Um Kompatibilität zu gewährleisten, müssen alle Bereiche sowohl unterhalb (für die 24-Bit-Adressierung) als auch oberhalb (für die 31-Bit-Adressierung) verfügbar sein.

Was hat sich geändert? Der Nukleus ist um die 16-MB-Grenze hochgewandert. Neu hinzugekommen ist die *Prefixed Storage Area*. Dies ist ein Bereich von 4096 Bytes mit Informationen, die in einem System immer an der gleichen absoluten Adresse

stehen müssen. Ein Beispiel dafür ist die *Common Vector Table* (CVT), ein Kontrollblock, der den Startpunkt für eine riesige Informationsverkettung im System darstellt. Auch für die Behandlung von Interrupts werden hier Bereiche zur Verfügung gestellt, damit das Program Status Word bei einer Unterbrechung gesichert werden kann. Da ein Interrupt in der ersten Phase ein von der Hardware gesteuerter Vorgang ist, müssen die entsprechenden Datenfelder an einer festen Stelle im Hauptspeicher gefunden werden.

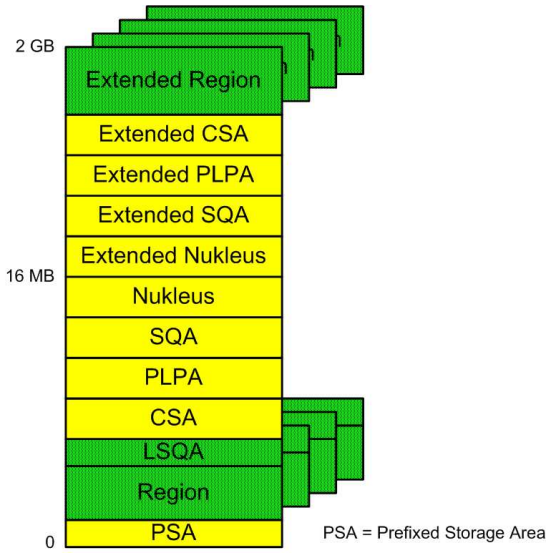


Abbildung 4.4: Layout des virtuellen Speichers in der eXtended Architecture

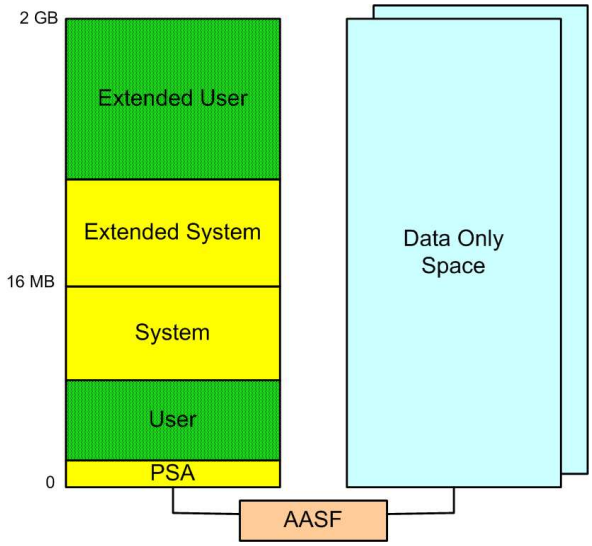
4.4 Erweiterungen der Enterprise System Architecture (ESA)

Auch bei der Einführung der *Enterprise System Architecture* (ESA) hat sich im Bereich Adressierung wieder viel getan. Die Adressierung wurde in diesem Fall in die Breite verändert. Zusätzlich zu den „normalen" Adressräumen können so genannte *Data Only*-Adressräume eingerichtet werden, die wie die anderen Adressräume auch bis zu 2 GB groß sein können. Über die *Advanced Address Space Facilities* (AASF) ist es möglich, bis zu 15 Data Only Spaces gleichzeitig zu adressieren. Der Vorteil dieser Technik liegt vor allem in der Möglichkeit der Trennung von Systemadressräumen und Daten.

So wurden diese Datenadressräume tatsächlich zunächst in erster Linie für DB2 eingerichtet. DB2 als Datenbankmanagement-System muss viele Daten verwalten, die vor der Einführung der Data Spaces in DB2-Systemadressräumen gehalten wurden.

In Verbindung mit Data Spaces wurde es möglich, eine strikte Trennung zwischen Systemadressräumen und Adressräumen mit reinen Daten aufzubauen, was neben den enormen Datenmengen, die nun verwaltet werden können, auch zu einer robusteren Umgebung geführt hat.

Abbildung 4.5: Layout des virtuellen Speichers in der Enterprise System Architecture



Die Adressierung der Data Only Spaces wird durch 16 zusätzliche Register erreicht. Das heißt, dass auch die Hardware diese Architektur unterstützen muss. Ein Register ermöglicht indirekt den Zugriff auf eine jeweils eigene Segmenttabelle, die den Startpunkt für die Adressierung der 2 GB des entsprechenden Data Space darstellt. Da eines der 16 Register für Verwaltungszwecke verwendet wird, können von einem Programm bis zu 15 Data Spaces gleichzeitig angesprochen werden.

Neben Byte-adressierbaren Data Only Spaces gibt es zusätzlich die Möglichkeit, *HiPer Spaces* einzurichten, die seitenweise adressiert werden und vor allem zur Datenpufferung dienen. HiPer steht hier für „High Performance".

4.5 64-Bit-Adressierung mit zSeries

Mit der XA-Architektur wurde 1981 die bis dahin eingesetzte 24-Bit-Adressierung (mit realer und virtueller Adressierung von max. 16 MB) erweitert auf die 31-Bit-Adressierung, mit der reale und virtuelle Adressbereiche bis zu maximal 2 GB angesprochen werden können.

Diese 2-GB-Beschränkung war in der Vergangenheit zunächst einmal für den Real Storage problematisch, was zur Folge hatte, dass die IBM zur Umgehung dieser

Einschränkung den *Expanded Storage* eingeführt hat. Dieser ist in erster Linie ein schneller Page Storage, wobei die Seiten (4-KB-Blöcke) mit synchronen Move-Instruktionen zwischen Real Storage und Expanded Storage hin und her geschoben werden.

Das führte zu höherer CPU-Belastung und im Falle eines Engpasses zu einem zusätzlichen Problem: Wenn der Expanded Storage voll wird, müssen Seiten auf den externen Speicher ausgelagert werden. Da es jedoch keine direkte Verbindung vom Expanded Storage zum I/O-Subsystem gibt, müssen die Seiten über den Real Storage auf den externen Speicher transferiert werden. Da dieser ja auch schon voll ist (sonst wäre der Expanded Storage nicht so intensiv genutzt worden), gibt es einen zusätzlichen Engpass.

Die 64-Bit-Adressierung schaffte hier endlich Erleichterung. Mit OS/390 V2R10 und z/OS V1R1 wurde zunächst die Unterstützung für den Real Storage eingeführt. Mit z/OS V1R2 begann dann die Unterstützung für den virtuellen Speicher. Programme können nun auf Speicherbereiche oberhalb der 2-GB-Grenze zugreifen.

Das z/OS und auch schon die Vorgängersysteme ermöglichten (zumindest seit der Einführung der Cross Memory Services) eine horizontale Skalierung, indem z. B. mehrere CICS-Regions eingesetzt oder die Möglichkeiten der Data/HiperSpaces genutzt wurden.

Die meisten E-Business-Anwendungen basieren auf C, C++ oder Java und erfordern eine vertikale Skalierung, d. h. einen größeren Adressraum.

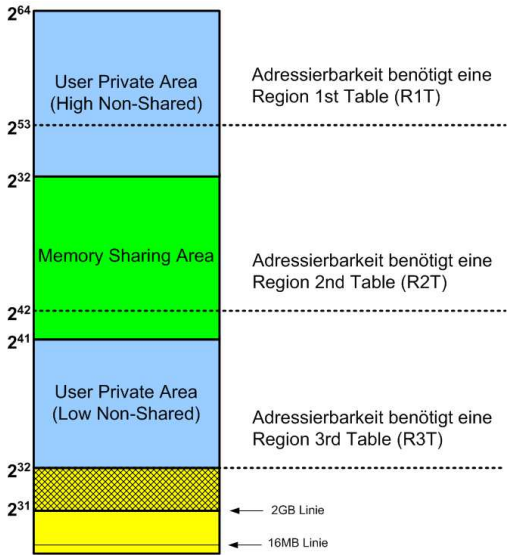
4.6 Der virtuelle 64-Bit-Adressraum

Ab z/OS V1R2 stehen für die Adressierung 2^{64} Bytes oder 16 EB (Exabytes) zur Verfügung. Dieser Adressraum ist 8 Milliarden-mal größer als der bisherige 2-GB-Adressraum. Ein weiterer Vergleich zur Verdeutlichung: Man würde ca. 107 Millionen PC-übliche 100-GB-Festplatten benötigen, um 16 Exabytes abzuspeichern. Hier kommt sicher die Frage auf, ob man derartig riesige Adressbereiche überhaupt benötigt. Diese Diskussionen gab es in der Vergangenheit immer wieder. Mit der Einführung der 31-Bit Adressierung war man Anfang der 1980-er Jahre auch der Meinung, dass die adressierbaren 2 GB virtueller Speicher „ewig" reichen würden.

Die Kompatibilität ist selbstverständlich gewährleistet, d. h. alle bisherigen Programme laufen unverändert weiter, da der Bereich unterhalb 2 GB unverändert abgebildet wird. Auch die 24-Bit-Adressierung wird unterhalb der 16-MB-Grenze weiterhin unterstützt.

Das System adressiert den Bereich zwischen 2 und 4 GB nicht. Dieser Bereich wird als *Balken* (*Bar*) bezeichnet. Dies bewirkt, dass es beim Ansprechen einer Adresse zwischen 2 und 4 GB im 64-Bit-Mode mit eingeschaltetem High Bit der 31-Bit-Adresse immer zu einem Program Check und damit zu einem Abbruch kommt.

Abbildung 4.6: 64-Bit-Adressraum



Es gibt derzeit keinen Bereich oberhalb der 4-GB-Grenze, der von den Adressräumen gemeinsam genutzt wird. Der Bereich für das Memory Sharing mit Hilfe der Region 2nd Table (R2T) wird von IBM jedoch für die künftige Nutzung reserviert.

4.7 Dynamic Address Translation mit 64-Bit

Die dynamische Adressübersetzung sorgt, wie schon erwähnt, für die Umsetzung einer virtuellen in eine reale Adresse. Sie wird vom *Virtual Storage Management* (VSM) durch einen Lookup-Prozess in Verbindung mit Tabellen durchgeführt. Die Tabellen werden Segment- und Page-Tabellen genannt. Im Falle der 31-Bit-Adressierung wird der virtuelle Speicher in 2048 Segmente von jeweils 1 MB Größe eingeteilt. Jeder Adressraum bekommt seine eigene Segmenttabelle. Ein Segment hat eine Pagetable, deren Einträge auf die 256 Seiten innerhalb des Segments zeigen.

Region-Tabellen

Bei der 64-Bit-Adressierung mit einem Adressbereich bis zu 16 Exabytes werden drei zusätzliche Levels von Übersetzungstabellen eingerichtet, die als *Region-Tabellen* bezeichnet werden. Man unterscheidet *Region Third Table* (R3T), *Region Second Table* (R2T) und *Region First Table* (R1T). Die Region Tables haben eine

Länge von 16 KB, pro Tabelle gibt es 2048 Einträge. Wenn zum ersten Mal Speicher oberhalb des Balkens (Bar) angefordert wird, wird die R3T angelegt. Wenn virtueller Speicher oberhalb von 4 TB angefordert wird, wird die R2T angelegt. Eine R2T hat 2048 Zeiger auf R3Ts und ermöglicht somit die Adressierbarkeit bis zu 8 PB. Eine R1T wird angelegt, wenn Adressen oberhalb 8 PB angefordert werden. Die R1T hat 2048 Zeiger auf R2Ts und ermöglicht somit eine Adressierbarkeit von 16 EB, was der 64-Bit-Adressbreite entspricht.

Die Region Tables werden nur erstellt, wenn sie notwendig werden, d. h. wenn eine Translation Exception eintritt, und zwar bei einer dynamischen Adressumsetzung bis zu fünf Tabellen. Allerdings startet die Übersetzung bei derjenigen, die für die Umsetzung der höchsten nutzbaren virtuellen Adressen benötigt wird.

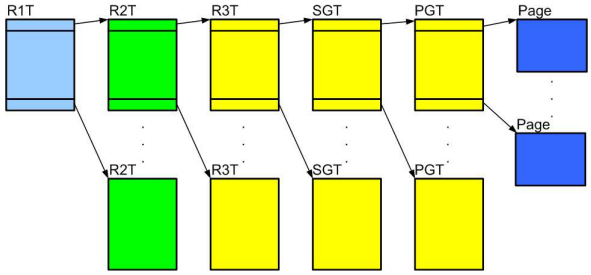


Abbildung 4.7: Tabellen für die 64-Bit-Adressierung

Die virtuelle Adresse

Die virtuelle Adresse, die wir von der 31-Bit-Adressierung kennen, besteht aus drei Teilen: Einem 11-Bit-Segmentindex, einem 8-Bit-Pageindex und einem 12-Bit Displacement.

Mit der 64-Bit-Adressierung kommen nun einfach drei weitere Teile hinzu, die jeweils in eine der drei Region-Tabellen zeigen.

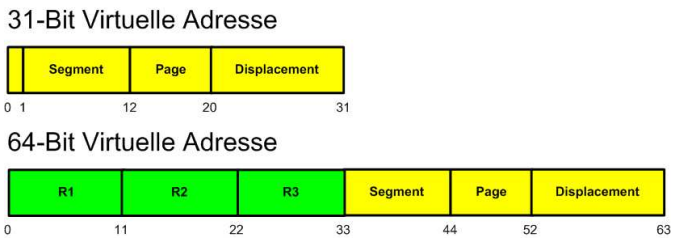
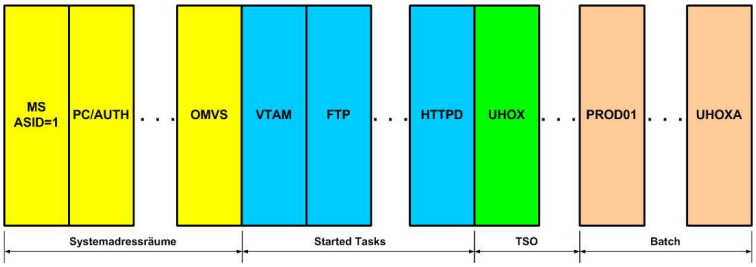


Abbildung 4.8: Region-Tabellen der 64-Bit-Adressierung

4.8 Adressräume im MVS

Ein Adressraum im MVS ist gekennzeichnet durch die Adressierung eines virtuellen Speicherbereichs. Gleichzeitig spielt in diesem Zusammenhang die Isolation der Adressräume untereinander eine große Rolle. Rein aufgrund der Adressierungstechnik ist es nicht ohne weiteres möglich, dass ein Adressraum auf die Bereiche eines anderen Adressraums zugreift. Dies ist vor allem wichtig in Umgebungen, wo viele Adressräume gleichzeitig aktiv sind.

Abbildung 4.9: Adressräume im MVS



Es gibt im MVS unterschiedliche Arten von Adressräumen:

Systemadressräume

Ein Systemadressraum ist ein Adressraum, der beim Starten des Systems (*Initial Program Load*, IPL) initiiert wird. Ein Beispiel dafür ist der Master Scheduler, der grundsätzlich immer als erster Adressraum gestartet wird und die *Address Space Identification* (ASID) 1 hat.

Weitere Beispiele sind der *Real Storage Address Space* (RSAP), der *Workload Manager* (WLM), der *Catalog Address Space* oder die *Program Call Authorization* (PCAUTH). Auch der Kernel von *UNIX System Services* (OMVS) ist mittlerweile als Systemadressraum in das System integriert.

Started Task

Ein *Started Task* entspricht einem Daemon in UNIX/Linux-Umgebungen. Es ist ein Adressraum, der mit dem START-Kommando von einer Konsole aus gestartet wird und mit einem STOP-Kommando beendet werden kann. Meist sind es Adressräume, die beim IPL des Systems gestartet werden und dann bis zur Systembeendigung aktiv bleiben. Beispiele dafür sind VTAM, TSO und DB2 oder auch Adressräume, die aus UNIX-Umgebungen als Daemon-Prozesse bekannt sind, wie TELNET oder FTPD.

Batchjob

Auch ein Batchjob ist ein eigenständiger Adressraum. Die Größe des privaten Bereichs wird durch den REGION-Parameter festgelegt.

TSO-Adressraum

Ein TSO-Adressraum steht für interaktive Verarbeitung nach dem Prinzip des Timesharing und entspricht einem Shell-Prozess in UNIX/Linux-Umgebungen. Jeder TSO User bekommt seinen eigenen Adressraum und ist damit weitgehend von anderen Adressräumen abgeschottet.

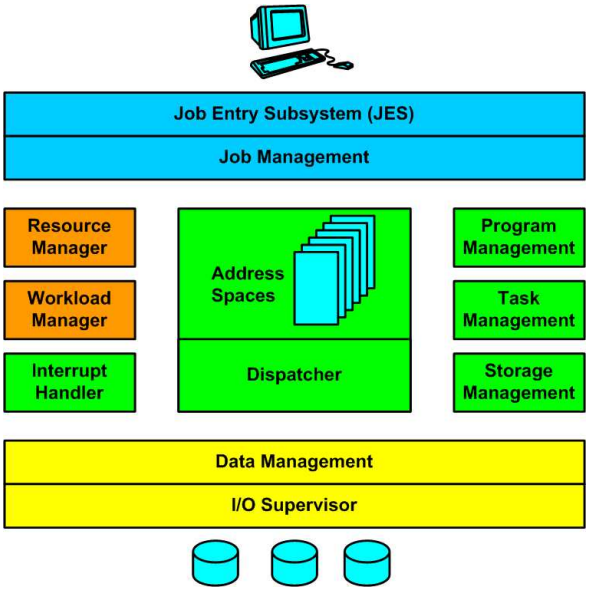
Kapitel 5

Die wichtigsten Systemkomponenten

In diesem Kapitel geht es darum, einige wichtige Systemkomponenten und deren Funktionen aufzuzeigen. Zwei Komponenten haben eine direkte Schnittstelle zu den Benutzern: Das ist zum einen das *Job Management* mit dem *Job Entry Subsystem* (JES) und das *Data Management* mit dem *I/O Supervisor* als Schnittstelle und Bindeglied zwischen einer logischen Datenanforderung und der physischen Speicherung der Daten auf einem Speichermedium. Dazwischen befinden sich Komponenten, die für die Systemverwaltung eine immens wichtige Rolle spielen. Dazu gehört der *Supervisor*, der alle Vorgänge innerhalb des Systems steuert, mit den Sub-Komponenten *Interrupt Handler* und *Dispatcher* sowie dem *Program Management*, *Task Management* und *Storage Management*.

In diesem Kapitel geht es um das Job Management, das Data Management und den Supervisor. Der System Resource Manager und der Workload Manager sind Thema in Kapitel 16.

Abbildung 5.1: Systemkomponenten



5.1 Job Management und JES

Komponenten des Job Management

In der folgenden Beschreibung sind die Komponenten, die dem Job Management angehören, typographisch hervorgehoben.

Operator-Aktionen

Wenn der Operator ein Kommando an der Konsole eingibt, setzt die *Communication Task* ein EXCP-Makro ab, das den *I/O Supervisor* veranlasst, die Ein-/Ausgabe für die Konsole durchzuführen.

Darauf interpretiert der *Command Processor* die Zeichenfolge des eingegebenen Kommandos und führt es aus. Dieser Teil des Job Management betrifft in erster Linie die Systembedienung. Wir werden hier nicht näher darauf eingehen.

Batch-Verarbeitung

Bei der Batch-Verarbeitung liest das *Job Entry Subsystem* (JES) einen Job mit Hilfe eines so genannten *Internal Readers* ein und interpretiert die Job Control Language (JCL) mit Hilfe des *Converter/Interpreter*. Der *Initiator* kontrolliert die Ausführung der einzelnen Jobsteps. Eine Komponente des Initiators, die *Allocation/ Unallocation Routine*, ordnet Geräte zu, die per JCL angefordert werden, und gibt sie wieder frei.

Timesharing

Die interaktive Schnittstelle für das Timesharing heißt im MVS-Umfeld Time Sharing Option (TSO) und ist mit einer Shell unter UNIX vergleichbar. Das LOGON der TSO-Benutzer wird intern ebenfalls mit der JCL umgesetzt und benutzt ebenfalls *Converter/Interpreter, Initiator* sowie die *Allocation/Unallocation Routine*, um das Terminal Monitor Program (TMP) aufzurufen. Von dieser JCL sieht der TSO-Anwender in der Regel jedoch nichts. Sie ist als so genannte LOGON-Prozedur in einer Systembibliothek abgelegt. Wir kommen darauf bei der Behandlung von TSO noch zurück.

An dieser Stelle ist anzumerken, dass die drei Bereiche, die hier als Benutzerschnittstelle zum Job Management aufgeführt sind, tatsächlich sehr unterschiedlich sind. Das gibt es in dieser Form in kaum einem anderen Betriebssystem. Dies hat zur Folge, dass ein Operator, ein TSO-Benutzer und der Benutzer, der einen Batchjob zum Laufen bringen will, jeweils mit einer unterschiedlichen Benutzerschnittstelle konfrontiert ist. Das macht das Leben für die Benutzer nicht gerade einfach, da für die jeweilige Benutzerschnittstelle auch immer eine eigene Syntax gelernt werden muss. In UNIX beispielsweise gibt es nur eine einzige Schnittstelle, nämlich die Shell. Ein Administrator hat zwar spezielle Berechtigungen und kann einige Befehle ausführen, die ein „normaler" Benutzer nicht zur Verfügung hat, die Schnittstelle und vor allem die Befehlssyntax ist jedoch für alle Benutzer dieselbe. Auch ein Batchjob in UNIX in Form eines Shellscripts hat die gleiche Syntax.

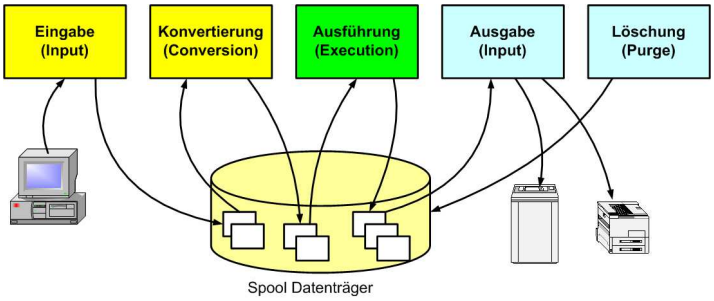
Das Job Entry Subsystem (JES)

Hinter dem JES steht die Umsetzung des Spooling-Konzepts. Im Wesentlichen geht es dabei um die Entkopplung der Eingabe/Konvertierung eines Jobs von der Ausführung und der Ausgabe von Daten auf einen oder mehrere Drucker. Das Job Entry Subsystem unterscheidet fünf Phasen, die ein Batchjob in seinem Lebenszyklus durchläuft.

Es sollte an dieser Stelle noch erwähnt werden, dass es zwei Varianten des JES gibt. Für den Einsatz des Spooling-Konzepts stehen als Alternativen *JES2* oder *JES3* zur Verfügung. JES3 ist vor allem für Mehrrechner-Konfigurationen konzipiert. Da die

überwältigende Mehrzahl von Unternehmen heute JES2 verwendet, wird hier in erster Linie darauf eingegangen.

Abbildung 5.2: Die fünf JES2-Phasen



INPUT
: Ein Job wird über den Reader (Internal Reader) eingelesen und auf dem Spool-Datenträger abgelegt. Er gelangt in die JES Input Queue und erhält eine Jobnummer.

CONVERSION
: Die JCL wird im Falle von Prozeduren ergänzt und auf Gültigkeit überprüft. Es wird ein so genannter Internal Text erstellt, der als Kontrollblockstruktur die Batch-Verarbeitung gegenüber dem JES repräsentiert.

EXECUTION
: Nach einem bestimmten Algorithmus und auf der Basis von Prioritäten wird ein Job zur Ausführung ausgewählt. Die Kontrollblöcke des ausgewählten Jobs werden dazu einem freien Initiator übergeben.

OUTPUT
: Die vom Programm erstellten Listen werden in der JES Output Queue auf dem Spool-Datenträger gehalten, bis ein freier Drucker bereit ist, die Listen auszudrucken, oder bis der User/Operator den Output löscht („cancelt").

PURGE
: Erst wenn der gesamte Output auf einen Drucker ausgegeben wurde, werden die von einem Batchjob belegten Bereiche auf dem Spool-Datenträger gelöscht.

Den Batchjobs wird für die Ausführung eine Jobklasse zugewiesen, die steuert, wie ein Job für die Ausführungsphase ausgewählt und behandelt wird. Diese Ausführungsklasse wird über den CLASS-Parameter der JCL zugewiesen, oder es wird, wenn dieser nicht in der JCL mitgegeben wird, ein Default-Wert zugeordnet, der in einem Parametersatz für das JES definiert wird.

Für die Verarbeitung eines Batchjobs wird ein *Initiator* benötigt, der für die Steuerung eines Batchjobs in der Ausführungsphase zuständig ist. Ein Initiator wird für eine oder mehrere Job-Klassen gestartet. Mit der Anzahl an Initiatoren für die jeweiligen Klassen kann somit gesteuert werden, wie viele Batchjobs einer bestimmten Klasse höchstens gleichzeitig im System aktiv sein können. Wenn ein Job gestartet wird und kein Initiator seiner Klasse verfügbar ist, weil entweder für diese Klasse keiner gestartet ist oder im Moment gerade alle Initiatoren der entsprechenden Klasse mit anderen Batchjobs belegt sind, muss der Batchjob warten, bis ein Initiator frei oder gestartet wird.

Die Anzahl der Initiatoren, die pro Klasse zur Verfügung stehen, konnten in der Vergangenheit nur über die Parameter für das JES und im laufenden Betrieb durch entsprechende Operator-Befehle beeinflusst werden. Inzwischen kann die Anzahl gestarteter Initiatoren auch vom Workload Manager (*Workload Managed Initiator*) verwaltet werden. Damit wird für die Batch-Verarbeitung eine deutlich flexiblere und dynamischere Arbeitsweise möglich.

Zur JCL kommen wir noch in Kapitel 7. Trotzdem hier ein Beispiel für den CLASS-Parameter:

```
//UGEW01 JOB CLASS=C ...
```

Der Batchjob mit dem Namen UGEW01 wird der Ausführungsklasse C zugeordnet und kann nur von einem Initiator zur Ausführung gebracht werden, der für die Klasse C gestartet wurde.

Auch für die Ausgabe werden wiederum Klassen zugeordnet, die steuern, auf welches Ausgabegerät (i. d. R. ein Drucker) die Daten ausgegeben werden sollen. Es handelt sich dabei um die so genannte *Message-Klasse* (MSGCLASS). Über den SYSOUT-Parameter kann ein individueller Datenbestand über eine DD-Anweisung direkt einer Ausgabeklasse zugeordnet werden. Auch hier wieder ein Beispiel:

```
//UHOXA JOB  CLASS=A,MSGCLASS=B
//S1    EXEC PGM=COBP1
//OUT1  DD SYSOUT=A
//OUT2  DD SYSOUT=F
```

Die Nachrichten des JES werden über die Message-Klasse in die Ausgabe-Queue für die Klasse B geleitet, die Daten über OUT1 in die Klasse A und die Daten über OUT2 in die Klasse F. Hinter den Klassen A und F können beispielsweise unterschiedliche Formulare angesteuert werden, die auf den Druckern eingerichtet sind, welche für die jeweilige Klasse gestartet wurden.

Das Job Entry Subsystem 2 (JES2) wurde übrigens ursprünglich im Rechenzentrum der NASA in Houston in den USA entwickelt. Diese Vergangenheit ist auch im neuesten z/OS noch erkennbar. Die JES-Meldungen an der Konsole und in den

Job-Outputs beginnen alle mit dem Präfix $HASP ..., wobei das Kürzel HASP für *Houston Automatic Spooling Package* steht.

In der Vergangenheit kam es immer wieder vor, dass bei einem Kunden eine Software entwickelt wurde, die sich auch für andere Kunden als nützlich erwies. IBM hat dann diese Software in ihr Betriebssystem integriert bzw. in ihr Produktportfolio mit aufgenommen. Der *Data Propagator* beispielsweise ist ein Werkzeug, um IMS-Daten (IMS ist eine hierarchische Datenbank) in DB2-Daten (DB2 ist eine relationale Datenbank) zu überführen; er wurde beim Schweizer Bankverein in Basel entwickelt und später von der IBM übernommen und als Produkt vertrieben.

System Display and Search Facility (SDSF)

Ein äußerst interessantes Werkzeug für die Einsicht in die JES2-Mechanismen ist SDSF. SDSF stand ursprünglich für *Spool Display and Search Facility*, wurde jedoch im Lauf der Zeit beträchtlich erweitert, so dass es umbenannt wurde in *System Display and Search Facility*. Ein äquivalentes Werkzeug steht mit dem *Flasher* auch in JES3-Umgebungen zur Verfügung.

SDSF ist sowohl ein Werkzeug für die Administration eines Systems als auch eines für Endbenutzer. Es ermöglicht die Steuerung, Verwaltung und das Anzeigen vielfältiger Systeminformationen in einer JES2-Umgebung.

SDSF stellt eine Reihe von Panels und Funktionen zur Verfügung. Man kann damit:

- Jobs in den verschiedenen Phasen überwachen
- Jobs kontrollieren (Hold, Purge, Release, Cancel)
- Jobs ansehen
- Initiator, Printer usw. kontrollieren
- Netzwerke und Knoten überwachen
- MVS- und JES-Kommandos absetzen

In der Regel können die Endbenutzer mit SDSF nur die eigenen Jobs sehen und beeinflussen.

Bestimmte Funktionen, wie beispielsweise das Absetzen von MVS- und JES-Kommandos, setzen spezielle Berechtigungen voraus, die über die *Resource Access Control Facilities* (RACF) gesteuert werden können. Mehr zu RACF und Security finden Sie in Kapitel 8.

Es gibt mehrere Möglichkeiten, eine SDSF-Session zu starten:

- aus TSO mit dem Kommando **SDSF**
- aus ISPF (ein menügesteuertes Hilfsmittel)
- SDSF-Aufrufe aus einem Batchjob

In der Praxis ruft man SDSF meist über das ISPF-Menüsystem auf.

Systemautomation mit Tools

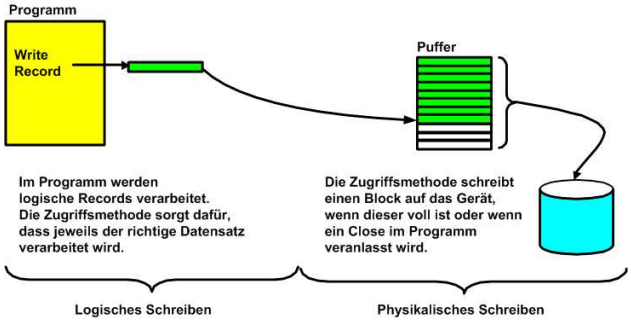
Beim Ablauf vor allem von Batch-Programmen und Started Tasks werden in der Praxis häufig Werkzeuge zur Automation eingesetzt, mit denen sich periodische Abläufe planen und definieren lassen. So wird beispielsweise die Verarbeitung eines Produktionsablaufs zu einem bestimmten Zeitpunkt gestartet. Der Ablauf besteht aus mehreren Jobs: Job B wird erst gestartet, wenn Job A zum Ende gekommen ist, und Job Y läuft nur, wenn Job X mit einem Return-Code größer als 8 beendet wurde. Es stehen in diesem Bereich diverse Produkte zur Verfügung, z. B. CA-7 von Computer Associates, BAGJAS von Bertelsmann, Control-M von BMC und OPC (*Operation Planning and Control*) von IBM, das inzwischen umbenannt wurde in *Tivoli Workload Scheduler* (TWS).

5.2 Data Management

Allgemein betrachtet stellt das Data Management die Brücke zwischen logischen und physischen Ein-/Ausgaben dar. Unter anderem soll das Data Management

- Ein-/Ausgaben kontrollieren (Steuerung über die so genannten Zugriffsmethoden)
- Speicherplatz auf externen Geräten verwalten (Anlegen, Erweitern und Löschen)
- eine späte Bindung zwischen Programm und Datei ermöglichen, d. h., dass z. B. die Parameter Puffergröße, Blockungsfaktoren, Gerätetypen und Geräteadressen erst zur Laufzeit relevant sind;
- Dateien über einen Namen automatisch auffinden (Katalogverwaltung)

Abbildung 5.3: Beispiel für das logische/physische Schreiben von Daten



Datenorganisation und Dateien

Um Daten im MVS abspeichern zu können, müssen Datasets eingerichtet werden. Im Vergleich zu anderen Betriebssystemen wie Windows oder UNIX/Linux ist dies leider komplizierter, da die Daten nicht als Bytestrom abgelegt werden, sondern – aus historischen Gründen – satzorientiert. Das nach wie vor häufigste Satzformat enthält Datensätze mit 80 Bytes, ebenfalls ein Relikt aus der „Lochkarten-Zeit", denn die Standard-Lochkarte konnte 80 Zeichen aufnehmen. Für Drucker beispielsweise beträgt die Standard-Satzlänge hingegen 133 Bytes (132 Bytes für Daten, 1 Byte als Drucker-Steuerzeichen).

Diese logischen Datensätze werden aus Effizienzgründen in Blöcke zusammengefasst. Ein Block ist die physische Übertragungseinheit zwischen Ein-/Ausgabegerät und Hauptspeicher. Um nicht bei jedem einzelnen Datensatz eine Ein-/Ausgabe durchführen zu müssen, die mit viel Overhead verbunden ist, werden in einem physischen Block mehrere logische Records übertragen.

Die maximale Größe eines Blocks ist abhängig vom Speichermedium. Eine große Herausforderung der Vergangenheit war es, die optimale Blockgröße für eine Datei in Abhängigkeit vom eingesetzten Medium zu ermitteln. Heute übernimmt diese Aufgabe das System selbst mit der Komponente *System Managed Storage* (siehe Kapitel 6.5).

Die Daten werden vom Betriebssystem mit Hilfe der Zugriffsmethoden verwaltet. Generell werden sequenzielle, index-sequenzielle und direkte Zugriffsmethoden unterschieden, wobei die index-sequenziellen und direkten Zugriffe von der *Virtual Storage Access Method* (VSAM) verwaltet werden.

Einen Spezialfall bilden so genannte *Partitioned Datasets* (PDS), die oft auch als *Partitioned Organized* (PO) *Datasets* oder auch als *Bibliotheken* bezeichnet werden. Eine Bibliothek besteht aus einem Inhaltsverzeichnis (Directory) und einem

Datenteil. Im Inhaltsverzeichnis befinden sich so genannte *Members* mit einem Zeiger auf den aktuellen Datenbestand im Datenteil.

Ziel von Bibliotheken ist es, mehrere Datenbestände unter einem gemeinsamen Dach unterbringen zu können. Beispielsweise kann ein COBOL- oder C-Programmierer eine Quelldatei für ein bestimmtes Projekt einrichten, in dem dann sämtliche Quellmodule als Members abgelegt werden. Dies ist vergleichbar mit einem Verzeichnis unter Windows bzw. UNIX/Linux – ein Vergleich, der jedoch hinkt, da in einem Windows- oder UNIX/Linux-Verzeichnis unterschiedlichste Datentypen (z. B. Quellcode und ausführbares File) abgelegt werden können, was im MVS nicht geht, da Quellcode und Lademodul (so wird ein ausführbares Modul im MVS bezeichnet) unterschiedliche Satzformate haben und somit nicht im gleichen Dataset untergebracht werden können.

Die Satzformate können eine feste Länge, eine variable Länge oder eine undefinierte Länge haben.

Aus Optimierungsgründen werden die Sätze in einem Dataset geblockt abgelegt. Ein Block ist die physische Übertragungseinheit, die bei einer Ein-/Ausgabe zwischen E/A-Gerät und Hauptspeicher übertragen wird. Damit nicht bei jeder Anforderung eines Datensatzes ein sehr aufwändiger Ein-/Ausgabevorgang initialisiert werden muss, wird jeweils ein Block mit mehreren Datensätzen übertragen und in einem Puffer im Hauptspeicher abgelegt. Der Zugriff auf die logischen Sätze und das Blocken und Puffern wird von der jeweiligen Zugriffsmethode verwaltet.

Es gibt noch einen weiteren wichtigen Unterschied zu Windows/UNIX/Linux-Umgebungen: Die Aufnahmekapazität eines Dataset ist begrenzt und wird in so genannten *Extents* verwaltet. Durch die Platzangabe (SPACE-Parameter) wird die Größe eines Extent festgelegt. Ein Standard-Dataset kann maximal 16 Extents haben; wenn diese Extents aufgebraucht sind und weitere Daten geschrieben werden sollen, gibt es eine abnormale Beendigung des Vorgangs, und das Dataset muss reorganisiert werden. Dies war vor allem in der Vergangenheit äußerst mühsam, da sich die Benutzer selbst um diese Reorganisation kümmern mussten. Inzwischen ist die Problematik durch den Einsatz von *System Managed Storage* (siehe Kapitel 6.5) deutlich entschärft, da sich SMS transparent für die Benutzer um die Organisation und Reorganisation der Daten kümmert.

Katalogverwaltung

Zentrale Bedeutung beim Umgang mit Dateien besitzt im MVS der Katalog. In jedem MVS muss mindestens ein Katalog definiert sein, der so genannte *Master-Katalog*. Optional können (und sollen!) *User-Kataloge* eingerichtet werden. Die User-Kataloge erleichtern vor allem die Systempflege und sorgen für eine bessere Verfügbarkeit und Robustheit des Systems, indem der Master-Katalog aus Benutzersicht nur für so genannte „Alias-Einträge" verwendet wird (einen pro Benutzer).

Ein Alias-Eintrag verweist dann auf einen User-Katalog. Die Dateien der einzelnen Benutzer werden dann in dem zugewiesenen User-Katalog verzeichnet. Ein Endbenutzer hat mit dem Einrichten der Umgebung in der Regel nichts zu tun. Er beantragt lediglich für den Zugriff auf den Mainframe eine Benutzerkennung. Ein Administrator sorgt dann dafür, dass die notwendige Umgebung eingerichtet wird.

Der Master-Katalog ist aus Betriebssicht eine kritische Ressource. Wenn er voll wird, bleibt das System stehen. Deshalb muss man unbedingt verhindern, dass der Master-Katalog überläuft und dass in ihm ständig neue Einträge für Dateien erstellt werden. Für die typischen Systemdateien trifft das beispielsweise zu: Es werden im laufenden Betrieb so gut wie keine neuen Systemdateien eingerichtet, dies geschieht etwa bei der Installation eines neuen Produkts. Ein neues Produkt wird jedoch nie in einem Produktionssystem installiert.

Katalogisierte Dateien haben den Vorteil, dass sie allein durch die Angabe des Dateinamens vom System lokalisiert und dem Benutzer zur Verfügung gestellt werden können. Umgesetzt auf die JCL heißt das beispielsweise, dass nur der Dateiname (DSN) und der Dispositionsparameter (DISP) für bestehende Dateien spezifiziert werden müssen.

In Verbindung mit SMS sind übrigens für nicht-temporäre Dateien ausschließlich katalogisierte Dateien erlaubt.

Der ***Master-Katalog*** enthält die Systemdateien, Verweise auf User-Kataloge und die erwähnten Alias-Einträge. Ein Alias-Eintrag ist identisch mit den höchsten Levels des Dateinamens. In früheren Versionen (vor MVS/ESA) gab es nur einen Level, den so genannten ***High Level Qualifier*** (HLQ). Seit MVS/ESA werden Multi-Level-Aliases unterstützt (maximal vier). Das erleichtert vor allem die Pflege mehrerer Systeme.

Die Nutzung von ***User-Katalogen*** hat Einfluss auf die Verfügbarkeit und die Performance der Kataloge und der darin verwalteten Informationen. MVS benutzt die Zugriffsmethode VSAM für die Katalogverwaltung. Für die Definition von Katalogen sowie für die Einrichtung von Benutzern und High-Level-Qualifiern wird das Utility ***Access Method Services*** (AMS bzw. IDCAMS) benutzt.

Namenskonventionen für Dateien

Das MVS hat eine sehr flache Dateistruktur. Ein Dateiname kann maximal 44 Zeichen lang sein und besteht aus maximal acht Zeichen langen, so genannten ***Qualifiern***, die durch Punkte voneinander getrennt werden. Die Punkte zählen bei der Bestimmung der Namenslänge mit! In der Praxis wird man versuchen, mit möglichst kurzen und dennoch sprechenden Namen zu arbeiten, und man wird häufig außerdem versuchen, mit drei Qualifiern auszukommen, da dies in Verbindung mit der Arbeit mit ISPF Vorteile hat. Beispiel:

```
UHOX.TEST.CNTL
```

UHOX ist in diesem Fall der *High Level Qualifier* (HLQ), der eine spezielle Bedeutung für die Katalogverwaltung, für die Security (RACF) und für die Speicherverwaltung (SMS) hat. Er entspricht einer User-Kennung, wie im dargestellten Beispiel, oder einer Gruppenbezeichnung.

Der letzte Qualifier (*Low Level Qualifier*, LLQ) hat ebenfalls eine wichtige Bedeutung für die Speicherverwaltung (SMS), da damit indirekt angegeben wird, welche Inhalte in der entsprechenden Datei hinterlegt sein sollen. Wichtig ist in diesem Fall, dass es für das jeweilige Unternehmen ein Namenskonzept gibt, dass die Benutzer dieses Namenskonzept kennen und sich auch daran halten. Es haben sich De-facto-Standards durchgesetzt. So impliziert beispielsweise ein LLQ „COBOL", dass in dem entsprechenden Dataset COBOL-Quellcode abgelegt wird.

So kann SMS in unserem Beispiel UHOX.TEST.PRJ15.CNTL anhand des LLQ erkennen, dass in der Datei JCL (Job Control Language) abgelegt werden soll; weiterhin kann beispielsweise automatisch erschlossen werden, dass eine PO-Datei eingerichtet und das Satzformat FB (*Fixed Blocked*) mit 80-stelligen Records sein soll.

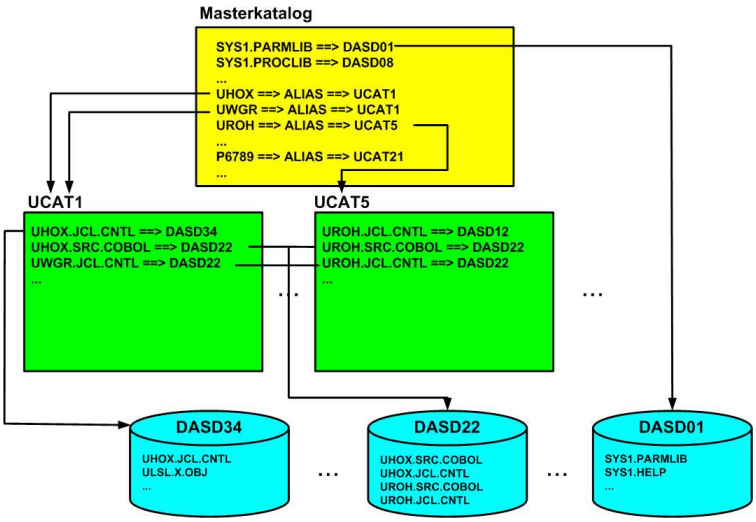


Abbildung 5.4: Beispiel einer Katalogstruktur

Der Alias-Eintrag UHOX zeigt im Beispiel aus Abbildung 5.4 auf den User-Katalog UCAT1. In diesem wird der Dateiname UHOX.JCL.CNTL mit einem Pointer auf das Volume DASD34 gefunden. Über das Inhaltsverzeichnis der Platte (VTOC) kann der physische Datenbestand lokalisiert werden.

Organisation der Daten auf einer Platte

Die Verweise auf Daten auf einem Plattengerät werden in Form von *Data Set Control Blocks* (DSCB) in einem Inhaltsverzeichnis eingetragen, das als *Volume Table of Contents* (VTOC) bezeichnet wird. Jedes Plattengerät hat einen Kennsatz (Label) auf der ersten Spur einer Platte. Aus einem Datensatz dieser ersten Spur gibt es einen Pointer auf das Inhaltsverzeichnis. Das Label der Platte sowie der VTOC werden beim Formatieren der Platte mit dem Utility ICKDSF erzeugt.

5.3 Der Supervisor

Wie der Name schon sagt, ist der Supervisor die Systemkomponente, welche für die zentrale Steuerung des Systems sorgt. Im Mittelpunkt stehen dabei die Unterbrechungsbehandlung und der Dispatcher. In diesem Zusammenhang spielen auch die Supervisor Calls eine Rolle, die dafür sorgen, dass über eine von insgesamt sechs Unterbrechungsarten eine Systemroutine aufgerufen wird. Weitere Unterkomponenten des Supervisors sind das Program Management, das Task Management und das Storage Management.

Unterbrechungsbehandlung und Dispatcher

Das Verständnis für Unterbrechungen (Interrupts) ist zwar nicht unbedingte Voraussetzung, um mit dem System zu arbeiten, hilft jedoch sehr, um die Eigenschaften und Qualitäten der Großrechnerarchitektur kennen und verstehen zu lernen. Das IBM-Großrechnersystem wird ausschließlich über Interrupts gesteuert – man spricht von einem solchen System daher auch als „Interrupt driven".

Was passiert nun bei einer Unterbrechung? Eine Unterbrechung ist ein elektronisches Signal, das die CPU bei der momentanen Arbeit unterbricht. Der Unterbrechungsmechanismus ist dafür verantwortlich, dass die Ursache der Unterbrechung analysiert wird, entsprechende Aktionen durchgeführt werden und zu einem späteren Zeitpunkt das unterbrochene Programm seine Arbeit fortsetzt. Im Zentrum einer Unterbrechung steht das *Program Status Word* (PSW), das ein spezielles Register in der CPU darstellt und die für eine Programmausführung wichtigen Informationen enthält, die den Status des laufenden Programms repräsentieren.

Das PSW besteht aus zwei Teilen, der Instruktionsadresse (Befehlszähler) und so genannten „Flags", die den Zustand des ausgeführten Programms beschreiben. Um das kontrollierende PSW in der CPU von abgespeicherten PSWs zu unterscheiden, wird es als *Current PSW* bezeichnet. Durch die Abspeicherung des Current PSW während eines Unterbrechungszyklus kann der Zustand eines unterbrochenen Programms ermittelt werden.

Wenn ein Unterbrechungssignal auftritt, wird die Kontrolle automatisch an eine Systemroutine übergeben. Diese Routine heißt *First Level Interrupt Handler* (FLIH). Für jeden Unterbrechungstyp gibt es einen FLIH, der die jeweilige Behandlung einleitet und in Abhängigkeit von den Besonderheiten des Interrupt an weitere Routinen (*Second Level Interrupt Handler*) weitergeben kann.

Unterbrechungstypen

Es gibt sechs Unterbrechungstypen:

I/O Interrupt
: Da Kanäle und CPU voneinander unabhängig Arbeit verrichten sollen, muss es für das Kanalsubsystem die Möglichkeit geben, der CPU zu signalisieren, dass ein Kanal seine Arbeit, nämlich die Abarbeitung eines I/O-Vorgangs, beendet hat. Dies geschieht mit einem I/O Interrupt.

SVC Interrupt
: Ein Programmierer schreibt bestimmte Programmteile wie z. B. die Abwicklung eines I/O-Vorgangs nicht selbst. Zum einen wäre dies äußerst mühsam, zum anderen wäre die Sicherheit des Systems gefährdet, da er dieses ja mit anderen Benutzern teilt; Systemroutinen allgemein sind natürlich äußerst kritische Routinen, die privilegierte Instruktionen benötigen. Für die Entwickler werden die Systemroutinen deshalb über SVC-Schnittstellen zur Verfügung gestellt, die sie aufrufen und die dann diese Arbeit erledigen. Der Aufruf der entsprechenden Systemroutine zieht einen SVC Interrupt nach sich.

External Interrupt
: Eine externe Unterbrechung kann unterschiedliche Ursachen haben. Beispiele sind die Systemkonsole, die Kommunikation zwischen CPUs oder auch der Timer. Mit Hilfe des Timers wird eine Uhrzeit eingestellt (sozusagen der „Wecker" gestellt), zu der beispielsweise eine Zeitscheibe abgelaufen ist.

Program Check Interrupt
: Mit Programmunterbrechungen wird jeder Programmierer konfrontiert. Ein Program Check Interrupt tritt beispielsweise auf bei Division durch Null, Überläufen, Speicherschutzverletzungen usw. In der Regel ist damit eine abnormale Beendigung (ABEND) verbunden, und der *Recovery Termination Manager* (RTM) wird aufgerufen.

Machine Check Interrupt
: Eine Machine-Check-Unterbrechung zeigt an, dass irgendwo in den Schaltkreisen des Systems ein Hardwarefehler aufgetreten ist. Es hat dann keinen Sinn, Berechnungen und logische Verknüpfungen auf einem System durchzuführen, das nicht richtig funktioniert. Dieser Interrupt kommt in der Praxis

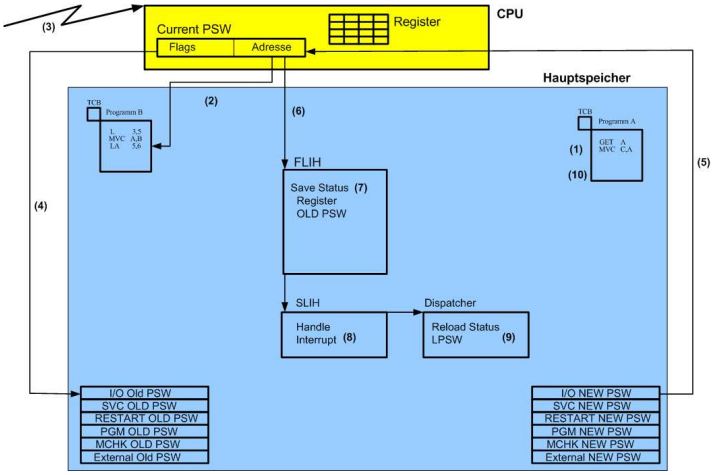
heute jedoch so gut wie nie vor, da die *Mean Time Between Failures* (MTBF) für die IBM-Hardware je nach Rechnerserie im Mainframe-Bereich irgendwo zwischen zehn und 30 Jahren liegt.

Restart Interrupt

Der Restart Interrupt kommt in der Regel nur beim Start des Systems zum Zuge, indem ein Operator definiert, von welcher Stelle aus das System gestartet werden soll.

Eine Unterbrechung läuft immer nach dem gleichen Grundschema ab :

Abbildung 5.5: Grundschema eines Interrupt



(1) Beispiel: Ein Programm A hat eine Ein-/Ausgabe initiiert und muss nun warten, bis das Kanalsubsystem die vom Programm benötigten Daten vom externen Gerät in den Hauptspeicher gebracht hat.

(2) Die Kontrolle wird deshalb an ein Programm B übergeben, das nun seinerseits CPU-Instruktionen abarbeitet.

(3) Nun hat das Kanalsubsystem den Ein-/Ausgabevorgang für das Programm A beendet. Es sendet deshalb ein Hardwaresignal an die CPU.

(4) Das gerade aktive PSW wird an einer vorbestimmten Stelle im Hauptspeicher abhängig vom Interrupt-Typ abgespeichert. Da es sich um einen I/O Interrupt handelt, wird es im Feld für das I/O OLD PSW abgespeichert.

(5) Ein neues PSW wird geladen und damit zum Current PSW. Da es sich um einen I/O Interrupt handelt, ist es das I/O NEW PSW.

(6) Das neue PSW zeigt mit seinem Instruktionszähler auf eine von der Unterbrechungsart abhängige Behandlungsroutine, die dann abgearbeitet wird. In diesem Fall ist es der I/O First Level Interrupt Handler.

(7) Der First Level Interrupt Handler sorgt für die Sicherung der Umgebung des unterbrochenen Programms, indem beispielsweise auch die vom Programm verwendeten Registerinhalte und das Old PSW in einem Kontrollblock abgespeichert werden, der das Programm repräsentiert und als *Task Control Block* (TCP) bezeichnet wird.

(8) Der Second Level Interrupt Handler (SLIH) analysiert dann die Ursache der Unterbrechung und veranlasst entsprechende Aktionen.

(9) Wenn die Unterbrechung abgearbeitet ist, kommt der Dispatcher zum Zuge, der aufgrund der ihm zur Verfügung stehenden Informationen unter Berücksichtigung der Prioritäten feststellt, welches Programm für die weitere Ausführung in Frage kommt. Nehmen wir an, das Programm A, das den I/O Interrupt verursacht hat, hat die höchste Priorität. Deshalb wird nun dessen Umgebung mit den Registerinhalten zurückgeladen. Mit LPSW wird das ursprünglich einmal gesicherte PSW in die CPU geladen.

(10) Das Programm A, das den I/O-Vorgang initiiert hatte, der nun beendet ist, kann seine Arbeit fortsetzen.

Storage Management

Zum Supervisor gehören zudem die Storage Manager. Hiervon gibt es drei: den *Virtual Storage Manager* (VSM), den *Real Storage Manager* (RSM) und den *Auxiliary Storage Manager* (ASM). Für den Expanded Storage gibt es keinen eigenen Storage Manager, er wird vom Real Storage Manager mitverwaltet.

Ein MVS-Programm wird im virtuellen Speicher gehalten, wobei nur die Teile im Hauptspeicher sein müssen, die gerade abgearbeitet werden. Inaktive Teile können auf einen Auxiliary Storage ausgelagert werden, der zumeist auf Plattengeräten eingerichtet wird, die dann auch als *Page-Platten* bezeichnet werden. Die Bereiche, die dafür genutzt werden, werden in Form von speziell formatierten Dateien eingerichtet. Diese Dateien sind in Slots formatiert. In einem Slot hat eine Page (4 KB) Platz. Der Platz auf dem externen Speicher wird vom Auxiliary Storage Manager verwaltet. Die aktiven Seiten im Hauptspeicher werden in Frames gehalten, die jeweils 4 KB groß sind und vom Real Storage Manager verwaltet werden.

Virtual Storage Manager (VSM)

Der Virtual Storage Manager verwaltet den virtuellen Speicher. Seine Hauptaufgabe ist es, die Requests auf die einzelnen Speicherbereiche entgegenzunehmen und die

angeforderten Bereiche zur Verfügung zu stellen. Die Requests werden in Form von GETMAIN- oder STORAGE OBTAIN-Makroanweisungen abgesetzt und mit den FREEMAIN- bzw. STORAGE RELEASE-Makros wieder freigegeben.

Auch gemeinsam genutzte Speicherbereiche (z. B. aus der Common Service Area) werden über Makros angefordert. Dies geschieht über Subpools, die definieren, aus welchem Bereich Storage angefordert wird.

Real Storage Manager (RSM)

Aufgabe des Real Storage Managers ist es, die Belegung der Frames im Hauptspeicher zu verwalten. Die wichtigsten Funktionen des RSM:

- Zuweisen von Hauptspeicher, um GETMAIN-Anforderungen zu erfüllen
- Fixen von Pages auf Anforderung, z. B. bei Ein-/Ausgabe-Vorgängen
- Zuweisen für Hauptspeicher, damit ein Adressraum eingeswappt werden kann

Der RSM verwaltet auch den Expanded Storage, der eingeführt wurde, um Paging und Swapping in einem bis zur zSeries-Architektur auf 2 GB begrenzten Hauptspeicher zu reduzieren. Da der Expanded Storage ab der zSeries-Architektur keine große Rolle mehr spielt, gehen wir hier nicht weiter darauf ein.

Der Auxiliary Storage Manager (ASM)

Der Auxiliary Storage Manager (ASM) verwaltet die Slots auf den externen Page-Dateien und die Übertragung von Seiten aus dem Hauptspeicher auf den Auxiliary Storage. Dies geschieht in Form von Paging (hier werden einzelne Seiten übertragen) oder Swapping (hier wird ein Adressraum ein- bzw. ausgelagert). Der ASM initiiert die dafür notwendigen Ein-/Ausgaben und verwaltet die Tabellen, die den aktuellen Zustand des Auxiliary Storage widerspiegeln.

Für ein effizientes Paging gibt es unterschiedliche Page-Dateien, die als PLPA, Common und Local Page Datasets eingerichtet werden. Um die Effizienz zu erhöhen, werden diese unterschiedlichen Page-Dateien auf unterschiedliche physische Geräte gelegt. Das war vor allem in früheren Zeiten wichtig, als aufgrund der beschränkten Speicherkapazitäten noch häufig Seiten mit Paging aus- und eingelagert werden mussten.

Heutzutage versucht man eher, so viel Hauptspeicher zur Verfügung zu stellen, dass ein Paging weitgehend verhindert wird. Die Page-Dateien werden in erster Linie nur noch für Backup des virtuellen Speichers genutzt. Aufgrund der immer billiger werdenden Hardware und der technischen Möglichkeiten (64-Bit-Adressierung) ist das in der Praxis auch kein Problem mehr.

Vor allem die Local Page Datasets, wo die privaten Regions der Benutzer aufbewahrt werden, werden aus Kapazitäts- und Performance-Gründen meist auf mehrere physische Geräte verteilt. Bei Engpässen können Page-Dateien auch dynamisch eingerichtet und in Betrieb genommen werden.

Die PLPA Page-Datei wird für das Backup der Module angelegt, die in den virtuellen PLPA-Bereich geladen werden. Dieses Laden ist nur beim Hochfahren (*Initial Program Load*) des Systems notwendig und auch wiederum nur dann, wenn der Parameter CLPA (*Create Link Pack Area*) beim Hochfahren angegeben wird. Dies ist im Grunde genommen nur notwendig, wenn ein Modul in der PLPA geändert oder ein neues Modul hinzugefügt wird.

Program Management

Basierend auf den Kontrollblöcken in der *Scheduler Work Area* (SWA), die bei der Initialisierung eines Batchjobs, eines TSO Users oder einer Started Task angelegt wird, wird ein Programm im Speicher lokalisiert oder aus einer entsprechenden Bibliothek geladen.

Die Reihenfolge der Programmsuche spielt dabei für das Verständnis der Abläufe eine wichtige Rolle:

1. Job Pack Area (JPA)
2. STEPLIB/JOBLIB
3. Link Pack Area (LPA)
4. Linklist-Verkettung

Wenn ein Modul nicht gefunden wird, gibt es eine abnormale Beendigung (ABEND) mit dem Code 806 und dem entsprechenden Hinweis, dass das Modul nicht gefunden wurde.

Erläuterungen:

Die Job Pack Area befindet sich in einem Adressraum und enthält Kopien von Lademodulen, die vorgängig bereits aufgerufen wurden.

Die STEPLIB/JOBLIB-Definitionen eines Benutzers haben Vorrang vor den für das System definierten Bibliotheken. Das bedeutet, dass man damit einen Programmaufruf eines Systemmoduls durch ein eigenes Modul ersetzen kann, wenn dieses sich in einer Bibliothek befindet, die als Steplib oder als Joblib definiert ist.

Beispiel JOBLIB/STEPLIB:

```
//GEWA    JOB
//JOBLIB  DD DSN=GEW.TEST.LOADLIB
```

```
//STEP1   EXEC PGM=X
// ...
//STEP2   EXEC PGM=Y
// ...
//STEP3   EXEC PGM=Z
//STEPLIB DD DSN=GEW.SPECIAL.LOADLIB
```

Eine Steplib gilt für einen einzelnen Jobstep, eine Joblib gilt für einen gesamten Job.

Die *Link Pack Area* (LPA) umfasst die *Pageable Link Pack Area* (PLPA), die *Fixed Link Pack Area* (FLPA) und die *Modified Link Pack Area* (MLPA). Die Link Pack Areas werden beim Starten des Systems aufgebaut, die PLPA nur dann, wenn beim Starten CLPA (Create Link Pack Area) als Systemparameter mitgegeben wird. Die LPAs enthalten Systemroutinen, einige SVC-Routinen, Zugriffsmethoden etc.

Das Ziel ist es, häufig verwendete (System-)Routinen im virtuellen Speicher zu halten, statt diese bei jedem Aufruf von einem externen Gerät per I/O-Vorgang zu laden.

Die Linklist-Verkettung besteht aus Systembibliotheken, die nach den entsprechend aufgerufenen Modulen durchsucht werden. Die SYS1.LINKLIB gehört grundsätzlich dazu. Weitere Bibliotheken werden durch entsprechende Definitionen in der PARMLIB im Member LNKLSTxx spezifiziert und damit mit der SYS1.LINKLIB verbunden (*concatenated*).

Supervisor Calls

Mit Hilfe von *Supervisor Calls* (SVCs) können Services vom Betriebssystem angefordert werden. Für die Services sind spezielle Systemberechtigungen notwendig, insbesondere das Ausführen so genannter privilegierter Befehle, die von einem „normalen" Programm nicht selbst ausgeführt werden dürfen. Einem normalen Programm steht in diesem Fall die SVC-Schnittstelle zur Verfügung. Damit wird gewährleistet, dass über einen SVC Interrupt die Privilegierung auf eindeutig definierte und vor allem kontrollierte Art und Weise erfolgt.

Der SVC-Befehl ist ein Maschinenbefehl mit einem zwei Byte langen Operanden, der die SVC-Nummer bezeichnet. Mit SVC xx sind somit maximal 256 (0-255) verschiedene SVCs aufrufbar.

Beispiele für SVCs:

- Lesen/Schreiben von Daten (EXCP bzw. SVC 0)
- Anforderung von virtuellem Speicher (GETMAIN bzw. SVC 10)
- Öffnen einer Datei (OPEN bzw. SVC 19)

- Schließen einer Datei (CLOSE bzw. SVC 20)
- Abbrechen eines Programmlaufs (ABEND bzw. SVC 13)
- Bandverwaltung CA1 (User SVC)

Absetzen von SVCs

Der SVC-Befehl selbst wird in der Regel nicht als Maschinenbefehl codiert (auch nicht von einem Assembler-Programmierer), sondern mit Hilfe entsprechender Makros aufgerufen.

```
CSECT PROGA
...
...
MACRO
+
+
+
SVC xx
...
...
```

Ein Makro ist eine Anweisung, die von einer Routine ersetzt wird, welche vordefinierte Maschinenbefehle enthält, um eine bestimmte Funktion zu erfüllen.

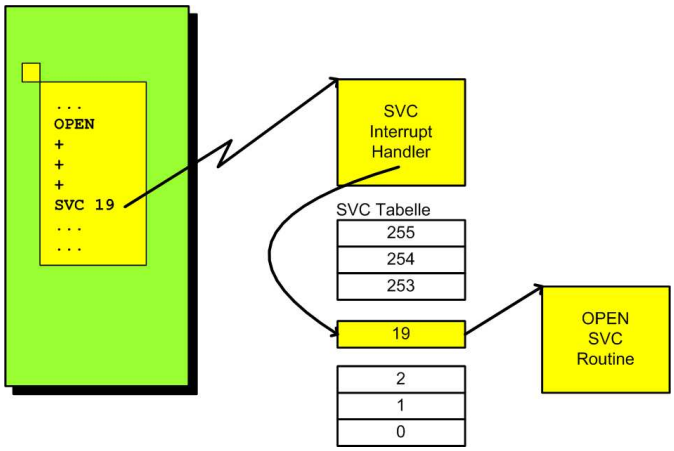


Abbildung 5.6: Ablauf eines SVC-Aufrufs

Ein SVC bewirkt immer eine Unterbrechung. Diese wiederum sorgt durch den Austausch des PSW für eine Verzweigung ins Betriebssystem. Dadurch wird grundsätzlich das Bit 15 im PSW auf 0 gesetzt, d. h., dass nach dem Interrupt unabhängig vom vorangegangenen Zustand im so genannten *Supervisor Mode* gearbeitet wird, der

die Ausführung privilegierter Befehle ermöglicht. Der Interrupt Handler findet über die SVC-Tabelle in Verbindung mit der entsprechenden SVC-Nummer die Adresse der angeforderten SVC-Routine.

Die wichtigsten SVCs (vom Typ 1, 2 und 6) befinden sich im Nukleus, die restlichen (vom Typ 3 und 4) in der Link Pack Area.

Reservierte SVCs

Die SVCs 0 bis 199 sind für das Betriebssystem reserviert. Sie werden derzeit zwar nicht alle genutzt, doch sollte man die reservierten auch nicht nutzen, da sie für zukünftige Systemerweiterungen eingesetzt werden. Die darüber liegenden Nummern können für so genannte „User SVCs" definiert werden. Mit User SVCs können Fremdanbieter von Systemprogrammen oder auch Systemprogrammierer eines Unternehmens eigene Routinen in das System integrieren, die nicht per Default vom Betriebssystem zur Verfügung gestellt werden. Das Parmlib *Member IEASVCxx* definiert die verfügbaren User SVCs. Um Konflikte mit den reservierten SVCs zu vermeiden, verwendet man die SVC-Nummern für die installationsspezifischen User SVCs von oben nach unten.

Die SVCs werden über eine SVC-Tabelle verwaltet. Neben der Entry-Point-Adresse stehen auch entsprechende Attribute in der Tabelle, die den SVC-Typ kennzeichnen und angeben, ob und welche Art von Lock gesetzt wird.

Inter Address Space Communication

Die Notwendigkeit, zwischen Adressräumen Informationen auszutauschen, trifft man in der Praxis sehr häufig an. In UNIX/Linux und auch anderen Umgebungen wird in diesem Zusammenhang der Begriff *Inter Process Communication* (IPC) verwendet.

Beispiele für eine derartige Kommunikation zwischen Systemroutinen:

- Das Job Entry Subsystem „redet" mit einem Batchjob.
- VTAM und TSO User kommunizieren miteinander.
- Ein beliebiger Adressraum greift auf den Catalog Address Space zu.
- Die IMS Control Region und eine Message Processing Region tauschen Informationen aus.

Das Problem dabei ist, dass ein Adressraum zunächst einmal aufgrund des Adressierungsmechanismus des MVS keinerlei Möglichkeiten hat, auf private Bereiche eines

anderen Adressraums zuzugreifen. Das ist gleichzeitig ein sehr gut funktionierender Schutzmechanismus. Mit den Adressierungsmechanismen des Betriebssystems hat man nur Zugriff auf den eigenen privaten Bereich.

Wie kann nun dennoch eine Kommunikation zwischen Adressräumen stattfinden? Hierfür gibt es zwischenzeitlich unterschiedliche Möglichkeiten.

Kommunikation über die CSA

Eine Möglichkeit, die schon sehr lange besteht, ist der Weg über die Systembereiche. Im virtuellen Speicherlayout gibt es unter anderem die *Common Service Area* (CSA). Dies war ursprünglich die einzige Möglichkeit für die Kommunikation zwischen Adressräumen. Man spricht hier auch von einer „asynchronen" Kommunikation. Dieser Begriff spielt deshalb eine Rolle, weil der Sender einer Information die Daten, die übermittelt werden sollen, in die CSA bringt (über spezielle und wohldefinierte Schnittstellen). Danach erfolgt ein so genannter *Adressraum-Switch*, d. h., dass der Empfangsadressraum die Kontrolle (und damit die Adressierungsmöglichkeit der CSA) bekommt, damit er die Daten aus der CSA holen und verarbeiten kann. Dass dieses Verfahren umständlich und vor allem mit Overhead verbunden ist, leuchtet ein. Deshalb gibt es inzwischen weitere Varianten für die Kommunikation zwischen Adressräumen.

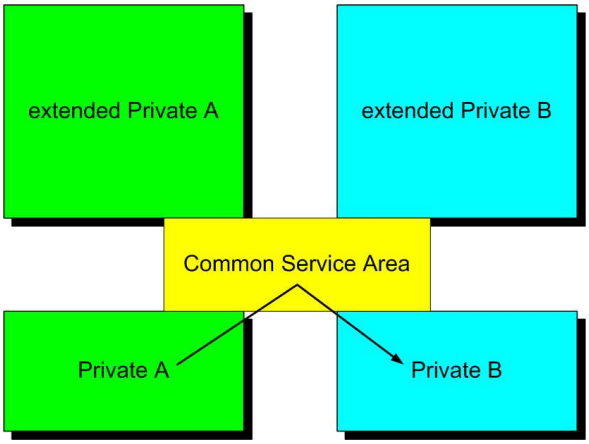


Abbildung 5.7: Kommunikation über die CSA

Bei der CSA-Kommunikation dürfen die Adressräume natürlich nicht direkt auf den gemeinsamen Speicherbereich zugreifen. Dies geschieht über definierte und kontrollierte Systemschnittstellen. Es werden so genannte *Service Request Blocks* (SRB) erstellt und abgearbeitet.

Kommunikation mit Cross Memory Services (CMS)

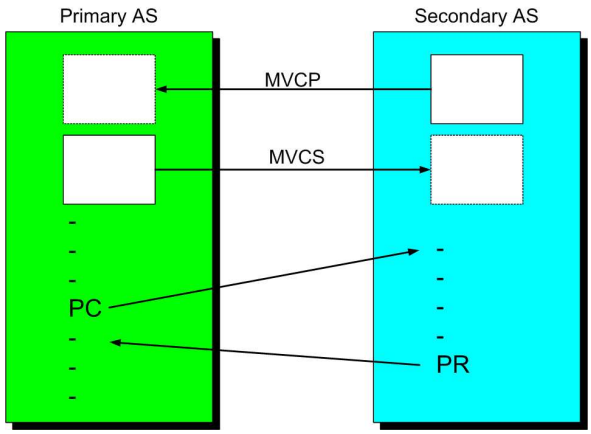
Mit der Extended Architecture (XA) wurde eine effizientere Methode eingeführt, die unter dem Begriff *Cross Memory Services* (CMS) oder auch *Dual Address Space* (DAS) zur Verfügung gestellt wurde. Mit CMS können zwei Adressräume synchron kommunizieren. Was dazu benötigt wird, ist ein zusätzliches Register, das die Adressierung des zweiten Adressraums über eine zweite Segmenttabelle ermöglicht. Dies bedeutet auch, dass CMS nicht nur von der Software, sondern auch von der Hardware unterstützt werden muss. Die entsprechenden Tabellen, über welche die Zugriffsrechte verwaltet werden, werden in einem – zum damaligen Zeitpunkt neuen – Systemadressraum mit dem Namen PC/AUTH verwaltet. PC steht für *Program Call* und AUTH für *Authorization*.

Diese Technik ermöglicht eine wesentlich effizientere Kommunikation. Das hat dazu geführt, dass der Masterscheduler-Adressraum deutlich entlastet werden konnte, indem viele Informationen, die in der Vergangenheit von ihm verwaltet wurden, in eigene Systemadressräume ausgelagert wurden. Beispiele dafür sind der CONSOLE- oder der CATALOG-Adressraum. Die Effizienz und die Robustheit des Betriebssystems wurde durch diese Funktionsteilung deutlich gesteigert.

Wenn im Cross Memory Mode gearbeitet wird, wird der Adressraum, in dem die Instruktionen ausgeführt werden, als *Primary Address Space* bezeichnet. Der Adressraum, in den oder von dem Daten übertragen werden, heißt *Secondary Address Space*.

Mit den Instruktionen MVCP wird vom Secondary in den Primary Address Space übertragen, mit MVCS vom Primary in den Secondary Address Space. Ein Secondary Address Space wird mit der Instruktion *Set Secondary Address Space* (SSAR) definiert. Mit dem Befehl *Program Call* (PC) wird bzgl. der Instruktionsausführung vom Primary in den Secondary Address Space gewechselt. Mit *Program Return* (PR) erfolgt die Rückkehr in den Primary Address Space.

Abbildung 5.8: Kommunikation mit Cross Memory Services



Cross Coupling Facility Communication

Eine Erweiterung zu diesen Konzepten ist die Nutzung von Data Spaces, wodurch das CMS-Konzept von zwei auf mehrere Adressräume gleichzeitig ausgeweitet wird. Außerdem wird mit der *Cross System Coupling Facility* (XCF) eine Software zur Verfügung gestellt, die eine Kommunikation zwischen mehreren Systemen in einer Sysplex-Umgebung realisiert. Data- und Hiperspaces wurden in Kapitel 4 schon behandelt.

XCF unterstützt die folgenden Services für eine Multisystem-Kommunikation:

Group Services
: Makros, um Informationen über Gruppen und Members zu erhalten und sich als Member an einer Gruppe an- bzw. abzumelden

Signalling Services
: Makros, um die Kommunikation zwischen Members einer Gruppe zu realisieren

Monitoring Services
: Makros, um Exit-Routinen anzustoßen und Statusänderungen von Members, Gruppen und Systemen auszutauschen

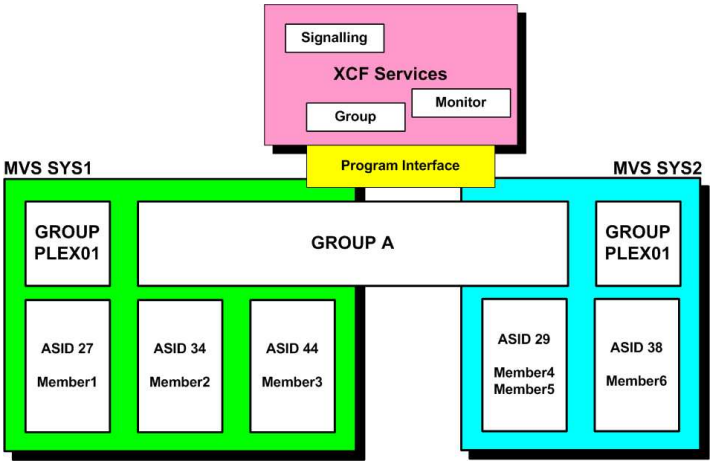


Abbildung 5.9: Kommunikation mit XCF

Serialisation von Ressourcen

Der Begriff „Serialisation" bezeichnet die Notwendigkeit, gleichzeitige Updates auf Ressourcen zu verhindern.

Unterschieden werden dabei:

- Enqueues für die Zugriffssteuerung auf Dateien
- Locks für die Zugriffssteuerung auf sonstige Systemressourcen

ENQ/DEQ-Mechanismen

Ein so genannter *shared ENQ* ermöglicht das gleichzeitige Lesen von Dateien. Ein *exclusive ENQ* dagegen lässt nur jeweils einen Zugriff zu einer gegebenen Zeit zu. Ein exklusiver ENQ auf eine Ressource muss warten, bis alle anderen möglichen User (auch solche mit shared ENQs) die Ressource wieder freigegeben (DEQ) haben.

Die Serialisationsroutinen und Queues für die Kontrolle der Ressourcen werden im GRS-Adressraum verwaltet. GRS steht für *Global Resource Serialization*.

Wenn auf Plattengeräte von unterschiedlichen Systemen zugegriffen wird, spricht man von *shared DASD*. Es gibt im Betriebssystem einen RESERVE/RELEASE-Mechanismus, der dafür sorgt, dass eine Platte nur von einem System gleichzeitig benutzt wird. Für häufige Zugriffe ist dieser Mechanismus nicht brauchbar, da für einen Zugriff jeweils die ganze Platte „reserviert" wird.

GRS-Verarbeitung

Um dieses Problem zu lösen, kann der ENQ/DEQ-Mechanismus auch über Systemgrenzen hinweg in Verbindung mit GRS eingesetzt werden. Die entsprechenden Systeme müssen zu diesem Zweck miteinander verbunden werden. In der Vergangenheit gab es nur die Möglichkeit, die Systeme über Channel to Channel (CTC) Connection in einer Ringform zu verbinden. In Verbindung mit Sysplex werden für die Kommunikation der ENQ/DEQ Requests XCF Services benutzt. In einer Parallel-Sysplex-Umgebung kann eine Star-Konfiguration mit einer Coupling Facility als Kommunikationszentrum eingesetzt werden, wodurch die Performanz deutlich gesteigert werden kann.

Lock-Serialisation

In Mehrprozessorkonfigurationen werden konkurrierende Zugriffe auf Systemressourcen (Systemcode, Kontrollblöcke etc.) über so genannte „Locks" verwaltet. Ein Lock ist ein Wort im Speicher, das eine Systemressource repräsentiert. Ein Lock muss angefordert werden, bevor auf eine Ressource zugegriffen werden kann. Die Komponente *Lock Manager* im MVS ist verantwortlich für das Setzen und Freigeben von Locks im Auftrag eines Adressraums.

Kapitel 6

Speichermedien und SMS

6.1 Speicher-Hierarchie

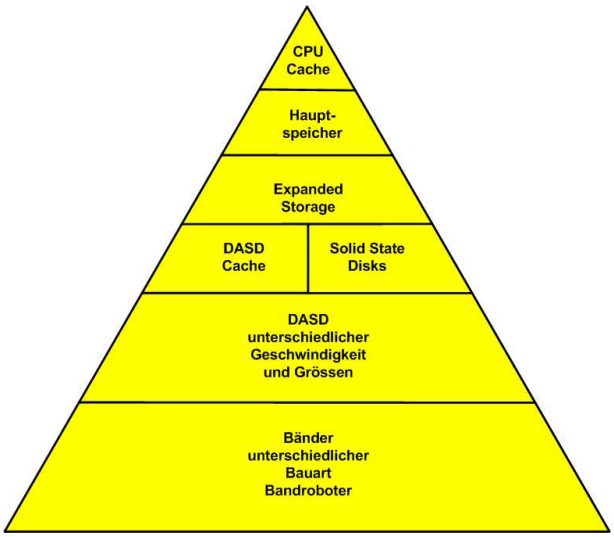
Die Speicherverwaltung hat im Wesentlichen zwei wichtige Anforderungen zu erfüllen: große Datenvolumen zu einem wirtschaftlichen Preis aufnehmen und Daten in akzeptabler Zeit für die Verarbeitung zur Verfügung stellen. Dies ist in der Regel nicht von einer einzigen Speicherebene zu bewerkstelligen. Datenspeicher mit großer Kapazität (z. B. Magnetplatten) haben relativ lange Zugriffszeiten, und Datenspeicher mit kurzen Zugriffszeiten haben nicht die erforderliche Kapazität oder sind zu teuer. Die Anforderungen werden deshalb über eine Speicherhierarchie abgedeckt, in der mehrere Ebenen wirkungsvoll zusammenarbeiten. Unterschiedliche Speichertypen haben in dieser Hierarchie verschiedene Zugriffszeiten, Größen und Preise. Je höher in der Hierarchie, desto teurer, kleiner und schneller sind die Speichermedien.

An der Spitze befindet sich der Cache in der CPU. Dieser ist abhängig vom CPU-Modell und hat zum Ziel, Daten aus dem Hauptspeicher zwischenzuspeichern, da-

mit nicht bei jeder neuen Instruktion die Dynamic Address Translation mit Zugriffen auf Segment- und Page-Tabellen erfolgen muss.

Die zweite Ebene bildet der Hauptspeicher. Er war bis zur Extended Architecture (XA) aufgrund der 24-Bit-Adresse auf 16 MB beschränkt, wobei er mit einem Trick bis zu 32 MB erweitert werden konnte. Bis zur zSeries-Architektur wurde dann mit einer 31-Bit-Adresse gearbeitet, womit maximal 2 GB adressiert werden konnten. Erst mit der 64-Bit-Adresse der zSeries-Architektur ist diese Grenze gefallen.

Abbildung 6.1: Hierarchie unterschiedlicher Speichermedien



Vor der zSeries-Architektur war die Beschränkung auf 2 GB tatsächlich ein großes Problem. Um dieses zu lindern, wurde die dritte Ebene mit dem Expanded Storage eingeführt. Wie der Name schon sagt, wird damit der Hauptspeicher erweitert. Im Grunde genommen handelt es sich um einen schnellen Seitenspeicher. Da er mit 32 Bits in Einheiten einer Seite (4 KB) adressiert wird, kommt man auf eine theoretische Größe von 16 TB. Diese Größe ist jedoch wirklich nur theoretisch, da der Verwaltungsaufwand und die Effizienz maximal das Vier- bis Sechsfache der Hauptspeichergröße erlauben. Mit Einführung der 64-Bit-Adresse wird der Expanded Storage überflüssig.

In der nächsten Ebene befinden sich zwei Speicherarten, die in Sachen Zugriffsgeschwindigkeit in einem ähnlichen Bereich liegen, in der Art und Weise, wie sie arbeiten, jedoch sehr unterschiedlich sind:

Der *DASD Cache* ist ein Zwischenspeicher, der die physischen Zugriffe auf ein Plattengerät reduzieren soll. Die Daten werden in einem Cache gepuffert. Dies funktioniert in Verbindung mit non-volatile (nicht-flüchtigem) Storage auch für Schreibzugriffe, die asynchron auf die Platte geschrieben werden.

Mit ähnlicher Zugriffsgeschwindigkeit gibt es das Speichermedium *Solid State Disk*. Dies ist ein elektronischer Speicher, der eine Platte simuliert. Im Vergleich zu einer Platte ist dieser Speicher jedoch deutlich teurer und spielt in der Praxis heute so gut wie keine Rolle mehr.

In der nächsten Ebene folgen die Plattenmedien. Diese können unterschiedlich groß und unterschiedlich schnell sein.

Zuletzt die Magnetbänder: In der Regel sind das heutzutage Kasettenmedien, die von Robotern verwaltet werden. Die Zeiten, in denen Bandoperatoren den ganzen Tag nichts anderes gemacht haben, als Magnetbänder in einem Bandarchiv zu suchen, in das Magnetbandgerät einzuhängen und danach wieder ins Archiv zurückzubringen, sind vorbei.

Weitere wichtige Begriffe in diesem Zusammenhang sind:

Internal Storage
: Die Speicher in der CPU und der Haupt- bzw. Expanded Storage werden auch als interner Speicher oder manchmal auch als Processor Storage bezeichnet. Diese Speichertechniken arbeiten elektronisch und sind flüchtig (*volatile*). Bei einem Stromausfall gehen die Inhalte verloren, wenn keine Vorkehrungen getroffen werden.

External Storage
: Zugriffe auf den externen Speicher erfolgen über das Kanalsubsystem und werden als Ein-/Ausgaben bezeichnet. Der Speicher ist nicht flüchtig (*non volatile*). Für den DASD Cache und die Solid State Disks, die auch auf elektronischem Weg arbeiten und somit zunächst ebenfalls flüchtig sind, müssen wiederum spezielle Vorkehrungen getroffen werden, damit die Inhalte nicht verloren gehen. Die Geräte arbeiten auf magnetischem oder optischem Weg und sind „removable".

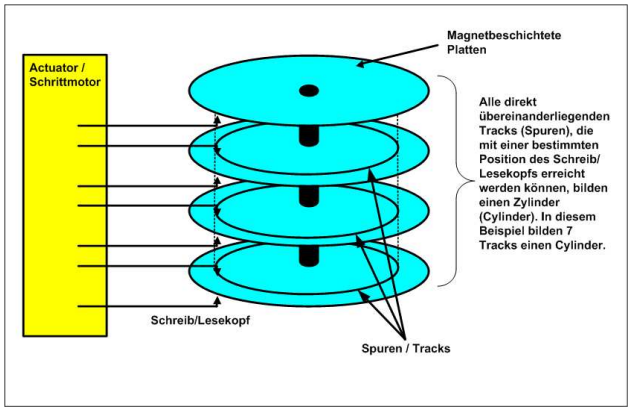
Externe Speichermedien
: Die wichtigsten Speichermedien im Großrechnerbereich sind nach wie vor Platten und Bänder. Platten oder Disks werden im IBM-Jargon oft auch als *Direct Access Storage Device* (DASD) bezeichnet. Bänder waren früher in erster Linie relativ große „Reels". Heute werden vermehrt Kassetten (Cartridges) verwendet.

6.2 Aufbau einer Disk

Ein *Direct Access Storage Device* (DASD) besteht aus mehreren magnetisch beschichteten Platten, die übereinander angebracht sind und sich um eine gemeinsame Achse drehen. Für jede Oberfläche gibt es einen Schreib/Lese-Kopf. Disks werden vor dem Gebrauch mit dem Utility ICKDSF initialisiert; unter anderem wird dabei

auch ein *Volume Table of Content* (VTOC), also ein Inhaltsverzeichnis, geschrieben und ein Plattenname (sechs Zeichen) vergeben. Die Plattennamen müsen in einem laufenden System eindeutig sein, d.h., es gibt im laufenden Betrieb keine zwei Plattengeräte gleichen Namens.

Abbildung 6.2: Aufbau eines DASD



Begriffe, die früher einmal sehr wichtig waren, sind „Track" (Spur) und „Cylinder" (Zylinder). Cylinder wurden oft auch aus Performance-Gründen definiert, da auf alle Spuren eines Cylinders ohne Schreib/Lese-Kopf-Bewegungen zugegriffen werden kann. Heute sehen die Benutzer meist nur noch „logische" Disks mit unterschiedlichen Eigenschaften, die beispielsweise die Perfomance und Verfügbarkeit betreffen.

JCL-Beispiele

Früher:

```
//OUT    DD  DSN=...,
//           DISP=(NEW,CATLG,DELETE),
//           SPACE=(CYL,(nn,nn,nn),RLSE),
```

oder

```
//           SPACE=(TRKS,(nn,nn,nn),RLSE),
```

oder

```
//           SPACE=(3120,(nn,nn,nn),RLSE),
//           ...
```

Die SPACE-Angabe bestimmt die Dateigröße. Es wird ein Primary und ein Secondary Space definiert. „Normale" Datasets können 16 Extents haben, d. h., ein Dataset kann i. d. R. 15-mal um den Secondary Space erweitert werden.

Heute:

```
//OUT    DD  DSN=...,
//           DISP=(NEW,CATLG,DELETE),
//           SPACE=5M,
```

6.3 Magnetbandgeräte und Magnetbänder

Ein Magnetband ist ein günstiges, aber langsames Speichermedium und findet heute hauptsächlich in Verbindung mit Datensicherungen oder für den Austausch von Daten zwischen unterschiedlichen Rechnern Verwendung.

Ursprünglich handelte es sich bei den Bändern um relativ große sogenannte *Reels* (z. B. Typ 3420), die vom Operator von Hand jeweils in ein Gerät eingespannt werden mussten.

Dieses Medium wurde heute weitgehend durch Magnetband-Kassetten (*Cartridges*) abgelöst (z. B. 3480 mit 250 MB, 3490 mit 1 GB, 3590 mit 20 GB), die wesentlich einfacher in der Handhabung sind und meist in Verbindung mit so genannten Tape-Robotern eingesetzt werden (s. unten). Inzwischen gibt es bereits Versuche mit einer Terabyte-Kasette.

JCL-Beispiel für Magnetbandzugriff:

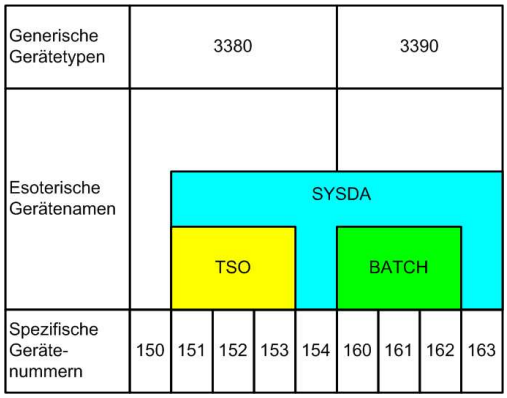
```
//OUT    DD  DSN=...,
//           DISP=(NEW,CATLG,DELETE),
//           UNIT=TAPE,
//           VOL=SER=TAP111,
//           LABEL=(SL,1)
```

Bei Tape-Verarbeitung wird keine Space-Angabe benötigt wie bei den Platten.

Gerätenamen

Mit der Systemgenerierung werden auch die zur Verfügung stehenden Gerätenamen definiert. Auf der Ebene der esoterischen Gerätenamen werden in der Regel De-facto-Standards wie SYSDA und TAPE verwendet.

Abbildung 6.3: Übersicht über Gerätenamen



6.4 Neue Speichertechnologien

Neue Applikationen verwenden vermehrt neue Technologien. Gegenüber traditionellen Informationstechniken, wo es vor allem um Daten in Textform ging, sind heute vermehrt Daten im Bereich Sprache (Voice), Bilder (Images) und Video im Einsatz. Man spricht hier auch von so genannten „New Data Sources".

Aber nicht nur diese New Data Sources sind der Grund, die Speichertechnologie zu überdenken, sondern auch die Art und Weise, wie die Daten heute zur Verfügung gestellt werden müssen. Um nur das Internet als neue Applikationsumgebung zu nennen, ist es zwingend, langfristig Daten aller Art elektronisch verfügbar zu machen (noch 1998 waren höchstens 10 Prozent aller Informationen elektronisch verfügbar).

Die heute zur Verfügung stehenden Speichermedien sind vor allem:

- magnetische Disks (DISC)
- magnetische Bänder (TAPE)
- optische Speichermedien (OPTICAL)

Bänder werden für Datenaustausch und Archivierung von Daten sowie für Migration und Backup eingesetzt.

Der Unterschied zwischen Migration und Backup ist, dass bei der Migration nur immer eine Kopie vorhanden sein muss, bei Backup aber meist mehrere Kopien in verschiedenen Versionen. Das führt dazu, dass mit der wachsenden Datenmenge

die Backup-Datenmenge zunimmt. Die Backup-Menge ist oft 10-, 20- oder sogar 30-mal größer als die der Originaldaten.

Gleichzeitig ist es heute zwingend, dass eine Applikation 24 Stunden an 7 Tagen die Woche verfügbar sein muss. Den täglichen „Housekeeping"-Jobs steht daher ein immer engeres Zeitfenster zur Verfügung, um gewünschte Sicherungen durchzuführen. In der Regel werden heute nicht mehr sämtliche Daten täglich gesichert, sondern nur noch jene, die auch tatsächlich verändert worden sind. Man spricht dann von einer inkrementellen Sicherung. Gleichzeitig steigt auch die Anzahl von Anfragen auf die gesicherten Daten.

Um diesen neuen Anforderungen gerecht zu werden, müssen Sicherung und Zugriff auf gesicherte Daten verbessert und Abläufe dazu möglichst automatisiert werden. Es wird dabei in Kauf genommen, dass bei selten benutzten Daten ein paar Sekunden Wartezeit auftreten, solange diese Daten überhaupt noch erreichbar sind.

Virtuelle Systeme

Vor mehr als 30 Jahren hatte IBM das *Virtuelle Speichermanagement* angekündigt. Die Idee dahinter war, dem Benutzerprogramm den Speicher zur Verfügung zu stellen, den es adressieren kann, obwohl oft gar nicht so viel Hauptspeicher installiert war. Nicht benutzte Teile wurden ausgelagert und bei Bedarf wieder eingelagert. Und das Ganze, ohne dass sich das Benutzerprogramm selbst darum kümmern musste.

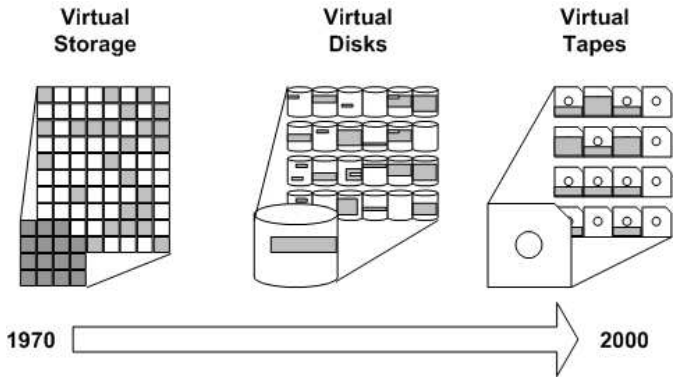


Abbildung 6.4: Entwicklung virtueller Systeme

Die erste Version virtueller Plattengeräte hat IBM mit dem *Mass Storage Subsystem* (MSS) realisiert. Das MSS hatte die Aufgabe, ganze Disks bei Bedarf zur Verfügung zu stellen oder auszulagern. Wurden die Disks nicht mehr verwendet, wurden sie auf so genannte Cartridges ausgelagert. Dies kam vor allem in der Batch-Verarbeitung zum Einsatz. Dadurch wurde es möglich, anstelle langsamer Tape Units so genannte MSS Volumes einzusetzen.

Heute sind die virtuellen Disks unter dem Virtual Array Subsystem (RAMAC) verfügbar. Virtuelle Disks haben einen ähnlichen Ansatz wie das virtuelle Speichermanagement, denn sie vermitteln dem Benutzer den Eindruck eines unbegrenzten Speichers. Die Hardware speichert tatsächlich jedoch nur die geschriebenen Daten. Interne Kompressionsverfahren verdichten diese innerhalb der Subsysteme ohne Einwirkung der Betriebssysteme und sind somit völlig autonom.

Die *Virtual Tapes Subsystems* (VTS) haben für den Benutzer dieselben Vorteile wie die Virtuellen Disks. Mit diesem Konzept kann der Benutzer ein virtuelles Band verlangen (SCRATCH oder ein ganz bestimmtes). Das verlangte Band wird nun aber nicht auf einer physischen Bandstation bereitgestellt, sondern der Mount erfolgt im VTS.

Die Vorteile liegen auf der Hand:

- kein physikalischer Mount (Geschwindigkeit)
- kein Zurückspulen (Bandstation nach einem Unmount ohne Wartezeit sofort verfügbar)
- nur genutzter Bandplatz wird wirklich geschrieben
- virtuelles Band kann jederzeit als physikalisches Band erstellt werden

Bisher wurde Speicherplatz in Disk Units geplant. Die Menge der Channels war dabei ein wichtiger Bestandteil, wenn es darum ging, die Daten möglichst effizient dem Endbenutzer zur Verfügung zu stellen.

Heute werden die Daten eher in Gigabytes und Zugriffspfaden definiert. Die Definition von Disk-Zugriffen hat heute auch keine große Aussagekraft mehr, da die Daten vielfach in einem Cache zwischengelagert und aus diesem Cache wieder gelesen werden.

Auch bei Bändern hat sich das geändert. Früher stand die Anzahl der Bänder oder der Bandstationen im Vordergrund, heute die Giga- (GB), Tera- (TB) und Petabytes (PB) auf Bändern und als Transferraten pro Minute, Stunde oder Tag.

Für die Verwaltung der Magnetbänder wird in großen Installationen meist ein Bandverwaltungssystem eingesetzt.

6.5 System Managed Storage (SMS)

Das MVS verfügt über ein historisch gewachsenes Data Management, dessen Grundlagen bereits erläutert wurden.

Die zu verwaltenden Datenmengen sind in den letzten Jahren allerdings exponentiell angewachsen, so dass in vielen Großunternehmen eine Speicherverwaltung „von Hand" schlichtweg unmöglich ist.

Das durchschnittliche jährliche Wachstum der Datenmengen, die auf dem Mainframe verwaltet werden, liegt bei 40 Prozent. Viele Unternehmen befinden sich bereits im dreistelligen Terabyte-Bereich, d. h., dass es nicht mehr lange dauert, bis große Unternehmen mit Petabytes fertig werden müssen.

Als Ergänzung zum ursprünglichen Data Management wurden darum die ***Data Facility Products*** (DFP) als Zusatzkomponenten angeboten. Dazu gehörten neben SMS auch die ***Resource Access Control Facility*** (RACF) als Security-Mechanismus und ***System Managed Storage*** (SMS) mit der Unterkomponente ***Hierarchical Storage Manager*** (HSM).

Inzwischen sind sowohl SMS als auch RACF Basiskomponenten des Betriebssystems.

Ziel von SMS

Das Ziel von SMS (der „offizielle" Name ist DFSMS, wobei das DF für ***Data Facility*** steht) ist es, logische und physische Sicht auf die Daten zu trennen, was den Endbenutzer von der Fragestellung entlastet, auf welchem physischen Datenträger seine Daten abgelegt werden. Es sollen nur noch logische Anforderungen definiert werden. Das System kümmert sich um die physische Speicherung, wobei Vorgaben und Richtlinien eines Storage-Administrators berücksichtigt werden.

DFSMS hat vier funktionelle Komponenten, die zu einem einzigen integrierten Paket zusammengefügt wurden:

DFSMSdfp
: Darunter fallen die „klassischen" Data-Management-Funktionen, viele Utilities, unter anderem auch ***Access Method Services*** (IDCAMS) sowie die ***Integrated Storage Management Facility*** ISMF, außerdem auch neuere Komponenten, die über Unix System Services implementiert werden, wie Network File System (NFS) und die Unterstützung des Service Message Block (SMB).

DFSMSdss
: Hinter den DFSMSdss (Data Set Services) steht der „Data Mover" des Betriebssystems. Darunter versteht man das Kopieren, Verwalten von Backups und Space-Management-Funktionen.

DFSMShsm
: Hinter DFSMShsm (Hierarchical Storage Manager) stehen ebenfalls Backup- und Space-Management-Funktionen sowie zusätzliche Funktionalitäten für Migration und Recovery von Daten. HSM und DSS arbeiten sehr eng zusammen.

DFSMSrmm

Der DFSMSrmm (Removable Media Manager) kümmert sich um Wechseldatenträger wie Magnetbänder und Magnetkassetten.

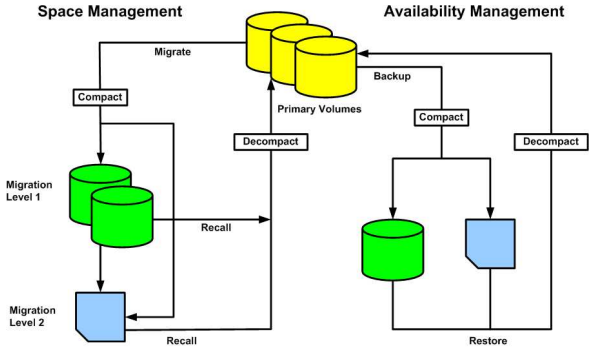
Der Hierarchical Storage Manager

Eine wesentliche Komponente von SMS ist der *Hierarchical Storage Manager*. Er kann auch ohne SMS eingesetzt werden, aber nicht umgekehrt; das bedeutet, der Einsatz von System Managed Storage setzt ein Produkt wie den Hierarchical Storage Manager voraus.

Der Hierarchical Storage Manager hat zwei Hauptfunktionen:

- Space Management für die Platzverwaltung und das Ein-/Auslagern von Dateien
- Availability Management für Backup/Recovery und Restores von Daten

Abbildung 6.5: Funktionen des HSM



Space Management

Bei Space Management geht es darum, knappen und teuren Plattenplatz vornehmlich für Daten und Informationen zu nutzen, die auch aktiv bearbeitet werden, und nicht von Datenbeständen, auf die schon längere Zeit (Wochen, Monate, Jahre) nicht mehr zugegriffen wurde.

Ein Beispiel: Ein COBOL-Programmierer legt eine Datei für seinen Quellcode für ein bestimmtes Projekt an und benutzt diese relativ häufig während der Entwicklungsphase. Dann ist diese Entwicklungsarbeit beendet, und die Datei wird nicht mehr benutzt. Er könnte nun natürlich seinen COBOL-Quellcode auf Band sichern und die Datei löschen, doch das ist aufwändig – so bleibt die Datei meist einfach auf der Platte liegen.

Das Space Management des HSM ist nun eine Systemkomponente, die die Nutzung der Dateien überwacht und automatisch solche, die nicht mehr benutzt werden, in der Speicherhierarchie „nach unten" verlagert bzw. auslagert (*migrate*). Wird eine Datei dann irgendwann doch wieder benötigt, holt es diese automatisch wieder zurück (*recall*).

Der Automatismus bezieht sich auf:

- Migration der Daten
- Recall der Daten
- Löschen temporärer Dateien
- Löschen abgelaufener Dateien
- Freigeben von nicht benötigtem Plattenplatz
- Space-sparende Funktionen (Extents, Compression etc.)

Volumes, die für die direkte Nutzung zur Verfügung stehen, werden als *Primary Volumes* oder *Migration Level 0 Volumes* bezeichnet. Wenn ein Primary Volume beispielsweise einen bestimmten Füllgrad erreicht, werden Dateien, die am längsten nicht mehr „angefasst" wurden, automatisch migriert.

Für die Migration stehen zwei Levels zur Verfügung. Der *Migration Level 1* wird von Plattengeräten gebildet, die beispielsweise eine besonders hohe Kapazität haben, dafür möglicherweise etwas langsamer sind. Außerdem werden die Daten bei der Migration komprimiert (*compacted*) und benötigen auf den Migrationsplatten deutlich weniger Platz. Wenn Daten auf den ML1-Platten weiterhin lange nicht angefasst werden, werden sie auf *Migration Level 2* migriert. Migration Level 2 sind zumeist Magnetbänder, die in der Regel von Robotern verwaltet werden.

Ein migriertes Volume wird bei entsprechenden Anzeigen unter ISPF, wo normalerweise der Plattenname angezeigt wird, als MIGRAT gekennzeichnet. Beim Zugriff erfolgt automatisch ein Recall. Wenn die Datei auf Migration Level 2 (Magnetband) ausgelagert ist, wird der Benutzer informiert, dass nun ein Recall von ML2 erfolgt, und er kann entscheiden, ob er darauf warten will oder ob der Recall im Hintergrund erfolgen soll.

Availability Management

Auch hier wieder ein Blick in die Vergangenheit: Bevor es einen Hierarchical Storage Manager gab, wurden sämtliche Platten in einem Rechenzentrum täglich vollumfänglich gesichert. Bei den erwähnten Beständen im dreistelligen Terabyte-Bereich wäre das gar nicht mehr durchführbar. Deshalb werden nur noch periodisch nach einem rollierenden System vollumfängliche Backups gefahren, und danach

nur noch inkrementelle Backups, d. h., dass nur noch Daten gesichert werden, die seit der letzten Sicherung verändert wurden.

Das *Availability Management* kümmert sich darum, dass von wichtigen Dateien immer eine aktuelle Kopie besteht, um in Problemfällen eine verloren gegangene oder zerstörte Datei wiederherstellen zu können. Es werden automatisch periodische Datensicherungen durchgeführt. Wie oft das geschieht und wie viele Generationen an Backups gehalten werden, wird vom Storage-Administrator festgelegt und ist zu einem großen Teil vom *Last Level Qualifier* einer Datei abhängig.

Funktionen des Availability Management:

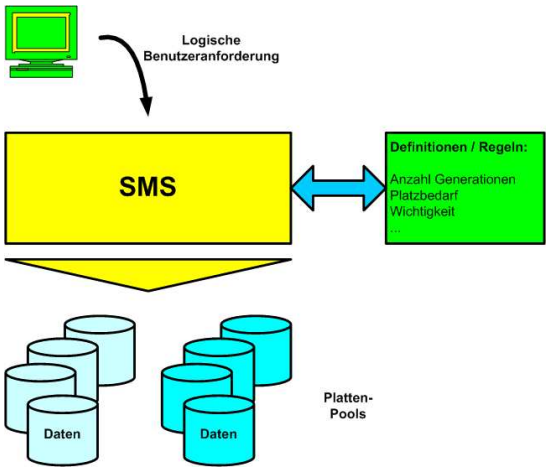
- Kopieren der Daten
- Inkrementelle Backups
- Reorganisation

System Managed Storage (DFSMS)

DFSMS ist ein Satz von Konstrukten, Schnittstellen und Routinen, der die DFSMS/MVS-Produkte nutzt, um die Speichersysteme einer Installation effizient zu verwalten.

Die Kernlogik von SMS (ACS-Routinen, ISMF, SMS-Konstrukte etc.) befindet sich in DFSMSdfp. DFSMShsm und DFSMSdss sind hauptsächlich in Verbindung mit der so genannten *Management Class* (eines von fünf SMS-Konstrukten) involviert. Mit SMS werden Verfügbarkeits- und Performance-Ziele definiert, Vorgaben für typische Dateien gemacht und die ganze Sicherungsverwaltung festgelegt.

Abbildung 6.6: Funktionen von SMS



Die Ziele von SMS

Vereinfachtes Anlegen von Dateien

Ohne SMS musste ein Benutzer die Geräteeinheit (UNIT) und den Gerätenamen (VOLUME) mitgeben, wenn ein Dataset angelegt werden sollte. Außerdem (und das ist noch mühsamer) musste der Speicherplatz definiert werden: in Einheiten von Spuren (Tracks) oder Zylindern (Cylinders), deren Kapazität vom jeweils eingesetzten individuellen Gerätetyp abhängig ist. Mit SMS ist das System für die Platzierung der Dateien zuständig. Die Dateigrößen werden über Defaults festgelegt bzw. können in KB bzw. MB angegeben werden.

Bessere Steuerung und Kontrolle

Der zur Verfügung stehende freie Platz auf den Speichergeräten wird automatisch überwacht. Es werden entsprechende Thresholds sowohl für Plattengeräte als auch für Bandgeräte definiert, bei deren Erreichen SMS adäquat reagiert.

Verbesserte I/O Performance

SMS verbessert die DASD I/O Performance, indem automatisch das Caching von Daten gesteuert wird.

Automatisches Space Management

SMS verwaltet automatisch den Plattenplatz und sorgt dafür, dass dieser nicht von alten und ungenutzten Dateien belegt wird. Über Systemvorgaben wird definiert, wie lange welche Datentypen ungenutzt auf welchen Speichergeräten liegen dürfen. Nicht verwendete Dateien werden automatisch migriert, um Platz für aktive und neu angelegte Dateien zu machen.

Automatisches Tape Space Management

Auch der verfügbare Platz auf Magnetbändern wird automatisch verwaltet. In Verbindung mit Robotersystemen kann die gesamte Bandverwaltung weitgehend automatisiert werden, ohne dass Operator-Eingriffe notwendig sind.

Verbesserte Verfügbarkeit

Die Backup-Vorgaben für DASD-Dateien werden in Abhängigkeit von Konventionen getroffen. Dies betrifft nicht nur „normale" Dateien, sondern auch CICS-, DB2- und andere Dateien.

VSAM- und HFS-Dateien

Datasets können logisch gruppiert werden, damit die entsprechend zusammengehörenden Dateien gemeinsam im Sinne von Backup/Recovery verwaltet werden. Dies wird realisiert durch den *Aggregate Backup and Recovery Support* (ABARS).

Vereinfachte Anpassung an unterschiedliche Gerätetypen

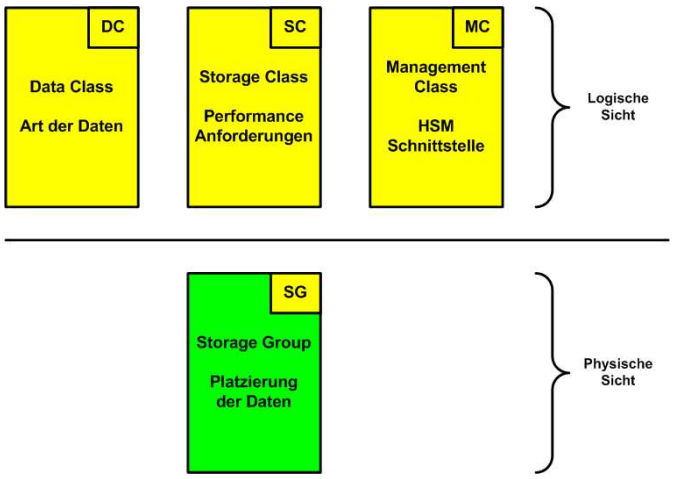
SMS kann automatisch neue Gerätetypen einbinden und nutzen, ohne dass die JCL angepasst werden muss, da keine UNIT- und VOL-Angaben gemacht werden müssen. Die „richtigen" Blockungsfaktoren werden automatisch eingesetzt.

Die Policy im Bezug auf das Storage Management eines Unternehmens wird über Klassen definiert, über die dann Verwaltung, Performance und Verfügbarkeit gesteuert werden.

ACS-Routinen

Mit Hilfe von *Automatic Class Selection* (ACS)-Routinen werden Automatismen implementiert, die auf Namenskonventionen beruhen.

Abbildung 6.7: SMS-Konstrukte



Die ACS-Routinen sorgen für den Automatismus. Unter anderem wird hier über Namenskonventionen gesteuert, wie und wo eine Datei angelegt wird.

ACS-Routinen werden in einer speziellen Scriptsprache geschrieben und in Objekte übersetzt, die in SMS eingebunden werden. Mit ACS-Routinen kann die Einhaltung von Namensstandards erzwungen werden. Wichtig sind hier vor allem die *High Level Qualifier* und die *Low Level Qualifier* (HLQ und LLQ).

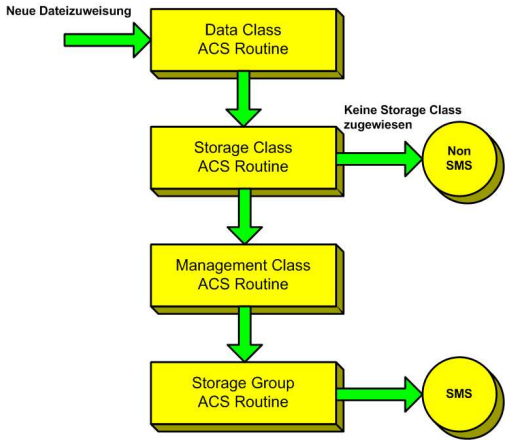


Abbildung 6.8: Ablauf von ACS-Routinen

Damit SMS „funktioniert", müssen Namenskonventionen eingeführt und vor allem auch eingehalten werden. Durch stimmige und durchgängige Konventionen wird der Umgang mit Datasets (für *alle* Benutzer) und die Verwaltung von SMS (für die Administratoren) sehr stark vereinfacht und rationalisiert.

Data Classes

Im Folgenden soll ein Beispiel von Konventionen für Data-Class-Namen gezeigt werden. Wichtig: Es ist nur ein Beispiel, in anderen Umgebungen können die Konventionen anders aussehen. Entscheidend ist jedoch, dass es Konventionen gibt.

Die ersten beiden Buchstaben eines SMS-Konstrukts bezeichnen den Typ. Für Data Classes ist dies „DC". Die dann folgenden Buchstaben und Ziffern kennzeichnen das Satzformat und die Satzlänge.

Die Datenklassen (Data Classes) von SMS werden in erster Linie durch die Low Level Qualifier (LLQs) gesteuert. Über die LLQs werden dann automatisch mit Hilfe der ACS-Routinen die Data Classes zugeordnet.

Tabelle 6.1: Beispiel-LLQs

LLQ	Data Class	Beschreibung
COBOL	DCPF80	COBOL Source Code
ESDS	DCVES	VSAM Entry Sequenced
H	DCPV255	C Header Files
LIST	DCSV255	Listen ohne ASA-Steuerzeichen
LOAD*	DCPULOAD	Load Libraries

SMS ist auf die Einhaltung der Namenskonventionen angewiesen. Wenn eine Datei mit dem Last Level Qualifier COBOL angelegt wird, geht das System davon aus, dass eine Bibliothek mit 80-stelligen Records angelegt werden soll. Darin können dann keine Lademodule abgelegt werden, da dafür andere Anforderungen bzgl. der Datensätze bestehen.

Eine Data Class kann explizit über den DC-Parameter der DD-Anweisung per JCL mitgegeben werden, oder sie wird implizit über die Automatic-Class-Selection-Routinen (ACS) zugewiesen.

Storage Classes

Bei einer *Storage Class* geht es um Ziele bezüglich Performance und Verfügbarkeit, beispielsweise um die Festlegung, ob Daten, die auf einem Plattengerät abgelegt werden, immer von einem Cache verwaltet werden (*always cache*), nie von einem Cache verwaltet werden (*never cache*) oder dann von einem Cache verwaltet werden, wenn dieser nicht überlastet ist (*may cache*).

Bei Attributen bezüglich Verfügbarkeit kann bestimmt werden, ob Plattenspiegelung (*Dual Copy*), Kopieren im laufenden Betrieb (*Concurrent Copy*), Kopieren über große Entfernungen (*Remote Copy*) oder Snapshot Features genutzt werden sollen.

Eine Storage Class kann explizit über den SC-Parameter der DD-Anweisung per JCL mitgegeben werden, oder sie wird implizit über die Automatic-Class-Selection-Routinen (ACS) zugewiesen.

Management Classes

Mit Hilfe der Management-Klassen werden Anforderungen bzgl. Aufbewahrungsdauer, Migration sowie Backup und Recovery spezifiziert. Sie sind im Prinzip die Schnittstelle zum Hierarchical Storage Manager.

Auch eine effiziente Verwaltung der Management Class ist auf die Einhaltung der Namenskonventionen angewiesen. Wenn beispielsweise eine Datei mit dem Last Level Qualifier COBOL angelegt wird, geht SMS davon aus, dass automatisch eine tägliche Sicherung gemacht werden soll. Wenn eine Datei mit Last Level Qualifier TMP angelegt wird, geht das System davon aus, dass es sich um eine temporäre Datei handelt, und macht keine automatische Sicherung.

Eine Management Class kann explizit über den MC-Parameter der DD-Anweisung per JCL mitgegeben werden, oder sie wird implizit über die Automatic Class Selection (ACS)-Routinen zugewiesen.

Storage Groups

Eine *Storage Group* ist eine Zusammenfassung von Geräten, die für bestimmte Daten verwendet werden sollen. So gibt es beispielsweise Gruppen von Page Volumes, DASD Volumes, Tape Volumes etc., die wiederum für bestimmte Verwendungszwecke unterteilt werden können, beispielsweise für DB2-Daten, für IMS-Daten, für Test, für Produktion etc.

Aufgrund der für eine Datei angeforderten Data Class, Storage Class und Management Class ist das System in der Lage, die richtige Storage Group für die Datei zu bestimmen und letztendlich die für das System und den Benutzer günstigste Platte zuzuweisen.

Aggregate Backup and Recovery Support (ABARS)

Mit ABARS können Daten anwendungsspezifisch zusammengefasst werden, um Backup/Recovery effizient zu organisieren. Da es um die Anwendungen geht, wird das Kürzel ABARS auch oft mit *Application Backup and Recovery Support* übersetzt.

Ziel ist es, alle notwendigen Daten für eine Anwendung als eine Einheit zu definieren, die dann in Bezug auf Backup/Recovery verwaltet wird. Wenn ein Katastrophenfall eintritt, können die Daten schnell, umfassend und vollständig lokal oder remote wiederhergestellt werden.

Für die Sicherung der Daten in Verbindung mit ABARS ist SMS Voraussetzung. Die Wiederherstellung der Daten kann auch in Systemen erfolgen, die kein SMS nutzen – jedoch nur unter der Voraussetzung, dass keine Organisationsformen eingesetzt werden, die SMS brauchen, wie beispielsweise PDS/E.

Interactive Storage Management Facility (ISMF)

Mit ISMF können Daten und Speichermedien interaktiv analysiert und verwaltet werden, mit interaktiven Schnittstellen zu den Funktionen Space Management und Backup/Recovery der Komponenten DFSMShsm und DFSMSdss sowie DFSMSrmm für die Bandverwaltung. Daneben sind noch weitere Produkte involviert wie beispielsweise RACF, DFSORT und ICKDSF.

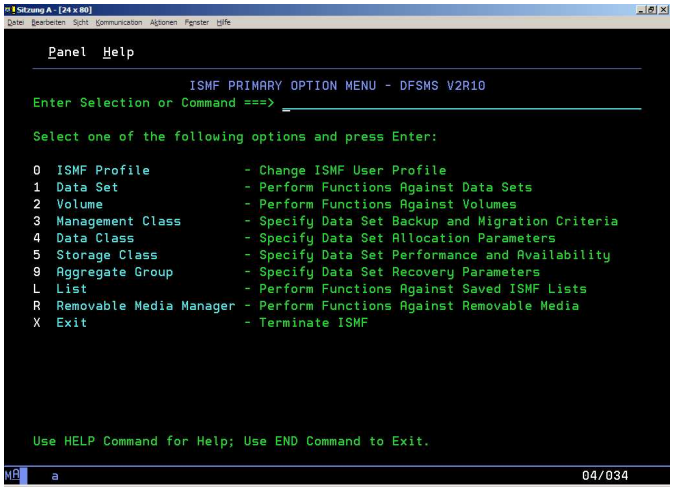
ISMF ist eine menügesteuerte ISPF-Anwendung. Man kann damit

- Informationen über Dateien und Volumes anzeigen
- Listen nach diversen Kriterien erzeugen
- Listen editieren und anpassen

- Dateien und Backup-Kopien löschen
- Dateien und Volumes kopieren
- Migration und Recall steuern
- Backup von Dateien und Volumes veranlassen
- Recovery von Dateien und Volumes veranlassen

Ein Storage-Administrator nutzt ISMF, um die unternehmensspezifischen Regeln zu definieren, und verwaltet damit die SMS-Klassen, Gruppen und ACS-Routinen. Die Daten werden in ein *Source Control Data Set* (SCDS) abgelegt und über ISMF oder per Operator-Kommando aktiviert.

Abbildung 6.9: ISMF Primary Option Menu



Mit Hilfe der *Data Tag Language* (DTL) kann ISMF auch eine grafische Schnittstelle unterstützen.

Die Arbeit mit ISMF gestaltet sich sehr komfortabel: Menügeführt werden Attribute definiert und aufgrund dieser Eingaben eine Liste erzeugt. Gegen diese Liste oder auch gegen individuelle Dateien können dann Space-Management- oder Backup/Recovery-Funktionen ausgelöst werden. Welcher Benutzer welche Funktionen aufrufen darf, hängt von den entsprechenden Berechtigungen ab, die wiederum im Security Server RACF festgelegt sind.

Kapitel 7

Batchverarbeitung und Timesharing

7.1 Batchverarbeitung mit Job Control Language (JCL)

Der IBM Mainframe war in den Anfangszeiten, als er im Jahr 1964 mit der Definition der /360-Architektur ins Leben gerufen wurde, ein reines Batchverarbeitungs-System, im Gegensatz zu UNIX-Systemen, die rund fünf Jahre später als reine Dialogsysteme konzipiert wurden.

In der jüngeren Vergangenheit war immer wieder zu hören, heutzutage sei keine Batchverarbeitung mehr vonnöten, weil alles interaktiv und online abgewickelt werde. Das sieht in der Praxis anders aus! Es gibt sehr viele Abläufe, die sinnvollerweise mit Batchverarbeitung realisiert werden. Ein typisches Beispiel dafür sind die täglichen Datensicherungen. Bei einer großen Schweizer Bank beispielsweise läuft eine tägliche Tagesendverarbeitung von über 40000 Batchjobs!

Selbst bei modernen E-Business Transaktionen beträgt der Batch-Anteil 50 Prozent oder mehr.

Ein weiteres Beispiel für einen Batchjob ist das Übersetzen und Binden eines Quellprogramms. Ob es sich dabei um ein COBOL- oder um ein C-Programm handelt, ist zweitrangig. Der Ablauf ist prinzipiell derselbe.

Wenn aus einem Quellprogramm ein Lademodul (ausführbares Programm) entstehen soll, sind dafür zwei Schritte notwendig. Der erste Schritt ist die Übersetzung des Quellcodes in ein Objektmodul, der zweite das Binden des Objektmoduls in ein ausführbares Lademodul. Diese beiden Schritte müssen mit der Job Control Language (JCL) entsprechend abgebildet werden.

Job Control Language (JCL)

Es gibt eine gute und eine schlechte Nachricht in Verbindung mit der *Job Control Language*. Zunächst die gute: Es gibt eigentlich nur drei verschiedene JCL-Anweisungen; die schlechte: diese haben eine Unzahl von Parametern und Subparametern, die die Handhabung in der Praxis nicht immer leicht machen. Dies betrifft vor allem die Parameter der DD-Anweisungen.

JOB

Die erste Anweisung, die immer vorhanden sein muss, ist die JOB-Anweisung. Sie grenzt die Batchjobs voneinander ab und definiert beispielsweise einen Jobnamen, eine Ausführungsklasse und Accounting-Informationen. Selbst wenn die JOB-Anweisung in vielen Installationen nicht explizit definiert werden muss, ist sie dennoch vorhanden: In diesen Fällen wird sie von einem so genannten *Submit-Exit* automatisch erstellt. Abhängig von der Benutzerkennung, die den Job abschickt, wird ein Jobname erstellt, die Accounting-Information eingefügt und eine Default-Jobklasse angenommen.

EXEC

Die zweite Anweisung ist die EXEC-Anweisung, die einen Jobstep definiert. Sie legt fest, welches Programm aufgerufen werden soll. In unserem oben erwähnten Fall des Übersetzens und Bindens eines Quellcodes werden zwei Jobsteps benötigt: einer für den Aufruf des Compilers und ein zweiter für den Aufruf des Binders, der im MVS-Umfeld auch als *Linkage Editor* bezeichnet wird. Darüber hinaus spielt die EXEC-Anweisung keine besondere Rolle. Etwas komplizierter wird es bestenfalls, wenn eine Ausführung nur unter bestimmten Bedingungen erfolgen soll. Auf derartige Spezialfälle gehen wir hier jedoch nicht weiter ein.

DD

Die dritte Anweisung ist die DD-Anweisung. DD steht für *Data Definition* und weist dem aufgerufenen Programm über so genannte DD-Namen

Ein-/Ausgabedaten zu. Die Eingabe für die Übersetzung ist beispielsweise der Quellcode, die Ausgabe das Objektmodul. Die Eingabe für den Linkage Editor ist das Objektmodul und die Ausgabe das Lademodul. Beispiel für einen Batchjob, mit dem ein Standard-Kopiervorgang aufgerufen wird:

```
//UWGR1    JOB   (007,T1),'GREIS',CLASS=C,MSGCLASS=A
//COPY     EXEC PGM=IEBGENER
//SYSIN    DD   DSN=UWGR.TEST.COBOL(A1),DISP=SHR
//SYSOUT   DD   DSN=UWGR.TEST.COBOL(A5),DISP=SHR
//SYSPRINT DD   SYSOUT=*
```

Details zu den einzelnen Parametern würden hier zu weit führen. Wer JCL in der täglichen Arbeit effizient nutzen will, sollte ein entsprechendes Seminar besuchen oder sich wenigstens ein gutes Buch über JCL besorgen[1] und die Systemliteratur zu Rate ziehen. Die DD-Namen aus dem obigen Beispiel SYSIN und SYSOUT entsprechen den STDIN- und STDOUT-Funktionen in UNIX/Linux-Umgebungen. Leider sind sie in der MVS-Umgebung nicht gleichermaßen standardisiert, so dass von einigen Utilities auch andere Namen verwendet werden. Man muss in diesem Fall die entsprechenden Benutzerhandbücher konsultieren, um herauszubekommen, welche DD-Namen in den jeweiligen Programmen verwendet werden.

Prozeduren

Eine Vereinfachung bieten Prozeduren, die für diverse Aufgaben per Default zur Verfügung stehen, von der jeweiligen Installation eingerichtet werden oder auch selbst codiert werden können. Für Abläufe und Aufgaben, die immer wieder benötigt werden, lohnt sich das auf jeden Fall.

Man unterscheidet so genannte *Instream-Prozeduren*, die in der JCL selbst definiert werden, und *katalogisierte Prozeduren*, die in Prozedur-Bibliotheken abgelegt sind.

Mit der PROC- (für *Procedure*) und der PEND-Anweisung (für *Procedure End*) werden Instream-Prozeduren eingeklammert.

In Prozeduren lassen sich symbolische Parameter definieren, die beim Aufruf der Prozedur durch aktuelle Werte ersetzt werden. Außerdem können EXEC- und DD-Anweisungen beim Aufruf ergänzt oder überschrieben werden.

Mit der SYS1.PROCLIB steht eine Standardbibliothek zur Verfügung, die in jeder MVS-Umgebung eingerichtet sein muss. Installationsspezifisch gibt es noch zahlreiche weitere Bibliotheken, die in eine Systemumgebung eingeklinkt und deren Prozeduren dann ebenfalls automatisch gefunden werden. Welche Bibliotheken eingebunden sind, wird über die JES-Definitionen (*Job Entry Subsystem*) gesteuert. Auch private Prozedurbibliotheken können erstellt und gepflegt werden. Sie

[1] Beispielsweise von Michael Winter, „MVS/ESA JCL", Oldenbourg Verlag, 3. Aufl. 1999.

müssen in der JCL, die eine private Prozedur aufruft, über eine JCLLIB-Anweisung bekannt gemacht werden.

Beispiel für den Aufruf einer Prozedur, über die beispielsweise ein COBOL-Programm übersetzt und gebunden werden kann:

```
//UWGR5       JOB   (007,T1),'GREIS',CLASS=C
//COPY        EXEC PROC=COBCL
//COB.SYSIN   DD   DSN=UWGR.TEST.COBOL(A1),DISP=SHR
//LKED.SYSIN  DD *
  NAME A1(R)
```

In der Prozedur COBCL sind zwei Jobsteps mit entsprechenden EXEC-Anweisungen definiert. Mit COB.SYSIN bezieht man sich auf die Eingabe für den Jobstep COB (Aufruf des Compilers) und mit LKED.SYSIN auf die Eingabe für den Jobstep LKED (Aufruf des Linkage Editor). Auch hier gilt wieder: Man muss diese Namen kennen, um sich darauf beziehen zu können.

Jobablauf-Steuerungen

Zweifellos eine Stärke der Batchverarbeitung auf dem Mainframe ist die Möglichkeit, die Abläufe von Batchjobs zu automatisieren. Eine automatische Abwicklung mehrerer tausend Batchjobs pro Tag ist eine enorme Integrationsaufgabe für das Systemmanagement. Wenn es darum geht, dass sich bestimmte Batchjobs nicht überschneiden dürfen, wenn Fehler abgefangen und Jobs in bestimmte Bearbeitungsläufe münden müssen, greifen entsprechende Werkzeuge. Um komplette Produktionen zu steuern, gibt es von IBM oder auch von ISVs (Independent Software Vendors) so genannte Jobablauf-Steuerungen. Damit lassen sich beispielsweise Zeit- und Event-gesteuerte Abläufe definieren. So kann festgelegt werden, dass beispielsweise ein Batchjob erst gestartet wird, wenn ein anderer ordnungsgemäß zum Ende gekommen ist. ISVs in diesem Bereich sind unter anderem die Unternehmen Computer Associates, BMC oder BETA. Das IBM-Produkt für die Funktionalität der Jobablauf-Steuerung hieß früher OPC (*Operation Planning & Control*) und wurde inzwischen umgetauft in TWS (*Tivoli Workload Scheduler*).

7.2 Timesharing mit TSO und ISPF

Time Sharing Option (TSO)

TSO steht für *Time Sharing Option*. Wie der Name schon sagt, war es ursprünglich eine Option, um in dem klassischen Batchsystem Timesharing betreiben zu können. Inzwischen ist es keine Option mehr, sondern unverzichtbarer und integraler Bestandteil des Betriebssystems.

MVS ermöglicht die parallele Unterstützung vieler Benutzer. Da ein Rechner jedoch immer nur einen Benutzer gleichzeitig bedienen kann, wurde ein Verfahren entwickelt, das die zur Verfügung stehende Rechenzeit zwischen den Benutzern aufteilt. Dieses Verfahren wird als Time Sharing bezeichnet. Der Rechner wird in Form von Zeitscheiben von zahlreichen Benutzern gleichzeitig benutzt, wobei es jedem einzelnen so scheint, als stehe ihm das System allein zur Verfügung.

Die Durchführung eines interaktiven Dialogs zwischen Benutzer und System wird über Kommandos vorgenommen, die TSO-Kommandos.

Beispiele für TSO-Kommandos:

LISTC
: listet Katalogeinträge auf

LOGOFF
: Abmelden von TSO

OMVS
: Aufruf von UNIX System Services

SEND
: Senden einer Nachricht an die Konsole oder an einen anderen Benutzer

Die Eingabe und das Versenden der Kommandos geschieht zeilenweise, im Gegensatz zu einer UNIX/Linux-Telnet Session, wo zeichenweise gearbeitet wird.

Das TSO hat die wichtige Eigenschaft, dass für jeden Benutzer ein eigener Adressraum und damit ein separater virtueller Speicher aufgebaut wird. In der privaten Region des Adressraums wird das so genannte *Terminal Monitor Program* (TMP) gestartet, das den READY-Prompt ausgibt. Das TMP entspricht einer Shell in UNIX/Linux-Umgebungen.

Damit man mit TSO arbeiten kann, muss man sich zunächst anmelden. Hierzu wird ein Terminal benötigt, das die 3270-Schnittstelle kennt. Da es in der Praxis keine „dummen" Terminals mehr gibt, sondern intelligente Arbeitsstationen verwendet werden, wird die 3270-Schnittstelle nachgebildet (emuliert). Die Datenübertragung zwischen einem Terminal und dem TSO entspricht einer Telnet-Session in UNIX/Linux-Umgebungen. Damit das 3270-Protokoll über eine TCP/IP-Kommunikationsschnittstelle funktioniert, wird das Protokoll TN3270 verwendet. Im Gegensatz zu einer Telnet-Session in Verbindung mit UNIX/Linux steht ein Telnet-Client nicht als Standard-Schnittstelle im Betriebssystem eines Clients zur Verfügung, sondern muss erst besorgt und installiert werden. Entsprechende Terminal-Emulatoren werden von diversen Unternehmen angeboten. Eine Suche in einer Suchmaschine im Internet liefert entsprechende Links.

Aufbau eines TSO-Adressraums

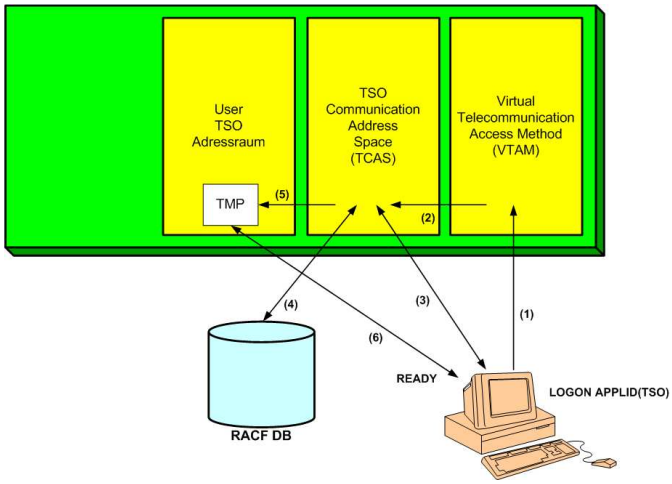


Abbildung 7.1: Ablauf des TSO-Prozesses

(1) Ein Benutzer meldet sich von einem Terminal aus am TSO an. Der Request geht an das VTAM.

(2) Das VTAM gibt den Request an den TSO Communication Address Space (TCAS) weiter.

(3) Der TCAS erfragt vom Benutzer die Anmeldedaten (UserID, Passwort, LOGON-Prozedur etc.).

(4) Für die Security-Überprüfung wird die RACF-Datenbank konsultiert.

(5) Bei erfolgreichem Security Check veranlasst der TCAS den Aufbau eines neuen Adressraums und startet darin das Terminal Monitor Program (IKJEFT01).

(6) Das TMP sendet eine READY-Meldung (Eingabeprompt) an den Benutzer.

Im TSO Logon-Bildschirm (Abbildung 7.2) werden die entsprechenden Parameter eingetragen, beispielsweise die Benutzerkennung und das Passwort. Die weiteren Parameter wie die Größe der Region (SIZE) oder die Abrechnungsinformationen (ACCT NMBR) müssen nicht jedes Mal neu eingegeben werden; die Eingaben bleiben zwischen den Sessions erhalten. Wenn keine Eingaben gemacht werden, gibt es für die meisten Parameter Standardwerte, die hergenommen werden.

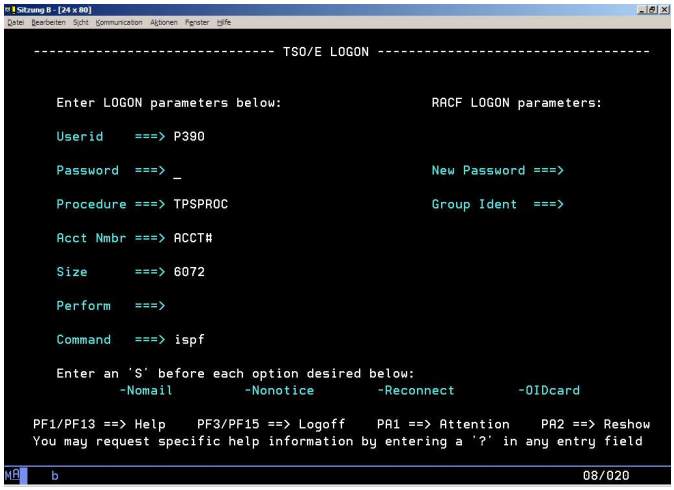


Abbildung 7.2: TSO/E Einstiegsmaske

Im Feld COMMAND kann ein TSO-Kommando oder eine REXX-Prozedur angegeben werden, die automatisch nach dem Aufruf des TMP ausgeführt wird. Dies entspricht dem Skript profile bzw. .profile in UNIX/Linux-Umgebungen.

Die LOGON-Prozedur:

```
//LOGONSTD    EXEC  PGM=IKJEFT01
//SYSPRINT   DD    TERM=TS
//SYSPROC    DD    DISP=SHR,DSN=SYS1.ISRCLIB
//ISPPLIB    DD    DISP=SHR,DSN=SYS1.ISRPLIB
//ISPMLIB    DD    DISP=SHR,DSN=SYS1.ISRMLIB
//ISPSLIB    DD    DISP=SHR,DSN=SYS1.ISPSLIB
//ISPTLIB    DD    DISP=SHR,DSN=SYS1.ISPTLIB
```

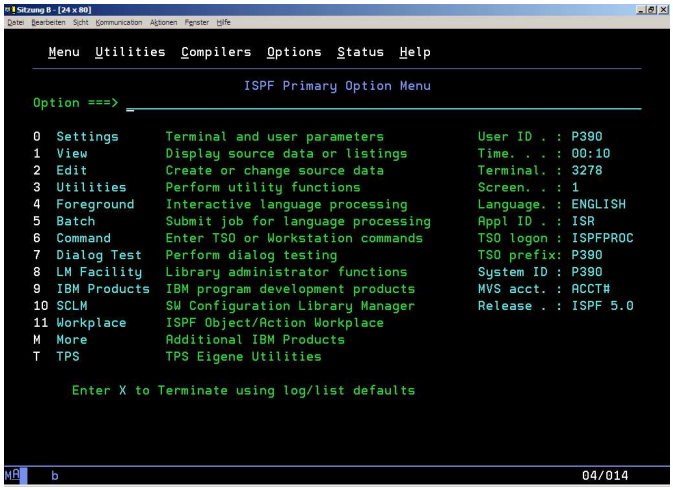
Über die LOGON-Prozedur werden dem Terminal-Monitor-Programm Bibliotheken zugewiesen, die für die Arbeit mit TSO benötigt werden. Ein typisches Beispiel dafür sind Bibliotheken, welche ISPF (*Interactive System Productivity Facility*) benötigt. Somit kann ISPF aufgerufen werden, ohne dass die Bibliotheken von Hand zugewiesen werden müssen.

7.3 Interactive System Productivity Facility (ISPF)

ISPF stellt Funktionalitäten zur Verfügung, die einen einfachen und effizienten Umgang mit dem System ermöglichen. Unter anderem stehen dahinter ein seitenorientierter Editor und zahlreiche Utilities.

Für die Arbeit benötigt ISPF das TSO (Time Sharing Option) als Trägersystem. ISPF besteht zum einen aus der *Programmer Development Facility* (PDF) und dem *Dialog Manager* (DM). Wenn jemand sagt, er arbeite mit ISPF, dann meint er in der Regel ISPF/PDF als menügesteuertes Produktivitäts-Werkzeug mit einem Editor und zahlreichen weiteren Hilfsmitteln. Der Dialog Manager gibt einem Entwickler Werkzeuge an die Hand, ein ISPF-Menüsystem um eigene Menüpunkte und Submenüs zu erweitern, und stellt APIs für Programmiersprachen zur Verfügung. Das Startmenü kann beispielsweise so aussehen:

Abbildung 7.3: ISPF Primary Option Menu



Da das Menüsystem installationsspezifisch angepasst werden kann, sieht es nicht überall identisch aus. Allerdings sind die Menüpunkte 1 bis 7 standardisiert und funktionieren in allen MVS-Umgebungen gleich. Dies hat den großen Vorteil, dass jemand, der mit ISPF umgehen kann, in allen MVS-Umgebungen sofort zurechtkommt. Eine Diskussion über den besten Editor ist ohnehin müßig; der beste Editor ist immer der, den man kennt und an den man sich gewöhnt hat. Wir kommen in Kapitel 12 noch einmal darauf zurück, wenn gezeigt wird, dass man im MVS unter UNIX System Services heute auch mit dem vi-Editor arbeiten kann, der in UNIX/Linux-Umgebungen sehr verbreitet ist.

7.4 Programmierung in einer TSO-Umgebung

In interaktiven Umgebungen spielen Skriptsprachen eine große Rolle, um Abläufe zu automatisieren und Kommandofolgen zu erzeugen und abzurufen. Mit dem gleichen Ziel wie die Skriptsprachen unter UNIX wurde auch für TSO eine Skriptsprache ins Leben gerufen, die CLIST (*Command List*) heißt und ursprünglich das

Ziel hatte, mehrere TSO-Kommandos in einem Dataset abzulegen. Beim Aufruf des Dataset werden die Kommandos nacheinander ausgeführt. In der Weiterentwicklung von CLIST wurden dann wie bei den UNIX-Skriptsprachen auch Programmierkonstrukte (Verzweigungen, bedingte Ausführungen etc.) eingefügt, so dass aus CLIST eine Programmiersprache geworden ist.

Ursprünglich in der VM-Umgebung entstanden, steht außerdem die Skriptsprache REXX (***Restructured Extended Executer***) zur Verfügung, die moderner, flexibler, umfangreicher und dennoch einfacher zu programmieren ist. REXX steht inzwischen auf allen IBM-Plattformen und zahlreichen weiteren Plattformen zur Verfügung und wurde auch ins MVS integriert. REXX kann alles, was mit CLIST gemacht werden kann – und einiges mehr, so dass REXX die Sprache CLIST voll ersetzen kann und soll. Als Fazit deshalb hier die Empfehlung: Neue Programme und Ablaufsteuerungen in Zusammenhang mit TSO, ISPF und dem Dialog Manager sollten nicht mehr mit CLIST, sondern ausschließlich mit REXX programmiert werden.

Eigenschaften von REXX

REXX zeichnet sich durch einige interessante Eigenschaften aus:

- REXX-Programme sind leicht zu lesen und zu schreiben. Die meisten Befehle sind einfach Worte aus dem englischen Sprachgebrauch wie beispielsweise SAY, PULL, IF-THEN-ELSE, DO, END etc.
- Beim Schreiben von REXX-Programmen gibt es wenig formale Zwänge: Groß- und Kleinschreibung werden nicht unterschieden, wo in einer Zeile eine Anweisung beginnt, spielt keine Rolle. Somit können die Programme durch Einrücken strukturiert werden. Eine Instruktion kann sich über mehrere Zeilen erstrecken. Ob REXX-Programme in Dateien mit fester oder variabler Länge abgespeichert werden, spielt für den Interpreter keine Rolle.
- Variablen müssen nicht vordefiniert werden. Eine nicht belegte Variable in einem REXX-Programm zeigt immer auf ihren zur Großschreibung konvertierten Eigennamen.

Kapitel 8

Mainframe Security

8.1 Die Umgebung

Der Mainframe stellt zweifellos eine starke und sichere Hardware-Plattform zur Verfügung. Die Speicherbereiche der einzelnen Anwendungen sind vor allem in den klassischen Mainframe-Betriebssystemen z/OS und z/VM konsequent voneinander abgeschottet. Die Hardware unterstützt in Kombination mit der Systemsoftware außerdem unterschiedliche Zustände eines Programms. Nur im so genannten ***Supervisor Mode*** können privilegierte Befehle ausgeführt werden. Der Normalzustand wird als ***Problem Program Mode*** bezeichnet, der durch das auf Eins gesetzte Bit 15 im Program Status Word (PSW) gekennzeichnet wird. Ist das Bit 15 auf Null gesetzt, befindet man sich im Supervisor Mode. Vom Problem Program Mode in den Supervisor Mode kommt man nur über einen Interrupt. Um vom nichtprivilegierten in den privilegierten Zustand zu kommen, muss deshalb ein Supervisor Call (SVC) aufgerufen werden, der einen Interrupt auslöst. Über den Interrupt wird das PSW ausgetauscht und das Bit 15 auf Null gesetzt. Es wird dann im privile-

gierten Zustand gearbeitet. Am Ende der SVC-Verarbeitung wird das ursprüngliche PSW wieder zurückgeladen, und der nicht-privilegierte Zustand ist wiederhergestellt.

Durch neue Technologien und vor allem durch die Vernetzung sind im Laufe der Zeit zusätzliche Herausforderungen entstanden, wenn es um Sicherheit geht. Neue eBusiness-Anwendungen sind oft so konzipiert, dass sie über mehrere Plattformen hinweg eingesetzt werden können. Aus diesem Grund muss die Security-Infrastruktur offene Sicherheitsstandards unterstützen, die plattformübergreifend funktionieren.

Die steigende Anzahl der Benutzer und vor allem auch die Tatsache, dass nicht nur firmenintern auf die Daten zugegriffen wird, sorgen dafür, dass die Sicherheit und die entsprechend notwendigen Maßnahmen einen neuen Stellenwert erhalten.

Ein weiterer wichtiger Punkt sind Verschlüsselungstechniken. Auf den IBM-Großrechnern wird seit 1991 Hardware-unterstützte Verschlüsselung angeboten. Seit 1997 ist diese Hardware-Unterstützung als Standard in die S/390-Prozessoren und danach auch in die zSeries-Rechner eingebaut. Damit wird der Tatsache Rechnung getragen, dass Datenverschlüsselung die wichtigste Grundlage für sichere Kommunikation in Zusammenhang mit dem Internet bildet. Damit die notwendige Performanz gewährleistet werden kann, müssen die Verschlüsselungstechniken von der Hardware unterstützt werden.

Im Zuge der Ausrichtung der verteilten Anwendungen an offenen Standards verschmilzt das z/OS die bewährte Sicherheitsarchitektur des Großrechners mit den besten und meistverbreiteten Sicherheitstechnologien. So wird beispielsweise ein LDAP-Server zur Verfügung gestellt, mit dem die Security-Verwaltung einfach und unkompliziert realisiert werden kann.

Weiterhin können sich User herkömmlich über UserID und Passwort authentifizieren und verifizieren. Es werden jedoch auch offene Standards wie *Kerberos* oder X.500-Zertifikate unterstützt.

Im Mittelpunkt der Security-Architektur des Mainframe steht die *Resource Access Facility* (RACF). RACF beinhaltet verschiedene Security-Features. Die wichtigsten sind zweifellos Identifikation und Verifikation sowie Zugriffskontrolle auf die unterschiedlichsten Ressourcen. Wir gehen darauf noch später in diesem Kapitel ein.

Bezüglich Logical-Partitioning-Konfigurationen (LPAR) wurde S/390 bzw. zSeries bereits 1995 von der *Information Technology Security Evaluation Criteria* (ITSEC) der Europäischen Kommission „E4-zertifiziert". Dies betrifft die Isolation der Workloads in unterschiedlichen Partitions und bescheinigt, dass diese Partitions aus Security-Sicht so gegeneinander abgeschottet sind, als ob sie unterschiedliche physische Server wären. Dies garantiert, dass eine Web-Anwendung in einer logischen Partition und eine Produktionsumgebung in einer anderen Partition sauber voneinander getrennt und isoliert betrieben werden können und dennoch gemeinsame Hardware-Ressourcen auf einem gemeinsamen physischen Server nutzen.

8.2 Verschlüsselung und Internet

Verschlüsselung ist zweifelsohne die wichtigste Technik für eine sichere Kommunikation über das Internet. Hinter der Verschlüsselungstechnik steht die Idee, dass Informationen mit Hilfe eines Schlüssels (Key) so umgeformt (*enciphered*) werden, dass sie auf dem Übertragungsweg nicht mehr entziffert werden können. Ein komplementärer Algorithmus ermöglicht auf der Empfängerseite die Rückformung (*deciphering*) in die ursprüngliche Information.

Secure Socket Layer (SSL)

Die wichtigste Technik in diesem Zusammenhang ist SSL (*Secure Socket Layer*). SSL ist ein so genanntes hybrides Verfahren, d. h. es arbeitet mit einer Kombination aus dem Private- und dem Public-Key-Verfahren. Der Hintergrund ist der, dass eine synchrone Verschlüsselung mit dem Private-Key-Verfahren, bei dem auf beiden Seiten ein identischer Schlüssel für die Ver- und Entschlüsselung benutzt wird, viel weniger aufwändig ist als eine Verschlüsselung nach dem Public-Key-Verfahren, bei dem auf beiden Seiten jeweils mit einem Schlüsselpaar gearbeitet werden muss. Ohne hier in Details zu gehen, kann man davon ausgehen, dass das Public-Key-Verfahren um den Faktor 100 mehr Rechenleistung benötigt.

Deshalb wird bei SSL über ein Handshaking-Verfahren nach dem Public-Key-Verfahren ein Schlüssel ausgetauscht, der als *Session Key* dann auf beiden Seiten genutzt werden kann. Die große Menge an Nutzdaten wird dann synchron mit dem Private-Key-Verfahren übertragen. Gerade für Sessions, über die große Datenmengen ausgetauscht werden, wie beispielsweise FTP, Telnet oder TN3270, spielt das eine sehr große Rolle.

Hardware-Unterstützung der Verschlüsselungstechniken

Die zSeries-Hardware unterstützt diverse Verschlüsselungstechniken, vor allem SSL. Es gibt zwei unterschiedliche Techniken, die von z/OS unterstützt werden: der *CMOS Cryptographic Coprocessor* und der neuere *Peripheral Component Interface Cryptographic Coprocessor* (PCICC). Als Richtwert kann gesagt werden, dass die Ver-/Entschlüsselung per Hardware etwa 17- bis 20-mal schneller ist als per Software.

Der CMOS Cryptographic Coprocessor ist eine reine Hardware-Implementation, ohne dass irgendwelcher Microcode ausgeführt wird. Er hat den FIPS (*Federal Information Processing Standard*) Level 140-1 Level 4 im Security-Rating des *National Institute of Standards and Technology* (NIST) der US-Regierung erlangt.

Der CMOS Cryptographic Coprozessor unterstützt sehr schnelle DES–Verschlüsselungen (*Data Encryption Standard*), Method Authentication Code Checking (MA-

Cing), Key Management und PIN-Funktionen. Außerdem wird RSA (Rivest, Shamir, Adleman) unterstützt, ein Algorithmus, der vor allem in Verbindung mit SSL eine bedeutende Rolle spielt. Vor allem von Finanzdienstleistern werden diese Funktionalitäten extensiv verwendet.

Somit eignet sich der CMOS Cryptographic Coprocessor hervorragend für die DES-Verschlüsselung. Allerdings ist er etwas unflexibel, wenn es um die Unterstützung neuerer Funktionen und um Änderungen geht. In Verbindung mit RSA ist die Performanz zwar nicht schlecht, hält jedoch nicht immer Schritt mit den exponentiell ansteigenden Anforderungen im Zusammenhang mit SSL-Verschlüsselungen.

Die Antwort auf diese Herausforderungen ist der PCI Cryptographic Coprocessor, der in die 4758-2 PCI Cryptographic-Coprozessorkarte integriert wird. Auch die 4758-2 PCICC-Karte entspricht dem eben erwähnten Security-Rating. Die Karte arbeitet mit einem Betriebssystem und einer C-Programmierumgebung für die Implementierung neuer Funktionen. Die auf der Karte implementierten Funktionen können über die *Integrated Cryptographic Services Facility* (ICSF) aufgerufen werden.

Mit PCICC wird eine exzellente RSA-Performance erreicht. Eine Karte ermöglicht ca. 135 RSA Handshakes pro Sekunde, während ein CMOS Cryptographic Coprocessor ca. 75 Handshakes pro Sekunde erreicht. Bei einem Einsatz von 16 PCICC-Karten und zwei CMOS Cryptographic Coprocessors können somit bis zu 2.000 SSL Handshakes pro Sekunde verarbeitet werden. In der neuesten z990-Reihe sind es gar bis zu 9.000 SSL Handshakes pro Sekunde.

Sowohl der CMOS Cryptographic Processor als auch der PCICC implementieren die *Common Cryptographic Architecture* (CCA) von IBM. Diese definiert einen Satz von kryptographischen Funktionen und eine Methode für die sichere Trennung von kryptographischen Schlüsseln, um sicherzustellen, dass Schlüssel nur für bestimmte spezifische Funktionen verwendet werden können. Ein derartiges Design bildet die Grundlage einer vernünftigen kryptographischen Implementierung und steht im Einklang mit kryptographischen Standards wie beispielsweise ANSI X9.24.

Die CCA stellt außerdem ein API zur Verfügung, mit dem auf Funktionen mit Hilfe von High Level Language Function Calls zugegriffen werden kann. Es handelt sich dabei um ein plattformübergreifendes API, das z/OS, AIX, Windows NT, Windows 2000 und OS/2 unterstützt.

Unter z/OS werden zwei zusätzliche APIs unterstützt. Das erste ist das *BSAFE Hardware API* (BHAPI), das über das BSAFE Toolkit genutzt werden kann, welches von der RSA Security Inc. entwickelt wurde. Außerdem können die kryptographischen Funktionen der Hardware über die *Open Cryptographic Services Facility* unter z/OS genutzt werden. Hierbei handelt es sich um eine Implementierung der *Common Data Security Architecture* (CDSA) für Anwendungen, die unter der Umgebung von UNIX System Services laufen. Der Zugriff auf die Funktionen wird über Access Control verwaltet.

8.3 SecureWay Security Server

Komponenten

Der *SecureWay Security Server for z/OS* besteht aus folgenden Hauptkomponenten:

Resource Access Control Facility (RACF)
: Siehe dazu Seite 125 ff.

DCE Security Server
: Der *DCE Security Server* ermöglicht eine Benutzer- und Server-Authentifizierung für Client/Server-Umgebungen, wie sie über die Standards der Distributed Computing Environment (DCE) der Open Software Foundation (OSF) definiert ist. Insbesondere werden die Kerberos-Technologien unterstützt, die am Massachusetts Institute of Technology (MIT) entwickelt wurden. Durch die Integration mit RACF ist es möglich, dass RACF-authentifizierte Benutzer auf DCE-verwaltete Ressourcen und Server zugreifen können, ohne dass sie sich zusätzlich DCE-authentifzieren müssen. Umgekehrt können DCE-verwaltete Server einen DCE-authentifizierten Benutzer in eine RACF-Identität umformen, die es ihm ermöglicht, auf RACF-verwaltete Ressourcen zuzugreifen.

z/OS Firewall-Technologien
: Mit den z/OS-Firewall-Technologien werden Sicherheitstechniken auf der zSeries-Plattform ermöglicht, die die Notwendigkeit von vorgelagerten Firewalls auf anderen Plattformen reduzieren oder eliminieren. Es stehen beispielsweise die gängigen Möglichkeiten wie IP-Paketfilterung, IP Security in Form von Tunneling und VPN sowie die Network Address Translation (NAT) zur Verfügung.

LDAP Server
: LDAP ermöglicht den sicheren Zugriff von Anwendungen und Systemen in einem Netzwerk auf Verzeichnisinformationen, die auf dem Mainframe gespeichert sind. Insbesondere wird oft über LDAP auf RACF-Profile zugegriffen und damit entsprechende Autorisierung und Authentifizierung ermöglicht.

Open Cryptographic Enhanced Plug-ins (OCEP)
: Der SecureWay Security Server ermöglicht eine einfache und effiziente Verwaltung von Server-Zertifikaten zum Schutz der privaten Server-Schlüssel. Eine wichtige Schnittstelle für Anwendungen wird durch die *Open Cryptographic Enhanced Plug-ins* (OCEP) zur Verfügung gestellt. Die Services können durch zwei Service Provider Modules angesprochen werden, die auch als Plug-ins bezeichnet werden: eines für die Data Library Services und eines

für den Trust Policy Manager. Die Schnittstellen entsprechen den Spezifikationen der *Common Data Security Architecture* (CDSA).
Anwendungen können die Module und APIs nutzen, um digitale Zertifikate und private Schlüssel aus einer RACF-Datenbank abzurufen.

Was heißt Sicherheit?

Wenn eine sichere Umgebung aufgebaut werden soll, muss zunächst einmal die physische Sicherheit gewährleistet sein. Aus historischen Gründen war dies in der Vergangenheit aus der Sicht des Mainframe meist gegeben, da dieser sich allein aufgrund der Größe und der notwendigen Infrastruktur in einem Rechenzentrum befand. Das oft zitierte „Glashaus" konnte nur von wenigen autorisierten Personen betreten werden. Andere hatten in den Räumen, in denen die Rechner standen, nichts verloren.

Inzwischen hat sich das Bild gewandelt. Das z/OS-Betriebssystem lässt sich heute im Extremfall sogar auf einem Laptop betreiben, wenngleich dann die Hardware-spezifischen Features bezüglich Security natürlich nicht zur Verfügung stünden. Dennoch werden die klassischen Mainframes nach wie vor in abgeschotteten Rechenzentren betrieben. Nur autorisierte Personen haben Zugang, und die Kontrolle findet oftmals mit biometrischen Verfahren (Fingerabdruck, Augenanalyse etc.) statt. In dezentralen Umgebungen sieht es oft anders aus. Die Rechner wurden meist von Fachabteilungen angeschafft, die Anwendungen dafür wurden entwickelt, und irgendwann wurde der Rechner in Betrieb genommen. Zugangskontrollen, Stromausfallschutz etc. wurde oft vernachlässigt. Zugegeben: Das hat wenig mit den technischen Möglichkeiten zu tun, sondern bezieht sich eher auf organisatorische Aspekte, die sich natürlich auch in diesen Umgebungen lösen lassen und die in vielen Unternehmen auch tatsächlich gelöst wurden.

Ebenfalls zur physischen Sicherheit gehört ein Backup- und Katastrophenplan, der in Verbindung mit den klassischen Betriebssystemen und *Geographically Dispersed Parallel Sysplex* von der Hardware und Software perfekt unterstützt wird.

Der zweite wichtige Aspekt ist dann die Sicherheit des Systems. Hierbei muss sehr viel beachtet werden, um die Integrität und die Sicherheit des Gesamtsystems gewährleisten zu können. So dürfen im z/OS bestimmte Programme nur aus APF-autorisierten Bibliotheken aufgerufen werden. Ein so genanntes Syslog-Journal schreibt sämtliche wesentliche Aktivitäten mit. Über die System Management Facility werden Informationen dauerhaft abgespeichert. Die zentrale Komponente für die Unterstützung und Gewährleistung dieser Funktionen ist die *Resource Access Control Facility.*

8.4 Resource Access Control Facility (RACF)

Der Zugriff auf Daten gestaltet sich heute wesentlich einfacher als früher. Endbenutzer ohne Systemkenntnisse haben die Möglichkeit, über definierte Schnittstellen und Anwendungen auf Daten zuzugreifen. Die Anzahl der Endbenutzer steigt täglich, und es sind nicht mehr nur die eigenen Mitarbeiter eines Unternehmens, sondern auch Partner und Kunden, die auf die Unternehmensdaten zugreifen.

Die Daten sind Unternehmenswerte und für viele Unternehmen zum wichtigsten Produktionsfaktor geworden. Die Datenschutzgesetze legen fest, welche sicherheitsspezifischen Belange zur Absicherung der Daten erfüllt sein müssen.

Datenschutz bedeutet nicht nur, vertrauliches Material unzugänglich zu machen, sondern es muss auch die versehentliche Zerstörung oder Verfälschung von Daten soweit möglich verhindert werden.

Anforderungen

Daraus ergeben sich die Anforderungen, die ein modernes Sicherheitssystem erfüllen muss.

Identifikation von Benutzern

Eine der wichtigsten Regeln ist, dass jeder Benutzer seine eigene Benutzerkennung haben muss. Das fördert die Eigenverantwortlichkeit, da bei der Analyse von Fehlerursachen in der Regel verfolgt werden kann, wer an welcher Stelle des Systems eine Änderung vorgenommen hat, die zu dem entsprechenden Fehler geführt hat. Im z/OS gibt es keinerlei Rechtfertigung mehr dafür, dass eine Benutzerkennung von mehreren Personen gemeinsam genutzt wird. Einen von vielen Personen genutzten *Superuser*, der alles kann und alles darf und dessen Aktivitäten oft nicht einmal protokolliert werden, gibt es im z/OS nicht. Alle Berechtigungen können granular und abhängig von der vom Benutzer ausgeführten Rolle so eingerichtet werden, dass unter Einhaltung der sicherheitsrelevanten Regeln jeder Benutzer seinen Job erfüllen kann, ohne dass er dabei gehindert wird.

Verifizierung

Durch ein Passwort muss geprüft werden können, ob der Benutzer auch die Person ist, für die er sich ausgibt. In vielen Fällen, vor allem bei Finanzdienstleistern, wird oftmals in Verbindung mit Smartcards und entsprechenden Lesern an den Endgeräten der Nachweis eines X.509-Zertifikats verlangt. In einigen Fällen werden sogar die Endbenutzer biometrisch überprüft.

Zugriffskontrolle

Geschützte Ressourcen dürfen nur von berechtigten Personen benutzt werden. Dies betrifft sowohl Hardware- als auch Software-Ressourcen, in erster Linie natürlich die Daten.

Kontrolle und Protokollierung

Der Zugang und Abgang von Benutzern sowie vor allem die Zugriffsverletzungen müssen umfassend und lückenlos protokolliert werden.

Es gibt natürlich auch im z/OS spezielle Benutzer, die für die Administration der Security zuständig sind. Allerdings gibt es wesentliche Unterschiede beispielsweise zu UNIX/Linux-Umgebungen. Im Gegensatz zum Superuser in UNIX/Linux ist das jedoch nicht ein „Alleskönner", sondern seine Berechtigungen sind speziell auf die Administration von RACF ausgerichtet. Ein Benutzer kann beispielsweise seine eigenen Ressourcen gegen den Zugriff anderer Benutzer schützen. Der Zugriffsschutz gilt auch für den RACF-Administrator, der wie jeder andere Benutzer auch an einem unberechtigten Zugriff gehindert wird. Der RACF-Administrator kann zwar, weil er der Administrator ist, den Zugriffsschutz auf die Ressourcen ändern, diese Änderung wird jedoch bereits wieder über SMF protokolliert, und diese Protokollierung kann auch ein RACF-Administrator nicht ohne weiteres ausschalten.

Benutzerschnittstelle

Der Aufbau der Security muss eine saubere und klare Struktur haben und einfach in der Anwendung und Handhabung sein.

Performance

Die Überprüfung und die Zugriffe auf die RACF-Datenbank müssen schnell erfolgen und hohe Transaktionsraten unterstützen, damit die Security keine Behinderung darstellt. Dies wird unter z/OS durch zahlreiche Features unterstützt. Unter anderem können beispielsweise Hiperspaces (virtuelle Datenräume) eingerichtet werden, damit nicht bei jedem Zugriff physisch auf die RACF-Datenbank zugegriffen werden muss.

Funktionsüberblick

Ein Datensicherungssystem muss, um von den Benutzern akzeptiert zu werden, so konzipiert sein, dass es einfach und wirkungsvoll, aber ohne großen administrativen Aufwand eingesetzt und gesteuert werden kann. In einem Unternehmen, das einen Mainframe betreibt, ist der Einsatz eines Security-Systems eine absolute Notwendigkeit. Das muss jedoch nicht immer und in jedem Fall RACF sein. Es gibt hier einige Software-Anbieter am Markt, die äquivalente Produkte anbieten. In der Praxis findet man beispielsweise neben RACF auch ACF2 (*Access Control Facility 2*) oder *Top Secret* vor. Beides sind Produkte von Computer Associates. Die Funktionalität dieser Produkte ist der von RACF sehr ähnlich.

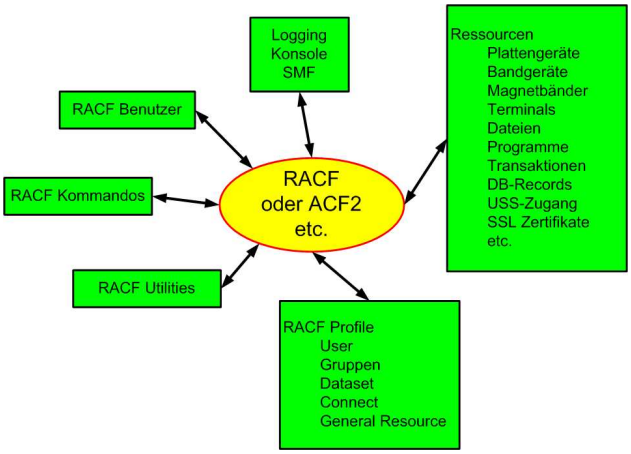


Abbildung 8.1: RACF im Überblick

Mit RACF können so gut wie alle Ressourcen in einem MVS verwaltet werden. Ob tatsächlich alles, was technisch möglich ist, in Verbindung mit RACF auch genutzt wird, muss unternehmensspezifisch entschieden werden. So ist es beispielsweise möglich, Magnetbänder mit RACF zu schützen. Diese Funktionalität wird jedoch meist von einem Bandverwaltungssystem übernommen. Des weiteren können mit RACF physische Terminals geschützt werden, was in den meisten Fällen nicht gewünscht wird, da die Benutzer die Möglichkeit haben sollen, sich mit ihrer jeweiligen Kennung und einem Verifizierungs-Mechanismus von jedem Gerät in einem Unternehmen am System anzumelden.

Ein z/OS-Systembenutzer muss grundsätzlich auch ein RACF-Benutzer sein, d. h. er muss im RACF mit seiner Kennung und entsprechenden Attributen und Parametern eingetragen sein. Dies betrifft auch die UNIX-spezifische Schnittstelle in Verbindung mit den UNIX System Services.

Die Benutzerschnittstelle wird mit Hilfe von RACF-Kommandos umgesetzt. Die Nutzung dieser RACF-Kommandos kann mit einer ISPF-Anwendung erleichtert werden, die eine menügeführte Schnittstelle zu den Kommandos zur Verfügung stellt. Außerdem gibt es für die Administration und vor allem auch für Auditoren zahlreiche Utilities für die Verwaltung und Überwachung der Security-Umgebung.

Die sicherheitsrelevanten Daten werden in einer RACF-Datenbank in Form von Profilen verwaltet. Man unterscheidet User-, Gruppen-, Dataset- und General-Resource-Profile.

Im Unterschied zu UNIX/Linux-Umgebungen sind die sicherheitsrelevanten Daten nicht direkt mit der Ressource verknüpft. Nehmen wir als Beispiel den Zugriffsschutz für Files bzw. Dateien. In UNIX/Linux-Umgebungen wird der Zugriffsschutz in Form der ***Permission Bits*** in einem ***File Security Packet*** abgelegt, das direkt

mit dem jeweiligen File verknüpft ist. Das hat zur Folge, dass mit dem Löschen des Files auch der entsprechende Schutz gelöscht wird. Wird das gleiche File wieder neu angelegt, so ist nicht gewährleistet, dass der Schutz der gleiche ist wie vor dem Löschen und Wiederanlegen. Beim Anlegen eines Files wird ein Default-Schutz eingerichtet, der über eine *umask* bestimmt wird.

Im Unterschied dazu werden im MVS die Schutzdaten in Form der erwähnten Profile und die zu schützenden Ressourcen getrennt. Dies hat einige Vorteile. So kann in Zusammenhang mit RACF mit so genannten „generischen Profilen" gearbeitet werden. Dies bedeutet, dass mit einem einzigen Profil eine komplette Umgebung eines Benutzers gesichert werden kann.

Beispiel für ein generisches Profil:

```
UHOX.*  UACC(NONE)
```

Mit diesem Profil werden über den High Level Qualifier (HLQ) UHOX die gesamten Ressourcen des Benutzers UHOX geschützt. Mit dem Parameter UACC(NONE) wird festgelegt, dass der Universalzugriff auf NONE gesetzt ist, d. h., dass per Default niemand auf die Ressourcen zugreifen darf. Wenn nun Ausnahmen zugelassen werden sollen, kann dies über spezifische PERMIT-Kommandos geschehen oder auch über das Einrichten zusätzlicher generischer Profile. So kann beispielsweise mit dem Profil

```
UHOX.TEST.*  UACC(READ)
```

festgelegt werden, dass auf alle Ressourcen, die mit UHOX.TEST beginnen, mit dem Universalrecht READ zugegriffen werden kann.

RACF-Struktur

Es gibt im RACF eine definierte Benutzerstruktur, die hierarchisch aufgebaut ist. Jeder ist für seine eigene Umgebung selbst verantwortlich. Zusätzlich können Gruppenverantwortliche eingesetzt werden, die auf Gruppenebene bestimmte Berechtigungen haben.

Man kann auch Owner von anderen Daten sein, wenn systemseitig für ein bestimmtes Projekt ein Qualifier definiert und einem Benutzer zugeordnet wird.

Jeder Benutzer ist mindestens einer RACF-Gruppe (Default-Gruppe) zugeordnet. Die Gruppen sind meist entweder nach organisatorischen oder projektspezifischen Gesichtspunkten geordnet, so dass beispielsweise die Mitarbeiter einer Abteilung der gleichen RACF-Gruppe angehören. Ein Benutzer kann allerdings auch problemlos an andere Gruppen „angehängt" werden und somit mehreren Gruppen zugehören. Dies geschieht mit Hilfe von Connect-Profilen.

Die RACF-Gruppenstruktur hat den Vorteil, dass Berechtigungen nicht nur auf Benutzerebene, sondern auch auf Gruppenebene vergeben werden können. Wenn man beispielsweise allen Mitarbeitern eines Projekts den Zugriff auf bestimmte Daten gestatten will, muss man nicht die einzelnen Benutzer kennen, sondern kann allen den Zugriff gestatten, die der Projektgruppe zugeordnet sind.

Für jeden Benutzer im System wird ein User-Profil in der RACF-Datenbank geführt, in dem auch seine Default-Gruppe eingetragen ist. Ein Benutzerprofil ist in mehrere Segmente eingeteilt. Die Gruppen werden in Gruppenprofilen verwaltet. Verbindungen von Benutzern zu Gruppen werden, abgesehen von der Default-Gruppe, über Connect-Profile verwaltet.

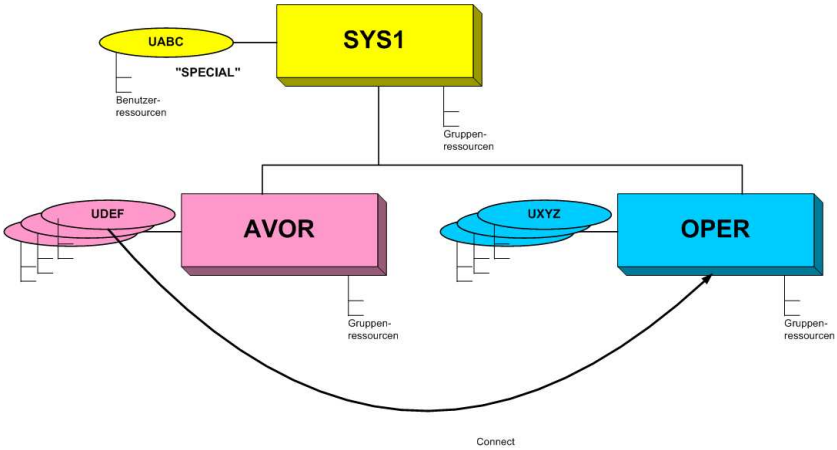


Abbildung 8.2: Hierarchie im RACF

Es werden Benutzerressourcen und Gruppenressourcen unterschieden.

Für die Benutzerressourcen ist in einem vorgegebenen Rahmen jeder Benutzer selbst verantwortlich. Das heißt, er kann bestimmen und selbst auch im RACF definieren, wer in welcher Form auf seine Ressourcen zugreifen kann.

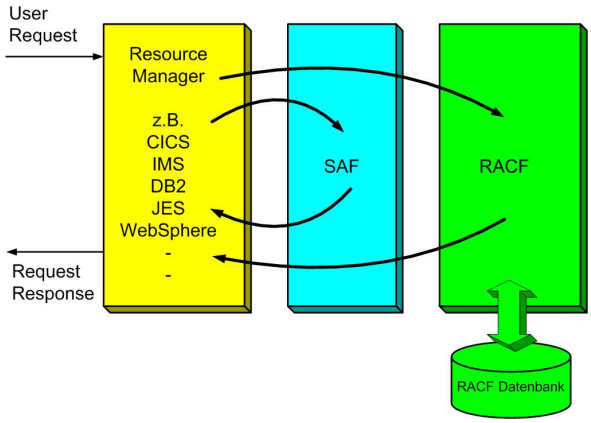
Für die Verwaltung der Gruppenressourcen ist der Owner der Gruppe verantwortlich. Das ist der Benutzer, der die Gruppe eingerichtet hat oder dem die Ownership von einem Administrator übertragen worden ist.

Die Administration kann zentral oder dezentral erfolgen. Beim zentralen Ansatz verwaltet ein Administrator oder eine zentrale Gruppe von Administratoren alle Berechtigungen. Der zentrale Administrator hat in seinem Profil das Attribut SPECIAL. Beim dezentralen Ansatz wird die Administration in die Gruppen delegiert. Es gibt dann auf Gruppenebene typischerweise jeweils einen Benutzer, der das Attribut SPECIAL auf Gruppenebene zugeordnet bekommt. Er kann dann die RACF-Administration für seine Gruppe und für untergeordnete Gruppen vornehmen.

System Authorization Facility (SAF)

Die *System Authorization Facility* (SAF) ist Teil des Betriebssystems und steht auch dann zur Verfügung, wenn die Security nicht mit RACF durchgeführt wird. Es gibt APIs und Schnittstellen, mit denen Systemprogramme und Resource Manager auf SAF zugreifen können. Je nach Umgebung kann SAF entsprechende Requests selbst beantworten oder greift auf RACF bzw. ein Alternativprodukt wie ACF2 oder Top Secret zurück. Die Schnittstelle zu den Benutzern ist immer ein Resource Manager.

Abbildung 8.3: Die System Authorization Facility (SAF)



Kommandos

Mit RACF-Kommandos können Profile angelegt, verändert und angezeigt werden.

Ein einfaches Kommando ist beispielsweise LISTUSER oder LU, mit dem der Zustand des eigenen Benutzerprofils angezeigt werden kann. Für die Anzeige fremder Profile benötigt man eine spezielle Berechtigung.

Weitere wichtige Kommandos sind:

ADDSD
: Definition des Schutzes einer Datei

ALTDSD
: Ändern des Dateischutzes

DELDSD
: Löschen eines Profils

PERMIT
: Zugriffsschutz für einen spezifischen User

Für die Administration wichtige Kommandos sind:

ADDUSER
: Hinzufügen eines Benutzerprofils

ALTUSER
: Ändern eines Benutzerprofils

CONNECT
: Verbinden eines Benutzers mit einer Gruppe

DELUSER
: Entfernen eines Benutzerprofils

REMOVE
: Lösen einer Verbindung zu einer Gruppe

PASSWORD
: Zurücksetzen eines Passworts

Anmerkung: Auch ein Administrator kennt die Passwörter der Benutzer nicht. Beim Zurücksetzen des Passworts definiert der Administrator lediglich ein Einstiegspasswort. Der Benutzer kann sich dann mit diesem Passwort anmelden. Dieses Passwort ist als „expired", d. h. als abgelaufen gekennzeichnet, und der Benutzer wird aufgefordert, ein neues, individuelles Passwort einzugeben.

Schutzkategorien

Für die Zugriffe auf Ressourcen werden Schutzkategorien definiert. Je nach Ressourcenart haben diese unterschiedliche Bedeutung. Für Dateien als meistgeschützte Ressourcen sind folgende Zuordnungen möglich:

NONE
: Kein Zugriff erlaubt

EXECUTE
: Erlaubnis zur Ausführung (nicht lesen!)

READ
: Leseerlaubnis

UPDATE
: Schreiben/Verändern erlaubt

CONTROL
: speziell für VSAM-Datenbestände

ALTER
: komplette Kontrolle incl. Schutzänderung

User-Attribute

User-Attribute weisen einem Benutzer bestimmte Security-spezifische Fähigkeiten zu, die sich entweder ständig auswirken oder wenn der Benutzer zu einer spezifischen Gruppe connected ist. Eine Fähigkeit, die ständig wirkt, wird auf dem System-Level definiert, ein Attribut, das sich nur auf eine spezifische Gruppe auswirkt, wird auf Gruppen-Level definiert.

Attribute können mit dem ADDUSER- oder dem ALTUSER-Kommando definiert werden. Es gibt folgende Attribute:

SPECIAL
: Ein Benutzer mit dem Attribut SPECIAL auf Systemebene kann alle RACF-Kommandos ausführen und hat volle Kontrolle über die Datenbank mit den RACF-Profilen. Das SPECIAL-Attribut kann auch auf Gruppenebene vergeben werden und ermöglicht damit die Kontrolle über alle Profile, die dem Geltungsbereich der jeweiligen Gruppe entsprechen.

AUDITOR
: Das AUDITOR-Attribut wird für Benutzer vergeben, die Überwachungs- und Audit-Funktionen ausüben. Das SPECIAL- und AUDITOR-Attribut sollte nicht ein und demselben Benutzer zugewiesen werden.

OPERATIONS
: Dieses Attribut wird für Backup/Recovery-Funktionen benötigt. Ein Benutzer mit dem Attribut OPERATIONS hat Zugriff auf alle RACF-geschützten Ressourcen in den Klassen DATASET, DASDVOL, GDASDVOL, PSFMPL, TAPEVOL, VMBATCH, VMMDISK, VMNODE und VMRDR.

CLAUTH
: Das CLAUTH-Attribut wird für RACF-Klassen definiert und ermöglicht es einem Benutzer, Profile in der entsprechenden Klasse zu definieren, zu ändern und zu löschen.

REVOKE
: Dieses Attribut verhindert das Anmelden und verweigert damit den Zugriff zum System. Dies macht bei temporärer Abwesenheit Sinn oder auch dann, wenn ein Benutzer eigentlich gelöscht werden sollte, weil er aus dem Unternehmen ausgeschieden ist, es jedoch noch RACF-geschützte Ressourcen

unter seiner Benutzerkennung gibt, die erst umorganisiert werden müssen. Ein weiterer Fall ist der automatische Revoke, wenn eine Benutzerkennung einen definierten Zeitraum lang nicht mehr verwendet worden ist.

GRPACC
: Wenn einem Benutzer das GRPACC (für Group Access) zugewiesen ist, können Profile für Gruppendateien, die der Benutzer definiert, automatisch auch von anderen Gruppenmitgliedern zugegriffen werden.

ADSP
: Das ADSP (*Automatic Data Set Protection*) sorgt dafür, dass automatisch ein diskretes Profil erstellt wird, wenn eine neue DASD- oder Tape-Datei erstellt wird. Das Attribut sollte heute nicht mehr vergeben werden, da man aus Effizienz- und Administrationsgründen sinvollerweise mit generischen Profilen arbeitet.

Segmente

Wenn ein Benutzer im RACF eingetragen wird, wird für diesen Benutzer ein Profil in der RACF-Datenbank erzeugt. Dieses Profil besteht aus dem eigentlichen RACF-Segment und optional weiteren Segmenten, je nachdem, welche Subsysteme und Komponenten dem Benutzer zur Verfügung stehen sollen. Ein Benutzer benötigt beispielsweise ein OMVS-Segment, wenn er Zugriff auf UNIX System Services haben soll. Darin steht dann seine UNIX UID, sein Home-Directory und die Shell, die für ihn gestartet wird.

Weitere Segmente sind zum Beispiel CICS, DCE, NETVIEW und TSO.

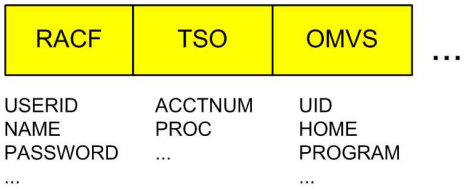


Abbildung 8.4: RACF-Segmente

Im RACF-Segment selbst sind die Basisdaten untergebracht wie USERID, NAME, OWNER, DFLTGRP, AUTHORITY und PASSWORD. Im TSO-Segment stehen beispielsweise die Account-Nummer und die LOGON-Prozedur.

Die RACF-Administration benutzt oft Batchjobs für bestimmte Arbeiten, da dadurch Fehler reduziert werden können und auch bessere Kontrollmöglichkeiten bestehen.

Beispiel:

```
//UGEWS  JOB 7326,'W.GREIS',MSGCLASS=X
//TSOBAT01 EXEC PGM=IKJEFT01
```

```
//SYSTSPRT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSTSIN   DD *
LD DA(??MARTIN.*??????AUTHUSER
LU MARTIN
/*
```

8.5 Security und LDAP

LDAP steht für *Lightweight Directory Access Protocol* und ist ein Standard, mit dem auf Namens- und Verzeichnisdienste zugegriffen werden kann. Ein Anwendungsgebiet von LDAP ist deshalb unter anderem auch die Authentifizierung von Benutzern.

Was die Berechtigungen von Benutzern angeht, sind diese noch nicht in der LDAP Protokoll-Spezifikation integriert. Dieser Aspekt wird implementations- oder herstellerspezifisch meist über *Access Control Lists* (ACLs) realisiert. Für künftige Versionen gibt es jedoch bereits einen Internet Draft.

Authentifizierungsstufen

Bezüglich Authentifizierung gibt es drei unterschiedliche Stufen:

- keine Authentifizierung
- Basis-Authentifizierung
- Simple Authentication and Security Layer (SASL)

Da es in der LDAP-Version 2 noch keine Verschlüsselungstechnik gab, wurden von einigen Herstellern eigene SSL Calls als LDAP APIs zur Verfügung gestellt.

Der Nachteil dabei ist, dass diese API-Aufrufe zwischen unterschiedlichen Herstellerimplementationen nicht kompatibel sind. Für LDAP 3 gibt es deshalb einen Vorschlag (Proposal), SSL bzw. TLS (*Transport Security Layer*) durch Erweiterungen zu unterstützen.

Keine Authentifizierung

Dies ist der einfachste Fall. Die Methode sollte auch nur dann verwendet werden, wenn Sicherheit kein Thema ist (was in manchen Fällen durchaus Sinn machen kann!) und wenn keine speziellen Zugriffskontrollen notwendig sind. Dieser Fall wird angenommen, wenn das DN-Feld (Distinguished Name) nicht ausgefüllt und

das Passwort nicht mitgegeben wird. LDAP geht dann automatisch davon aus, dass eine so genannte „anonymous" Session gewünscht wird. Die Zugriffe werden dann gemäß der Vorgaben für anonyme User gewährleistet (oder eben nicht).

Basis-Authentifizierung

Dieser Sicherheitsmechanismus wird ausgehandelt, wenn eine Verbindung zwischen einem Client und einem Server aufgebaut wird. Beim Einsatz der Basis-Authentifizierung identifiziert sich der Client dem Server gegenüber mit einem DN und einem Passwort. Das mitgelieferte Passwort wird mit dem auf dem LDAP Server abgelegten Passwort verglichen. Bei Übereinstimmung wird der entsprechende Zugriff gewährt. Das Passwort geht in diesem Fall meistens unverschlüsselt über das Netz. Einige Implementationen unterstützen Base64-Verschlüsselung, die jedoch auch nicht als sehr sicher eingestuft werden darf.

Simple Authentication and Security Layer (SASL)

SASL ist ein Framework, um zusätzliche Sicherheitsmechanismen für verbindungsorientierte Protokolle einzubringen. SASL wird ab LDAP Version 3 unterstützt, um vor allem die Schwachstellen bezüglich Authentifizierung in der Version 2 auszuräumen. Der Layer war ursprünglich dazu gedacht, um für das IMAP-Protokoll bessere Authentifizierungsmechanismen zur Verfügung zu stellen. Eine generelle Erweiterung ermöglicht inzwischen den Einsatz, wenn es allgemein um die Zusammenarbeit zwischen Protokollen und Authentifizierungen geht. SASL ist im RFC 2222 beschrieben.

In SASL werden die Verbindungsprotokolle wie IMAP, LDAP etc. über Profile repräsentiert. Ein Profil wiederum kann als Protokollerweiterung gesehen werden, die es ermöglicht, dass die Protokolle und SASL zusammenarbeiten. Eine komplette Liste der SASL-Profile gibt es beim Information Sciences Institute (ISI). Jedes Protokoll, das SASL unterstützt, muss um ein Kommando erweitert werden, um einen Authentifizierungsmechanismus zu unterstützen und einen Authentifizierungsaustausch vornehmen zu können, damit die Vertraulichkeit eines Informationsaustausches gewährleistet wird.

LDAP V3 beinhaltet ein derartiges Kommando (**ldap_sasl_bind()**). Die Schlüsselparameter:

dn

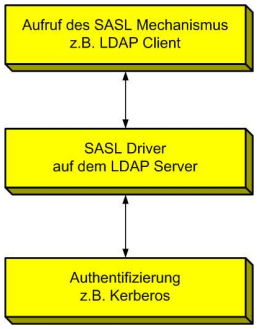
der Distinguished Name des Eintrags, über den man sich verbinden will; das ist in der Regel die UserID, mit der man sich anmeldet.

mechanism
: die Bezeichnung der Security-Methode, die verwendet werden soll. Beispiele dafür sind Kerberos Version 4, S/Key, GSSAPI, CRAM-MD5 und EXTERNAL. Es gibt auch einen ANONYMOUS-Mechanismus, der eine Anmeldung als „anonymer" Benutzer ermöglicht. In Verbindung mit LDAP ist der gebräuchlichste Mechanismus SSL bzw. TLS, die als EXTERNAL-Mechanismen unterstützt werden.

credentials
: Dies sind willkürliche Daten, die einen DN identifizieren. Format und Inhalt der Parameter hängen vom gewählten Mechanismus ab. Es kann ein beliebiger String wie beispielsweise eine E-Mail-Adresse sein, die einen User identifiziert.

Durch den SASL-Bind-API-Funktionsaufruf ruft eine Client-Anwendung den SASL-Protokolltreiber auf dem Server auf, der sich mit dem Authentifizierungssystem, das im SASL-Mechanismus definiert ist, verbindet, um die Authentifizierungsinformation für den User zu bekommen.

Abbildung 8.5: Ablauf des SASL-Prozesses



SASL bildet somit eine Vermittlungsfunktion zwischen einem Authentifizierungssystem und einem Protokoll wie LDAP.

Natürlich muss auch der Server den SASL-Mechanismus unterstützen, da sonst der Authentifizierungsmechanismus nicht funktioniert. Ob und wie ein Server SASL unterstützt, kann mit einem Browser und einer URL festgestellt werden:

```
ldap://<ldap server>/?supportedsaslmechanisms
```

Auf die Möglichkeiten der LDAP-URLs gehen wir noch ein.

Wie somit dargestellt wurde, ermöglicht SASL den Einsatz als Framework, um den Beteiligten die Auswahl eines bestimmten Sicherheitsmechanismus zu ermöglichen. Die entsprechende Aushandlung des Verfahrens wird im Klartext vorgenommen. Wenn sich Client und Server geeinigt haben, ist die Verbindung robust gegenüber der Modifizierung der Authentifizierungsidentitäten.

Ein Angreifer könnte nun versuchen, den Verhandlungsvorgang abzuhören und einen Beteiligten vortäuschen, der den geringsten Sicherheitslevel haben will. Um dies zu verhindern, sollten Clients und Server so konfiguriert sein, dass sie auch für die Verhandlung einen minimalen Sicherheitsmechanismus anwenden.

SSL und TLS

Wie bereits erläutert, sind SSL und TLS die Mechanismen, die normalerweise für SASL und LDAP verwendet werden.

Das SSL-Protokoll wurde entwickelt, um für TCP/IP zwischen Transport- und Anwendungsprotokollen eine Ebene (Layer) einschieben zu können, die sich sowohl um Authentifizierung als auch um die sichere Übertragung kümmert.

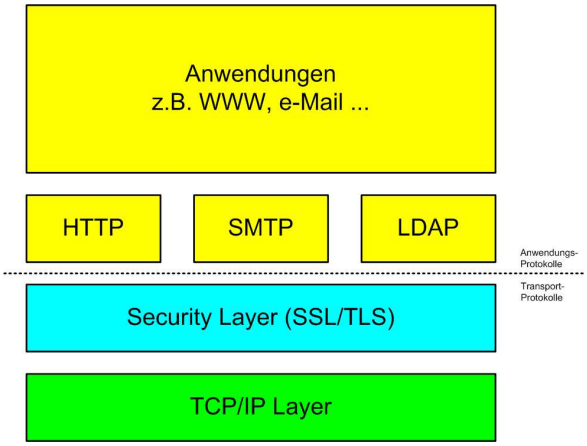


Abbildung 8.6: Schema: Beziehung zwischen Anwendungs- und Transportebene

SSL wurde von Netscape entwickelt. Transport Layer Security (TLS) ist ein offener Standard der IETF und kann als Nachfolger von SSL angesehen werden. TLS basiert auf SSL 3.0 mit nur wenigen Unterschieden, vor allem ist er kompatibel zu SSL 3.0. Es kann somit davon ausgegangen werden, dass TLS SSL in vollem Umfang ersetzen wird. Die folgenden Anmerkungen treffen sowohl für SSL als auch für TLS zu.

Es wird im Folgenden nun davon ausgegangen, dass der Server bereits ein Schlüsselpaar erzeugt hat. Dies wird normalerweise beim Aufsetzen eines LDAP Servers erledigt.

Zwischen Client und Server erfolgt ein so genanntes „Handshaking", um entsprechende Informationen auszutauschen.

1. In einem ersten Schritt fordert der Client vom Server den Aufbau einer SSL/TLS Session an. Er gibt gleichzeitig seine Optionen mit, die er für den Aufbau unterstützt und vorschlägt.

2. Der Server antwortet mit seinen SSL/TLS-Optionen sowie mit einem Zertifikat, das unter anderem den öffentlichen Schlüssel des Servers sowie den Namen der Zertifizierungsstelle, die das Zertifikat ausgestellt hat, und das Ablaufdatum enthält.

3. Der Client fordert nun den Server auf, seine Identität zu beweisen. Dies ist notwendig, damit gewährleistet ist, dass das Zertifikat nicht von einem Dritten gesendet wurde, der das Zertifikat bei einer früheren Session abgehört hat.

4. Der Server sendet eine Nachricht zurück, einschließlich eines so genannten *Message Digest* (so etwas wie eine Prüfsumme), der mit dem privaten Schlüssel verschlüsselt wurde. Der Client kann den Message Digest mit dem öffentlichen Schlüssel des Servers entschlüsseln und vergleicht ihn mit dem eigenen errechneten Wert. Bei Übereinstimmung ist die Identität des Servers gesichert, und der Authentifizierungsprozess ist abgeschlossen.

5. Als Nächstes müssen sich Client und Server auf einen geheimen symmetrischen Schlüssel einigen, der für die Verschlüsselung der Session-Daten verwendet wird. Die (oft umfangreichen) Session-Daten werden mit einem symmetrischen Verschlüsselungsverfahren realisiert, da dies wesentlich weniger Ressourcen benötigt. Der Client erzeugt deshalb einen Session Key, verschlüsselt diesen mit dem öffentlichen Schlüssel des Servers und schickt diesen zum Server. Nur mit dem privaten Schlüssel des Servers kann der Session Key wieder entschlüsselt werden.

6. Der Server entschlüsselt den Session Key und sendet eine Testnachricht zurück, die mit dem Session Key verschlüsselt wurde. Wenn diese Testnachricht gelesen werden kann, können nun verschlüsselt die Daten ausgetauscht werden.

LDAP unter z/OS

Vor allem wenn es um Security geht (Policies, Passwörter, X.509-Zertifikate), werden künftig LDAP-Verzeichnisse verwendet, da LDAP ein offener Standard ist, der von allen Herstellern unterstützt wird.

Allerdings werden an einen LDAP Server für derart kritische Informationen auch entsprechend hohe Anforderungen gestellt. Dies betrifft die Verfügbarkeit, Performance, Fehlertoleranz und natürlich Security in hohem Maße. Wenn ein LDAP Server für diesen Bereich nicht läuft, bedeutet das, dass sich niemand an einem System anmelden kann.

Ein LDAP Server unter z/OS bietet sich dann an, wenn die Anforderungen betreffend Transaktionsvolumen und Verfügbarkeit besonders hoch sind. Die notwendige Fehlertoleranz (Vermeidung eines Single Point of Failure) und ein dynamisches

Workload Management ist in Verbindung mit z/OS beispielsweise mit einer Parallel-Sysplex-Konfiguration erreichbar.

Es können mehrere Server mit den gleichen LDAP-Daten zur Verfügung gestellt werden. Eine derartige Konfiguration wird als *Multiserver Mode* bezeichnet. In Zusammenarbeit mit Workload Manager/Domain Name System in Verbindung mit dem Communication Server for z/OS wird ein Workload Balancing erreicht: Anforderungen werden zu dem System geleitet, das gerade die meisten Ressourcen zur Verfügung stellen kann.

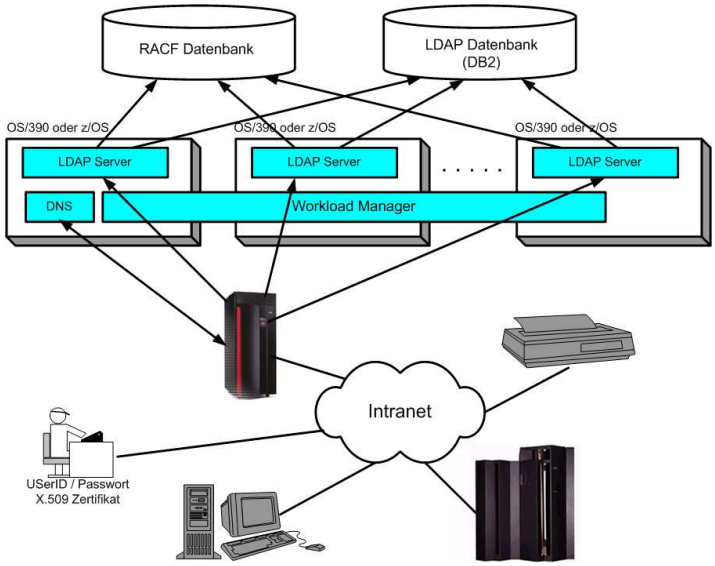


Abbildung 8.7: Beispielkonfiguration mit LDAP

In einer Parallel-Sysplex-Umgebung werden auf unterschiedlichen Systemen mehrere LDAP Server gestartet. Jeder dieser Server verfügt über die gleiche Funktionalität. Alle LDAP Server greifen gemeinsam auf die Verzeichniseinträge zu, die in einer DB2-Datenbank in Verbindung mit Data Sharing verwaltet werden.

Zugriff auf RACF

Einige Informationen, die in RACF abgelegt sind, speziell die Informationen in User- und Gruppenprofilen, sind natürlich auch als Verzeichnisdaten interessant. Damit diese Daten nun jedoch nicht doppelt gehalten werden müssen, stellt RACF eine LDAP-Schnittstelle zur Verfügung.

Mit LDAP Services und entsprechenden Attributen können RACF-Profile abgefragt und modifiziert werden. Die Schnittstelle zwischen RACF und LDAP macht im Grunde genommen nichts anderes, als LDAP Requests in RACF-Kommandos umzuset-

zen. Die entsprechende RACF-Berechtigung für das Absetzen der Kommandos muss natürlich vorhanden sein.

LDAP greift auf die Informationen mit LDAP-Attributnamen zu. Die UserID heißt beispielsweise **racfid** und der Username **racfProgrammerName**. Die komplette Attributliste ist in den Files **slapd.at.racf** und **slapd.oc.racf** zu finden, die standardmäßig im USS-Verzeichnis **/usr/lpp/ldap/etc** liegen.

Da die Anforderungen bezüglich Security in einem Verzeichnis sehr unterschiedlich sein können, muss auch ein entsprechender Security Level zu definieren sein. Die Abfrage einer E-Mail-Adresse oder eines Public Key sollte beispielsweise allen (auch nicht identifizierten) Benutzern zugänglich sein; das zu einer UserID gehörige Passwort dagegen natürlich nur mit Identifikation und entsprechender Berechtigung.

Anforderungen

Authentifizierung
: Der Nutzer muss seine Identität beweisen. Dies geschieht beispielsweise mit Zertifikaten in Verbindung mit SASL/SSL.

Zugriffskontrolle
: Der Directory Server liefert nur Daten aus, wenn die entsprechende Berechtigung vorhanden ist. Diese Funktion wird über Access Control Lists (ACLs) realisiert.

Integrität
: Die zuverlässige Speicherung und unverfälschte Übertragung muss gewährleistet sein. Mögliche Veränderungen auf dem Transportweg müssen erkannt werden. Auch hier kann wiederum SSL eingesetzt werden.

Vertraulichkeit
: Sensitive Daten müssen sicher gespeichert und übertragen werden können. User-Passworte werden verschlüsselt abgespeichert.

Funktionen des z/OS LDAP Server

Multiple-Concurrent-Database-Instanzen
: Der z/OS LDAP Server kann so konfiguriert werden, dass er auf mehrere Datenbanken gleichzeitig zugreifen kann. Er kann beispielsweise so aufgebaut sein, dass er Zugriff auf RACF hat und gleichzeitig anwendungsspezifische Informationen in DB2-Datenbanken gespeichert hat.

Robuste Datenbank

Der z/OS LDAP Server stützt sich auf eine TDBM-Backend-Datenbank, die auf DB2 basiert. Eine TDBM-Datenbank ist besser skalierbar als eine RDBM-Datenbank, die vor Release 10 genutzt wurde. Aus Kompatibilitätsgründen kann zwar auch weiterhin die RDBM-Variante genutzt werden, dies ist jedoch nicht empfehlenswert.

Laden von großen Datenmengen

Mit dem Utility **ldif2tdbm** können große Datenmengen in eine DB2 TDBM geladen werden.

Access Control

Die Zugriffskontrolle auf die Informationen, die in dem LDAP Server gespeichert sind, wird über eine leistungsfähige Access Control Facility gewährleistet, die bis auf Attributebene granular arbeiten kann. Sie funktioniert sowohl für die TDBM- als auch für die RDBM-Version.

Replikation

Der z/OS LDAP Server kann in Form eines Master/Slave-Replikationsschemas konfiguriert werden. Dies spielt in größeren Umgebungen, wo unter anderem auch Aspekte wie Verfügbarkeit und Performance beachtet werden müssen, eine wichtige Rolle.

Verschlüsselung

Der LDAP Server kann so aufgesetzt werden, dass die Daten verschlüsselt übertragen werden. Auch die Client-Authentifizierung mit X.509-Zertifikaten wird unterstützt.

Einsatz mehrerer Server

Eine Konfiguration kann so aufgebaut werden, dass mehrere Server-Instanzen mit der gleichen Datenbank arbeiten. Die Instanzen können sowohl im gleichen System-Image als auch in einem Sysplex-Verbund laufen.

Dynamisches Workload-Management

Der z/OS LDAP Server kann so konfiguriert werden, dass das dynamische Workload Management in einer Sysplex-Umgebung in Verbindung mit TCP/IP unterstützt wird. Abhängig von der Workload können in unterschiedlichen Systemen mehrere Server-Instanzen gestartet werden.

Zugriff auf RACF-Daten

Über den z/OS LDAP Server kann mit dem LDAP-Protokoll auch auf RACF User- und Gruppenprofile zugegriffen werden. Dies funktioniert auch in Sysplex-Umgebungen, wenn die RACF-Daten geshared werden. Die LDAP-Zugriffe auf RACF werden von einem zusätzlichen Backend verwaltet, der als SDBM bezeichnet wird.

Dynamic Schema
: Wenn ein TDBM Backend eingesetzt wird, kann ein Schema über das LDAP-Protokoll dynamisch geändert werden.

SASL-External-Bind/Client- und -Server-Authentifizierung
: Der z/OS LDAP Server ermöglicht es den Client-Anwendungen, für die Kommunikation mit einem Server über SSL Zertifikate zu nutzen.

Unterstützung des Root DSE
: Der z/OS LDAP Server unterstützt Suchoperationen auf das Root des Verzeichnisbaums, wie im RFC 2251 beschrieben.

Extended Group Membership Searching
: Dies ermöglicht es einem LDAP Server, Distinguished Names (DN) in unterschiedlichen Backends zu finden.

Anonyme Authentifizierung
: Diese Funktion ist wichtig für den Lesezugriff auf nicht-sensitive Daten. Beispiele dafür sind E-Mail-Adressen, Telefonnummern, Raumnummern etc. Die anonyme Authentifizierung wird ganz einfach mit einem leeren DN vorgenommen.

Konfiguration des z/OS LDAP Servers

Der Name des LDAP Server Daemons unter z/OS ist **slapd** und stimmt somit mit der Namensbezeichnung für andere UNIX-Umgebungen überein.

In dem Verzeichnis **/usr/lpp/ldap/examples/sample_server** finden Sie ein Beispiel, wie ein z/OS LDAP Server aufgesetzt werden kann.

DB2-Konfiguration für LDAP

Bevor der eigentliche LDAP Server aufgesetzt werden kann, muss DB2 für LDAP konfiguriert werden. Vorausgesetzt wird mindestens die Version 5 von DB2. Wenn mit dem Multi-Server Mode gearbeitet werden soll, muss eine DB2 Sharing Group mit Members auf den beteiligten Systemen, auf denen eine LDAP Server-Instanz laufen soll, eingerichtet werden.

Abhängig von dem oder den ausgewählten Backends müssen Bufferpools, TEMP Space, TEMP-Dateien etc. eingerichtet werden. Hier ist eine enge Zusammenarbeit mit der DB-Administration notwendig.

Mit dem DB2 **runstats**-Kommando kann die Konfiguration nach dem Laden von Daten auf optimale Queries hin überprüft werden.

Es steht ein Beispieljob in der Datei DSNHLQ.SDSNSAMP(DSNTIJCL) zur Verfügung. Damit kann die DB2-CLI-Umgebung aufgesetzt werden, um dem LDAP Server die Nutzung des Call Level Interface zu ermöglichen. So wird das Binding des CLI-Plans (z. B. mit dem Plannamen DSNACLI) durchgeführt. Der Job muss mit einer UserID gestartet werden, die entsprechende DB-Berechtigungen hat.

Hier ein Beispiel für ein DB2-CLI-Initialisierungsfile:
DSNHLQ.SDSNSAMP(DSNAOINI)

Aufsetzen von RACF

Um über LDAP auf RACF zugreifen zu können, muss RACF natürlich zunächst einmal installiert sein. Die RACF Subsystem Function muss definiert und aktiviert sein, damit der LDAP Server mit dem SDBM Backend kommunizieren kann.

Aktivieren von SSL

Damit der LDAP Server SSL unterstützt, müssen die z/OS Cryptographic Services System SSL installiert sein und über STEPLIB, LPALIB oder LINKLIST angehängt werden.

Aktivieren von OCSF und ICSF für Passwortverschlüsselung

Der LDAP Server nutzt die *Open Cryptographic Services Facility* (OCSF) für MD5 und SHA Hashing der Passwörter in den TDBM oder RDBM Backends. Er verwendet darüber hinaus sowohl OCSF als auch *Integrated Cryptographic Service Facility* (ICSF), um die DES-Verschlüsselung zu realisieren.

Aufsetzen des LDAP Servers

Da LDAP sinnvollerweise als Started Task im MVS laufen sollte, empfiehlt es sich, eine separate UserID für den LDAP Server einzurichten.

Als Standard wird LDAPSRV für die UserID vorgeschlagen. Als Produktionsverzeichnis wird /etc/ldap angenommen.

Die UserID muss mit UID 0 versehen werden, Leseberechtigung auf BPX.DAEMON und Update-Berechtigung auf BPX.SERVER haben.

Beispiel für die RACF-Kommandos:

```
ADDGROUP LDAPGRP SUPGROUP(SYS1)OMVS(GID(2))
ADDUSER LDAPSRV DFLTGRP(LDAPGRP)OMVS(UID(0)PROGRAM ('/bin/sh'))
```

```
PERMIT BPX.DAEMON CLASS(FACILITY)ID(LDAPSRV)ACCESS(READ)
PERMIT BPX.SERVER CLASS(FACILITY)ID(LDAPSRV)ACCESS(UPDATE)
```

Konfigurations-File des LDAP Servers

Der LDAP Daemon hat den Namen **slapd**; es gibt ein Konfigurations-File mit dem Namen **slapd.conf**.

Das Konfigurations-File wird per Default in das Verzeichnis **/usr/lpp/ldap/etc** installiert. Es sollte in das Verzeichnis **/etc/ldap** kopiert werden, damit es dort entsprechend angepasst werden kann. In dem Konfigurations-File sind zahlreiche Defaults gesetzt. Es enthält jedoch kein Datenbank-Suffix.

In dem File gibt es die folgenden Bereiche:

- Global Section
- SDBM Section
- TDBM Section
- RDBM Section

Betrieb des LDAP Servers

Der LDAP Server benötigt beim Start den Zugriff auf DLLs, die sich in einem PDS befinden (**GLDHLQ.SGLDLNK**).

LINKLIST/STEPLIB

Bei der Serverpack-Installation ist der HLQ **GLD**. Die Module müssen entweder über die **LINKLIST** oder über ein **STEPLIB DD**-Statement in der Startprozedur eingebunden werden.

Wenn der LDAP Server aus OMVS gestartet wird, muss im Falle einer Steplib die **STEPLIB**-Umgebungsvariable gesetzt werden.

SCEERUN

Der LDAP Server benötigt außerdem das **SCEERUN** Dataset. Auch dieses muss entweder über die LINKLIST oder über eine Steplib eingebunden werden.

Die Started Task mit dem Namen **LDAPSRV** muss in einer dem System angehängten Prozedurbibliothek stehen und im RACF als Started Task definiert werden. Ein Beispiel für die RACF-Kommandos:

```
RDEFINE STARTED LDAPSRV.STDATA((USER(LDAPSRV))
SETROPTS RACLIST(STARTED)REFRESH
```

Danach kann der Server von der Konsole aus mit einem Startkommando gestartet werden:

```
S LDAPSRV
```

Es gibt einige Parameter, über die beispielsweise eine Portnummer, ein anderes Konfigurations-File oder der Debugging Level etc. mitgegeben werden können.

Bei erfolgreichem Start erscheint die Meldung:

```
GLD 122I SLAPD is ready for requests
```

LDAP Utilities unter z/OS

In den meisten LDAP SDKs gibt es Kommandos, die in einer Kommandozeile des Betriebssystems ausgeführt werden können. Die Tools nutzen die LDAP APIs und können als Beispielanwendungen verwendet werden. Sie stehen auch unter z/OS zur Verfügung.

Die wichtigsten Programme:

- ldapsearch
- ldapadd
- ldapmodify
- ldapdelete
- ldapmodrdn

Jedes der Utilities entspricht einer LDAP-Protokoll-Operation. Die Tools können in Zusammenarbeit mit einem Scripting Tool (z.B. einem Shellskript oder einer REXX-Prozedur) auch kombiniert werden. Sie können beispielsweise auch in Web-basierten CGI-Programmen eingesetzt werden.

Ein Beispiel für eine Suchfunktion:

```
ldapsearch -h mvs012 -b "ou=marketing, o=tps-data, c=ch" -s one
cn="Wolfram Greis" cn email
```

Utilities für die Verwaltung von TDBM und RDBM

Hierbei geht es um Utilities, mit denen Daten in den LDAP Server gebracht bzw. aus dem LDAP Server geholt werden können.

Tabelle 8.1: TDBM- und RDBM-Utilities

Funktion	TDBM	RDBM
Laden von Daten in die Datenbank	ldif2tdbm	ldif2db
Entladen von Daten aus der Datenbank	tdbm2ldif	db2ldif

Die Utilities können aus der USS Shell, aus Batchjobs per JCL, aus REXX-Prozeduren oder direkt aus TSO aufgerufen werden.

Aufruf aus der Shell

Um die Utilities aus der Shell heraus aufrufen zu können, müssen einige Umgebungsvariablen gesetzt werden. Beim Start eines der Utilities wird ein File **/etc/ldap/slapd.envvars** gelesen. Dieses File kann über die Umgebungsvariable **LDAP_SLAPD_ENVVARS_FILE** zugewiesen werden. Außerdem muss **/usr/sbin** als Pfad für aufrufbare Programme hinzugefügt werden. Sofern das PDS nicht mit der LINKLIST verkettet ist, muss die STEPLIB auf **GLDHLQ.SGLDLNK** gesetzt sein.

Aufruf per JCL

Es werden bei der Installation Beispieljobs mitgeliefert, die in **GLDHLQ.SGLDSAMP** abgelegt werden.

Hinweis: Der Input für das **ldif2db**-Utility (**GLDLD2DB**) muss, unabhängig davon, wie es aufgerufen wird, im Hierarchical File System von USS abgelegt werden. Dies gilt auch dann, wenn über SYSIN aus einem Batchprogramm gelesen wird. Das bedeutet, in der JCL muss ein **PATH** Keyword definiert werden, das in das HFS verweist.

Beispiel:

```
//SYSIN DD PATH=/ldap/db2input/db001.ldif, ...
```

Aufruf aus TSO

1. Definition des PDS, in dem die LDAP-Servermodule zu finden sind.

   Beispiel:

   ```
   tsolib act dsn('GLDHLQ .SGLDLNK')
   ```

2. Zuweisung über SYSEXEC

```
concatd f(SYSEXEC)da('GLDHLQ .SGLDEXEC')
```

3. Beim Aufruf kann mit der Option -f ein Konfigurations-File mitgegeben werden.

   Beispiel:

```
ldif2db -f "//'datasetname '"-i /tmp/ldif.1
```

4. Alternativ dazu kann ein Konfigurations-File auch über einen DD-Namen mit einem ALLOCATE-Kommando zugewiesen werden.

```
alloc da('datasetname ') fi(config) shr
```

   Der Aufruf erfolgt dann analog zum Aufruf aus der Shell, mit dem Unterschied, dass die Utilities ldf2tdbm statt ldif2tdbm und tdbm2ldf statt tdbm2ldif heißen, da sie sonst den TSO-Namenskonventionen nicht mehr entsprächen.

Das Password Encryption Utility

Mit diesem Administrations-Utility (db2pwden) kann ein Passwort im Klartext in ein verschlüsseltes Passwort für ein DB2 Backend umgewandelt werden.

Es kann entweder aus der USS Shell oder aus TSO aufgerufen werden. Für den Aufruf aus der Shell müssen entsprechende Umgebungsvariablen gesetzt und /usr/sbin in der Pfaddefinition eingetragen sein. Die STEPLIB muss auf GLDHLQ.SGLDLNK gesetzt sein, sofern die Datei nicht mit der LINKLIST verkettet ist.

Das Utility kann nur von einem LDAP-Administrator aufgerufen werden. Beispiel:

```
db2pwden -h ushost -p 391 -D "cn=admin"-w "secret"-b "o=university,c=US"
```

LDAP URLs

Hinter einer URL (Unified Resource Locator) steht ein definierter Standard, um eine Ressource im Internet oder Intranet zu lokalisieren. Bekannt sind die URLs, die mit http, ftp oder telnet beginnen.

Da LDAP sich inzwischen zu einem bedeutsamen Protokoll für das Internet entwickelt hat, wurde auch ein URL-Format (RFC 2255) für LDAP-Ressourcen definiert.

Die Syntax:

```
ldap[s]://[<host>[:<port>]][/[<dn>[?[<attributes>
          [?[<scope>][?[<filter>[?<extensions>]]]]]]
```

Die Bedeutung im Einzelnen:

ldap[s]
: das Protokoll, in diesem Fall LDAP; mit dem Zusatz „s" wird eine SSL-Verbindung definiert.

host
: Name des Hosts oder die IP-Adresse

port
: Portnummer; als Standard wird für LDAP 389 und für LDAP über SSL 636 angenommen.

dn
: Distinguished Name, der als Basis für die Suchoperation verwendet werden soll

attributes
: eine durch Kommata getrennte Liste von Attributen, die zurückgegeben werden sollen; werden keine Attribute explizit definiert, werden alle verfügbaren Attribute zurückgegeben.

scope
: Geltungsbereich des Suchvorgangs; möglich sind base, one oder sub; Default ist base.

filter
: der Search-Filter, der verwendet werden soll; wird keiner mitgegeben, wird objectClass=* angenommen und alle Einträge zurückgegeben.

Beispiele:

```
ldap://mvs012.tps-data.com

ldap://mvs012.tps-data.com:9389/ou=marketing,c=de

ldap://mvs012.tps-data.com/cn=Wolfram%20Greis
      ,ou=Professional%20Services,o=tps-data,c=CH
```

Das letzte Beispiel ist etwas erklärungsbedürftig. Da in einer URL bestimmte Zeichen (z. B. ein Leerzeichen) nicht verwendet werden dürfen, müssen diese mit einem Prozentzeichen eingeleitet werden. Danach folgt der hexadezimale ASCII-Wert des Zeichens. Eine Leerstelle hat den hexadezimalen ASCII-Wert 20. Diese so genannten „unsafe characters" sind im RFC 1738 definiert.

Die LDAP URLs sind sehr flexibel. Man kann damit alles von einem LDAP Server bis hin zu einem Attribut eines Eintrags abfragen. LDAP URLs können auch in Anwendungen verwendet werden. Es gibt bereits Diskussionen, wie LDAP URLs künftig auch in einem Domain Name Service (DNS) gespeichert werden können.

Kapitel 9

Connectivity

9.1 Das OSI-Referenzmodell

Es gibt zahlreiche Netzwerkarchitekturen, die in unterschiedlichen Systemumgebungen eine mehr oder weniger große Rolle spielen. Das wichtigste Unterscheidungskriterium betrifft die Herkunft der Architektur. Es geht darum, ob die Architektur von einem Hersteller (z. B. IBM mit der System Network Architecture oder Siemens mit Transdata) oder von einem unabhängigen Gremium (z. B. Open System Interconnection mit dem OSI-Modell oder die Internet Engineering Task Force, IETF, mit TCP/IP) definiert wird.

Da es bei der Vernetzung in der heutigen Zeit immer mehr darum geht, heterogene Systeme von unterschiedlichen Herstellern zusammenzubringen, haben die proprietären Netzwerkarchitekturen keine Chance mehr, sich am Markt gegenüber den offenen Architekturen durchzusetzen.

Unabhängig davon arbeiten heute alle Architekturen mit einer Schichtenarchitektur. Dies hat den Vorteil, dass Funktionalitäten der Datenübertragung in definierten

Schichten untergebracht werden können. Jede Schicht hat eine ganz bestimmte Aufgabe und dient als Grundlage für die jeweils höhere Schicht. Die Schichten müssen sich nur jeweils um die Schnittstellen zur Nachbarschicht kümmern. Was in diesen Schichten im Einzelnen passiert, kümmert sie nicht.

Wenn auch für die Praxis nicht immer so relevant, wie man es sich im Sinne von „De jure"-Standards wünschen würde, spielt das OSI-Modell eine wichtige Rolle bei der Einordnung und Abgrenzung von Kommunikationslösungen, Architekturen und Standards. Es bildet eine so genannte Referenz, mit deren Hilfe es für Netzwerk-Spezialisten einfach wird, sich in Diskussionen darüber zu einigen, worüber man eigentlich redet.

Die sieben OSI-Schichten

Das OSI-Referenzmodell teilt die Datenübertragungsfunktionen in sieben Schichten ein.

Während der Datenstrom in vertikaler Richtung beim Sender von oben nach unten (von Schicht sieben zu Schicht eins) und beim Empfänger von unten nach oben (von Schicht eins zu Schicht sieben) geleitet wird, findet die logische Kommunikation horizontal zwischen den jeweils gleichen Schichten statt. Diese logische Kommunikation wird in Form von Protokollen realisiert.

Die Zusammenfassung der Schichten wird auch als Protokollstack bezeichnet. Verfolgt man die Daten einer Anwendung auf ihrem Weg durch den Protokollstack, stellt man fest, dass diese Daten in der jeweiligen Schicht mit einem Header und einem Trailer versehen, also regelrecht verpackt werden. Diese Header und Trailer sind die Basis für die Kommunikation der Schnittstellen zwischen den Schichten und für die horizontale Kommunikation mit Hilfe der Protokolle.

Die Informationen in den Headern und Trailern dienen der Fehlererkennung und enthalten wichtige Steuerfunktionen. Für die Arbeit der Protokolle verpackt eine Schicht auf der Senderseite Informationen in Header und Trailer und gibt sie an die darunter liegende Schicht weiter. Die komplementäre Schicht auf der Empfängerseite erhält die Daten mit Header und Trailer von der darunter liegenden Schicht. Sie entpackt und interpretiert die Daten in Header und Trailer, führt die entsprechenden Arbeitsschritte durch und gibt die Daten schließlich an die darüber liegende Schicht weiter. Diese verfährt analog, bis die Daten bei der Zielanwendung angekommen sind.

Schicht 1

Die unterste Schicht kümmert sich um das Übertragungsmedium und das physikalische Umfeld für die Datenübertragung. Deshalb wird diese Schicht auch ***physikalische Schicht, Bitübertragungsschicht*** oder im englischen Sprachraum auch ***Network Layer*** genannt, da es um die Schnittstelle zum darunter liegenden Netzwerk geht.

Schicht 2

Die Aufgabe der Schicht zwei ist im Wesentlichen die Absicherung und Kontrolle der fehlerfreien Übertragung des Bitstroms. Sie wird deshalb als *Sicherungsschicht* und im englischen Sprachgebrauch als *Physical Layer* bezeichnet. Bei den LAN-Standards ist diese Schicht zweigeteilt.

Die Schicht 2a bildet den *Medium Access Control (MAC) Layer* mit den wesentlichen Funktionen der IEEE-Standards 802.3 (Ethernet), 802.5 (Token Ring), 802.11 (Wireless LAN), FDDI etc.

Die Schicht 2b wird als *Logical Link Control (LLC) Layer* bezeichnet, die sich in erster Linie um die Erfüllung des IEEE-Standards 802.2 kümmert.

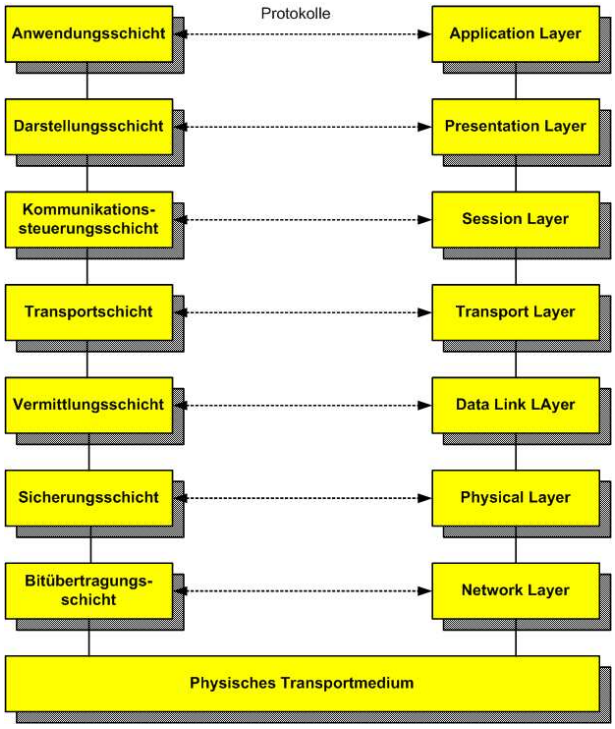


Abbildung 9.1: Das OSI-Referenzmodell

Schicht 3

Die Schicht drei kümmert sich um Routing-Funktionen, die den Aufbau von logisch strukturierten Netzen ermöglichen. Sie wird als *Vermittlungsschicht* oder im englischen Sprachraum als *Data Link Layer* bezeichnet. Im LAN-Bereich werden beispielsweise auf dieser Ebene die Standards Internet Protocol (IP), das XEROX Network System Protocol (XNS) oder das IPX Protocol von Novell eingesetzt.

Schicht 4

Die Schicht vier stellt eine transparente Datenübertragung zwischen Endsystemen zur Verfügung. Sie wird als *Transportschicht* oder im englischen Sprachraum als *Transport Layer* bezeichnet. Die Standards auf dieser Ebene stellen unterschiedliche Dienstklassen zur Verfügung. Man unterscheidet zwischen verbindungsorientierten und verbindungslosen Protokollen insbesondere im Hinblick auf Übertragungssicherheit und Fehlerkorrektur. Beim Beispiel TCP/IP Protokollstack etwa kümmert sich auf dieser Ebene das Transmission Control Protocol (TCP) um eine sichere und garantierte Übertragung zwischen den Endsystemen und ist verbindungsorientiert, während das User Datagram Protocol (UDP) auf der gleichen Ebene keine Prüfungen und Sicherungen vornimmt und verbindungslos arbeitet.

Schichten 5–7

Die Schichten fünf bis sieben werden als Anwendungsschichten bezeichnet. Die Schicht 5, *Sitzungsschicht* oder *Session Layer*, sorgt für die Prozesskommunikation und die Umsetzung und Darstellung der Informationen. Die Schicht 6, *Darstellungsschicht* oder *Presentation Layer*, kodiert/dekodiert die Daten für das jeweilige System. Die Schicht 7 schließlich wird als *Anwendungsschicht* (*Application Layer*) bezeichnet und stellt anwendungsspezifische Protokolle zur Verfügung.

9.2 System Network Architcture (SNA) versus TCP/IP

In Verbindung mit dem IBM Mainframe gibt es zwei wesentliche Netzwerk-Architekturen. Die proprietäre Architektur ist die System Network Architecture (SNA), die aufgrund der Vergangenheit auch heute noch eine Rolle spielt. Vor allem gibt es viele Begriffe, die aus der SNA-Welt kommen und auch in heutigen Umgebungen relevant sind. Die herstellerunabhängige Architektur, die auch auf dem Mainframe heute Standard ist, ist TCP/IP.

In diesem Zusammenhang einige Worte zu den Unterschieden zwischen SNA und TCP/IP: SNA und TCP/IP haben grundverschiedene Philosophien. Jede dieser Philosophien hat Stärken und Schwächen. Das ändert jedoch nichts daran, dass sich TCP/IP inzwischen durchgesetzt hat, weil damit eben eine weltweite Kommunikation möglich ist, da TCP/IP heute auf allen Plattformen verfügbar ist.

Hierarchischer Aufbau versus Peer-to-Peer

Während SNA einen streng hierarchischen Aufbau mit „Steuerzentralen" im jeweiligen Host hat, fehlt bei TCP/IP diese Steuerzentrale. In TCP/IP gibt es nur gleichbe-

rechtigte Kommunikationspartner. Deshalb wird ein derartiges Netzwerk auch als Peer-to-Peer-Netzwerk bezeichnet.

Hierzu muss die Herkunft von TCP/IP berücksichtigt werden. Die Vorgängerversion von TCP/IP wurde vom Verteidigungsministerium (Department of Defense) in den USA initiiert. Übrigens auch der Grund, warum man in Verbindung mit TCP/P auch oft von den „DoD-Protokollen" spricht. Die Vorgabe war, ein Netzwerk aufzubauen, das im Ernstfall auch nach dem Ausfall eines beliebigen Knotens im Netz noch funktionsfähig ist. Das führte zu einem Netzwerkdesign, das ohne zentrale Steuerung funktionieren muss.

Art der Verbindung

Ein weiteres wichtiges Unterscheidungskriterium ist die Art und Weise, wie eine Verbindung aufgebaut und aufrechterhalten wird. Während SNA eine Punkt-zu-Punkt-Verbindung aufbaut, funktioniert TCP/IP paketorientiert.

SNA und Telefon SNA Sessions können mit der klassischen Telefonverbindung verglichen werden, wo nach dem Kommunikationsaufbau eine feste Verbindung zwischen zwei Kommunikationspartnern besteht. Wenn diese Verbindung unterbrochen wird, muss sie neu aufgebaut werden. Vorteil dieser Kommunikationsart ist die geringere Anfälligkeit bezüglich Abhören und Manipulation der zu übertragenden Daten.

TCP/IP dagegen kann mit der Paket/Briefpost verglichen werden, wo die Informationen paketweise zwischen den Kommunikationspartnern verschickt werden. Das Netzwerk ist aus beliebig vielen Paketen zusammengesetzt. Der Transportweg ist nicht eindeutig vorgegeben. Wenn die Informationsmenge groß ist, werden die Informationen in mehrere Pakete verpackt und auf die Reise geschickt. Die Pakete werden mit der Empfängeradresse versehen und können auf unterschiedlichen Wegen zum Kommunikationspartner übertragen werden. Das Protokoll TCP/IP ist verantwortlich dafür, dass die Pakete komplett beim Empfänger eintreffen und die Reihenfolge der Pakete geordnet wird.

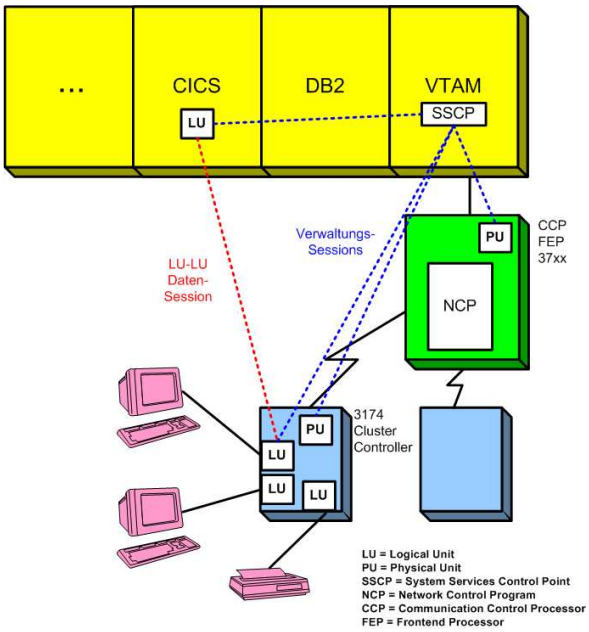
9.3 Die System Network Architecture (SNA)

Die klassische SNA-Konfiguration

Die klassische SNA-Architektur ist hierarchisch aufgebaut und hat mit dem *System Services Control Point* (SSCP) eine zentrale Steuerungskomponente. Der Adressraum, in dem diese Steuerzentrale eingerichtet ist, heißt *Virtual Telecommunication Access Method* (VTAM) und kann in einem Betriebssystem nur einmal vorhanden sein.

Der Eintrittspunkt in ein SNA-Netz ist eine so genannte *Logical Unit* (LU), in der klassischen Konfiguration auf der einen Seite ein Endgerät (im Allgemeinen ein Terminal oder ein Drucker) und auf der anderen Seite eine Anwendung. Für die Kommunikationssteuerung in den Zwischenknoten gibt es so genannte Physical Units, die sich unter anderem um die Leitungssteuerung kümmern.

Abbildung 9.2: Die SNA-Konfiguration



Das VTAM kommuniziert über eine Kanalschnittstelle mit einem Vorrechner, der als *Communication Control Processor* (CCP) oder manchmal auch als Front End Processor bezeichnet wird. In diesem Vorrechner läuft das *Network Control Program* (NCP), das die Kontrolle und Steuerung der Verbindungen übernimmt. An den CCP wiederum sind lokal oder remote Cluster Controller angeschlossen, an diesen dann wiederum die Endgeräte.

Das meistgenutzte Endgerät war in solch einer Konfiguration ein „dummes" Terminal. Das Attribut „dumm" bezieht sich darauf, dass in dem Gerät keine eigene Verarbeitung geschieht.

Das 3270-Protokoll

Zwischen einem Terminal und einem Endgerät wird für die Datenübertragung das so genannte 3270-Protokoll eingesetzt. Dies ist ein zeichenorientiertes Protokoll, das für die Kommunikation zwischen einem dummen Terminal und einer 3270-

Anwendung auf dem Mainframe sorgt. In der Praxis findet man heute jedoch kaum mehr dumme Terminals vor, sondern intelligente Endgeräte wie PCs sowie UNIX oder Linux Workstations.

Damit über diese Endgeräte mit 3270-Anwendungen gearbeitet werden kann, wird eine Terminal-Emulation benötigt, die ein dummes Terminal simuliert. Es gibt diverse Produkte am Markt, die dafür eingesetzt werden können. Diese müssen auf dem Endgerät installiert werden, da die entsprechende Software dort in der Regel nicht vorinstalliert ist, wie man das bei einer Telnet Session im Allgemeinen erwarten kann. Eine Alternative dazu ist das Produkt *Host on Demand* von IBM (es gibt auch hier Fremdprodukte am Markt), das die 3270-Schnittstelle über einen Internet Browser zur Verfügung stellt; dies hat den großen Vorteil, dass dann auf der Endgeräte-Seite keine Software mehr installiert werden muss.

LU 6.2 und APPC

Die oben dargestellte klassische SNA-Konfiguration gibt es heute nur noch sehr selten. In moderneren Konfigurationen werden beispielsweise Cluster Controller in LANs emuliert und LANs wiederum direkt über einen *Open Systems Adapter* (OSA) an den Mainframe angeschlossen. Die Endgeräte sind nicht mehr dumme Terminals, sondern PCs, UNIX-Workstations und andere Rechner.

Das bedeutet, dass eine Logical Unit mit Intelligenz versehen sein kann und in der Lage ist, Programme abzuarbeiten. Wenn man sich vorstellt, dass die Logical Unit im „Außenbereich" kein dummes Terminal mehr ist, sondern ein Rechner mit Verarbeitungsmöglichkeiten, ist der Schritt nicht mehr weit zu einem Kommunikationsmodell, wo beide Kommunikationspartner durch jeweils eine Anwendung repräsentiert werden. Dieses Kommunikationsmodell wird durch einen speziellen Typ von Logical Unit umgesetzt, der als LU 6.2 bezeichnet wird. Die Technik, die darauf aufsetzt, hat den Namen *Advanced Program to Program Communication* (APPC) und bietet in SNA-Umgebungen die Möglichkeit, eine Client/Server-Umgebung mit API-Schnittstellen aufzubauen. Das entsprechende Pendant in einer TCP/IP-Umgebung ist die Socket-Kommunikation. Darauf kommen wir im weiteren Verlauf noch zurück.

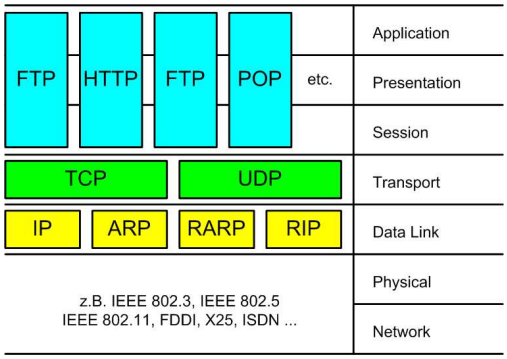
9.4 TCP/IP-Architektur

DoD-Protokolle

Im engeren Sinne bezeichnet TCP/IP zwei Protokolle, nämlich das *Transmission Control Protocol* (TCP) auf der OSI-Schicht 4 und das *Internet Protocol* (IP) auf der OSI-Schicht 3. Im weiteren und zumeist gebrauchten Zusammenhang meint man

eine große Protokollfamilie einschließlich der Anwendungsprotokolle, zu denen unter anderem auch HTTP gehört, das wichtigste Protokoll im World Wide Web. Die gesamte Protokollfamilie wird, wie schon erwähnt, oft auch als „DoD-Protokolle" bezeichnet.

Abbildung 9.3: TCP/IP Stack



TCP/IP-Schichten

TCP/IP ist eine Architektur, die aus vier Schichten aufgebaut ist. Die unterste Schicht, die den OSI-Ebenen 1 und 2 entspricht, wird als *Network-Access-Schicht* bezeichnet und ist nicht in TCP/IP selbst definiert. Vielmehr werden lediglich die Schnittstellen zu allen auf diesen Schichten verbreiteten Protokollen festgelegt. Damit werden die Vorteile dieser Schichten-Ausrichtung deutlich. Was bereits vorhanden ist, muss nicht neu erfunden werden. Über die Schnittstellen werden die notwendigen Funktionalitäten zusammengebracht. So funktioniert TCP/IP beispielsweise über ein Ethernet LAN, ohne dass dieses in TCP/IP selbst definiert werden muss.

Der *Internetwork Layer* entspricht der OSI-Schicht 3. Auf dieser Ebene arbeitet das *Address Resolution Protocol* (ARP), das *Reverse Address Resolution Protocol* (RARP), das *Internet Protocol* (IP) und das *Internet Control Message Protocol* (ICMP). Daneben gibt es auf dieser Ebene noch einige Routing-Protokolle wie beispielsweise das *Routing Information Protocol* oder das *Open Shortest Path First* (OSPF) Protocol.

Auf der Transport-Schicht, die auch in TCP/IP als *Transport Layer* bezeichnet wird, gibt es das Transmission Control Protocol (TCP) als verbindungsorientiertes und das User Datagram Protocol (UDP) als verbindungsloses Protokoll.

Auf der Ebene der Anwendungsprotokolle gibt es in Verbindung mit TCP/IP eine Vielzahl an Protokollen. Am häufigsten kommen Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), Post Office Protocol (POP3), Simple Network Management Protocol (SNMP) und das Telnet-Protokoll zum Einsatz.

Sockets und TCP/IP

Sockets sind eine Kombination von IP-Adresse, Protokoll (TCP oder UDP) und Portnummer. Die Portnummer ist vor allem dann wichtig, wenn nicht die Standard-Portnummer (das ist beispielsweise die Nummer 80 für einen Webserver) verwendet wird. Mit der IP-Adresse wird festgelegt, mit welchem Rechner kommuniziert werden soll, und über die Portnummer wird dafür gesorgt, dass die Daten an den richtigen Prozess übergeben werden. Die Anwendungen kommunizieren mit den Transportschichten fast immer entweder über das TCP-Protokoll, wenn eine gesicherte und verbindungsorientierte Kommunikation gewünscht wird, und über UDP, wenn eine verbindungslose Kommunikation gewünscht wird.

Bei den Definitionen im Services-File (**/etc/services**) ist erkennbar, dass beispielsweise das Protokoll **daytime** über den Port 13 entweder mit TCP oder mit UDP aufgebaut werden kann. Hierbei handelt es sich um jeweils unterschiedliche Sockets.

Socket-Kommunikation

Funktionell soll dasselbe erreicht werden wie mit APPC/LU 6.2 in einer SNA-Umgebung; d. h. es geht um Program-to-Program-Kommunikation, im Falle der Socket-Kommunikation in einer TCP/IP-Umgebung.

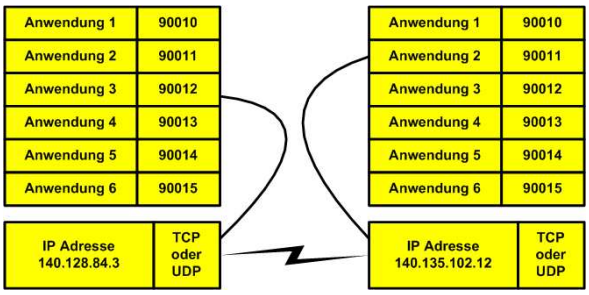


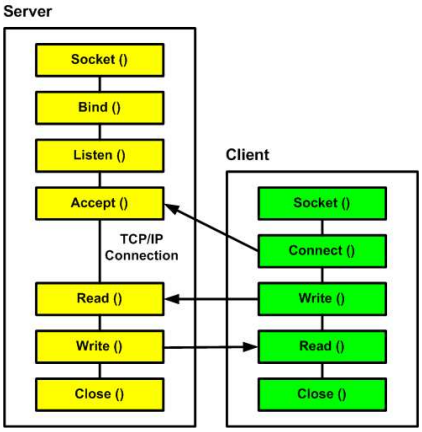
Abbildung 9.4: Überblick: Socket-Kommunikation

Das wichtigste der drei bei einer Socket-Kommunikation beteiligten Elemente (IP-Adresse, Protokoll, Port) ist die Portnummer, welche die Verbindung zu einem Prozess (einer Anwendung, einem Programm, einem Server) herstellt.

In TCP/IP-Umgebungen sind zahlreiche Standard-Ports (*Well-known Ports*) definiert, die als reserviert betrachtet werden sollen. Der Vorteil dieser Standard-Ports ist in erster Linie, dass Clients, die über das entsprechende Protokoll kommunizieren, mit der Standard-Portnummer vorkonfiguriert werden können, so dass die Portnummer beim Aufbau einer Verbindung nicht mitgegeben werden muss. Werden vom Standard abweichende Nummern verwendet, muss die jeweils genutzte Nummer beim Aufbau einer Verbindung spezifiziert werden.

Beispiele für Standard-Ports sind **20/21** (FTP), **23** (Telnet), **80** (HTTP, Webserver) oder **443** (HTTPS, SSL/Webserver).

Abbildung 9.5: Beispiel für eine Kommunikation über Sockets



Ob es sich hierbei um eine Telnet-, FTP-, HTTP- oder sonstige Verbindung handelt, spielt keine Rolle. Das Prinzip ist immer dasselbe.

9.5 Der Mainframe am Netz

Damit der Mainframe über ein Netzwerk mit der Außenwelt kommunizieren kann, muss er erst einmal physisch an dieses Netz angeschlossen werden. Die heute in der Praxis am meisten vorzufindenden Netzwerke sind *Local Area Networks* (LANs) und *Wide Area Networks* (WANs).

LANs findet man in räumlich begrenzten Umgebungen (Gebäude, Firmengelände etc.) vor, während WANs über größere Entfernungen eingesetzt werden, wobei in der Regel öffentliche Kommunikationseinrichtungen genutzt werden.

Der Mainframe unterstützt zahlreiche Netzgeräte. Die wichtigsten sind:

- 3172 LAN Channel Station (LCS)
- Channel-to-Channel (CTC)
- Open System Adapter (OSA)

Vor allem der Open System Adapter spielt in modernen Netzwerken eine große Rolle, da damit beispielsweise direkte Verbindungen in LAN-Umgebungen eingerichtet werden können, ohne dass Vorrechner oder Terminal Controller benötigt werden.

Die Services, die für ein Netzwerk-Management des Mainframe notwendig sind, werden in dem eNetwork Communications Server zusammengefasst. Die Anwendungsdaten können über die SNA-Architektur, über TCP/IP oder auch über eine Kombination der beiden Technologien erreicht werden.

Virtual Telecommunication Access Method (VTAM)

Die *Virtual Telecommunication Access Method* (VTAM) bildet die zentrale Steuerung für alle Netzwerk-Aktivitäten.

VTAM wird als Adressraum von der MVS-Konsole aus als Started Task gestartet. Die entsprechende Startprozedur befindet sich in der SYS1.PROCLIB. Über eine VTAMLST DD-Anweisung werden die Netzwerk-Definitionen eingelesen, und über eine VTAMLIB DD-Anweisung wird eine Bibliothek mit VTAM-spezifischen Lademodulen zugewiesen.

Beim Start sucht VTAM zunächst nach einem Member ATCSTRxx in der VTAMLST-Bibliothek, da sich in diesem Member Startparameter für VTAM befinden sowie ein Verweis über den CONFIG-Parameter auf die Startup-Liste (ATCCONxx) der Major Nodes (Hauptknoten), die gestartet werden sollen.

Typischerweise beinhaltet dieses Member meist zumindest einen Eintrag für den TSO Application Major Node und den Local 3270 Terminal Major Node. Weitere Einträge hängen beispielsweise davon ab, ob CICS verwendet wird oder ob ein Vorrechner mit einem Network Control Program (NCP) eingesetzt wird, etc.

Einige Spezialitäten, die mit dem Mainframe zusammenhängen, sollen hier erwähnt werden.

Telnet

In einer modernen Mainframe-Umgebung arbeiten oft die unterschiedlichsten Leute zusammen. Im Entwicklungsbereich arbeiten beispielsweise Entwickler an Transaktionsprogrammen, die in COBOL mit CICS APIs geschrieben werden. Diese arbeiten in der Regel mit dem ISPF-Editor, der eine 3270-Schnittstelle voraussetzt. Auf der anderen Seite gibt es im gleichen Unternehmen Entwickler, die mit C, C++ oder Java arbeiten. Die Entwicklung selbst findet zwar in der Regel meist auf einer Arbeitsstation mit einer integrierten Entwicklungsumgebung statt. Im Falle von E-Business-Anwendungen mit Java könnte dies beispielsweise der WebSphere Studio Application Developer sein. Oft müssen die Entwickler irgendwelche Anpassungen machen und Konfigurations-Files editieren. Viele dieser Entwickler kennen die Mainframe-Umgebung nicht und haben noch nie mit TSO und ISPF gearbeitet. Dafür kennen sie beispielsweise UNIX und den vi-Editor in- und auswendig.

Es ist deshalb meistens notwendig und sinnvoll, in Verbindung mit Telnet auf dem Mainframe zwei Port-Adressen zur Verfügung zu stellen, da in den meisten Umge-

bungen eine Telnet Session mit Telnet 3270 (TN3270) für die Arbeit mit TSO und ISPF und alternativ eine UNIX-spezifische Telnet-Session für den Zugang zu den UNIX System Services (USS) benötigt wird. Der Standard-Port 23 steht jedoch nur ein einziges Mal zur Verfügung. Man muss sich deshalb überlegen, welcher der beiden Alternativen der Standard-Port zugeordnet wird.

Weist man beispielsweise der TN3270 den Standard-Port 23 zu, so muss dem UNIX-Telnet für den Zugang zu USS ein anderer Port, beispielsweise der Port 10023 zugewiesen werden. In **/etc/services** von UNIX System Services und im TCP/IP Profile Dataset gibt es dann einen Eintrag **telnet** mit der Port-Zuweisung 10023 für den USS-Zugang und einen Eintrag **otelnet** mit der Port-Zuweisung 23 für den TN3270-Zugang. Das „o" im Eintrag **otelnet** steht für *Open Edition*, den Namen für die UNIX-Schnittstelle im MVS, bevor es dann zu UNIX System Services umbenannt wurde.

FTP

Mit FTP werden Files zwischen Rechnern übertragen. Hierbei müssen im Falle von Textübertragungen oft Code-Umwandlungen vorgenommen werden, da Daten auf dem Mainframe im EBCDIC-Code und Daten auf dezentralen Systemen (UNIX, Windows) im ASCII-Code abgespeichert werden. Außerdem kann FTP in Verbindung mit der Resource Access Control Facility (RACF) Zugriffskontrollen vornehmen.

NFS

In heterogenen Umgebungen wird oft das *Network Files System* (NFS) eingesetzt, um Filesysteme über Netzwerk- und Plattformgrenzen auszudehnen. NFS arbeitet plattformunabhängig und ist unabhängig vom Transportprotokoll. Erreicht wird das durch die beiden Protokolle *Remote Procedure Call* (RPC) und *External Data Representation* (XDR). Gerade auf dem Mainframe ist der Einsatz von NFS eine interessante Option, da hier Daten in MVS-Dateien über NFS integriert werden können. Das bedeutet, dass ein Benutzer auf einem UNIX-System auf einen Zweig im hierarchischen Filesystem zugreift und – ohne dass er es merkt – mit Daten in einer MVS-Datei arbeitet.

TCP/IP-Definitionen im MVS

Für den Start von TCP/IP wird eine Prozedur in der **SYS1.PROCLIB** eingerichtet. Der Name der Started Task muss mit dem Namen **SUBFILESYSTYPE** im **BPXPRMxx**-Member der **SYS1.PARMLIB** übereinstimmen. Außerdem wird ein Parameter-Satz benötigt, der in der als Membernamen **TCPDATA** in einer Datei abgelegt wird, auf die in der Started Task mit dem **DD**-Namen **SYSTCPD** verwiesen wird.

Kapitel 10

Datenbanken auf dem Mainframe

10.1 Datenbankmodelle

Die Vorteile eines Datenbanksystems gegenüber konventionellen Dateisystemen können folgendermaßen zusammengefasst werden:

- Es gibt eine gemeinsame Datenbasis für alle Anwendungen.
- Die Konsistenz der Daten wird sichergestellt.
- Redundanz entfällt oder wird bewusst und kontrolliert eingesetzt.
- Die Anwendungsprogrammierung wird effizienter, da die Entwickler nur die relevanten logischen Eigenschaften der Daten, nicht jedoch deren Organisation und physische Ablage kennen müssen.
- Die Abhängigkeit zwischen Anwendung und Daten wird eliminiert. Das DBMS liefert die Daten in der Form, wie sie von der Anwendung benötigt werden. Ände-

rungen in der Datenorganisation führen in der Regel nicht zu Änderungen der Anwendungen.

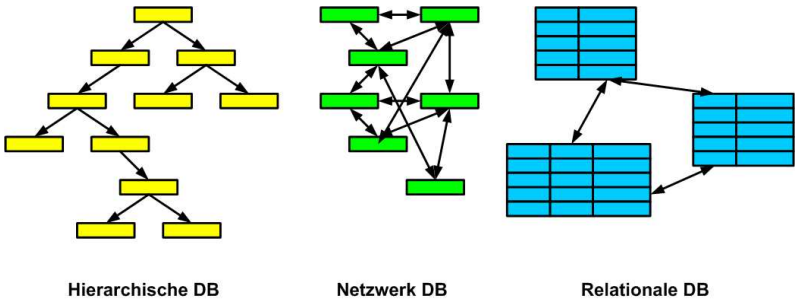
- Das Datenbanksystem bietet große Flexibilität für Datenauswertungen.
- Das Datenbanksystem kann zentral beispielsweise über „constraints" die Plausibilität von Daten überprüfen.
- Das Datenbanksystem kann zentral Mechanismen zur Wiederherstellung einer korrekten Datenbank nach dem Auftreten von Fehlern bereitstellen. Dieser automatisierbare Vorgang wird auch als „Recovery" bezeichnet.

Es werden drei verschiedene Datenbankmodelle unterschieden:

- Hierarchisches Modell
- Netzwerk-Modell
- Relationales Modell

Alle drei Modelle haben unterschiedliche Strukturen und Stärken. Von der Organisationsform her ist beispielsweise eine hierarchische Datenbank äußerst sparsam im Umgang mit Ressourcen, dafür mangelt es an Flexibilität. Eine relationale Datenbank dagegen ist ressourcenintensiver – vor allem, was den Hauptspeicher angeht –, bietet dafür jedoch die größte Flexibilität in den DB-Abfragen. In der Praxis findet man zumeist keine Reinform der jeweiligen Modelle vor, sondern Mischformen, die die Schwächen der Modelle ausgleichen sollen.

Abbildung 10.1: Datenbank-Modelle



Hierarchisches Modell

Die Beziehungen können durch eine Baumstruktur dargestellt werden. Jeder Datensatz kann nur einen Owner haben. Sind diese Strukturen in einer Anwendung entsprechend programmiert und abgebildet, wird auf diese Daten extrem schnell

und effizient zugegriffen. Dafür sind so genannte „Ad-hoc-Abfragen" mühsamer und zeitaufwändiger.

Ein typischer Vertreter dieses Modells ist das *Information Management System* (IMS) von IBM, das jedoch sehr an Bedeutung verloren hat, vor allem im Vergleich mit der relationalen Datenbank DB2. Neue Anwendungen werden in der Regel nicht mehr mit IMS erstellt. Deshalb wird hier auch nicht weiter auf IMS eingegangen.

Netzwerk-Modell

Bei einem Netzwerk-Modell kann ein Datensatz mehrere Owner haben. Diese Methode wird auch CODASYL-Datenbank genannt, weil sie aus einer *Data Base Task Group* (DTBG) der *Conference on Data System Languages* (CODASYL) hervorgegangen ist. Auch das Netzwerk-Modell spielt in der Praxis heute keine große Rolle mehr.

Relationales Modell

In einem relationalen Modell werden die Daten in zweidimensionalen Tabellen dargestellt. Diese Struktur bietet die größte Datenunabhängigkeit und Flexibilität in Design und Handhabung.

Das relationale Modell spielt heute in der Praxis die größte Rolle. Deshalb wird es hier noch etwas vertiefter dargestellt; danach wird der wichtigste Vertreter auf dem Mainframe, das DB2, behandelt.

10.2 Datenbank-Tabellen

Ein relationales Datenbank-Management-System (RDBMS) kommuniziert mit seinen Benutzern über zweidimensionale Tabellen.

Spalte (Column)

Eine Tabelle aus Spalten (oder auch Attributen). Jede Spalte hat eindeutige Eigenschaften. Das relationale Modell kennt eine Kennzeichnung für nicht bekannte Werte („Null-Wert" oder auch „Missing Value"). Es ist eine der Aufgaben des RDBMS, diese Werte unterschiedlich zu Nullen oder Leerstellen zu behandeln, da ein Nullwert beispielsweise bei der Formulierung von Auswahlbedingungen besonders behandelt werden muss.

Abbildung 10.2: DB2: Tabellenspalte

KUNDE

KNR	NAME	PLZ	ORT	STRASSE
37468	SCHULTZ	80805	MÜNCHEN	BERGSTRASSE 12
87635	LEHMANN	20097	HAMBURG	NORDERSTRASSE 101
46539	SEIBERT	50567	STUTTGART	ZETTACHRING 10
37469	KOHLER	55130	MAINZ	TETTELWEG 5
75392	MAIER	78337	ÖHNINGEN	WOLFERMOOS 12

Zeile (Row)

Eine Zeile (auch als *Tupel* bezeichnet) besteht aus einer eindeutigen Kombination von Spalteninhalten. Die Reihenfolge der Zeilen ist unwesentlich, da ein relationales System immer die Gesamtmenge einer Tabelle behandelt.

Abbildung 10.3: DB2: Tabellenzeile

KUNDE

KNR	NAME	PLZ	ORT	STRASSE
37468	SCHULTZ	80805	MÜNCHEN	BERGSTRASSE 12
87635	LEHMANN	20097	HAMBURG	NORDERSTRASSE 101
46539	SEIBERT	50567	STUTTGART	ZETTACHRING 10
37469	KOHLER	55130	MAINZ	TETTELWEG 5
75392	MAIER	78337	ÖHNINGEN	WOLFERMOOS 12

Datenwert (Value)

Der Schnittpunkt von Spalte und Zeile adressiert einen Datenwert. Wiederholgruppen sind unzulässig.

10.3 Structured Query Language (SQL)

Die Sprache zur Abfrage und Manipulation von Daten in einem relationalen DB-System heißt *Structured Query Language* (SQL). Mit ihr wird definiert, *was* gemacht werden soll, aber nicht, *wie* es gemacht werden soll. Die Sprache ist, wie der Name schon sagt, sehr stark strukturiert und kennt nur relativ wenige Befehle, die dann allerdings sehr viele Unterangaben und Parameter beinhalten können. Die Sprache ist eingeteilt in drei Befehlsgruppen (hier mit einigen Beispielen):

Data Definition Language (DDL)

CREATE: Anlegen von Objekten und Namensvergabe
ALTER: Verändern von Objekten
DROP: Löschen von Objekten

Data Manipulation Language (DML)

SELECT: Anzeige bzw. Auswahl von Daten
INSERT: Einfügen von Daten

UPDATE: Verändern von bestehenden Daten
DELETE: Löschen von Daten

Data Control Language (DCL)
GRANT: Vergabe von Privilegien
REVOKE: Entziehen von Privilegien

Beispielabfrage:

```
SELECT KNR, NAME, STADT
FROM KUNDE
WHERE PLZ=3
ORDER BY KNR
```

10.4 Das Datenbanksystem DB2

Historie von DB2

Die Ursprünge von DB2 gehen zurück auf die Arbeit von C. F. Codd im IBM-Labor San Jose und seine Abhandlung „A Relational Model of Data for Large Shared Data Banks" im Juni 1970. Mit einem Team wurde in den Folgejahren das *System/R* realisiert und eine Structured English Query Language eingeführt, aus der dann SQL hervorging. Aus dem System/R wurde später (1983) dann DB2.

Die Vorgängermodelle waren hierarchische Modelle oder Netzwerk-Modelle. Dies ist auch nicht weiter verwunderlich, da viele Daten, mit denen man täglich umgeht, hierarchisch geordnet sind. Denken Sie an File-Systeme, Organigramme, Gelbe Seiten etc. Die entsprechenden Daten wurden dann ebenfalls hierarchisch organisiert und in hierarchischen Datenbanken abgelegt.

Das relationale Modell ist einfacher und ermöglicht leistungsfähige Konstrukte für die Gewährleistung der Datenintegrität. Die Möglichkeit, die Datenstruktur einfach zu ändern, erhöht vor allem die Produktivität der Entwickler und der Administratoren. Die Flexibilität ist im Vergleich zu den anderen Modellen enorm gesteigert.

Eine erste DB2-Version wurde am 7. Juni 1983 angekündigt. Der Durchbruch gelang mit der Version 2.1 im Jahr 1988, als die referentielle Integrität unterstützt wurde und zahlreiche Verbesserungen bezüglich der Performance integriert wurden.

Die Version 5 kann dann als weiterer Meilenstein angesehen werden, da damit E-Business, Business Intelligence (Data Warehousing und Data Mining) und ERP-Systeme wie SAP und Peoplesoft unterstützt wurden.

In DB2 kann heute praktisch jegliche Art von elektronischen Informationen gespeichert werden. Neben den traditionellen relationalen Daten sind dies beispielsweise auch:

- Strukturierte und unstrukturierte Binärdaten
- Dokumente und Texte in beliebigen Sprachen
- Grafiken und Bilder
- Multimediadaten (Audio und Video) etc.

DB2 spielt auch eine immer wichtigere Rolle im E-Business-Software-Portfolio. Hinter dem Begriff E-Business steht die Idee, die wichtigsten Geschäftsprozesse in Internet-Prozesse zu transformieren. Wichtige Aspekte dabei sind:

- Enterprise Resource Planning (ERP)
- Customer Relationship Management (CRM)
- Supply Chain Management (SCM)

Im Mittelpunkt der E-Business-Anwendungen steht heute oft ein Portal als „Single Point of Access" bezüglich Informationen, Anwendungen, Geschäftsprozessen und Wissen. In diesem Zusammenhang müssen auch Internet-Standards wie HTML und XML unterstützt werden.

DB2 auf unterschiedlichen Plattformen

Hier geht es zwar um DB2 auf dem Mainframe. Da jedoch in einer typischen Umgebung heute zumeist mehrere Plattformen zusammenarbeiten, ist es sinnvoll, zu wissen, welche unterschiedlichen Produkte auf den diversen Plattformen zur Verfügung stehen.

DB2 Everyplace

DB2 Everyplace (DB2E) ist eine Mini-Datenbank, die lediglich ca. 100 KB groß ist. Sie ist gedacht für Klein- und Kleinstgeräte wie digitale Organizer, Mobiltelefone und andere so genannte „embedded devices". DB2E läuft beispielsweise auf dem Palm-Betriebssystem, unter Windows CE, dem EPOC-Betriebssystem und QNX Neutrino.

Die Idee dabei ist, dass relationale Daten auf diesen Kleingeräten mit Datenquellen auf anderen DB2-Systemen vom PC bis zum Mainframe synchronisiert werden können. In Zusammenarbeit mit dem Produkt Mobile Connect kann die Synchronisation auch mit einer beliebigen ODBC-konformen Datenbank wie beispielsweise Oracle oder MS Access erfolgen.

DB2 Satellite Edition

Die DB2 Satellite Edition (DB2 SE) ist dazu gedacht, Systeme zu unterstützen, die die meiste Zeit losgelöst von einem Unternehmenssystem arbeiten und sich von Zeit zu Zeit mit den Unternehmenssystemen verbinden, um Daten auszutauschen. Das Satellitensystem kümmert sich nicht um die Datenbankverwaltung. Die Benutzer können sich auf ihre Arbeit konzentrieren und müssen nicht einmal wissen, dass sie von einer Datenbank dabei unterstützt werden. Diese Version eignet sich somit für verteilte Systeme, die immer wieder repliziert werden müssen. Die DB2 Satellite Edition ist nur für Windows-Systeme (und künftig wahrscheinlich auch für Linux) verfügbar.

DB2 Personal Edition

Die DB2 Personal Edition (DB2 PE) ist ein voll funktionierendes DB-System, mit dem auf einer entsprechenden Arbeitsstation auch Datenbanken eingerichtet werden können. Es kann auch als entfernter Client für einen DB2-Server verwendet werden, da es auch die DB2-Client-Komponenten enthält. Das Produkt läuft unter Windows, OS/2 und Linux.

DB2 Workgroup Edition

Die DB2 Workgroup Edition (DB2 WE) ist für den Einsatz in LANs gedacht. Das Produkt unterstützt sowohl lokale als auch entfernte Clients. Für die Verbindung zu anderen Systemen werden die Protokolle TCP/IP, NetBIOS, IPX/SPX, Named Pipes und APPC unterstützt. Das Produkt ist für die Plattformen Windows, OS/2, AIX, Solaris, HP-UX und Linux verfügbar.

DB2 Enterprise Edition

Die DB2 Enterprise Edition (DB2 EE) ist eine skalierbare Version, die auch symmetrische Multiprozessoren und massiv-parallele Cluster unterstützt. Außerdem werden damit auch unstrukturierte Daten wie Bilddaten, Audio, Video etc. und XML mit objektrelationalen Fähigkeiten unterstützt. Das Produkt ist für die Plattformen Windows, OS/2, AIX, Solaris, HP-UX und Linux verfügbar.

DB2 Enterprise-Extended Edition

Die DB2 Enterprise-Extended Edition (DB2 EEE) ermöglicht die Aufteilung von Daten über Cluster-Konfigurationen (SMP und MPP) hinweg. Für die Benutzer sieht es so aus, als befinde sich die Datenbank auf einem einzigen System. Die SQL-Anweisungen werden parallelisiert, was zu deutlichen Geschwindigkeitsvorteilen führen kann. Das Produkt läuft auf den gleichen Plattformen wie die Enterprise Edition.

DB2 for z/OS

DB2 for z/OS ist als Unternehmens-DB-Server für höchste Ansprüche konzipiert. Es ist die Version, die am besten von allen skaliert und die höchste Verfügbarkeit aufweist – vor allem dann, wenn DB2 in einer Parallel-Sysplex-Umgebung eingesetzt wird.

Unterstützung von Standards

Offene Standards bilden die Grundlage für künftige E-Business-Anwendungen. Nur in Verbindung mit Standards ist es möglich, Anwendungsprogramme zu entwickeln, die ohne oder zumindest mit nur geringen Modifikationen in unterschiedlichen Datenbank-Umgebungen laufen können.

Wichtige Standards in diesem Zusammenhang sind:

- Structured Query Language (SQL99, ursprünglich mit SQL3 bezeichnet)
- Java Data Base Connectivity (JDBC)
- Structured Query Language for Java (SQLJ)

Gehen wir auf die wichtigsten Aspekte noch etwas näher ein:

SQL-Standard (SQL99)

Ein DBMS verfügt über eine Datenbankabfragesprache. DB2 verwendet hierfür wie fast alle relationalen Datenbanken die Structured Query Language (SQL). SQL ist heute weitgehend standardisiert.

Der SQL99 ANSI/ISO-Standard erweitert bzw. ersetzt den SQL92-Standard. Erweiterungen betreffen beispielsweise objektrelationale Fähigkeiten, Trigger und viele eingebaute Funktionen für die Datenanalyse.

SQLJ und JDBC

JDBC und SQLJ sind die beiden wichtigsten Standards, um Datenbanken aus Java heraus anzusprechen und somit E-Business-Anwendungen zu ermöglichen.

JDBC basiert auf dem SQL Call Level Interface (SQL/CLI), das über ISO/IEC 9075-3:1999 standardisiert ist. Es handelt sich dabei um ein Application Programming Interface für Java-Anwendungen, das dynamisches SQL für den Zugriff auf relationale Datenbanken verwendet.

Die SQLJ-Spezifikation besteht aus folgenden Teilen: Teil 0 definiert die SQLJ-Sprachsyntax und die Semantik für eingebettete (embedded) SQL in Java-Anwendungen. Das American National Standards Institute (ANSI) definiert diesen Teil im Dokument ANSI X3.135.10:1998. Teil 2 spezifiziert Erweiterungen wie die Integration von Java-Klassen in einer SQL-Datenbank und den Aufruf von statischen Java-Methoden als SQL Stored Procedures oder Funktionen. ANSI definiert diesen Teil im Dokument ANSI NCITS 3311.1:1999.

DB2 for z/OS unterstützt derzeit SQLJ Teil 0 und ein Subset aus Teil 2.

DB2-Strukturen

Hauptkomponenten von DB2 for z/OS sind die Datenbank-Services, die *System Services*, der *Internal Resource Lock Manager* (IRML) und die *Distributed Data Facility* (DDF).

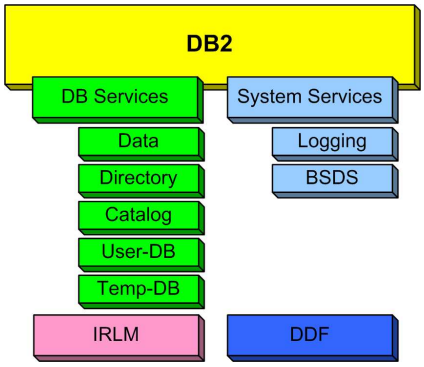


Abbildung 10.4: Hauptkomponenten von DB2

Die Elemente, die DB2 verwaltet, können in zwei Kategorien eingeteilt werden: Datenstrukturen, die von Endbenutzern genutzt und für die Organisation der Benutzerdaten verwendet werden, sowie Systemstrukturen, die von DB2 selbst verwaltet und genutzt werden.

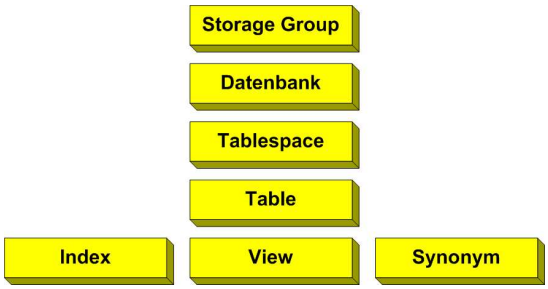


Abbildung 10.5: Überblick über die DB2-Strukturen

DB2-Datenstrukturen

Tabellen

Tabellen sind logische Strukturen, die von DB2 verwaltet werden. Tabellen setzen sich aus Spalten (Columns) und Zeilen (Rows) zusammen. Die Zeilen müssen sich an keine feste Reihenfolge halten. Die Spalten dagegen sind in der Reihenfolge angeordnet, in der sie bei der Tabellendefinition spezifiziert werden.

Am Schnittpunkt zwischen Spalte und Zeile findet man in der Tabelle einen Wert (Value). Eine Spalte ist ein Satz von Werten mit jeweils demselben Datentyp. Jede Tabelle muss mindestens eine Spalte haben. Die Anzahl Zeilen kann auch Null sein.

DB2 greift auf die Daten zu, indem ein Bezug auf den Inhalt und nicht auf die Lokation oder Organisation auf dem Speichermedium geschaffen wird.

DB2 unterstützt unterschiedliche Arten von Tabellen:

Basistabelle (Base Table)
: Eine Basistabelle ist eine Tabelle, die mit der SQL-Anweisung CREATE TABLE erzeugt wird und persistente Benutzerdaten enthält.

Temporäre Tabelle (Temporary Table)
: Eine temporäre Tabelle ist eine Tabelle, die mit der SQL-Anweisung CREATE GLOBAL TEMPORARY TABLE oder DECLARE GLOBAL TEMPORARY TABLE erzeugt wird. Temporäre Tabellen werden für die Speicherung von Daten verwendet, um SQL-Zwischenresultate zu verwalten. Beispielsweise kann eine temporäre Tabelle verwendet werden, um ein großes Result Set zwischenzuspeichern, das anschließend sortiert wird. Eine temporäre Tabelle existiert nur so lange wie der Anwendungsprozess, der sie erzeugt hat, und wird auch nur innerhalb des Anwendungsprozesses verwendet.

Ergebnistabelle (Result Table)
: Eine Tabelle, die entsteht, wenn mit einer SQL-Anweisung eine Tabelle in einer Datenbank abgefragt wird, enthält ein so genanntes Result Set und wird als Result Table bezeichnet. Im Gegensatz zu einer Basistabelle oder einer temporären Tabelle wird eine Result Table nicht mit einer CREATE-Anweisung erzeugt.

Views

Ein View ist eine alternative Sichtweise auf die Daten in einer oder mehreren Tabellen. Es handelt sich dabei um eine Spezifikation einer Result Table, die mit einem Namen versehen ist.

Es können beispielsweise Views erzeugt werden, um Daten aus unterschiedlichen Basistabellen zu kombinieren. Ein View wird mit der SQL-Anweisung CREATE VIEW erzeugt. Die Spezifikation eines Views in anderen SQL-Anweisungen hat den gleichen Effekt wie eine SQL-SELECT-Anweisung.

Views können für unterschiedliche Zwecke definiert werden:

- Zugriffssteuerung auf eine Tabelle und einfachere Verwendung
- Autorisierungskontrolle, indem der Zugriff auf einen View, aber nicht der Zugriff auf die Basistabelle gewährt wird

- Sichtbarmachen eines Teils einer Tabelle
- Anzeige einer Zusammenfassung von Daten aus einer Tabelle
- Kombinieren von Tabellen auf sinnbringende Art und Weise

Table Spaces

Sämtliche Tabellen werden in *Table Spaces* verwaltet. Table Spaces sind Speicherstrukturen in Form von Dateien, in denen eine oder mehrere Tabellen verwaltet werden. Technisch gesehen handelt es sich dabei um einen VSAM/LDS Cluster. Die Größe von Table Spaces ist nur durch DB2- bzw. VSAM-Einschänkungen limitiert. Derzeit ist die maximale Größe 64 GB. Die interne Organisation geschieht in 4 KB oder 32 KB Pages, die in einem bzw. in acht VSAM CIs abgelegt werden. Bei der Wahl der Page-Größe muss die maximale Zeilenlänge berücksichtigt werden, da sich eine Zeile nicht über mehrere Pages verteilen kann. So etwas wie „Spanned Records", wie man sie von VSAM kennt, gibt es in DB2 nicht. Es werden segmentierte und partitionierte Table Spaces unterschieden.

Segmentierte Table Spaces
: Ein segmentierter Table Space ist ein Table Space, der mehrere Tabellen enthalten kann. Er ist aus Gruppen von Seiten (Pages) zusammengesetzt, die als Segmente bezeichnet werden.

Partitionierte Table Spaces
: Ein partitionierter Table Space kann nur eine einzige Tabelle enthalten. Er wird unterteilt in separate Speichereinheiten, die als Partitions bezeichnet werden. Jede Partition gehört zu einem Bereich von Schlüsselwerten. Jede Partition kann separat von anderen Partitions von Utilities und mit SQL-Anweisungen gleichzeitig bearbeitet werden. Über einen Partitionsindex wird gesteuert, welche Daten in welche Partition kommen.

Index Spaces

Ein Index Space ist ein weiteres Speicherkonstrukt und enthält einen Index. Wenn ein Index mit der CREATE INDEX-Anweisung erzeugt wird, wird dieser Index automatisch in der gleichen Datenbank wie die Tabelle definiert.

Datenbanken

In DB2 ist eine Datenbank ein Satz von Table Spaces und Index Spaces. Eine Datenbank wird mit der CREATE DATABASE-Anweisung erzeugt. Wann immer ein Table Space erzeugt wird, wird er explizit oder implizit einer existierenden Datenbank zugewiesen.

Storage Groups

Eine Storage Group bildet die Einheit an physikalischem Plattenspeicher, die an DB2 zur Verwaltung übergeben wird. Eine Storage Group kann sich über mehrere Plattengeräte erstrecken und wird mit dem SQL-Befehl CREATE STOGROUP erzeugt. Allerdings kann eine Storage Group nur auf Platten desselben Gerätetyps angelegt werden.

DB2-Systemstrukturen

DB2 verfügt über umfassende Infrastrukturen für Integrität und Performance sowie für das Recovery von Daten. Im Gegensatz zu den Datenstrukturen, die von Benutzern erzeugt und verwendet werden, steuert DB2 den Inhalt und den Zugriff auf die Systemstrukturen selbst. Die wichtigsten Systemstrukturen, auf die hier näher eingegangen wird, sind: der Systemkatalog, die Aktiv- und Archiv-Logs, das Bootstrap Dataset und die Buffer Pools

Der Systemkatalog (Data Dictionary)

Die Drei-Ebenen-Architektur eines DB-Systems ermöglicht es, zwischen der konzeptionellen Gesamtsicht einer Datenbank, der internen Sicht und den unterschiedlichen externen Benutzersichten zu unterscheiden.

Die Informationen dieser Ebenen werden jeweils in eigenen Schemata modelliert und zusammen in einem ***Data Dictionary*** (Systemkatalog) abgelegt und verwaltet. Ein DBMS verfügt dazu über einen Dictionary Manager. Hierbei handelt es sich um ein Programm, welches den Katalog in einer bestimmten Weise organisiert und gegebenenfalls Informationen anderen Komponenten des DBMS, den Anwenderprogrammen oder den Benutzern zur Verfügung stellt. Bei diesen Informationen handelt es sich um so genannte ***Metadaten***, das sind Daten über die Daten, die in einem DBMS abgespeichert sind.

Zu diesen Metadaten gehören: zeitinvariante Daten über die Schemata einer Datenbank; nur teilweise zeitinvariante Daten wie Indizes, Zuordnung von Tabellen zu Table Spaces, Views und Zugriffsrechte; zeitvariante Informationen wie Statistiken als Grundlage für den Optimizer.

Der Katalog wird wie jede vom System verwaltete Datenbank selbst wieder in Tabellen organisiert. Die Tabellen sind mit dem DB-Namen SYSIBM vordefiniert und werden vom System selbst automatisch oder auf Veranlassung durch Benutzer bzw. den Datenbankadministrator (DBA) gepflegt.

Die Einträge in diesen Tabellen werden ebenfalls automatisch verwaltet, z. B. dann, wenn ein Benutzer eine neue Tabelle, einen Index oder einen View erzeugt, und können mit SELECT-Anweisungen abgefragt werden. Jeder Benutzer kann den Ka-

talog abfragen (SELECT). Ein DBA hat zusätzlich auch Berechtigungen für INSERT, DELETE, UPDATE und ALTER.

Aktiv- und Archiv-Logs

DB2 zeichnet alle Datenänderungen und andere signifikante Events in einem Log auf. DB2 kann diese Änderungen in einem Fehlerfall dann anhand des Logs nachfahren oder auch auf einen bestimmten Punkt zurücksetzen.

DB2 schreibt jeden Log-Datensatz auf eine Datei auf Platte. Dies ist das Aktiv-Log. Wenn dieses voll wird, kopiert DB2 den Inhalt des Aktiv-Logs auf eine Platten- oder Banddatei. Hierbei handelt es sich dann um ein Archiv-Log.

Bootstrap Dataset

Das Bootstrap Dataset (BSDS) enthält wichtige Informationen für DB2, wie beispielsweise den Log-Namen. DB2 nutzt diese Informationen im BSDS für System-Restarts und für andere Aktivitäten, für die die Logfiles benötigt werden.

Buffer Pools

Buffer Pools sind virtuelle Speicherbereiche, in denen DB2 temporär Seiten von Table Spaces oder Index Spaces ablegt. Wenn ein Anwendungsprogramm auf eine Tabellenzeile zugreift, wird die Seite mit dieser Zeile in einem Buffer abgelegt. Wenn sich die benötigten Daten bereits in einem Buffer befinden, geht der Zugriff natürlich deutlich schneller, als wenn die Seite von einem Plattengerät geholt werden muss. In Verbindung mit Parallel Sysplex werden Group Buffer Pools verwendet.

Packages und Plans

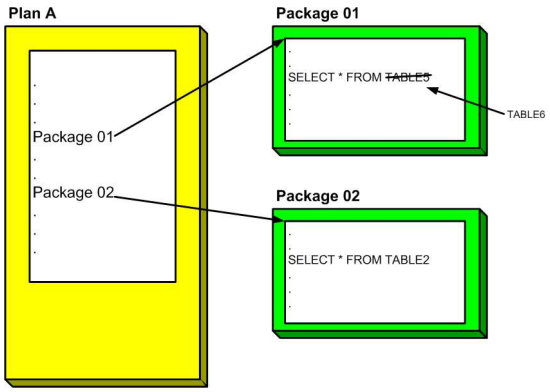
Ein *Package* enthält Kontrollstrukturen, die DB2 nutzt, wenn SQL-Anweisungen ausgeführt werden. Packages werden bei der Programmvorbereitung erzeugt. Diese Kontrollstrukturen sind so etwas wie „gebundene" oder ausführbare Formen für SQL-Anweisungen.

Ein *Application Plan* bezieht sich auf einen Anwendungsprozess auf eine lokale Instanz von DB2, spezifiziert Verarbeitungsoptionen und enthält entweder eine Liste von Package-Namen oder eine gebundene Form der SQL-Anweisungen.

Jede DB2-Anwendung benötigt einen Application Plan. Packages machen Anwendungsprogramme flexibler und erleichtern die Wartung. Wenn Packages verwendet werden, muss beispielsweise nicht jedes Mal ein kompletter Plan neu gebunden

werden, wenn nur eine einzelne SQL-Anweisung geändert wird. Wenn beispielsweise wie in Abbildung 10.6 die Tabelle im Package 01 geändert wird, muss nur das Package 01 neu gebunden werden.

Abbildung 10.6: DB2: Packages und Plans



Verteilte Daten

Vielfach müssen Daten in unterschiedlichen Quellen und an unterschiedlichen Lokationen verwaltet werden. Eine verteilte Umgebung schafft mehr Flexibilität und wird benötigt, um Ressourcen in unterschiedlichen Lokationen zusammenzubringen. Von verteilten Daten spricht man, wenn Daten in einem anderen DBMS angesprochen werden sollen.

Der wichtigste Standard für die Verbindung unterschiedlicher DB-Systemen ist die *Distributed Relational Database Architecture* (DRDA), die inzwischen auch von der Open Group abgesegnet ist.

Integration von DB2 in z/OS

DB2 wird als formales Subsystem von z/OS betrieben. Prozesse unter z/OS werden in separaten Regionen (Regions) aufgebaut, die als Adressräume (Address Spaces) bezeichnet werden. DB2 ist auf mehrere unterschiedliche Adressräume verteilt.

DB2 arbeitet mit anderen z/OS-Subsystemen und -Komponenten zusammen. Beispiele dafür sind der z/OS Security Server und Parallel Sysplex.

DB2 Utilities werden meist über Batchjobs gestartet. Anwendungen, die auf DB2-Ressourcen zugreifen, können auf dem gleichen z/OS-System oder auch auf anderen Betriebssystemen laufen. Werden sie in anderen Systemen betrieben, wird auf die DB2-Ressourcen mit Hilfe der Distributed Data Facility (DDF) zugegriffen. Mit Hilfe so genannter Attachment Facilities wird die Connectivity hergestellt.

DB2 und der Security Server

Um die Zugriffe auf z/OS-Ressourcen zu kontrollieren, wird der Secure Way Security Server for z/OS oder ein äquivalentes Produkt verwendet. Wenn eine Session aufgebaut wird, überprüft der Security Server die Identität des Benutzers, um einen nicht autorisierten Zugang zu verhindern. Außerdem können mit RACF auch DB2-Ressourcen geschützt werden.

Attachment Facilities von DB2

Die Attachment Facilities steuern, wie eine DB2 Session aufgebaut wird. Sie stellen eine Schnittstelle zwischen DB2 und anderen Umgebungen dar. Für z/OS-Umgebungen sind dies beispielsweise CICS, IMS, TSO, CAF und RRS.

CICS Attachment Facility

CICS (Customer Information Control System) ist ein Anwendungsserver, der sich um die Transaktionsverwaltung von Anwendungen kümmert. In einer CICS-Umgebung kann die CICS Attachment Facility für den Zugriff auf DB2 genutzt werden. Nach dem Start von DB2 kann DB2 von einem CICS Terminal aus betrieben werden.

CICS und DB2 können unabhängig voneinander gestartet und gestoppt werden, und es kann sowohl auf DB2-Daten als auch auf CICS-Daten zugegriffen werden. Über eine Option können Verbindungen zwischen CICS und DB2 automatisch aufgebaut werden. Im Falle von Fehlersituationen koordiniert CICS das Recovery sowohl für CICS-Daten als auch für DB2-Daten.

IMS (Information Management System)

IMS ist ein Datenbanksystem. IMS umfasst einen IMS-hierarchischen Datenbankmanager, einen IMS-Transaktionsmanager und Datenbank-Middleware-Produkte für den Zugriff auf IMS-Datenbanken und Transaktionen. In einer IMS-Umgebung kann die IMS Attachment Facility für Zugriffe auf DB2 genutzt werden. Die IMS Attachment Facility nimmt Requests entgegen und setzt diese für den Zugriff auf DB2 mit Hilfe von Exit-Routinen um, die Teil des IMS-Subsystems sind. IMS verbindet sich normalerweise automatisch, d. h. ohne Operator-Eingriff, mit dem DB2-System. Im Falle von Fehlersituationen koordiniert IMS das Recovery sowohl für IMS- als auch für DB2-Daten.

TSO (Time Sharing Option)

TSO ist ein interaktives System, das den Benutzern den Rechner zeitscheibenweise zur Verfügung stellt. In TSO- wie auch in Batch-Umgebungen können die Call Attachment Facility (CAF) und die Resource Recovery Services

(RRS) für Zugriffe auf DB2 verwendet werden. TSO ermöglicht es autorisierten DB2-Benutzern, Datenbanken und Anwendungsprogramme zu erzeugen, zu modifizieren und zu verwalten. Mit der TSO Attachment Facility kann auf DB2 sowohl interaktiv als auch per Batch zugegriffen werden. Interaktiv wird der Zugriff über eine Terminal-Schnittstelle gemacht, im Falle von Batchjobs wird das Terminal Monitor Program (TMP, d. h. IKJEFT01) aufgerufen, und über SYSTSIN wird der Input übergeben.

Folgende zwei Kommandoprozessoren stehen zur Verfügung:

- Der DSN-Kommandoprozessor läuft als TSO-Kommandoprozessor und nutzt die TSO Attachment Facility.
- DB2 Interactive (DB2I) ist eine ISPF-Anwendung, über die letztendlich wieder der DSN-Kommandoprozessor aufgerufen wird. Über entsprechende Panels ist diese Benutzerschnittstelle allerdings einfacher zu bedienen.

Die Call Attachment Facility ist eine Alternative für TSO- und Batch-Anwendungen, die enge Anbindungen an eine Session-Umgebung benötigen. Der Zustand der Verbindungen wird über spezielle Connection-Funktionen gesteuert, die CAF zur Verfügung stellt.

Distributed Data Facility (DDF)

Die *Distributed Data Facility* (DDF) ermöglicht Client-Anwendungen, die in einer Umgebung laufen, die DRDA (*Distributed Relational Data Architecture*) unterstützt, den Zugriff auf DB2-Server.

Außerdem können DB2-Anwendungen auf Daten auf anderen DB2-Servern zugreifen oder auf andere entfernte relationale Datenbanksysteme, die DRDA unterstützen. DDF unterstützt sowohl TCP/IP- als auch SNA-Netzwerke.

Mit DDF können bis zu 150.000 verteilte Threads gleichzeitig mit einem einzigen DB2-Server verbunden sein.

DDF nutzt spezielle Methoden für die Übermittlung von Abfrageergebnissen, die den Netzwerkverkehr minimieren. Auch Stored Procedures werden unterstützt, um die Verarbeitung verteilter Zugriffe zu minimieren.

DB2-Adressräume

Wegen der Komplexität und im Sinne einer funktionalen Trennung wird die Arbeit von DB2 heute auf mehrere Adressräume verteilt.

DSNMSTR

Der wichtigste Adressraum, der auch für die Gesamtkoordination verantwortlich ist, ist der DSNMSTR, der auch als System-Service-Adressraum bezeichnet wird. Er ist auch zuständig für das Schreiben von Protokollsätzen auf das Recovery Log und die Verwaltung des Boot Strap Dataset (BSDS), ein Verzeichnis der Log-Dateien, das beim Systemstart und in Fehlerfällen als wichtigste Informationsquelle dient.

Tatsächlich ist das Schreiben von Log-Sätzen immens wichtig. Es geht gar nichts mehr (nicht einmal das Stoppen von DB2!), wenn DSNMSTR nicht in die Log-Dateien schreiben kann. Das ist vor allem auch für das Verständnis des Systembetriebs wichtig.

DSNDBM1

Ein weiterer Adressraum mit dem Namen DSNDBM1 ist für die Database Services zuständig. Wann immer eine SQL-Abfrage bearbeitet werden soll, wird er aktiv. Er kontrolliert die Buffer- und die RID-Pools sowie die Systemdatenbanken. Wenn DB2 gestartet ist, sind DSNMSTR und DSNDBM1 permanent aktiv.

Lock Manager IRLM

Um die Integrität der Daten bei konkurrierenden Zugriffen sicherstellen zu können, wird ein Lock Manager benötigt. Ursprünglich wurde dieser für das hierarchische DBMS der IBM geschrieben und dann auch für DB2 verwendet. Es hat immer wieder für Verwirrung gesorgt, dass dieser Lock Manager den Namen *IMS Resource Lock Manager* (IRLM) hatte. Um Missverständnissen künftig vorzubeugen, wurde er inzwischen umgetauft in *Internal Resource Lock Manager*, so dass die gebräuchliche Abkürzung IRLM erhalten geblieben ist.

Distributed Data Facility (DDF)

Der jüngste Adressraum ist der DDF-Adressraum, der benötigt wird, wenn ein lokales DB2 auf ein entferntes DB-System zugreifen will. Die entfernte Datenbank kann eine beliebige DRDA-fähige Datenbank sein.

Systemdatenbanken

Zur Speicherung von Steuerungs- und Verwaltungsinformationen des DB2-Systems dienen die vier Systemdatenbanken **DSNDB06, DSNDB01, DSNDB04** und **DSNDB07**.

DB2-Katalog (**DSNDB06**)

Alle Objekte, die von einem DB2-System verwaltet werden, sind in den DB2-Katalogtabellen beschrieben. Eingebettet in mehrere Table Spaces befinden

sich DB2-Tabellen, die von SQL-Anweisungen bearbeitet und abgefragt werden können. Der DB2-Katalog wird vom DB2-System permanent miteinbezogen, um z. B. durch das Abgleichen von Ressourcenanforderungen mit Kataloginformationen eine günstige Zugriffsstrategie auf die Daten zu erreichen.

DB2 Directory (DSNDB01)
: Das DB2 Directory besteht aus Objekten, die Informationen über

- Utility Restart
- Application Plans und Packages
- Benutzerdatenbanken
- Tablespace-Recovery

beinhalten. Diese Objekte können nicht mit SQL beeinflusst werden. Sie werden mit entsprechenden DB2-Kommandos bearbeitet.

User Default-Datenbank (DSNDB04)
: Wenn eine Tabelle beim Anlegen nicht einer bestimmten Datenbank zugeordnet wird, wird sie automatisch in der Datenbank DSNDB04 angelegt. Sie dient hauptsächlich für Test- und Übungszwecke.

Temporäre Datenbank (DSNDB07)
: Bestimmte SQL-Befehle benötigen für die Zwischenspeicherung von Ergebnissen eine temporäre Speichermöglichkeit. Außerdem wird diese Datenbank auch als Überlaufspeicher für interne Arbeitsbereiche des Systems genutzt. Der Administrator sollte hierauf ein Auge haben, da eine starke Nutzung dieser Datenbank oft auf zu klein angelegte Pufferbereiche hinweist.

Logging, Backup und Recovery

Oberstes Gebot für eine relationale Datenbank ist die Einhaltung der Konsistenz der verwalteten Objekte. Dies wird von DB2 in erster Linie über das Logging sichergestellt. Tritt irgendwo ein Fehler auf, so müssen die Daten automatisch wieder in einen konsistenten Zustand gebracht werden können.

Alle Aktionen in der Datenbank, die Veränderungen bewirken, werden über das Logging entsprechend mitaufgezeichnet. Dies betrifft nicht nur Veränderungen per DML, sondern auch Veränderungen über DDL- und DCL-Befehle, da diese Datenänderungen im Katalog zur Folge haben.

Alle Anforderungen, die mittels SQL an DB2 gestellt werden, werden zunächst in zeilenweise Anforderungen zerlegt, und die Abarbeitung jeder Zeile wird in einem Log-Satz dokumentiert. Diese Log-Sätze werden in einen Puffer geschrieben und tragen als gemeinsame Kennzeichnung die relative Byte-Adresse (RBA) des ersten

Satzes dieser *Unit of Recovery* (UOR). Mit Ende der UOR, bei Durchführung des Commit-Protokolls, werden die Daten dann in das Active Log geschrieben.

Da die gesamten Log-Informationen auf Einzeilenverarbeitung innerhalb von Pages und Table Spaces basieren, besteht ein direkter Zusammenhang mit der physikalischen Abspeicherung der Daten.

Werden physisch gespeicherte Daten geändert, ohne dass entsprechende Log-Sätze geschrieben werden, so können diese nicht wiederhergestellt werden. Das ist beispielsweise der Fall bei Einsatz des REORG- bzw. des LOAD-Utility (mit Option LOG=NO). DB2 verlangt deshalb dann, dass neue Ausgangsdaten erstellt werden, indem das Objekt in den COPY-Pending-Zustand versetzt wird.

Auch im Falle von Hardwarefehlern muss es möglich sein, den Zustand vor Eintritt des Fehlers wiederherzustellen. Damit dies möglich ist, gibt es das COPY-Utility, mit dem der Inhalt von Table Spaces in Form von so genannten *Image Copies* gesichert werden kann. Diese sind die Grundlage für Recovery-Aktivitäten.

Das COPY-Utility kennt zwei Betriebsarten:

Full Image Copy
: Damit werden alle Pages eines Table Space gesichert. Full Image Copies müssen nach Ablauf des REORG-Utilities oder nach Einsatz des LOAD-Utilities (mit der Option LOG=NO) durchgeführt werden, da ansonsten keine Recovery möglich ist.

Incremental Image Copy
: Damit werden Pages gesichert, die seit dem letzten Lauf des COPY-Utilities verändert wurden. Besonders bei Table Spaces mit geringem Änderungsaufkommen spart dies natürlich viele Ressourcen. Konsolidiert werden kann entweder durch ein neues Full Image Copy oder mit Hilfe des Mergecopy-Utilities.

Interaktive Schnittstellen

Zwei wichtige Schnittstellen zu DB2 sind DB2 Interactive und Sequential Processing Using File Input (SPUFI).

DB2 Interactive (DB2I) kann benutzt werden, um DB2-Operationen menügesteuert aufzurufen. DB2I läuft unter TSO/ISPF. Unter anderem werden folgende Aufgaben unterstützt:

- Ausführung von SQL-Anweisungen mit Hilfe von SPUFI
- Aufruf von DCLGEN
- Vorbereiten eines DB2-Programms

- Aufruf des DB2 Precompiler
- Ausführung von BIND/REBIND/FREE-Kommandos
- Aufruf eines SQL-Programms
- Aufruf von DB2-Kommandos
- Ändern von DB2I-Default-Werten

Sequential Processing Using File Input (SPUFI) ist eine Option im DB2I-Menü, um SQL-Anweisungen interaktiv aufzurufen. In einem Input Dataset können mehrere Queries für die Ausführung definiert werden. Wenn nicht explizit eine COMMIT-Anweisung codiert wird, bilden alle SQL-Anweisungen eine Unit of Work.

Nach der Ausführung werden der SQLCODE und der SQLSTATE angezeigt.

Kapitel 11

Transaktionsverarbeitung

11.1 Einführung

DB/DC steht für *Data Base/Data Communication* und ist ein Begriff für die so genannte Online-Verarbeitung (im Gegensatz zu Timesharing oder Batch). Diese Art der Verarbeitung ist gekennzeichnet durch Transaktionen, die meist relativ wenig Ressourcen benötigen. Dafür sind die Anforderungen bezüglich Verfügbarkeit und Antwortzeiten sehr hoch.

So genannte *Transaction-Processing*-Anwendungen (TP), an die sich ein Benutzer anmeldet, arbeiten transaktionsorientiert und zumeist in Verbindung mit einem Datenbanksystem. Für diese Art Verarbeitung stehen auch oft Begriffe wie Transaktionsverarbeitung oder *Online Transaction Processing* (OLTP). Das System muss hier zusätzliche Services bereitstellen, damit die Integrität der Daten gewährleistet wird. Es reicht nicht mehr – wie für Batchjobs –, dass eine tägliche Sicherung der Daten gemacht wird. Die Daten müssen online mutierbar und die Datenintegrität muss zu jedem Zeitpunkt der Verarbeitung gewährleistet sein.

In einer Anwendung hat der Begriff „Transaktion" im Prinzip die gleiche Bedeutung wie im täglichen Leben auch: Es handelt sich um ein Ereignis bzw. einen Geschäftsvorgang zwischen zwei oder mehr Beteiligten. Objekte, die von Transaktionen betroffen sind, können Geld, Waren, Informationen, Dienstleistungen etc. sein. Eine „Buchhaltung" muss dafür sorgen, dass der Vorgang aufgezeichnet wird.

Die Steuerung eines derartigen Vorgangs und die Buchhaltung sollen für Verlässlichkeit, Geschwindigkeit, Skalierbarkeit und Kostenoptimierung sorgen.

Aufbau einer Transaktion

Eine Online-Transaktion besteht aus der Ausführung eines Programms, das (meistens) eine administrative Funktion ausführt, indem auf eine gemeinsam benutzte Datenbank zugegriffen wird. Beispiele dafür sind eine Flugbuchung oder eine Geldtransaktion zwischen verschiedenen Konten.

Eine *Transaction Processing Application* (TP-Anwendung), oft auch einfach als Online-Anwendung bezeichnet, ist eine Zusammenfassung von mehreren Transaktionsprogrammen, hinter denen eine entsprechende Geschäftstätigkeit steht. Die erste Online-Anwendung, die einen sehr hohen Bekanntheitsgrad erfahren hat, war ein Flugreservierungssystem mit dem Namen SABRE. SABRE wurde in den 60er Jahren als Joint Venture zwischen IBM und American Airlines entwickelt.

Damals bereits war es das bei weitem umfangreichste DV-Projekt, das jemals aufgesetzt wurde, und auch heute noch gehört SABRE zu den weltweit größten OLTP-Systemen. Es sind weltweit ca. 300.000 Arbeitsplätze involviert, und in Spitzenzeiten werden mehr als 10.000 Transaktionen pro Sekunde abgearbeitet.

Ein Transaktionsprogramm macht im Wesentlichen drei Dinge:

1. Es nimmt Eingaben von einem Client entgegen. Der Client kann ein Sichtgerät sein, das von einer Person bedient wird. Es kann sich dabei jedoch beispielsweise auch um einen Barcode-Leser, einen Kassenscanner oder einen Sensor an einem Roboter handeln.

2. Die eigentliche Arbeit wird ausgeführt.

3. Eine Antwort auf den Request wird erzeugt und oft zurück an das aufrufende Gerät gesendet.

ACID-Merkmale

Anwendungstransaktionen müssen die so genannten „ACID"-Eigenschaften umsetzen. Das Kürzel ACID steht für

Atomicity
: Eine Transaktion wird zur Sicherung der Datenintegrität entweder ganz oder gar nicht durchgeführt.

Consistency
: Die Daten müssen immer gültig sein. Eine Transaktion führt einen konsistenten Datenbankzustand in einen anderen konsistenten Datenbankzustand über.

Isolation
: Jede Änderung und alle Transaktionen müssen unabhängig voneinander sein. Eine Transaktion sieht die zugrunde liegenden Datenbanken so, als gäbe es keine nebenläufigen Transaktionen. Die Änderungen einer Transaktion werden bis zum Abschluss dieser Transaktion vor parallelen Transaktionen verborgen, so dass keine ungewollten Wechselwirkungen entstehen.

Durability
: Rollback und Recovery im Fehlerfall sind sicherzustellen. Die Änderungen einer abgeschlossenen Transaktion gehen nicht mehr verloren, auch nicht bei Hardware-Fehlern oder Fehlern in der Systemsoftware.

Diese ACID-Eigenschaften nun in jede einzelne Anwendung einbauen zu müssen, wäre jedes Mal für die Entwickler eine gewaltige Aufgabe. Viele Aspekte dabei haben mit dem eigentlichen Geschäftsvorfall wenig zu tun, sondern sind technischer Natur. Ein so genannter Transaktionsmonitor kann hier für Erleichterung sorgen, indem er sich um die ACID-Eigenschaften kümmert, ohne dass sich der Anwendungsprogrammierer darum kümmern muss.

Während eine Transaktion abläuft, werden alle Veränderungen zunächst einmal auf einer provisorischen Basis vorgenommen und in einem Logfile mitgeführt. Erst wenn ein definierter Zustand erreicht wird, gibt es ein *commit*, und die Änderungen werden dauerhaft abgespeichert. Treten während der Transaktion Probleme auf, bevor der entsprechende Synchronisationspunkt erreicht wird, können die provisorischen Änderungen mit Hilfe des Logfiles vollständig zurückgenommen werden.

11.2 TP-Monitor

Ein TP-Monitor ist ein Software-Produkt, das Online-Transaktionen erzeugt, ausführt und verwaltet und sich dabei um die ACID-Eigenschaften kümmert.

Zumeist beinhaltet ein TP-Monitor drei Funktionsbereiche:

- Ein Application Programming Interface (API) als Schnittstelle zu Laufzeitfunktionen, die die Ausführung von Transaktionen unterstützt. Hierzu gehört beispiels-

weise das „Klammern" von Transaktionen mit START, COMMIT und ABORT oder die Kommunikation zwischen Programmen innerhalb einer Transaktion.

- Programmentwicklungs-Werkzeuge, um Transaktionen zu erstellen, wie beispielsweise ein Compiler, der die Anweisungen einer Programmiersprache in das oben erwähnte API vornimmt.
- Eine System-Management-Schnittstelle, um die Funktionen zu überwachen und die Ausführung der Transaktionsprogramme zu steuern.

TP-Monitore auf unterschiedlichen Plattformen

Zahlreiche Hersteller bieten Transaktionsmonitore auf unterschiedlichen Plattformen an. Beispiele:

- ACMS von DEC
- CICS von IBM
- Complete von der Software AG
- Encina von Transarc
- IMS/DC von IBM
- Pathway/TS von Tandem
- TOP END von AT&T
- Tuxedo von Bea

Der Datenfluss eines TP-Monitors folgt meist dem folgenden Schema:

1. Umformung der Eingabe von einem Bildschirm (oder einem anderen Eingabegerät) in ein standardisiertes Nachrichtenformat, das der TP-Monitor versteht.
2. Analyse des Request-Headers um festzustellen, welche Transaktion angefordert wird und welches Programm den Request verarbeiten soll.
3. Starten der Transaktion und Aufruf des Programms, das typischerweise DB-Aufrufe beinhaltet.
4. Commit der Transaktion, wenn sie erfolgreich abgeschlossen werden konnte, oder Abort der Transaktion, wenn sie nicht erfolgreich abgeschlossen werden konnte.
5. Senden einer Antwort an die Quelle, von der der Request ausgegangen ist.

11.3 Architektur eines TP-Monitors

Abbildung 11.1 zeigt eine so genannte „3-Tier-Architektur": Es handelt sich also um eine Architektur, die in drei Ebenen aufgeteilt ist.

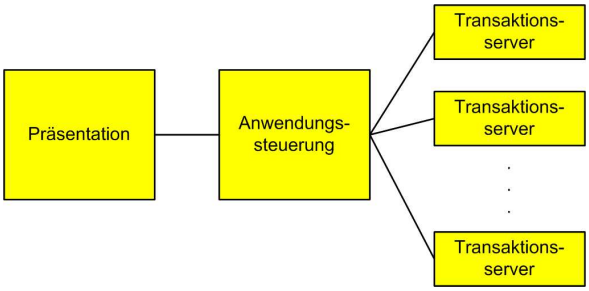


Abbildung 11.1: 3-Tier-Architektur

Die Präsentationsebene ist dafür zuständig, dass die Eingaben, die normalerweise über Formulare oder Menüs getätigt werden, in ein standardisiertes Nachrichtenformat umgeformt werden. Die Nachricht wird dann weitergeleitet an die Anwendungssteuerung.

Die Anwendungssteuerung leitet den Request dann wiederum weiter an einen geeigneten Transaktionsserver und ruft das Transaktionsprogramm auf, das dann die eigentliche Arbeit macht.

Bei einer 2-Tier-Architektur kommuniziert der Präsentationsserver direkt mit dem Transaktionsserver oder einem Datenbankserver, ohne dass ein TP-Monitor eingesetzt wird. Meistens wird die Präsentationsschicht mit Hilfe eines 4GL-Werkzeugs erstellt wie VisualBasic, Powerbuilder oder Delphi. Diese Technik wird auch oft als *TP Light* bezeichnet, da auf den eigentlichen TP-Monitor verzichtet wird. Eine derartige Lösung ist weniger skalierbar und verkraftet nur eine geringe Anzahl gleichzeitiger Requests.

11.4 Aufgabe eines TP-Monitors

Ein TP-Monitor stellt eine Software-Umgebung zur Verfügung, in der die Anwendungsprogramme laufen und über die die Benutzer mit dem System interagieren. Ein TP-Monitor bildet eine einheitliche Schnittstelle zu Systemkomponenten, die von den TP-Anwendungen benötigt werden, wie beispielsweise zu einem Datenbanksystem, zu Kommunikationsdiensten und zu Komponenten des Betriebssystems. Die Entwickler müssen sich nicht um die individuellen Komponenten kümmern, sondern nutzen lediglich die Schnittstellen.

Vor allem die System-Management-Funktionen eines TP-Monitors wie Lastverteilung, Fehlererkennung und -behebung, Performance Monitoring und die Möglichkeiten des Eingriffs in kritischen Situationen unterscheiden den Einsatz eines TP-Monitors von einer TP-Light-Implementation. Sehen wir uns die drei Ebenen einer 3-Tier-Architektur noch etwas genauer an.

Präsentationsschicht

Der Präsentationsserver sammelt die Eingaben eines Endbenutzers und formt daraus eine standardisierte Nachricht für den TP-Monitor. Diese standardisierte Nachricht wird an die Anwendungssteuerung weitergegeben. Der Vorteil dabei ist, dass die Anwendungssteuerung nichts über die diversen Eigenheiten der Geräte, von denen die Eingaben kommen, wissen muss. Damit wird die Präsentation unabhängig von den anderen Komponenten. Die Präsentationsschicht hat zwei Unterebenen, eine, die mit dem Endbenutzer kommuniziert und für die Sammlung der Eingaben und Formatierung der Ausgaben zuständig ist, und eine, die die standardisierte Nachricht für die Anwendungssteuerung erstellt.

Eingabeschnittstelle

Meist handelt es sich bei der Eingabeschnittstelle um einen Bildschirm mit Tastatur, der ein „dummes" Terminal, ein PC, eine Workstation etc. sein kann. Der Benutzer wählt einen Menüpunkt aus, indem er auf ein Icon drückt, einen Pull-Down-Menüpunkt markiert oder mit dem Cursor eine Stelle markiert und die Eingabetaste drückt. Dadurch wird dem System mitgeteilt, welche Transaktionsart gewünscht wird. Normalerweise wird daraufhin eine Eingabemaske angezeigt, über die Daten eingegeben werden, die für die entsprechende Anfrage relevant sind. Die Präsentationssoftware sorgt dann für eine Validierung, mit der sichergestellt wird, dass sich die Eingaben in einem vorgegebenen Rahmen bewegen.

Formular-Manager

Um diese Eingabemasken zu erstellen, verfügen die TP-Monitore über geeignete Werkzeuge, die auch als Formular-Manager bezeichnet werden. Diese unterstützen das Erstellen der Eingabemasken und die Validierung der Eingabefelder. Der Formular-Manager kompiliert die Menüdefinitionen in eine interne Repräsentation, die zur Laufzeit von der Präsentationssoftware benutzt wird, um die Eingabemasken darzustellen und die Eingaben entgegenzunehmen. Die Formularsoftware erzeugt normalerweise auch eine Satzdefinition, die mit den Ein-/Ausgabefeldern korrespondiert. Diese Satzdefinitionen werden dann in das Transaktionsprogramm übernommen, ohne dass sie von den Programmierern eingegeben werden müssen.

Erstellen der standardisierten Nachricht

Nachdem die Eingaben entgegengenommen wurden, erzeugt die Präsentationssoftware die standardisierte Nachricht für die Anwendungssteuerung. Jeder TP-Monitor definiert dafür sein eigenes Format. In der Vergangenheit zumindest gab es noch keinen Standard, der hier sinnvollerweise genutzt werden konnte. Die meisten TP-Monitore beinhalten in der Request Message die folgenden Informationen:

- einen Identifizierer des Benutzers, meist eine Benutzerkennung
- einen Identifizierer des Geräts, meist die Netzwerkadresse, über welche die Eingabe getätigt wurde. Über das Kommunikationssystem kann dann über einen Mapping-Mechanismus der Gerätetyp bestimmt werden.
- einen Request-Typ, der als Identifizierer für die Transaktionsart dient
- Request-spezifische Parameter, die von der Art des Requests abhängig sind

Authentifizierung

Eine weitere Funktion, die in der Präsentationsschicht angesiedelt ist, ist die Authentifizierung. Hierbei geht es darum, den Benutzer und/oder das Gerät, über das kommuniziert wird, zu identifizieren. Dies geschieht über ein Passwort oder immer häufiger – vor allem bei kritischen Anwendungen – auch über ein Zertifikat, das beispielsweise auf einer Smartcard untergebracht ist.

Anwendungssteuerung

Ein Request enthält, wie bereits erwähnt, ein Feld mit dem Request-Typ. Eine der wichtigsten Aufgaben der Anwendungssteuerung ist es, den Request-Typ in Verbindung zu bringen mit dem Programm, das den Request abarbeitet. Wenn der Request eine Rückgabe erfordert, was zumeist der Fall sein wird, muss auch dieser von der Anwendungssteuerung an den Präsentationsserver zurückgeleitet werden, von dem der Request ausgegangen ist. Normalerweise ist die Anwendungssteuerung auch für die Klammerung einer Transaktion zuständig, d. h. sie setzt die **Begin-**, **Commit-** und **Abort**-Anweisungen ab.

Routing

Das Routing kümmert sich um das Senden eines Request an einen geeigneten Transaktionsserver, der den Request verarbeiten kann. Der Request-Typ wird dazu mit einem Server Identifier verknüpft und dann losgeschickt. Für dieses Mapping wird oft ein Verzeichnisdienst verwendet, um der Dynamik von Konfigurationen gerecht zu werden.

Sessions und Security

Sessions werden benötigt, damit Zustände, die zwischen Kommunikationspartnern aufgebaut werden, zwischengespeichert werden können. Diese Zustände reflektieren beispielsweise Adressinformationen, um die Sessionpartner schnell lokalisieren zu können, ohne jedes Mal „teures" Nachschlagen durchführen zu müssen. Oft sind auch Security-Informationen involviert, damit die beteiligten Partner sicher sein können, dass der andere Partner berechtigt ist, Messages und Informationen auszutauschen, und damit die Security nicht bei jeder Nachricht überprüft werden muss.

Exception Handling

Da sich die Anwendungssteuerung um die Abgrenzung einer Transaktion kümmert, muss sie auch definieren, was passiert, wenn eine Transaktion aus systemtechnischen Gründen nicht erfolgreich abgeschlossen werden kann. Dies wird über einen Exception Handler realisiert, der aufgerufen wird, wenn eine Transaktion abgebrochen wird oder wenn im System fehlerhafte Situationen eintreten.

Transaktionsserver

Ein Transaktionsserver ist das Anwendungsprogramm, das die eigentliche Arbeit bei einem Request-Aufruf durchführt. Meist wird dieses Programm aus Portabilitätsgründen in einer höheren Programmiersprache wie COBOL, PLI, C oder C++ geschrieben, das oft wiederum eingebettete Aufrufe für Datenbanken enthält.

11.5 Two-Phase Commit (2PC) bei verteilten Transaktionen

Wenn eine Transaktion Daten auf zwei oder mehr Systemen in einer verteilten Umgebung vornimmt, müssen auch dort die ACID-Anforderungen gewährleistet sein. In diesen Umgebungen ist das Einhalten dieser Merkmale eine noch größere Herausforderung, da jedes System für sich Probleme machen kann und die Umgebung natürlich komplexer wird. Die Lösung dafür ist ein spezielles Vorgehen, das auch als Protokoll bezeichnet werden kann und unter dem Begriff ***Two-Phase Commit*** bekannt ist.

Das Besondere daran ist, dass eine Transaktion auf einem System die Updates bereits „committet" haben kann, während ein zweites System keinen Commit melden kann, weil sich irgendein Problem ergeben hat.

Um das Vorgehen bei einem Two-Phase Commit (2PC) zu verstehen, müssen wir die Architektur und die Vorgehensweise des Transaction Manager, der dafür verantwortlich ist, genauer unter die Lupe nehmen. Das Standardmodell eines derartigen

TP-Monitors mit einem integrierten Transaction Manager wurde mit dem *Customer Information Control System* (CICS) von IBM eingeführt und wird inzwischen von X/Open unterstützt.

In diesem Modell kommuniziert der Transaktionsmanager mit Anwendungen, Resource Managern und anderen Transaktionsmanagern. Unter „Ressourcen" werden Datenbanken, Warteschlangen, Messages und andere Objekte verstanden, die in einer Transaktion angesprochen werden können. Jeder Resource Manager verfügt über Vorgänge, die nur abgeschlossen werden dürfen, wenn die Transaktion, die den Vorgang auslöst, „committet".

Der Transaktionsmanager verarbeitet die grundlegenden Transaktionsmechanismen für die Anwendungen – Start, Commit und Abbruch. Eine Anwendung ruft *Start* auf, um die Ausführung einer neuen Transaktion zu kennzeichnen. Sie ruft *Commit* auf, um die Transaktion auszuführen, und *Abort* (Abbruch), um die Transaktion abzubrechen.

Der Transaktionsmanager kümmert sich unter anderem um die „Buchhaltung", die für die Transaktionen durchgeführt werden muss, um die Atomizität garantieren zu können, wenn eine oder mehr Ressourcen involviert sind.

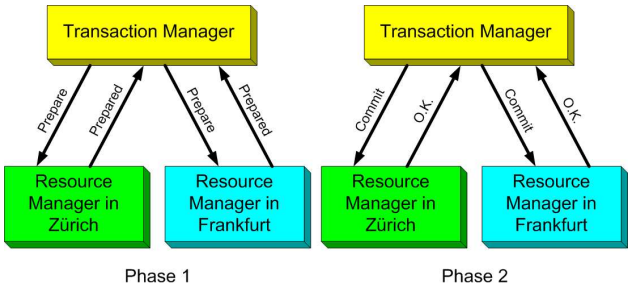
Typischerweise gibt es einen Transaktionsmanager pro Knoten in einer verteilten Umgebung. Wenn eine Anwendung den Start einer Transaktion auslöst, vergibt der Transaktionsmanager eine eindeutige Kennzeichnung, den so genannten *Transaction Identifier*. Während der Ausführung einer Transaktion führt der Transaktionsmanager alle Zugriffe auf die Resource Manager mit, die von der Transaktion veranlasst werden. Dies setzt eine enge Zusammenarbeit zwischen den Anwendungen, den Resource Managern und dem Kommunikationssystem voraus. Wann immer eine Transaktion auf einen Resource Manager zugreift, muss dafür gesorgt werden, dass der Transaktionsmanager davon erfährt. Damit ist gewährleistet, dass bei einem Commit der Transaktionsmanager alle Resource Manager kennt, die von der Transaktionsausführung betroffen sind und mit denen er den Ablauf des 2PC-Protokolls abstimmen muss.

Wenn ein Transaktionsprogramm die Ausführung abschließen will und die Commit-Anweisung aufruft, wird diese an den Transaktionsmanager weitergeleitet, der dann das 2PC-Protokoll ausführt. Bei einem Abbruch muss der Transaktionsmanager dafür sorgen, dass die Updates, die eine Transaktion vorgenommen hat, rückgängig gemacht werden.

Im Falle eines 2PC sendet der Transaktionsmanager zwei Folgen von Nachrichten an die Beteiligten aus. Jede dieser Folgen steht für eine Commit-Phase. In der ersten Phase werden alle Resource Manager benachrichtigt, dass sie sich auf den Commit vorbereiten und die Ergebnisse der Transaktion dauerhaft abspeichern, aber noch nicht abschließen sollen. Das bedeutet, die Resource Manager sind für den Commit vorbereitet (prepared). Wenn der Transaktionsmanager von allen Resource Managern eine positive Rückmeldung bekommen hat, ist die Vorbereitung abge-

schlossen. Dann kommt die zweite Phase, in der die Resource Manager aufgefordert werden, den Commit durchzuführen.

Abbildung 11.2: Ablauf des Two-Phase Commit



Ein Two-Phase Commit ist immer dann erforderlich, wenn eine Transaktion auf mehr als einen Resource Manager zugreift.

Eine der Schlüsselfragen beim Design von TP-Anwendungen ist die Frage, ob es notwendig ist, dass mehrere Resource Manager involviert sein müssen, da das 2PC-Protokoll natürlich einen Overhead hat, der die Performance der Transaktionen beeinflusst. Allerdings kann die Verteilung auch Vorteile bringen, da die Skalierbarkeit und die Verfügbarkeit verbessert werden. Eine wichtige Rolle spielt hierbei der Begriff *Logical Unit of Work* (LUW), der für eine Verarbeitung nach dem Motto „ganz oder gar nicht" steht. D.h., dass eine Transaktion entweder vollständig verarbeitet wird und die Daten in einen konsistenten Zustand gebracht werden, oder dass sie, falls eine Transaktion nicht vollständig verarbeitet werden kann, in den Zustand vor dem Start der Transaktion zurückgesetzt werden können (*Rollback*).

Ein Restart-/Recovery-Konzept spielt hierbei eine wichtige Rolle.

11.6 Architektur von CICS

CICS ist ein Transaktionsmonitor von IBM, der 1968 ursprünglich einmal für den klassischen Mainframe in einem Joint Venture zwischen Michigan Bell und IBM entwickelt wurde. Inzwischen gibt es CICS auch für andere Plattformen, wobei jedoch vor allem auf UNIX-Systemen die Codebase von Encina verwendet wird, ein Transaktionssystem, das von der Firma Transarc entwickelt wurde.

CICS steht für *Customer Information Control System* und verarbeitet weltweit täglich die meisten Transaktionen: 490 der Fortune-500-Unternehmen setzen CICS ein. Allein auf dem Mainframe gibt es weltweit rund 50.000 Lizenzen. Von 30 Millionen Benutzern werden täglich mehr als 300 Milliarden Transaktionen aufgerufen. Schätzungsweise knapp eine Million Programmierer sind in irgendeiner Form von CICS betroffen. Es existiert weltweit mehr als eine Milliarde Lines of Code in Verbindung mit CICS, hauptsächlich in der Programmiersprache COBOL geschrieben.

CICS läuft inzwischen auf nahezu allen Plattformen vom PC unter Windows (NT und 2000) über UNIX (HP-UX, Solaris, Sinix, OSF/1, AIX, Linux), VSE, OS/400 bis zum Mainframe (MVS, OS/390, z/OS).

CICS wurde im Jahre 1968 von einem IBM-Kunden entwickelt, später dann von IBM als Standardprodukt übernommen und inzwischen zum Industriestandard ausgebaut.

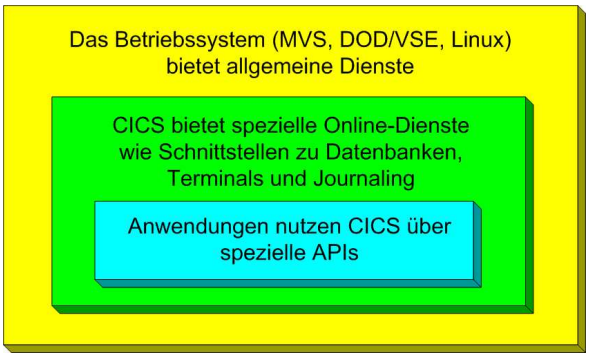


Abbildung 11.3: Aufbau von CICS

CICS und MVS

CICS ist ein Transaktionsmonitor, der in ein Betriebssystem (z. B. MVS) eingebettet ist. Die Komponenten zwischen CICS und dem Betriebssystem müssen optimal aufeinander abgestimmt werden, um die in einer Installation vorgegebenen Ziele auf der Basis der unterschiedlichen Workloads zu erreichen. CICS stellt in diesem Zusammenhang eine Workload dar.

Das Ziel eines sinnvollen CICS-Einsatzes sind hoher Durchsatz und kurze Antwortzeiten. Einflussfaktoren dabei sind die zur Verfügung stehende Hardware, das Anwendungsprofil (Workloads), die RZ-Organisation (z. B. Automatisationsgrad), die Konfiguration und Definition des Betriebssystems sowie die Konfiguration und Definition des CICS-Systems.

Voraussetzung zur Erreichung optimaler Antwortzeiten ist die optimale Anpassung des CICS an die jeweiligen Gegebenheiten sowie ein rechtzeitiges Erkennen und Beseitigen von Engpässen. Das CICS-System darf nicht isoliert betrachtet werden, sondern immer im Zusammenhang mit anderen Komponenten eines Rechenzentrums, vor allem des Betriebssystems. Deshalb spielt auch die Entwicklung der Betriebssysteme aus der Sicht von CICS eine entscheidende Rolle. Einschränkungen der Architektur wurden Schritt für Schritt aufgehoben, um die Effizienz zu erhöhen, größere Transaktionsraten zu ermöglichen und weitere Möglichkeiten zu bieten, die in vorangegangenen Architekturen undenkbar waren.

Vor allem anderen sind hierbei die Erweiterungen in Verbindung mit dem virtuellen Speicher erwähnenswert, von denen vor allem die DB/DC-Systeme profitieren konnten. Während beispielsweise ein Batchjob oder auch ein TSO-Benutzer einen jeweils eigenen Adressraum zugewiesen bekommt, müssen sich in Verbindung mit CICS oder einem Datenbanksystem viele Benutzer einen Adressraum teilen.

Die wichtigsten Meilensteine in diesem Zusammenhang sind die Aufhebung der 16-MB-Grenze mit der Extended Architecture (XA) Anfang der 80-er Jahre, die Nutzung von Data/HiPer Spaces mit der Enterprise System Architecture sowie die Unterstützung von Data Sharing in Verbindung mit Parallel Sysplex.

Aus Sicht der Programmierung wurde das CICS API auf der Basis von Makros zur Verfügung gestellt, was zu sehr proprietärem und oft auch schwer verständlichem Code führte. Inzwischen wird ausschließlich mit einer API auf Kommandobasis gearbeitet. Die Schnittstelle ist damit deutlich einfacher und verständlicher geworden.

Das CICS Command Level API besteht aus **EXEC CICS** …-Kommandos, die in den Code von Standard-Programmiersprachen eingebettet werden.

CICS wird beispielsweise unterstützt von COBOL, PL/I, C, C++, RPG, Assembler und inzwischen auch Java. Die am häufigsten verwendete Programmiersprache in Verbindung mit CICS ist nach wie vor COBOL. In ein COBOL-Programm können CICS-Anweisungen eingebettet werden, ähnlich, wie man das auch in Verbindung mit Datenbanken kennt, wo SQL-Anweisungen in den Code einer höheren Programmiersprache eingebettet werden.

Beispiel (Ausschnitt)

```
PROCEDURE DIVISION.
001-MAIN SECTION.
        EXEC CICS HANDLE CONDITION
                LENGERR (001-LENGTH-ERROR)
                ERROR  (001-GENERAL-ERROR)
        END EXEC.
        PERFORM 001-SEND.
        PERFORM 001-RECEIVE.
        PERFORM 001-DEBIT.
        PERFORM 001-CREDIT.
        PERFORM 001-SUCCESS.
        EXEC CICS RETURN.
        END EXEC.

001-SEND SECTION.
        EXEC CICS SEND
                MAP ('MAPDBCR')
                MAPSET ('DBCRSET')
                MAPONLY
                ERASE
        END EXEC.

001-RECEIVE SECTION.
```

```
        EXEC CICS RECEIVE
                MAP ('MAPDBCR')
                MAPSET ('DBCRSET')
                INTO (INPUT-DBCR-AREA)
        END EXEC.
...
```

Der Code läuft zunächst durch einen Preprocessor, der die CICS-Anweisungen herauslöst und das Gesamtprogramm für die Kompilierung vorbereitet.

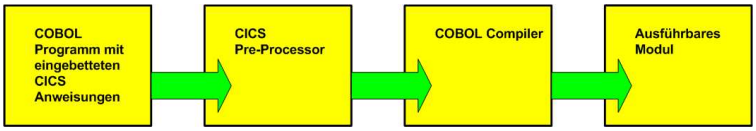


Abbildung 11.4: CICS-Verarbeitung

Basic Mapping Support (BMS) in CICS ist eine Schnittstelle für die Präsentation eines User Interface für 3270-Anwendungen. Über die 3270-Schnittstelle werden klassische CICS-Anwendungen für so genannte „dumme Terminals" realisiert.

Inzwischen gibt es auch weitere, zum Teil auch objektorientierte Schnittstellen, die auf Enterprise Java Beans (EJBs), CORBA und IIOP basieren.

11.7 Die wesentlichen CICS-Komponenten

Die Mainframe-Version von CICS läuft als Started Task oder als Batchjob in einem Adressraum.

Es gibt drei Hauptkomponenten in CICS mit jeweils definierter Funktionalität:

Nukleus

Der Nukleus enthält die CICS-Systemmodule und -Tabellen, die *Common System Area* (CSA) und die *Common Work Area* (CWA). Hier sind die grundlegenden Steuermechanismen von CICS untergebracht.

CICS hat sehr viele Systemmodule, die alle eine spezifische Steuerfunktion wahrnehmen. Da eine Installation sehr flexibel bezüglich Namenskonventionen etc. sein kann, hat CICS zahlreiche modifizierbare Parameter, die in Tabellen abgelegt und von den Systemmodulen zugegriffen werden können.

Die Common System Area (CSA) verwaltet die Pointer zu allen CICS-Kontrollblöcken und Storage Blocks. Der Anwendungsprogrammierer hat damit in der Regel nichts zu tun.

Die Common Work Area (CWA) ist ein installationsdefinierter Bereich. Sie enthält Informationen, die in CICS resident gehalten werden und von Anwendungsprogrammen verändert werden können. Die CWA wird beispiels-

weise benutzt, um Daten zwischen Anwendungsprogrammen auszutauschen, wobei die Programme zu unterschiedlichen Tasks gehören können. Die CWA wird in der *System Initialization Table* (SIT) als WRKAREA eingetragen und muss für die Benutzung im Programm adressiert werden können.

Resident Area

Die Resident Area in CICS ist der Bereich, in dem die Anwendungsprogramme gehalten werden. Diese Programme werden auf Anforderung vom CICS geladen. Der entsprechende Bereich wird wieder freigegeben, wenn er nicht mehr benötigt wird, d. h., wenn der Task endet.

Bestimmte Programme, die häufig verwendet werden und performancekritisch sind, werden beim Start von CICS bereits geladen. Diese Programmnamen werden in eine spezielle CICS-Tabelle eingetragen und als residente Programme bezeichnet.

Dynamic Storage Area (DSA)

Die Dynamic Storage Area ist der Arbeitsbereich für CICS-Systemmodule und Anwendungsprogramme. Dies sind beispielsweise Daten von einem Terminal, Working Storage Sections eines COBOL-Programms oder Arbeitsbereiche, die von CICS-Systemprogrammen verwendet werden. Auch diese Bereiche werden nur für die Dauer des jeweiligen Task belegt.

Das Domain-Konzept

Die CICS-Komponenten wiederum sind (inzwischen) in funktional gegliederte Bereiche organisiert, die als „Domains" bezeichnet werden. Der Vorteil dabei ist, dass diese Domains aus funktional selbstständigem Code bestehen und ausschließlich eigene Ressourcen verwalten. Zwischen diesen Modulen gibt es sauber definierte Schnittstellen, die eine vernünftige Wartbarkeit ermöglichen.

Die Domains kommunizieren über genau definierte Gates, deren vierstelliger Name Rückschlüsse auf die Funktion ermöglicht.

So fordert beispielsweise die Statistics Domain mit Hilfe eines definierten Token über das STST Gate andere Domains dazu auf, Statistikdaten zu übermitteln. Die Kommunikation zwischen den Gates erfolgt grundsätzlich über den Kernel und nie direkt von Gate zu Gate. Damit bildet der Kernel die zentrale Steuerung. Die Vermittlungsfunktion für die Requests von Gate zu Gate über den Kernel wird auch als *Kernel Linkage* bezeichnet.

Für den Zugriff auf Datenobjekte und deren Verarbeitung sind vor allem die folgenden Domains von Bedeutung:

Kernel Domain

Die Kernel Domain ist für Task Switching und für Program Checks verantwortlich.

Parameter Domain
: Diese versorgt andere Domains mit Systemeinstellungen aus der System Initialization Table (SIT).

Local/Global Catalog Domain
: Die Local/Global Catalog Domain speichert aktuelle Session-Daten, um für einen Emergency Restart oder für einen Warmstart die Datenintegrität sicherstellen zu können.

Dispatch Domain
: Die Dispatch Domain legt die Verarbeitungsreihenfolge von anstehenden Tasks (innerhalb des CICS!) fest. Die Steuerung erfolgt über entsprechend festgelegte Prioritäten. Außerdem werden Deadlocks erkannt und aufgelöst. Notfalls wird auch eine Task Purge durchgeführt.

Lock Manager Domain
: Die Lock Manager Domain verwaltet CICS-interne Sperrmechanismen, die aus Gründen der Datenintegrität gesetzt und wieder gelöst werden können.

Timer Domain
: Die Timer Domain unterstützt die zeitabhängige Verarbeitung und hilft beim Erkennen von Deadlocks, indem entsprechende Zeitüberschreitungen gemeldet werden.

Storage Manager Domain
: Diese Domain verwaltet die Dynamic Storage Area (DSA) sowie die Extended Dynamic Storage Area (EDSA). Sie weist den Systemkomponenten und den Benutzertasks Speicherbereiche über so genannte Subpools zu. Sie informiert regelmäßig die Loader-, Dispatcher- und Application Domain über den Speicherstatus. Außerdem ist sie für Storage Recoveries zuständig.

Application Domain
: Die Application Domain enthält Komponenten für das Teminal Management, File Management, Transient Data Management und weitere Dienste.

Program Manager Domain
: Die Program Manager Domain verwaltet die Autoinstall PCT. Früher mussten Programme und Maps von Hand in die Systemtabellen eingetragen werden. Diese Domain erkennt ein fehlendes Objekt und trägt es selbstständig ein.

Transaction Manager Domain
: Die Transaction Manager Domain enthält Komponenten für die Transaktionsverwaltung.

Security Manager Domain
: Die Security Manager Domain kümmert sich um die Schnittstellen zu RACF. Sie versorgt beispielsweise Hintergrund-Tasks mit einer User ID.

Komponenten des CICS Nukleus

Die Komponenten, die zum CICS Nukleus gehören (Terminal Control, Task Control, Program Control, Storage Control und File Control), befinden sich in der Region des CICS-Adressraums. Die Steuerung und die Zusammenarbeit zwischen den Komponenten wird über Tabellen realisiert, so gibt es beispielsweise die TCT, PCT, PPT, FCT etc.

Die so genannte COMMAREA dient als temporärer Zwischenspeicher beispielsweise für Session-Zustände, die von unterschiedlichen Transaktionen abgerufen werden können.

Terminal Control

Das ursprüngliche CICS-Kommunikationsprotokoll basiert auf SNA (vgl. Abschnitt 9.2). Die Komponente CICS Terminal Control verarbeitet die Terminal Input Requests, die über die *Virtual Telecommunication Access Method* (VTAM) hereinkommen. Dass die Daten heute meist nicht mehr über ein SNA-Netz, sondern über ein TCP/IP-Netz übertragen werden, spielt dabei eine untergeordnete Rolle. Das Protokoll ist bei klassischen Anwendungen dennoch das 3270-Protokoll, das über ein Telnet-3270-Protokoll über TCP/IP gefahren werden kann.

Wenn ein Request bei CICS ankommt, kann das Terminal, das den Request gesendet hat, über die CICS *Terminal Control Table* (TCT) identifiziert werden.

Die ersten vier Zeichen der Input Message werden als *Transaction Identifier* (TRID) identifiziert. Diese Information wird an die CICS Task Control weitergegeben, die eine entsprechende CICS Task startet.

Task Control

Die Task Control kümmert sich um die Zustände der CICS Tasks. Viele werden parallel abgearbeitet und konkurrieren um Ressourcen. CICS verwaltet ein eigenes internes Multitasking. Die Informationen über sämtliche Tasks werden in der CICS Program Control Table (PCT) mitgeführt. Für jede CICS Task gibt es dort einen Eintrag.

Wenn über Terminal Control ein neuer Task gestartet wird, wird für diesen Task über die CICS Storage Control ein Speicherbereich angefordert, und es werden entsprechende Task-bezogene Kontrollblöcke eingerichtet und in die Verkettung der aktiven Tasks eingehängt. Dann wird die Program Control bemüht, damit das entsprechende Anwendungsprogramm, das in der PCT definiert ist, aufgerufen werden kann.

Program Control

Die Komponente CICS Program Control ist verantwortlich für das Laden und Beenden von Anwendungsprogrammen. Wenn über die Task Control eine

Transaktion aufgerufen wird, ist die Komponente Program Control zuständig für die Verknüpfung des entsprechenden Task mit dem jeweiligen Anwendungsprogramm. Auch wenn oft viele Tasks das gleiche Anwendungsprogramm benötigen, wird dennoch das Programm nur ein einziges Mal geladen. Jeder Task durchläuft dann unabhängig von anderen Tasks das Anwendungsprogramm, d. h. die physische Kopie des Programms kann von vielen Tasks gleichzeitig genutzt werden. Das Programm muss deshalb „reentrant" programmiert werden.

Storage Control

Unter MVS hat das CICS selbst die volle Kontrolle über den virtuellen Speicher. Die Komponente Storage Control weist dynamisch Speicher zu, kontrolliert die Zugriffe und gibt den Speicher wieder frei.

Dynamische Speicherzuweisungen werden für Programme, I/O-Bereiche, Work Space usw. benötigt. Die CICS Storage Control verwaltet diverse Storage Pools für unterschiedliche Verwendungszwecke.

Die so genannte CICS COMMAREA ist ein spezieller Speicherbereich. Sie unterstützt die Zuordnung von Daten über Transaktionsgrenzen hinweg, beispielsweise um Session-Zustände zu verwalten.

File Control

Die Komponente CICS File Control unterstützt gemeinsame Zugriffe auf Plattengeräte. Die entsprechenden Kontrollblöcke und Puffer werden über die CICS File Control Table (FCT) verwaltet. CICS gewährleistet, dass keine zwei Anwendungen gleichzeitig Updates auf den gleichen Datensatz durchführen.

Timer Services

Über die Timer Services von CICS können beispielsweise bestimmte Transaktionen zu einer definierten Uhrzeit gestartet werden, Zeitintervalle überwacht und andere zeitabhängige Funktionen realisiert werden.

Datenkommunikation

In CICS lassen sich zwei Kommunikationsarten unterscheiden:

Data Communication

Kommunikation mit Endgeräten (z. B. Drucker, Terminals, POS-Geräte, Bankautomaten)

Interproduct Communication

Kommunikation mit anderen CICS-Systemen oder anderen Systemen (z. B. IMS).

Als Netzwerkarchitektur kann LU 6.2, TCP/IP, ICF (OS/400) oder NETBIOS eingesetzt werden. Die einfachste Form ist dabei eine CICS-Kommunikation zwischen CICS und einem nicht-programmierbaren Terminal. Die nächste Stufe ist die Kommunikation zwischen CICS und einer programmierbaren Arbeitsstation. Eine weitere Möglichkeit ist eine CICS-to-CICS-Kommunikation.

Die Interproduct Communication kann in folgende Kategorien eingeteilt werden:

Function Shipping (Remote Data Access)
: Function Shipping ermöglicht einem Anwendungsprogramm, das in einem CICS-System läuft, auf Ressourcen zuzugreifen, die einem anderen CICS-System gehören. In dem System, dem die Ressource gehört, wird eine Transaktion aufgerufen, welche die entsprechende Anforderung erfüllt.

Transaction Routing (Remote Application Invocation)
: Transaction Routing ermöglicht den Aufruf einer Transaktion in einer anderen CICS-Region. Es ist beispielsweise üblich, für die CICS-Benutzer eine Terminal Owning Region (TOR) für die Netzwerkressourcen, eine Application Owning Region (AOR) für Transaktionen und Programme und eine Resource Owning Region (ROR) für die Datenressource zur Verfügung zu stellen. Bei diesen Ressourcen handelt es sich beispielsweise um Dateien, temporäre Storage Queues, transiente Datenqueues und Zugriffe auf DB-Manager wie DBCTL oder DB2.

Distributed Program Link (Remote Procedure Call)
: Distributed Program Link (DPL) ermöglicht es einem Anwendungsprogramm, das in einer CICS-Umgebung läuft, ein Programm in einer anderen CICS-Umgebung aufzurufen. Das entspricht in etwa einem Remote Procedure Call in anderen Umgebungen.

Asynchronous Processing (Remote Message oder Queue)
: Asynchronous Processing ermöglicht einer Transaktion in einer CICS-Umgebung, eine Transaktion in einer anderen CICS-Umgebung aufzurufen. Diese Transaktionen können dann unabhängig voneinander weiterarbeiten.

Distributed Transaction Processing (Peer-to-Peer Cooperative Processing)
: Distributed Transaction Processing ermöglicht einer Transaktion in einer CICS-Umgebung, mit anderen Transaktionen in anderen Systemen synchron zu kommunizieren. Die Transaktionen sind für diese Art Verarbeitung speziell vorbereitet (Design und Codierung).

CICS Client-Kommunikation

Für CICS Clients, die CICS-Systeme als Server nutzen, gibt es zwei APIs:

External Call Interface (ECI)

Das ECI ermöglicht es einer Non-CICS-Anwendung, z. B. von einer Arbeitsstation aus ein CICS-Programm auf einem CICS Server aufzurufen. Für das aufgerufene Programm unterscheidet sich diese Form des Aufrufs nicht von einem Distributed Program Link (DPL), der von einem CICS-Programm abgesetzt wird.

Das ECI ist somit ein flexibles Verfahren für Client/Server-Anwendungen. Die Präsentationslogik (GUI) kann beispielsweise sauber im Sinne von C/S von der Business-Logik getrennt werden.

External Presentation Interface (EPI)

Das EPI ermöglicht es einer Non-CICS-Anwendung, gegenüber dem CICS Server wie ein oder mehrere 3270 Terminals zu erscheinen. Diese Anwendung kann somit beispielsweise CICS-Transaktionen starten und 3270-Datenströme zum Server senden bzw. vom Server empfangen. Für den Server sieht das Ganze wie „normales" Transaction Routing aus.

Steuerprogramme und Tabellen

Eine neue Transaktion muss zunächst mit einem vierstelligen Transaktionscode in die Program Control Table (PCT) eingetragen werden. Dieser Eintrag erfolgt i. d. R. mit einer speziellen Service-Transaktion, die CEDA heißt.

Damit die Transaktion ihre Aufgabe erfüllen kann, muss der Name des damit verbundenen Start-Moduls ebenfalls in die PCT aufgenommen werden.

Beim Start von CICS legt die Kernel Domain mehrere ***Task Allocated Stacks*** (TAS) an, deren maximale Anzahl durch den MXT-Parameter (maximale Anzahl Tasks) in der System Initialization Table (SIT) festgelegt wird.

Erkennt nun Terminal Control, dass ein Transaktionsaufruf vorliegt, so wird von der Dispatcher Domain ein Kontrollblock in der EDSA für einen Task angelegt, die so genannte ***Dispatcher Task Area*** (DTA). In diesem Kontrollblock werden Informationen abgelegt. Diese bestimmen beispielsweise, ob der entsprechende Task „ready", d. h. lauffähig ist oder ob er sich in einem Wartezustand befindet.

Das Anlegen des Kontrollblocks für das Task Dispatching in der EDSA sowie die Vorbereitungen für den Start werden als Task-Initialisierung bezeichnet.

Nun erhält der Transaction Manager (XM) die Kontrolle und durchsucht die PCT, um den eingegebenen Transaktionscode auf Gültigkeit zu prüfen. Ist dieser gültig, dann belegt der XM ein ***Task Queue Element*** (TQE) mit einem Token und ruft die

Dispatcher Domain auf, die die DTA angelegt hat und mit einem ATTACH einen Task erzeugt. Nun geht die Kontrolle an den Kernel, der den Task Allocated Stack mit Informationen über den Task versorgt.

Jetzt hat die Transaktion eigentlich alles, um beim nächsten Dispatching berücksichtigt zu werden. Beim „First Dispatch", nachdem CICS die Kontrolle erhalten hat, wählt der Dispatcher den nächsten Task aus der „Ready-Queue" und übergibt deren Kernel Token an die Kernel Domain. Um den Überblick zu behalten, merkt sich die Kernel Domain Informationen über alle in CICS befindlichen Tasks im Task Allocation Stack (TAS). Somit kann der Kernel den zugehörigen TAS adressieren und meldet dem Dispatcher sein „OK" zurück. Der XM übergibt die Kontrolle dann per XCTL (Transfer Control) an das Startprogramm, das vom Program Control Program (PCP) in der Processing Program Table (PPT) gefunden wurde. Jetzt existiert ein lauffähiger Task im System.

11.8 CICS-Systemmodule

Wie bereits erwähnt, gibt es in CICS viele Systemmodule und entsprechende Tabellen, mit denen die Verarbeitung gesteuert wird. Die meisten dieser Tabellen müssen entsprechend angepasst werden, wenn eine neue CICS-Anwendung installiert wird.

Terminal Control Program (TCP)
: macht das Betriebssystem darauf aufmerksam, dass Informationen von einem Terminal empfangen werden sollen; das TCP benutzt eine Terminal Control Table (TCT), um gültige Terminals und deren Attribute repräsentieren zu können.

Task Control Program (KCP)
: kümmert sich um die Zustände der CICS Tasks; es ist dafür verantwortlich, dass die Transaktionskennung verifiziert wird, eine Task Control Area (TCA) bereitgestellt wird und eine Task Identification Number (Task ID) vergeben wird. Die Task ID ermöglicht es dem CICS, alle aktiven Tasks mitzuverfolgen, auch wenn mehrere Tasks auf das gleiche Anwendungsprogramm zugreifen. Die meisten Tasks können gleichzeitig (Multitasking) verarbeitet werden. Das KCP weist die entsprechenden Prozessorzeiten zu.

  Die Fähigkeit, in einem CICS-System mehrere Tasks im gleichen Anwendungsprogramm gleichzeitig laufen lassen zu können, ist unter anderem davon abhängig, ob das jeweilige Anwendungsprogramm „reentrant" geschrieben worden ist. Die Verifikation der Transaktion wird vom KCP mit Hilfe der Program Control Table vorgenommen. In der PCT sind alle gültigen Transaktionen mit den damit verbundenen Programmen verzeichnet.

Program Control Program (PCP)
: ist dafür verantwortlich, dass der ausführbare Code eines Anwendungsprogramms in den residenten Bereich von CICS geladen wird, wenn er sich nicht schon dort befindet. Wenn es sich bei dem Programm um ein Command-Level-COBOL-Programm handelt, wird auch eine Kopie der Working Storage Section des Programms in die Dynamic Storage Area geladen. Diese Möglichkeit, eine einzige Kopie eines Programms und mehrere Kopien von Speicherbereichen verwalten zu können, ist Voraussetzung dafür, dass Programme reentrant geschrieben werden können. Die Program Processing Table (PPT) wird vom PCP benutzt, um festzustellen, welche Programme geladen sind.

File Control Program (FCP)
: steuert die Ein-/Ausgaben auf externe Daten; die Parameter für diese externen Daten werden in einer File Control Table (FCT) gehalten. Diese Parameter beinhalten beispielsweise die Satzlänge, die Schlüssellänge und die verwendete Zugriffsmethode (z. B. VSAM) sowie Informationen über Berechtigungen (Read oder Update).

System Management

CICS stellt von Haus aus einige Transaktionen für die Steuerung des Systems und seiner Ressourcen zur Verfügung. Die CICS-eigenen Transaktionen sind vier Zeichen lang und starten mit dem Zeichen „C".

Die beiden wichtigsten Transaktionen bzgl. System Management sind:

Master Terminal Transaction
: Mit der Master Terminal Transaction können Änderungen und Anpassungen im laufenden System vorgenommen werden. Oft können dadurch Probleme eingegrenzt und gelöst werden, ohne dass das CICS-System heruntergefahren werden muss. Auch für die Verwaltung von Datenbanken spielt die Master Terminal Transaction eine Rolle, beispielsweise für die dynamische Zuweisung und Wegnahme von Datenbeständen.

  Beispiele für Aktivitäten über das Master Terminal:

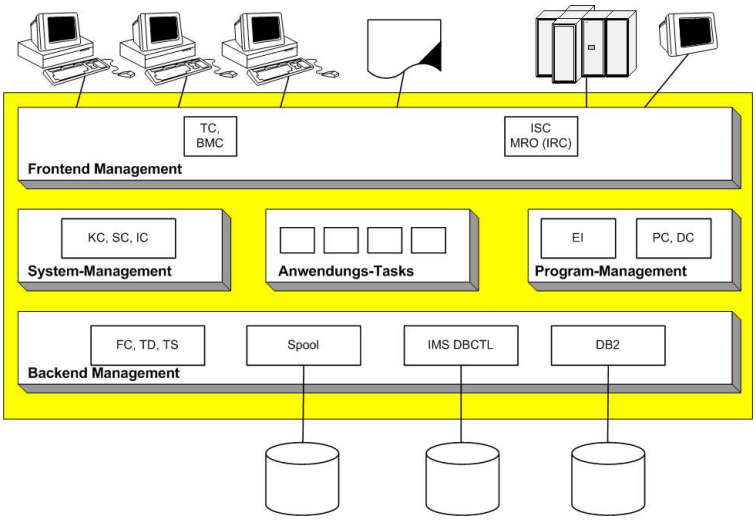
  - Steuerung der Anzahl von Tasks und der Anzahl gewisser Task-Arten
  - Entfernen eines Task aus dem System
  - Starten/Stoppen von Trace-Funktionen und Monitoring
  - Switchen von Dump Datasets
  - Open und Close von Interregion Communication Connections
  - Installieren neuer Programmversionen
  - Verwaltung des Message Routing

Resource Definition Online (RDO) Transaction

Wie der Name schon sagt, wird die Resource Definition Online Transaction verwendet, um dynamisch im laufenden CICS-System Ressourcen definieren zu können.

11.9 Funktionsarchitektur von CICS

Abbildung 11.5: CICS-Architektur im Überblick



Frontend Management

Das Frontend Management erfüllt die Kommunikationsaufgaben von und zu lokalen und entfernten Terminals und Systemen.

TC (Terminal Control)

Hier werden Terminals und Drucker sowie die Kommunikation mit anderen Systemen verwaltet. Je nach CICS-System werden hier vorhandene Betriebssystemkomponenten bzw. Erweiterungen genutzt (z. B. VTAM, BTAM, CMS).

BMS (Basic Mapping Support)

Hier wird die logische und physische Unabhängigkeit zwischen dem Nachrichtenstrom einer Anwendung und den physischen Geräteeigenschaften realisiert.

ISC (Inter System Communication)
: Kommunikation mit entfernten Systemen (anderes Betriebssystem lokal oder remote)

Multi Region Operation (MRO)
: Kommunikation mit anderen CICS Regions innerhalb eines Betriebssystems

System Management

Das System Management kümmert sich um die Task-Abwicklung und Speicherverwaltung. CICS wurde in der Vergangenheit des öfteren umstrukturiert, um die Erweiterungen im Speicherbereich (31-Bit-Unterstützung, 64-Bit-Unterstützung) nutzen zu können.

KC (Task Control)
: Aktivieren und Deaktivieren von Tasks, Einrichten einer Logical Unit of Work (LUW)

SC (Storage Control)
: Verwaltung eines CICS-Adressraums mit der Aufteilung in feste und dynamische Bereiche sowie in CICS- und benutzerspezifische Bereiche

IC (Interval Control)
: Zeitmanagement innerhalb von CICS

Anwendungs-Tasks

Hier werden die Benutzeranforderungen abgewickelt. Jeder Task erhält bei der Initiierung eine eindeutige Task-Nummer. Während der Ausführung eines Task werden aus dem dynamischen Speicher des Adressraums die Speicheranforderungen erfüllt.

Programm-Management

Das Programm-Management lädt die angeforderten Anwendungsprogramme, verwaltet diese im Speicher und übergibt bzw. übernimmt die Steuerung. Außerdem erfolgt hier die Fehlerbehandlung.

PC (Program Control)
: Laden von Programmen; Verwaltung des Speicherbereichs eines Programms und Übergabe der Programmsteuerung

DC (Dump Control)
: Abbruchkontrolle und Fehlerbehandlung

EI (Executive Interface)
: Kommandoschnittstelle für Anwendungsprogramme (API)

Backend Management

Das Backend Management verwaltet Daten auf internen und externen Datenträgern oder gibt die Steuerung an externe Datenhaltungs- oder Datenbanksysteme ab.

FC (File Control)
: Verwaltung von VSAM- und BDAM-Dateien

TD (Transient Data Control)
: Verwaltung von sequentiellen Zwischenspeicher-Queues; es geht dabei um adressrauminternen wie auch adressraumübergreifenden Datenaustausch.

TS (Temporary Storage Control)
: Verwaltung dynamischer Zwischenspeicherdaten intern im Speicher oder auf VSAM-Datenträger

Spool
: Nutzung der System-Reader- und System-Writer-Einrichtungen unter JES (Job Entry Subsystem) oder VSE-Power

DL/I (Data Language I)
: Anschluss einer IMS DB oder DL/I-Datenbank und Übergabe der Steuerung

DBRC (Database Recovery Control)
: Recovery-Kontrolle mit Hilfe von IMS DB

DB2
: Anschluss einer DB2-Lokation

11.10 CICS-Programme und Linking

Ein Programm ist die kleinste ersetzbare Einheit einer Anwendung. Programme werden übersetzt und gebunden (*linked*) und damit in ein ausführbares Programm überführt. Für die Erstellung des Programms können unterschiedliche Programmiersprachen verwendet werden.

CICS verfügt über mehrere Möglichkeiten, wie Programme in Verbindung mit CICS sich gegenseitig aufrufen können bzw. wie Programme, die außerhalb einer CICS-Umgebung laufen, CICS-Programme aufrufen können.

Programmaufrufe können sein:

Synchron
: Die Programmsteuerung wird erst nach Beendigung des aufgerufenen Programms an das aufrufende Programm zurückgegeben.

Asynchron
: Das aufrufende Programm setzt seine eigene Ausführung fort, nachdem ein anderes Programm aufgerufen wurde.

Synchrone Aufrufe

Es gibt zwei Möglichkeiten, wie ein Programm ein anderes Programm innerhalb einer CICS-Umgebung aufrufen kann:

```
EXEC CICS LINK
EXEC CICS XCTL
```

Mit LINK wird die Programmsteuerung an ein anderes Programm übergeben, und das aufrufende Programm bekommt die Kontrolle nach Beendigung des Programms zurück. Mit XCTL (Transfer Control) wird die Programmsteuerung an ein anderes Programm übergeben, jedoch geht die Kontrolle nicht mehr an das aufrufende Programm zurück.

In beiden Fällen wird die so genannte COMMAREA (Communication Area) genutzt, um Parameter zu übergeben und Rückgabewerte zurückzugeben.

Asynchrone Aufrufe

Ein asynchroner Programmaufruf eines anderen Programms in einer CICS-Umgebung erfolgt mit einem EXEC CICS START-Kommando, wobei Daten über die FROM Area ausgetauscht werden können. Alternativ kann auch *WebSphereMQ* eingesetzt werden.

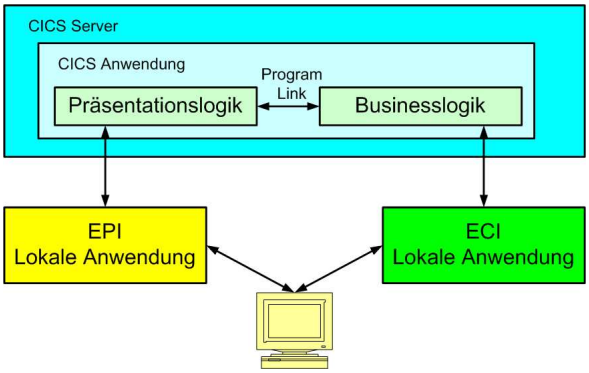
CICS-Aufrufe „von außen“

CICS bietet inzwischen zahlreiche Möglichkeiten, wie Programme „von außen“ ein CICS-Programm aufrufen können. Die wichtigsten Möglichkeiten sind das *External Call Interface* (ECI) und das *External Presentation Interface* (EPI).

External Call Interface (ECI)
: Hier wird ein CICS-Programm so aufgerufen, als ob der Aufruf über ein LINK-Kommando erfolgt wäre. Das ECI nutzt die COMMAREA für den Datenaustausch.

External Presentation Interface (EPI)
Hier wird ein CICS-Programm so aufgerufen, als ob es von einem 3270-Gerät aus aufgerufen wurde.

Abbildung 11.6: CICS, ECI und EPI



In Verbindung mit einem Webserver kann beispielsweise das CICS Transaction Gateway (CTG) eingesetzt werden, das sowohl das ECI als auch das EPI unterstützt.

Für den Einsatz in Verbindung mit Java gibt es ECI- und EPI-Schnittstellen zum CICS Transaction Gateway. Außerdem kann ein CICS-Programm auch über das Internet Inter-ORB Protocol (IIOP) aufgerufen werden.

11.11 CICS-Datenfluss

Mit den beschriebenen CICS-Systemmodulen und -Tabellen sieht der CICS-Datenfluss wie folgt aus:

1. Daten werden von einer Arbeitsstation eingegeben und mit der Eingabetaste betätigt.
2. Das TCP erkennt, dass Daten empfangen werden sollen, holt sich über die TCT den Gerätetyp und das verwendete Protokoll und sorgt dafür, dass die Daten in der Dynamic Storage Area abgelegt werden.
3. Das KCP überprüft die Transaktionskennung mit Hilfe der PCT und weist eine Task ID und ein Programm zu.
4. Das PCP sieht in der PPT nach und lädt ggf. das ausführbare Programm in den residenten Bereich von CICS. Wenn es sich um ein Command-Level-COBOL-Programm handelt, wird zusätzlich die Working Storage Section in die Dynamic Storage Area gebracht.

5. Die Kontrolle wird dann an das Anwendungsprogramm übergeben, das damit Zugriff auf die von der Workstation empfangenen Daten und ggf. auf die Working Storage Section bekommt. Wenn bei der Ausführung Ein-/Ausgaben auf externe Geräte gemacht werden sollen, wird die Steuerung zwischenzeitlich an das FCP mit entsprechenden Parametern aus der FCT übergeben, damit die I/Os durchgeführt werden können.

6. Wenn die Verarbeitung abgeschlossen ist und die entsprechende Antwort an die Workstation zurückgeschickt werden soll, wird das TCP mit der TCT wieder aktiviert, das dafür sorgt, dass die Daten richtig aufbereitet an das richtige Gerät geschickt werden.

7. Wenn der Task abgeschlossen ist, wird die Steuerung an CICS zurückgegeben, und alle belegten Speicherbereiche und anderen Ressourcen werden für die Nutzung wieder freigegeben.

11.12 CICS und E-Business

Dass große Unternehmen Millionen in ihre Datenhaltung auf dem Mainframe investiert haben, ist eine Binsenweisheit. Dass diese Daten dort gut aufgehoben sind, ist ebenfalls eine Tatsache, die immer weniger angezweifelt wird. Am E-Business führt kein Weg mehr vorbei, wenn man nicht ins Hintertreffen geraten will. Wie die Daten „Web-fähig" gemacht werden, ist die derzeit größte Herausforderung, der sich die Unternehmen stellen müssen. Es geht um die Integration von Host-Daten und Host-Anwendungen mit Web-Anwendungen, auf die mit Internet-Infrastrukturen zugegriffen werden kann.

Eine so genannte *Host Access Solution* bildet die Schnittstelle zwischen einer externen Anwendung und einer Host-Anwendung. Folgende Funktionalitäten sind gefordert:

- Entgegennahme des Request für den Host-Zugriff von der externen Anwendung
- Aufbau und Verwaltung einer Session zwischen der externen Anwendung und der Host-Anwendung
- Entgegennahme des Daten-Input von der externen Anwendung und Aufbereitung der Daten für den Input an die Host-Anwendung
- Entgegennahme der Output-Daten von der Host-Anwendung und Aufbereitung der Daten für die externe Anwendung

Das Ziel der traditionellen Host-Access-Lösungen ist der einfache und transaktionssichere Zugriff mit schnellen Antwortzeiten auf die Host-Daten.

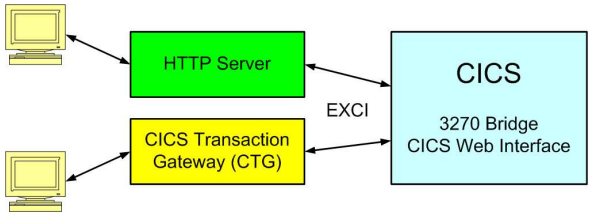
In der Vergangenheit bauten die entsprechenden Anwendungen auf dem 3270-Protokoll auf, d. h. sie sind zeichenorientiert und vom 3270-Protokoll abhängig, das von einem „dummen" Terminal ausgeht. Um Zugriffe mit Internet-Strukturen zu ermöglichen, muss eine Webschnittstelle zur Verfügung gestellt werden.

IBM hat bereits zahlreiche Erweiterungen der Mainframe-Systeme und Subsysteme implementiert, welche die Integration mit externen Anwendungen und vor allem auch mit E-Business-Anwendungen erleichtern. Ab dem CICS Transaction Server 1.3 beispielsweise wurden zwei wesentliche Schnittstellen implementiert:

- CICS Web Support
- 3270 Bridge Interface

CICS Web Support (CWS) ermöglicht die direkte Kommunikation mit CICS auf dem Mainframe von einem HTML Client aus, ohne dass ein Gateway dazwischengeschaltet werden muss.

Abbildung 11.7: CICS-Web-Support



Das *3270 Bridge Interface* ermöglicht die Entwicklung von Softwarekomponenten, die den Datenfluss in und aus CICS heraus abgreifen, bevor ein 3270-Datenstrom erzeugt bzw. als Eingabe erwartet wird. Das Bridge Interface arbeitet so, dass eine Softwarekomponente praktisch die Basic-Mapping-Support-Funktionalität ersetzt.

Eine Anmerkung zu BMS: Wenn eine Transaktion BMS aufruft, spezifiziert sie den Namen einer 3270 Screen „Map" sowie einen Satz von Feldern und Werten, die in Zusammenhang mit dem Screen genutzt werden sollen. Über die Map ist dann festgelegt, wo auf dem Bildschirm die Felder gesetzt sind, und BMS erzeugt daraus einen 3270-Datenstrom. BMS nimmt auch den 3270 Input entgegen und gibt diesen mit den entsprechenden Feldwerten an die Anwendung weiter. Eine CICS-Transaktion, die mit BMS arbeitet, „denkt" somit oft in Feld/Werte-Paaren und nicht in 3270-Datenströmen.

Das CICS Web Interface (CWI) ermöglicht einen direkten Zugriff von einer TCP/IP-Schnittstelle und HTML zu CICS.

Zwei Ansätze stehen hierbei zur Verfügung: Web-to-Host Gateways und Reengineering der Host-Anwendungen.

Web-to-Host Gateways

Eine Basislösung bilden hier zunächst einmal Programme, die in Form von Applets die klassische 3270-Schnittstelle emulieren. Die Session wird dabei als TN3270 über eine TCP/IP-Verbindung realisiert. Zwei Vorteile können hierbei ausgenutzt werden:

- Die Host-Anwendung kann 1:1 genutzt werden.
- Auf der Client-Seite muss keine Software (Terminal-Emulation) installiert werden.

Die entsprechend angebotenen Lösungen (z. B. Host-on-Demand von IBM) sind heute funktionell den auf den Clients zu installierenden Emulationen weitgehend ebenbürtig.

Diese Lösung eignet sich vor allem dann, wenn die Benutzer bereits mit der 3270-Schnittstelle vertraut sind und kein GUI brauchen oder wollen.

Reengineering einer Host-Anwendung

Eine erweiterte Form bilden die Lösungen, die den Endbenutzern einer zeichenorientierten 3270-Anwendung eine grafische Benutzeroberfläche anbieten, die über die zeichenorientierte Schnittstelle „übergestülpt" wird. Die am weitesten fortgeschrittenen Lösungen konvertieren den 3270-Datenstrom bzw. den Bildschirminhalt in andere Datenobjekte, die von einer komplett neuen Web-Anwendung genutzt werden können.

Dieser Ansatz ermöglicht das Erstellen intuitiver Schnittstellen, die die Host-Anwendungen für eine breitere Nutzerschicht verfügbar machen. Es können dann auch zusätzliche Funktionalitäten in Form von JavaScript, ActiveX oder Java eingebaut werden, um Daten zu manipulieren.

Allerdings besteht dann eine Verknüpfung mit den Host Screens, so dass Änderungen an den Host Screens auch Änderungen an der Benutzerschnittstelle notwendig machen. Wenn eine Anwendung mit Hunderten von Screens arbeitet, ist diese Lösung in der Regel nicht optimal, da der Entwicklungsaufwand zu groß ist und die Wartbarkeit zu wünschen übrig lässt.

Hinter den meisten traditionellen Lösungen steht der 3270-Datenstrom als Kommunikationsschnittstelle zwischen der Host-Anwendung und dem Web-to-Host Gateway. Das Web-to-Host Gateway realisiert dann ein so genanntes „Screen Scraping" oder „Stream Scraping", um Informationen aus dem 3270-Bildschirminhalt bzw. dem Datenstrom herauszuziehen.

Ob dieses Scraping auf der Client-Seite oder auf der Server-Seite passiert, spielt in diesem Zusammenhang eine untergeordnete Rolle, es handelt sich prinzipiell um

den gleichen Vorgang: Es werden Zeilen- und Spaltenkoordinaten verwendet, um die zu verarbeitenden Daten zu identifizieren. Dies ist natürlich nicht besonders wartungsfreundlich, wenn eine Host-Anwendung geändert wird.

Der Vorteil dieser Lösung ist eine „GUIfizierung" einer Benutzeroberfläche, ohne dass die Host-Anwendung selbst angefasst werden muss.

XML und CICS

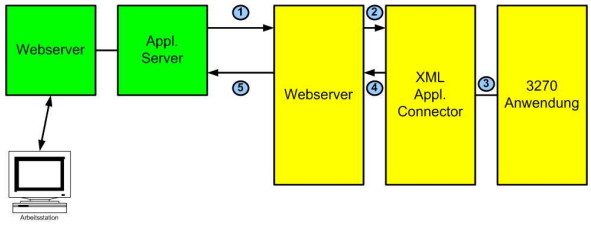
Das Problem des Screen Scraping in Verbindung mit einer Datenerkennung ist gelöst, wenn diese Daten in XML konvertiert werden. Die XML-Daten können dann sowohl für die Präsentation der Daten als auch für die Weiterverarbeitung in anderen Anwendungen verwendet werden. XML bietet eine strukturierte und standardisierte Schnittstelle für den Datenaustausch zwischen Host-Anwendungen und anderen XML-fähigen Anwendungen.

Die XML-Syntax kann nicht nur problemlos gelesen und interpretiert werden, sondern auch einfach von einer Anwendung geparsed oder direkt in eine Datenbank importiert werden. Die meisten Datenbanken (einschließlich beispielsweise DB2) unterstützen bereits den Import und Export von XML-Daten.

Der Vorteil dabei ist, dass die Informationen absolut unabhängig davon sind, wie und wo sie angezeigt bzw. weiterverarbeitet werden sollen. Mit Hilfe einer ebenfalls standardisierten XML Style Language Transformation (XSLT) können XML-Dokumente in ein beliebiges anderes Format konvertiert werden.

Die Basisarchitektur einer Konfiguration, die mit einem XML Application Connector arbeitet, sieht somit wie in Abbildung 11.8 aus.

Abbildung 11.8: Basisarchitektur einer Konfiguration mit einem XML Application Connector



1. Die externe Anwendung sendet einen HTTP Request an den Host. Der Request und die Input-Daten können beispielsweise als Query String oder als XML-Daten übermittelt werden.

2. Der Host nimmt den Request entgegen und prüft, ob der Zugriff erlaubt ist. Wenn die Autorisierung gegeben ist, wird der Request an den XML Application Connector zur Weiterverarbeitung übergeben.

3. Der XML Application Connector startet die Transaktion auf dem Host, übergibt die entsprechenden Input-Daten und nimmt die Antwort der Transaktion entgegen.

4. Der XML Application Connector konvertiert die Daten für die Ausgabe in XML.

5. Die Daten werden an die externe Anwendung als HTTP Response zurückgegeben.

Flexibilität

Da der XML Application Connector XML für die Kommunikation zur externen Anwendung verwendet, werden keine Bildschirmfelder zur Identifikation benötigt. Dadurch ist die Flexibilität deutlich größer. Die Abhängigkeit zwischen der CICS- und der Client-Anwendung wird gelöst. Wenn in der Host-Anwendung die Feldlokation geändert wird, ist die Client-Seite nicht davon betroffen, da mit XML die Elemente über Tags und nicht über die Lokation der Felder identifiziert werden.

Skalierbarkeit

Da die CICS-Daten direkt von und zu XML konvertiert werden, entfällt das Parsen des 3270-Datenstroms. Da der XML Application Connector auf dem Host läuft, entsteht auch kein Overhead für die TN3270-Konvertierung, SNA Stacks oder HLLAPI-Schnittstellen.

Security

Der Aufruf und die Verarbeitung einer CICS-Transaktion über den CWS und den XML Application Connector ist genauso sicher wie der Aufruf von einem CICS Terminal aus. Der XML Application Connector arbeitet mit den Standard-Mainframe- und Internet Security-Methoden. Das bedeutet:

Der UserID/Passwort-Schutz mit RACF, ACF2 oder TopSecret kann genutzt werden, um die Zugriffe auf die Ressourcen zu kontrollieren.

Die Client-Authentifizierung über CWS oder USS stellt sicher, dass Clients und externe Applications autorisiert sind, sich mit der Host-Anwendung zu verbinden. Verschlüsselung mit SSL schützt den Datenstrom zwischen Webserver und der Client-Anwendung.

Die Verbindung ist oft sogar sicherer: Wenn von einem 3270 Terminal oder über eine Emulation auf Host-Anwendungen zugegriffen wird, geschieht das in vielen Fällen ausschließlich über UserID/Passwort-Schutzmechanismen. Oft wird zwischen Terminal/Emulation und Host im Klartext übertragen. Da der XML Application Connector mit einem Webserver zusammenarbeitet, der die Authentifizierung der Clients (beispielsweise mit Zertifikaten) verlangt und die Daten mit SSL verschlüsselt übertragen kann, ist diese Form sogar sicherer als eine herkömmliche 3270-Emulation für den Host-Zugriff.

Performance

Aufpassen muss man vor allem bei großen Transaktionsraten in Zusammenhang mit der Performance, da das Verpacken und Entpacken der XML-Daten natürlich auch Overhead verursacht. Viele ehrgeizige Projekte in diesem Umfeld sind daran gescheitert.

Kapitel 12

UNIX System Services

12.1 Der „offene“ Mainframe

UNIX System Services (USS) hat zum Ziel, die Vorteile von intelligenten Arbeitsstationen, die Flexibilität offener Systeme und die Stärken des klassischen MVS in Einklang zu bringen.

USS bedeutet einen eindeutigen Schritt in Richtung offener Systeme. Darunter ist vor allen Dingen eine POSIX-Konformität zu verstehen. Erst mit USS wird der Einsatz eines Webservers und der Einsatz von WebSphere unter z/OS möglich.

Dass der Schritt unbedingt notwendig war, um das Überleben des Mainframe zu sichern, ist inzwischen unbestritten. Neue Anwendungen auf dem Mainframe, beispielsweise auf der Basis von J2EE in Zusammenwirkung mit den bestehenden Legacy-Anwendungen, sind zukunftsweisend. Die Unterstützung und Konformität mit den entsprechenden, im Folgenden aufgeführten Standards ist unbedingte Voraussetzung, um in einer modernen Umgebung eine Integrationsrolle zu übernehmen.

IEEE-Arbeitsgruppe 1003

Die mit OMVS (Vorgängervariante von USS) eingeführten APIs basieren auf den Definitionen 1003.1 und 1003.2 der IEEE-Arbeitsgruppe 1003. Diese Arbeitsgruppe wurde ins Leben gerufen, um die verschiedenen UNIX-Derivate zu konsolidieren. Die Definitionen, die sowohl die C-Programmierschnittstelle als auch eine interaktive Umgebung einschließen (eine so genannte Shell), sind besser bekannt unter dem Namen POSIX.

POSIX

Portable Operating Systems Interfaces, ursprünglich noch mit dem Zusatz „for UNIX", der jedoch heute weggelassen wird, da die Schnittstellen systemunabhängig definiert sind; allgemein kann gesagt werden, dass eine POSIX-konform programmierte Anwendung auch problemlos und ohne großen Migrationsaufwand unter UNIX System Services läuft.

SPEC1170

Daneben begannen weitere Gruppierungen, sich mit dem Thema Portabilität auseinanderzusetzen. Sie übernahmen die von der POSIX-Arbeitsgruppe erarbeiteten Definitionen und untersuchten außerdem zahlreiche UNIX-Anwendungen nach häufig benutzten Funktionen. Auf der Basis dieser Untersuchungen wurden Schnittstellen definiert, mit welchen diese Funktionen plattformunabhängig aufgerufen werden können. Diese über 1.100 definierten Funktionsaufrufe wurden mit dem Namen *SPEC1170* versehen.

X/Open Portability Guide (XPG)

Eine weitere Organisation, die X/Open, beschäftigt sich mit der Definition von Standardschnittstellen. Sie veröffentlicht diese Schnittstellen als *X/Open Portability Guides*, abgekürzt XPG. Ein Teil dieser Portability Guides definiert die von einem System erwarteten Schnittstellen, um als X/Open-konform zu gelten. In XPG4 sind die oben erwähnten SPEC1170 enthalten.

XPG4.2

X/Open überprüft die UNIX-Funktionen von Kandidaten, die das Branding erlangen wollen, anhand des X/Open Portability Guide 4.2, um sicherzustellen, dass die definierten Schnittstellen wie vorgesehen implementiert sind.

HW/SW-unabhängig

All diese Definitionen setzen weder eine bestimmte Hardware noch ein bestimmtes Software-Produkt voraus. Die Definitionen beschreiben lediglich, *was* zu implementieren ist, und nicht, *wie* es zu implementieren ist.

MVS und UNIX

Mit MVS/ESA SP 5.2.2 hat IBM im November 1995 die Bedingungen für das so genannte „Base Branding" erfüllt. Mit OS/390 Version 3 wurde inzwischen das „Full Branding" erreicht, auch verwendet in Zusammenhang mit dem

Begriff „Single UNIX" oder auch „UNIX95". Die wesentlichen Erweiterungen zu UNIX98 wurden in OS/390 R7 bereits integriert.

USS ist keine Emulation und auch keine Migration eines anderen UNIX-Systems, sondern die konsequente Umsetzung der oben erwähnten Standardschnittstellen.

12.2 USS im Überblick

Es geht in diesem Kapitel weniger um die Basisfunktionen eines UNIX-Systems, da es dazu jede Menge Literatur gibt. Vielmehr sollen die Unterschiede und Gemeinsamkeiten aufgezeigt werden, um den UNIX-Spezialisten die Merkmale von MVS und den MVS-Spezialisten die Merkmale von UNIX näherzubringen.

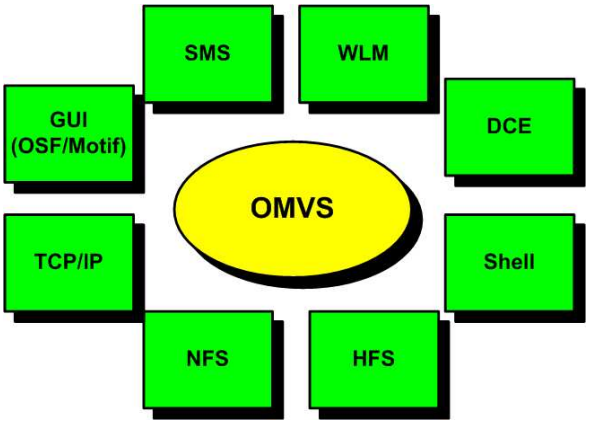


Abbildung 12.1: Basisfunktionen von UNIX System Services

Der Kern von UNIX System Services führt auch heute noch die Bezeichnung OMVS. OMVS steht für *OpenMVS* und wurde zu Beginn der Aktivitäten als Kürzel für eine Started Task vergeben. Das heißt, dass in den Anfangszeiten die UNIX-Schnittstelle als Started Task gestartet und gestoppt werden konnte. Inzwischen gehört der USS-Kernel zum System und wird beim Systemstart als Systemadressraum eingerichtet. USS ist somit inzwischen Bestandteil des Betriebssytems. Das heißt, dass es ein z/OS ohne USS nicht mehr gibt.

Der Name wurde nach der Bezeichnung OMVS noch zweimal geändert. Zunächst wurde der Begriff *OpenEdition* eingeführt, danach, nachdem der Name „UNIX" nicht mehr Markenzeichen eines Unternehmens war, sondern von X/Open verwaltet und für Systeme vergeben wurde, die den offenen Standards entsprechen, wurde die Systemkomponente in UNIX System Services (USS) umgetauft.

Die Basisfunktionen von UNIX System Services nach Abbildung 12.1:

GUI (Graphical User Interface)

Ein Standard in UNIX-Umgebungen ist unter dem Begriff X-Window und OSF/Motif bekannt. Dieser Standard wird auch in Verbindung mit UNIX System Services vollumfänglich unterstützt.

SMS (System Managed Storage)

System Managed Storage kann auch in Verbindung mit dem Hierarchical File System von USS genutzt werden. Die Daten für das System und vor allem auch für die Benutzer von USS werden in einem UNIX-spezifischen, hierarchisch organisierten Filesystem gehalten. Für die Benutzer wird unter dem Verzeichnis **/u** (leider nicht **/home**, wie in allen anderen UNIX-Derivaten) ein benutzereigenes Home-Directory eingerichtet, das in der Praxis meist von einem Feature mit der Bezeichnung „Automount" verwaltet wird. Das hat zur Folge, dass die Verzeichnisse mit der entsprechenden HFS-Datei, die den jeweiligen Container für die Daten bilden, nur dann am Filesystem gemountet sind, wenn auch damit gearbeitet wird.

Sind die HFS-Dateien inaktiv, können sie wie jede andere MVS-Datei auch vom Hierarchical Storage Manager des SMS ausgelagert werden.

WLM (Workload Manager)

Einer der wesentlichen Unterschiede zwischen UNIX und MVS ist die Art und Weise, wie Workloads verwaltet und Prioritäten gesetzt werden. Der Workload Manager des MVS wird mit Zielen versorgt, die den Service-Level-Begriffen für Workloads entsprechen.

Das bedeutet, dass beispielsweise für interaktive Arbeit Responsezeiten definiert werden, die das System einzuhalten versucht. Vergleichbar den Responsezeiten von TSO werden somit auch für die interaktive Arbeit mit der Shell Reponsezeiten definiert. Es gibt derzeit kein anderes UNIX-Derivat, das so etwas kann.

DCE (Distributed Computing Environment)

Der Standard DCE der Open Software Foundation wird ebenfalls vollumfänglich unterstützt. Allerdings spielt das in der Praxis heute nicht mehr die große Rolle, da DCE nicht die Wertigkeit hat, die man diesem Standard in der Vergangenheit vorausgesagt hat. Vor einigen Jahren noch dachte man, dass man in verteilten Umgebungen um DCE nicht herumkäme. Inzwischen gibt es zahlreiche Alternativen, die vor allem weniger komplex sind als DCE.

Shell

Eine Shell ist die interaktive Schnittstelle für ein UNIX-System, vergleichbar dem TSO innerhalb des MVS. Neben UNIX-spezifischen Shells steht in Verbindung mit USS vor allem auch eine ISPF-Shell zur Verfügung. Darauf wird im weiteren Verlauf dieses Abschnitts noch eingegangen.

HFS/NFS

Die Daten in einem UNIX-System werden in einem hierarchisch organisierten Filesystem gehalten, dem HFS. Es kann mit dem als De-facto-Standard bekannten ***Network File System***, das von Sun entwickelt wurde, über Netzwerk- und Plattform-Grenzen ausgeweitet werden. Dieses NFS wird von fast allen Betriebssystemen unterstützt. So kann man beispielsweise von USS aus auf einen bestimmten Zweig des HFS zugreifen und befindet sich physisch plötzlich im Filesystem eines Sun Solaris-Systems oder eines Windows-Systems. Umgekehrt kann man beispielsweise von einem HP-UX aus physisch auf Daten im HFS von UNIX System Services zugreifen oder gar, über einen so genannten „External Link", auf eine MVS-Datei. Die Benutzer merken davon in der Regel nichts.

TCP/IP

Wenn es um die Vernetzung unterschiedlicher Plattformen geht, kommt man um TCP/IP nicht herum. Es gibt in der Praxis heute kein z/OS-System mehr, das nicht für TCP/IP konfiguriert ist. Gerade der Einsatz von TCP/IP setzt wiederum voraus, dass UNIX System Services implementiert wird, da TCP/IP inzwischen nur noch in Verbindung mit USS funktioniert.

12.3 Die USS Shells

Was ist eine Shell?

Die Anwender eines Computers möchten auf vielfältige Weise mit ihrem Rechner arbeiten. Sie möchten Programme und Kommandos starten, Dateien kopieren, anderen Anwendern eine Nachricht schicken usw. Damit dies möglich ist, wird eine entsprechende Schnittstelle benötigt. Auf einem MVS-System ist diese Schnittstelle das TSO. In Unix-Systemen heißt diese Schnittstelle „Shell", zu Deutsch „Schale", weil sie den Unix-Kern wie eine Schale umgibt.

Bei der Shell handelt es sich um einen Kommandointerpreter, d. h. die eingegebenen Kommandos werden dem Betriebssystem übergeben und ausgeführt. Die funktionale Trennung zwischen Kern und Schale ist ein wichtiges Konzept von Unix. Die Benutzerschnittstelle wird dadurch unabhängig vom Betriebssystem, kann leicht ersetzt und erweitert werden. Das hat seine Vor- und Nachteile. In der Tat gibt es für Unix eine Reihe verschiedener, austauschbarer Shells. Wir kommen darauf noch zurück.

Die USS Shell und die Utilities sind eine Portierung der ***InterOpen POSIX Shell and Utilities*** von Mortice Kern Systems, einer kanadischen Firma, und wurden von IBM lizensiert. Beim Aufruf der USS Shell erscheint neben dem IBM Copyright auch das Copyright von MKS. Die Shell basiert auf UNIX System V mit weiteren Features aus der Korn Shell.

Auf einem Unix-System werden Aktivitäten in Form weitgehend unabhängiger Einheiten, der Prozesse, verwaltet. Die Shell ist ein Prozess unter vielen anderen gleichzeitig auf dem System aktiven Prozessen. Sie wird in der Regel gestartet, nachdem ein Benutzer sich am System angemeldet hat. Unter MVS wird die Shell durch das TSO-Kommando OMVS gestartet. Welche Shell dabei aufgerufen wird, ist in einem speziellen RACF-Segment auf Benutzerebene definiert. Jeder Benutzer der OMVS-Shell muss somit auch im RACF definiert sein. Per Default stehen in einem z/OS-System die Korn Shell und die C-Shell zur Verfügung.

Eintragung im RACF

Bevor ein Anwender mit der USS-Shell arbeiten kann, muss er vom RACF her entsprechend eingerichtet werden.

In einem speziellen OMVS-Segment muss dafür eine UserID, eine Default-Gruppe und die Default-Shell eingetragen sein.

Beispiel für das Einrichten eines OMVS-Segments für einen bestehenden User:

```
ALTUSER UHOX OMVS(UID(234) HOME('/u/uhox?) PROGRAM(?/bin/sh?))
```

Das File passwd, das in UNIX-Systemen verwendet wird, gibt es im z/OS nicht.

Zugriff auf die USS-Shell

Auf die USS-Shell kann auf unterschiedliche Weise zugegriffen werden:

- mit dem OMVS-Kommando unter TSO/E. Dann handelt es sich um eine 3270-Schnittstelle.
- mit dem rlogin-Kommando als asynchrone Terminalschnittstelle
- mit dem telnet-Kommando als asynchrone Terminalschnittstelle, die in UNIX-Systemen als Standard verwendet wird.

Mit dem TSO-Kommando OMVS wird eine UNIX-spezifische Shell über die 3270-Schnittstelle aufgerufen. Diese Shell hat einige Eigenheiten im Vergleich mit einer Telnet Session. So gibt es einige Shell-Kommandos, die ausschließlich unter dieser Umgebung zur Verfügung stehen: Beispielsweise kann mit dem Kommando oedit direkt der ISPF-Editor aufgerufen werden. In der Shell gibt man einfach das Kommando oedit und als Argument das File an, das editiert werden soll. Wenn es das File noch nicht gibt, wird es angelegt.

Beispiel:

```
oedit .profile
```

Startparameter für die OMVS-Shell

Eine weitere wichtige Eigenschaft besteht darin, dass beim Start der OMVS-Shell Parameter mitgegeben werden können. Hier nur die wichtigsten: Mit dem Parameter CONVERT kann eine Codepage für eine Zeichenkonvertierung mitgegeben werden. Der Parameter NOSHAREAS bewirkt, dass die Shell in einem separaten Adressraum aufgebaut wird. Der Standard ist SHAREAS, d. h., dass die Shell im TSO-Adressraum aufgebaut wird, was im Normalfall auch sinnvoll ist. Mit dem Parameter SESSIONS können von vornherein mehrere Sessions gestartet werden, zwischen denen dann analog wie unter ISPF hin und her geswitcht werden kann.

Eine wichtige Funktion besitzt der Parameter ESCAPE. In einer Telnet-Session werden oft Tastenkombinationen eingeben, wie beispielsweise Ctrl-C, um einen laufenden Vorgang abzubrechen. In einer 3270-Umgebung funktionieren diese Tastenkombinationen nicht, da das 3270-Protokoll nicht zeichenorientiert, sondern mit einem Terminal-Puffer arbeitet. Die Kommandos werden über die Tastatur in einen Puffer eingegeben und erst mit dem Auslösen der Daten-Freigabe-Taste an das System übermittelt. Im Gegensatz dazu werden die Telnet-spezifischen Eingaben zeichenweise direkt an das System übermittelt und interpretiert.

Um nun auch unter OMVS ein Ctrl-C, Ctrl-D oder sonstige Escape-Sequenzen eingeben zu können, muss ein Escape-Zeichen definiert werden. Dies kann beim Start von OMVS mit dem ESCAPE-Parameter mitgegeben werden. Man wählt sinnvollerweise ein Zeichen auf der Tastatur, das bei der normalen Arbeit nicht benötigt wird. Ein Beispiel dafür ist das Paragraphenzeichen (§).

Beispiel für den Start von OMVS mit Parametern:

```
OMVS ESCAPE('$') SESSIONS(3)
```

Eingabe von TSO-Kommandos

Aus der OMVS-Shell heraus können auch TSO-Kommandos abgesetzt werden. Hierfür setzt man vor das TSO-Kommando einfach das Shell-Kommando tso.

Beispiel:

```
tso LISTC
```

Auch die in Verbindung mit USS neuen TSO-Kommandos wie OGET oder OPUT werden oft aus der Shell heraus abgesetzt.

Eine weitere häufige Nutzung ist die Eingabe von tso ISHELL, um aus OMVS die ISPF-Shell aufzurufen. Sobald man sich in der ISHELL befindet, kann mit Split Screen (PF2) im „ganz normalen" ISPF gearbeitet werden.

Anpassen der Shell

Um die Shell an eigene Belange anzupassen, gibt es unterschiedliche Möglichkeiten. Die wichtigste ist die Anpassung der Datei profiles, entweder für alle Benutzer im /etc-Verzeichnis oder pro individuellem Benutzer im File .profile im jeweiligen Home-Verzeichnis.

In diesem File werden vor allem Variablen gesetzt. Der Inhalt einer Variablen kann mit einem Dollarzeichen und dem Variablennamen abgefragt werden. In der Variablen HOME steht beispielsweise das aktuelle Home-Verzeichnis. Der Inhalt der Variablen HOME kann mit dem Kommando

```
echo $HOME
```

auf den Bildschirm geholt werden. In dem Profile-File werden in erster Linie Variablen gesetzt. Die Variablen werden bei der Definition auch gleich exportiert. Das hat zur Folge, dass beim Aufruf einer Subshell die Variablen auch dorthin exportiert werden. Die wichtigste Variable ist die PATH-Variable. Sie entspricht einer JOBLIB/STEPLIB und der Linklist-Konkatenation im MVS. Über die Variable PATH wird definiert, in welchen Verzeichnissen nach ausführbaren Programmen gesucht wird.

Beispiel für ein Profile-File:

```
export PATH=$PATH:/java/bin:/usr/prj1/bin
export PS1='LOGNAME:$PWD>'
echo 'profile executed'
```

Die ISPF-Schnittstelle (ISHELL)

Mit Hilfe der ISPF-Oberfläche (ISHELL) können Anwender und Administratoren viele Aufgaben und Funktionen über ISPF-Menüs und ISPF-Dialoge abwickeln.

ISPF-Funktionen für Anwender:

- Verwaltung von Directories
- Editieren und Browsen von Files mit ISPF
- Kopieren von Files/Datasets
- Verwaltung normaler Files und FIFO Files
- Symbolic Links, External Links und Hard Links verwalten

Dies hat den großen Vorteil, dass Benutzer, die bisher nur mit dem MVS gearbeitet haben, sofort mit UNIX System Services arbeiten können, ohne entsprechendes Wissen über UNIX-Kommandos und vor allem die zahlreichen Optionen dieser Kommandos zu haben.

Dennoch ist es sinnvoll, dass sich MVS-Spezialisten auch mit den Möglichkeiten der UNIX-Kommandos auseinander setzen, da es viele Aufgaben gibt, die mit den entsprechenden UNIX-Kommandos effizienter gelöst werden können als mit der ISPF-Shell.

Beispiel: Inhalte (HTML-Files, Bilder etc.) für den z/OS Webserver sind im Hierarchical Files System von UNIX System Services abgelegt. Nun sollen die Permission Bits dieser Files inklusive der Files in Unterverzeichnissen so gesetzt werden, dass sie von allen gelesen werden dürfen (d. h., das OTHER-Bit soll auf read gesetzt werden). Wer UNIX kennt, erledigt das mit einem einzigen Kommando. Wenn es dagegen mit der ISPF-Shell erledigt werden soll, würde es bei über 100 Files eine Stunde dauern.

Zusätzliche ISPF-Funktionen für Administratoren:

- Montieren und Demontieren von File-Systemen
- Liste von Usern mit diversen Sortierkriterien erstellen
- USS-User und -Gruppen verwalten
- Erzeugen von Character Special Files
- Einrichten von File-Systemen etc.

Auch für Administratoren ist die ISPF-Shell eine große Hilfe. So können beispielsweise Filesysteme menügeführt abgefragt, gemountet und unmountet werden. Die Benutzer- und Gruppenverwaltung kann damit ebenso wie zahlreiche weitere Dinge erledigt werden.

12.4 Das Hierarchical File System (HFS)

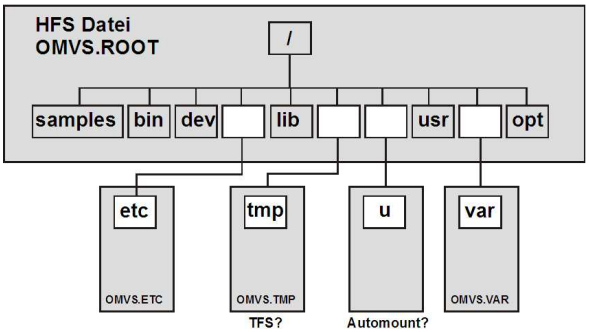
Ein Posix File System ist hierarchisch strukturiert. Alle Files befinden sich in irgendeinem Directory. Jedes Directory ist wiederum an einem höheren Directory aufgehängt, mit Ausnahme des Root Directory auf der obersten Stufe. Die Länge der File-Namen (incl. Pfaddefinition) ist mit 1023 Zeichen praktisch unbeschränkt. Dies gilt auch für die Anzahl der Hierarchien, die lediglich durch den verfügbaren Plattenplatz beschränkt ist.

Das USS-Filesystem ist vom Prinzip her wie jedes andere UNIX-Filesystem aufgebaut. Dies betrifft beispielsweise Standardverzeichnisse, die in allen UNIX-Derivaten anzutreffen sind und auch jeweils die gleiche Funktion und die gleichen Inhalte haben.

UNIX-Freaks fühlen sich hier gleich zuhause. Der wichtigste Unterschied ist, dass die Benutzer (leider) im Verzeichnis u und nicht wie in anderen UNIX-Systemen üblich im Verzeichnis home eingeklinkt werden.

Ein weiterer Unterschied ist, dass die Filesysteme in MVS-Dateien organisiert sind, die als Behälter für die Files im Hierarchical File System dienen. Es gibt immer ein Root-Filesystem, das in unserem Beispiel in einer MVS-Datei mit dem Dateinamen OMVS.ROOT untergebracht ist.

Abbildung 12.2: Aufbau des HFS



Das Root-Filesystem kann so organisiert werden, dass Zweige des Filesystems, die im täglichen Betrieb verändert werden, aus dem Root-Filesystem herausgenommen werden. Diese werden in separate MVS-Dateien abgelegt, die dann auch aus Performance-Gründen und anderen Überlegungen heraus auf unterschiedliche Platten gelegt werden können.

Das Root-Verzeichnis kann dann mit dem Attribut read-only versehen werden. Das hat den großen Vorteil, dass Files in dieser HFS-Datei auch nicht aus Versehen verändert oder gelöscht werden können.

Auch für den Superuser gilt dieser Schutz. Wenn ein Superuser versucht, ein File in einer Read-only-Datei zu ändern oder zu löschen, bekommt er eine entsprechende Nachricht. Er kann dann, wenn er die Änderung wirklich vornehmen will, die HFS-Datei dynamisch auf Read/Write ändern, um seine Arbeit zu erledigen. Danach wird das Filesystem wieder auf Read-only gesetzt.

Bezüglich der Inhalte der Verzeichnisse unter USS können teilweise Vergleiche zum MVS gezogen werden. So entspricht beispielsweise das Verzeichnis /samples der MVS-Datei SYS1.SAMPLES, das Verzeichnis /bin entspricht der SYS1.LINKLIB, da darin die Basismodule des Systems untergebracht sind. Es handelt sich um so genannte Binärfiles, die den MVS-Lademodulen entsprechen. Das /lib-Verzeichnis entspricht den MVS-Zusatzbibliotheken wie beispielsweise der SYS1.COBLIB.

Zu dem /dev-Verzeichnis gibt es keine Entsprechnung, da die Methoden, wie Geräte verwaltet und angesprochen werden, im MVS und in einem UNIX-System vollkommen unterschiedlich sind. Sämtliche Geräte in einem UNIX-System werden als File im Verzeichnis /dev repräsentiert. Die Kommunikation mit einem Gerät erfolgt dann auf die gleiche einfache Art und Weise wie die Kommunikation mit einem File.

12.5 Zusammenspiel zwischen USS und MVS

Zwischen MVS und USS gibt es eine sehr enge Beziehung. Es sind ja keine unterschiedlichen Systeme, sondern lediglich unterschiedliche Schnittstellen, die in einem gemeinsamen Betriebssystem zur Verfügung gestellt werden.

So können beispielsweise Files aus dem Hierarchical File System problemlos mit dem MVS-Dateisystem ausgetauscht werden. Das geht sehr einfach über die ISHELL. Es wird dort zwischen Data Sets (MVS-Dateisystem) und Files (USS-HFS) unterschieden.

Außerdem stehen die TSO-Befehle OGET, OGETX, OPUT, OPUTX, OCOPY zur Verfügung, mit denen Files in PO- bzw. PS-Dateien kopiert werden können und umgekehrt.

Auch die JCL unterstützt USS und das USS Filesystem. Ein großer Vorteil, wenn es darum geht, irgendeine UNIX-Aktivität in einen Batchjob einzubinden. Ein Batchjob kann dann problemlos mit den klassischen MVS-Automatisationsprodukten (Jobablaufsteuerung, Systemautomation) verwaltet werden.

```
//TPS0001  JOB  (TPS,10365),GREIS,MSGCLASS=X
//BPXBATCH EXEC PGM=BPXBATCH,
//         PARM=?SH ls -l;date?
//STDOUT   DD   PATH='/u/tps000/ausgabe',
//         PATHOPTS=(OCREAT,OWRONLY),
//         PATHDISP=KEEP
//STDERR   DD   DISP=SHR,DSN=TPS000.OE.DATA(ERR)
```

Das Programm, das aufgerufen wird, heißt BPXBATCH. Die Eingabe kann entweder über den PARM-Parameter der JCL oder über die STDIN DD-Anweisung mitgegeben werden. Die Ausgabe wird über die STDOUT DD-Anweisung entweder in das Hierarchical File System oder in eine MVS-Datei geschrieben. Mögliche Fehlernachrichten werden über die STDERR DD-Anweisung ausgegeben.

12.6 USS und Security

Auch in Verbindung mit Security wird versucht, die Vorteile des Mainframe in die UNIX-Welt einzubringen. Das betrifft zunächst einmal die RACF-Komponente des z/OS Security Server.

Ein klassisches UNIX/Linux-System verwaltet ein User Definition File mit der Bezeichnung /etc/passwd und einem optionalen /etc/shadow. Diese Files gibt es in UNIX System Services nicht. Vielmehr werden diese Informationen unter z/OS von RACF verwaltet.

Sowohl UNIX/Linux als auch das MVS erfordern eine Identifizierung mit einer Benutzerkennung und eine Verifizierung mit Passwort. UNIX verwendet intern eine

numerische Kennung, die als UID bezeichnet wird. Eine einem Benutzer zugewiesene Benutzerkennung wie beispielsweise **UHOX** wird mit dieser numerischen Kennung verknüpft. Diese Verknüpfung findet über das **/etc/passwd**-File oder im Falle von USS in Verbindung mit dem RACF statt.

UNIX ermöglicht das Zuweisen derselben numerischen Kennung zu mehreren Benutzernamen. Das sollte unbedingt vermieden werden, da Benutzer mit gleicher numerischer Kennung von den Berechtigungen her als ein und derselbe Benutzer identifiziert werden. Das heißt, Benutzerkennungen mit der gleichen numerischen UID haben volle Zugriffsberechtigung auf ihre jeweiligen Ressourcen und können auch gegenseitig Prozesse killen.

Auch in UNIX gibt es ein Gruppenkonzept. User mit gleichen Anforderungen bzgl. Security werden in Gruppen zusammengefasst. Jeder User gehört einer Default-Gruppe an, die in **/etc/passwd** definiert ist.

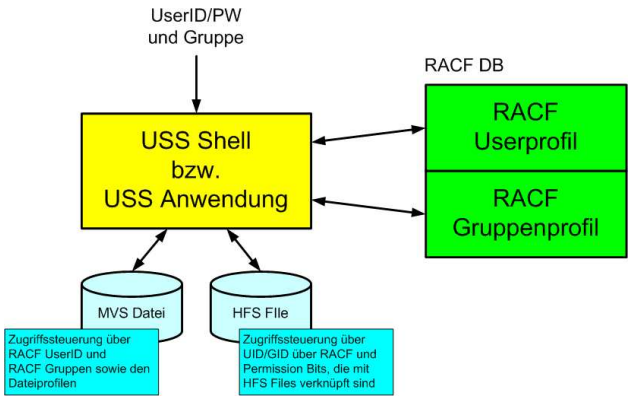
In Verbindung mit USS werden die Informationen im RACF-Profil hinterlegt. Die Zuweisung einer UID, die Zuweisung eines Home Directory und die Definition der Shell, die beim Start aktiviert wird, werden dabei in einem speziellen OMVS-Segment eingetragen.

Auch für die RACF-Gruppen gibt es ein OMVS-Segment, in das eine Group ID (GID) eingetragen wird.

Benutzerverwaltung und Zugriffsrechte

Alle Benutzer in einem MVS, einschließlich derer, die USS-Funktionen nutzen, müssen sich an die MVS-Standards bezüglich Identifikation und Verifikation halten. Sie benötigen deshalb eine MVS-Benutzerkennung und ein Passwort sowie eine Default-Gruppe. Wollen sie USS-Funktionen nutzen, müssen sie zusätzlich ein OMVS-Segment im RACF-Profil aufweisen, über das zumindest eine numerische UID zugewiesen wird.

Abbildung 12.3: USS und RACF



Für die Identifikation mit Benutzerkennung und Passwort werden im MVS grundsätzlich die Informationen in den RACF-Profilen benutzt. Befindet man sich dann in UNIX System Services, werden für die Zugriffe auf das Hierarchical File System die UNIX-spezifischen Sicherheitsmechanismen mit Permission Bits verwendet.

Innerhalb des Systems kommt es dann darauf an, ob auf MVS-Dateien oder auf das Hierarchical File System zugegriffen wird.

Im Falle von MVS-Dateien werden die Dataset-Profile von RACF für den Zugriffsschutz herangezogen. Im Falle des Hierarchical File System werden die Permission Bits herangezogen, die sich im *File Security Packet* (FSP) befinden, das mit einem File verknüpft ist.

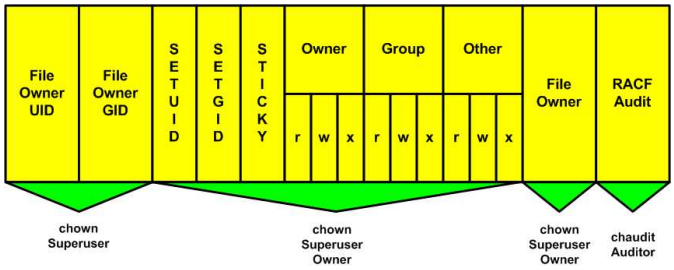


Abbildung 12.4: USS und FSP

Dieses FSP enthält unter anderem die Informationen über die Besitzrechte (Ownership), d. h., welchem User und zu welcher Gruppe das FSP gehört.

Die UNIX/Linux Files unterscheiden drei Berechtigungsarten: **read (r)**, **write (w)** und **execute (x)**. Diese Berechtigungen werden als File Permission Bits gesetzt und jeweils auf den Ebenen **User, Group** und **Other** vergeben. Mit der **read**-Berechtigung kann ein File gelesen, mit der **write**-Berechtigung kann es geschrieben mit der **execute**-Berechtigung ausgeführt werden. Da auch Verzeichnisse im Sinne von UNIX/Linux Files darstellen, gelten diese Berechtigungen auch für Verzeichnisse.

Eine wichtige Anmerkung betrifft die **write**-Berechtigung auf Verzeichnisebene. Wenn ein Benutzer auf Verzeichnisebene die **write**-Berechtigung hat, kann er ein File in dem Verzeichnis löschen, und zwar selbst dann, wenn er auf das File selbst überhaupt keine Berechtigung hat. Damit soll ausgedrückt werden, dass mit **write**-Berechtigungen auf Verzeichnisebene äußerst sorgfältig umgegangen werden muss.

Eine zweite Anmerkung betrifft die **execute**-Berechtigung. Für die Ausführung von Binärfiles reicht die **execute**-Berechtigung aus. Für die Ausführung von Scripts ist zusätzlich die **read**-Berechtigung notwendig, da die im Script befindlichen Befehle gelesen werden müssen, damit sie interpretiert und zur Ausführung gebracht werden können.

Der Superuser

Ein großes Problem bezüglich Security in UNIX/Linux ist der so genannte Superuser (oft auch als „root-User" oder einfach „root" bezeichnet), der durch die UID 0 gekennzeichnet ist. Im Vergleich mit RACF entspricht die Superuser-Berechtigung einem Benutzer, der die Attribute SPECIAL und OPERATIONS in sich vereint. Außerdem hat er volle Berechtigung auf alle Ressourcen und wird in keiner Weise kontrolliert. Wenn in UNIX ein Benutzer mit der UID 0 auf eine Ressource zugreifen will, kann er das grundsätzlich mit voller Berechtigung, und seine Zugriffe werden auch nicht protokolliert.

Im Gegensatz dazu kann man die Berechtigungen eines RACF-Administrators mit dem Attribut SPECIAL durchaus eingeschränken. Jeder RACF-Benutzer kann seine eigenen Ressourcen auch gegenüber Zugriffen des RACF-Administrators schützen. Versucht dieser, auf eine geschützte Ressource zuzugreifen, wird er abgewiesen. Aufgrund des SPECIAL-Attributs kann er allerdings den Schutz dann so ändern, dass der Zugriff in der gewünschten Form für ihn möglich ist. Allerdings wird diese Änderung des RACF-Schutzes über die Komponente System Management Facility (SMF) protokolliert. Auch ein RACF-Administrator kann diese Protokollierung nicht ohne fremde Hilfe ausschalten.

Um die Superuser-Berechtigung von RACF verwalten zu können, müssen zunächst die RACF Facility für die Verwaltung der Superuser-Berechtigungen eingeschaltet und dann mit der RACF PERMIT-Anweisung die Berechtigungen für den bzw. die Benutzer vergeben werden.

Beispiel:

```
RDEFINE FACILITY BPX.SUPERUSER UACC(NONE)
PERMIT BPX.SUPERUSER CLASS(FACILITY) ID(UHOX) ACCESS(READ)
```

Prozesse können in Verbindung mit USS über die Shell, die ISPF-Schnittstelle oder mit Operator-Kommandos angezeigt und verwaltet werden.

Kapitel 13

Webserver und Application Server

13.1 E-Business und der Mainframe

Die Geschäftswelt von heute fordert von der IT vor allem Schnelligkeit und Flexibilität. Heterogene Entwicklungsteams müssen stabile Anwendungen in Rekordzeit zur Verfügung stellen können, damit ein Unternehmen wettbewerbsfähig bleibt. Die Integration von neuen Technologien und bestehenden Anwendungen sowie der sichere und zuverlässige Zugriff auf riesige Datenmengen stellen eine große Herausforderung dar. Es muss eine Infrastruktur mit Prozessen und Werkzeugen verfügbar gemacht werden, um ein Unternehmen zu E-Business on Demand zu befähigen.

Die Wiederverwendung von Ressourcen ist ein entscheidender Faktor, um Kosten zu sparen. Bereits getätigte Investitionen müssen geschützt werden. Entscheidend dabei ist eine Kombination der Web- und Internet-Technologien mit komponen-

tenbasierter Entwicklung und Integration der bestehenden Plattformen und Anwendungssysteme. Die Java 2 Enterprise Edition (J2EE) spielt hierbei vor allem in heterogenen Umgebungen die größte Rolle.

Die Integration ins Web mit Web Services wird ebenfalls ein wichtiger Faktor werden.

Der CICS Transaction Server ist ein Bindeglied für die Kommunikation von Java-basierten Programmen und Enterprise Java Beans (EJBs) mit existierenden Transaktionssystemen. Das CICS Transaction Gateway (CTG) ermöglicht die Integration von Anwendungen, die auf einem WebSphere Application Server laufen, in CICS-basierte Kernanwendungen auf dem Mainframe. Das CTG implementiert inzwischen auch die *J2EE Connector Architecture* (JCA) und ermöglicht damit die direkte Nutzung der EJB-Technologie ohne Application Server.

Die E-Business Möglichkeiten in Verbindung mit dem Mainframe stehen in engem Zusammenhang mit den UNIX System Services, die im Kapitel 12 bereits behandelt wurden. Hierbei spielen dann natürlich die Einsätze von Webservern und Application Servern eine zentrale Rolle.

13.2 Warum den Mainframe als Webserver einsetzen?

Auf Großrechnersystemen wie dem z/OS liegen die meisten Geschäftsdaten der großen Unternehmen und Organisationen. Je nach Analyse schwanken die Aussagen zum Anteil der auf dem Mainframe verwalteten Daten zwischen 60 und 80 Prozent. Diese Daten können über die Internet-Technologie auf einfache und effiziente Art sowohl in Intranets als auch weltweit für das Internet verfügbar gemacht werden.

Wie bei allen Internet-Strategien ist der Hauptvorteil dabei die Plattformunabhängigkeit auf der Client-Seite. Speziell in Verbindung mit dem MVS ist der direkte Zugang zu den auf den Hosts liegenden Daten erwähnenswert.

Da der Mainframe vom Ursprung her für geschäftskritische Daten ausgerichtet ist, liegen seine Stärken vor allem in den Bereichen Kapazität, Verfügbarkeit, Performance, Sicherheit und Integrität. Wie diese Stärken in der Vergangenheit für Datenbanken, Transaktionssysteme und Batchverarbeitung konsequent ausgenutzt wurden, sind sie künftig auch für Internet-Technologien nutzbar, denn Internet-Anwendungen sind inzwischen genauso geschäftskritisch wie herkömmliche Anwendungen, und ein Systemausfall ist mit immensen Kosten verbunden.

Entwicklung des Mainframe-Webservers

Internet Connection Server (ICS)

Bereits seit 1995 gibt es die Möglichkeit, einen Webserver unter MVS einzusetzen. Er wurde zunächst *Internet Connection Server* (ICS) genannt und konnte als separates Produkt bezogen werden. Das Basisbetriebssystem zum damaligen Zeitpunkt, ab dem der ICS eingesetzt werden konnte, war MVS/ESA 5.2.2.

Der ICS unterstützte noch kein SSL (*Secure Socket Layer* für verschlüsselte Übertragung) und war deshalb in erster Linie für den Einsatz in Intranets gedacht.

Internet Connection Secure Server (ICSS)

Im September 1996 kam eine neue Version heraus, die unter anderem auch Verschlüsselungstechniken mit SSL oder S-HTTP ermöglichte. Demzufolge wurde der Server dann auch *Internet Connection Secure Server for OS/390* genannt. S-HTTP ist eine alternative Verschlüsselungstechnik zu SSL, spielt in der Praxis jedoch so gut wie keine Rolle mehr und wird inzwischen vom z/OS-Webserver auch nicht mehr unterstützt.

ICSS Version 2 Release 2

Im März 1997 wurde Release 2 für OS/390 1.3 mit neuen und erweiterten Funktionen verfügbar. Dieses Release wurde dann umbenannt in *Domino Go Webserver for OS/390 (DGW) Release 4.6.0* und war für OS/390 ab Version 2.4 ausgerichtet.

Domino Go Webserver Release 4.6.1

Im November 1997 wurde der *Domino Go Webserver for OS/390 4.6.1* verfügbar. Dieser Webserver stand auch für zahlreiche andere Plattformen zur Verfügung. Neue Funktionen waren vor allem der Java Servlet Support, die Zertifikatsauthentifizierung und Unterstützung von FASTCGI.

Domino Go Webserver Release 5.0

Im Juni 1998 kam die Version 5 des DGW heraus. Die Domino Go Certificate Authority wurde implementiert, eine Exportversion für eine 128-Bit-Verschlüsselung stand für Finanzdienstleister zur Verfügung. Außerdem wurden LDAP sowie SSL für mehrere IP-Adressen unterstützt.

IBM HTTP Server

Auch hier geht es zunächst lediglich wieder um einen neuen Namen für das gleiche Produkt. Der IBM HTTP Server ist nichts anderes als der Domino Go Webserver.

Es ist unter z/OS (leider) *nicht* der Apache Webserver! Hier muss man aufpassen. Auch auf anderen Plattformen (AIX, Solaris, Linux, Windows etc.) heißt

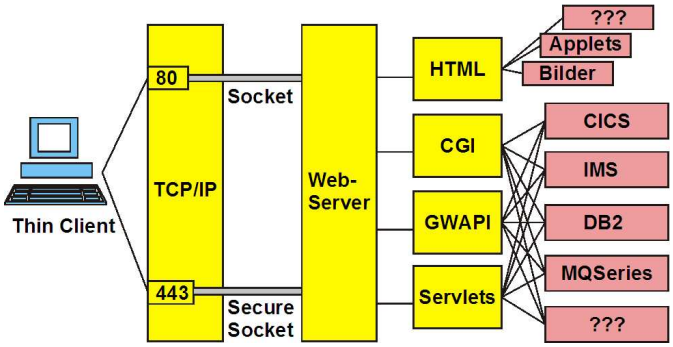
der IBM Webserver inzwischen HTTP Server, basiert dort allerdings auf dem Apache Webserver, der von IBM um SSL-Funktionalität erweitert wurde.

Einsatz des Mainframe-Webservers

Selbstverständlich muss für den Einsatz eines Webservers unter z/OS eine TCP/IP-Umgebung zur Verfügung stehen. Für die Kommunikation mit einem Webserver stehen die Standardports 80 und 443 zur Verfügung, wobei die Daten über den Port 443 mit SSL verschlüsselt übertragen werden.

Wenn es um dynamisch erzeugte Informationen geht, die von einem Client angefordert werden, stehen unterschiedliche Schnittstellen zur Verfügung.

Abbildung 13.1: Webserver-Strukturen



Das Common Gateway Interface (CGI)

Von der Historie her gab es zunächst das *Common Gateway Interface* (CGI), das jedoch einige gravierende Schwächen aufweist: So ist die Performance bei hohen Zugriffsraten problematisch, da bei jedem Request ein neuer Prozess mit einer komplett neuen Umgebung aufgebaut werden muss. Die Security in Verbindung mit dem CGI ist schwer kontrollierbar, und nicht zuletzt sind die CGI-Programme oft proprietär. Es klingt zwar nach einem allgemeinen Standard, wenn man den Begriff Common Gateway Interface hört, dies betrifft jedoch nur die Schnittstelle und die Art und Weise, wie Daten zwischen Client und Server ausgetauscht werden, nicht jedoch das Programm selbst. In einer IBM-Umgebung ist beispielsweise die Scriptsprache REXX weit verbreitet, die auch für die CGI-Programmierung eingesetzt werden kann. Dass dann ein REXX-Programm jedoch auch in einer Sun Solaris-Umgebung läuft, ist natürlich nicht selbstverständlich.

Webserver APIs

Um vor allem das Problem der Security und der Performance zu lösen, haben die Hersteller der Webserver APIs entwickelt, um die Schwächen zu eliminieren. Beispiele:

- GWAPI für den Domino Go Webserver
- NSAPI für den Netscape und iPlanet Webserver
- ISAPI für den Internet Information Server etc.

Die Problematik dabei: Die Schnittstellen sind proprietär, d. h. sie stehen nur für den jeweilig eingesetzten Webserver zur Verfügung. So sind die GWAPIs eben nur auf dem Domino Go Webserver bzw. IBM HTTP Server nutzbar, die NSAPI lassen sich nur in einer Netscpape-Webserver-Umgebung nutzen usw. Für systemspezifische Einsätze sind diese APIs jedoch sehr gut geeignet. So kann beispielsweise ein z/OS Webserver in wenigen Minuten so konfiguriert werden, dass man über einen Browser mit einem speziellen Schlüsselwort in der URL auf die Inhalte einer PS- oder PO-Datei zugreifen kann. Realisiert wird dies über ein GWAPI-Programm, das dann lediglich im Konfigurationsfile des Webservers entsprechend definiert werden muss.

Servlets

Im Zuge des Java-Hypes entstand dann die Idee, analog zu Applets auf der Client-Seite Java in Form von Servlets auf der Server-Seite einzusetzen. Entsprechende Erweiterungen wie Java Server Pages (JSPs) und Enterprise Java Beans (EJBs) setzen dann jedoch als Ergänzung zum Webserver einen Application Server voraus.

Komponenten und JavaBeans

In der Softwareentwicklung wird immer mehr auf Komponenten-Technologien gesetzt. Das Ziel ist eine effizientere und produktivere Entwicklung von Software mit vor allem kürzeren Entwicklungszeiten.

Ein Vergleich mit der Hardware drängt sich auf, weil dort schon sehr lange mit Komponenten gearbeitet wird. Kein Hersteller von elektronischen Geräten käme heute noch auf die Idee, ein Radiogerät oder ein Fernsehgerät aus einzelnen Transistoren zusammenzulöten. Vielmehr werden fertige Komponenten gekauft und zusammengesteckt. Ein typisches Beispiel sind auch die PCs: Selbst ein Anwender ohne Elektronikkenntnisse kann einen PC mit einer Netzwerkkarte vom Hersteller X, einer Soundkarte vom Hersteller Y und einer Grafikkarte vom Hersteller Z erweitern.

Analog wird diese Idee im Softwarebereich umgesetzt. Komponenten unterschiedlicher Hersteller sollen eingekauft und zusammengefügt werden können. Im Zusammenhang mit Java hat sich der Begriff „Beans" (Bohnen) durchgesetzt, wenn es um wiederverwendbare Software-Komponenten geht.

Mit den JavaBeans wurde ein allgemeines Komponentenmodell für die Java-Programmierung spezifiziert, das vorwiegend auf der Client-Seite zum Einsatz kommt, aber auch auf der Server-Seite eingesetzt werden kann. Eine Alternative auf der Server-Seite sind die Enterprise Java Beans, die Bestandteil der J2EE (Java 2 Enterprise Edition)-Spezifikation sind.

EBCDIC vs. ASCII

Was im Zusammenhang mit dem Einsatz eines Webservers unter z/OS beachtet werden muss, ist die Tatsache, dass die Daten unter z/OS im Gegensatz zu fast allen dezentralen Systemen (Windows, UNIX, Linux etc.) im EBCDIC-Code abgelegt werden. Dies betrifft auch die Daten im Hierarchical File System von USS.

Inhalte für den Webserver unter z/OS werden ebenfalls im Hierarchical File System abgelegt. Es kommt nun darauf an, ob diese Daten im EBCDIC-Code oder im ASCII-Code abgelegt werden; meist ist es sinnvoll, beide Varianten zu ermöglichen.

Daten im EBCDIC-Code abzulegen hat den Vorteil, dass diese Daten dann auch direkt vom z/OS bzw. aus Unix System Services beispielsweise mit dem ISPF-Editor verändert werden können.

Daten im ASCII-Code abzulegen hat den Vorteil, dass die Daten mit einem beliebigen HTML-Editor erzeugt werden können und ohne Konvertierung per FTP oder über eine NFS-Konfiguration ins Hierarchical File System übertragen werden.

Über das Konfigurationsfile des Webservers (**httpd.conf**) kann dann abhängig von der File Extension gesteuert werden, wie die angeforderten Daten behandelt werden. Meistens richtet man es so ein, dass Daten mit der Endung **.html** als EBCDIC-Daten abgelegt werden. Die Daten werden dann vor dem Versand nach außen in ASCII konvertiert. Daten mit der Endung **.htm** werden als BINARY gekennzeichnet und bei Requests ohne Konvertierung gesendet.

Zugriff auf MVS-Daten

Mit Hilfe des Webserver-APIs GWAPI kann der Webserver so konfiguriert werden, dass über einen beliebigen Browser auf Daten aus MVS-Dateien (sequentiell organisierte oder PO-Dateien) zugegriffen werden kann. Das Default-Schlüsselwort dazu heißt MVSDS. Wird dann in der URL beispielsweise Folgendes eingegeben:

```
www.hostname.com/MVSDS/'SYS1.PARMLIB'
```

wird das Directory der Datei SYS1.PARMLIB unter der Voraussetzung angezeigt, dass dem Benutzer vom RACF her der Zugriff auf die SYS1.PARMLIB gewährt wird.

WebSphere

Der Webserver wird künftig (meistens) mit einem Application Server zusammenarbeiten.

Hinter dem Begriff *WebSphere* steht eine ganze Palette von Produkten. IBM hat alles, was in irgendeiner Form mit E-Business zu tun hat, unter das Dach WebSphere gepackt. Weitere Produkte, die ebenfalls häufig auf dem Mainframe eingesetzt werden, sind beispielsweise *WebSphereMQ*, der *WebSphere Portal Server* und *WebSphere Host Integration*. Wir gehen hier in erster Linie auf den *WebSphere Application Server* ein.

Im WebSphere Application Server wurden in der Vergangenheit oft so genannte Connector Servlets zur Verfügung gestellt, um Verbindungen zu Datenbanken, zu ORBs, zu Transaktionssystemen und zu weiteren Middleware-Produkten zu ermöglichen.

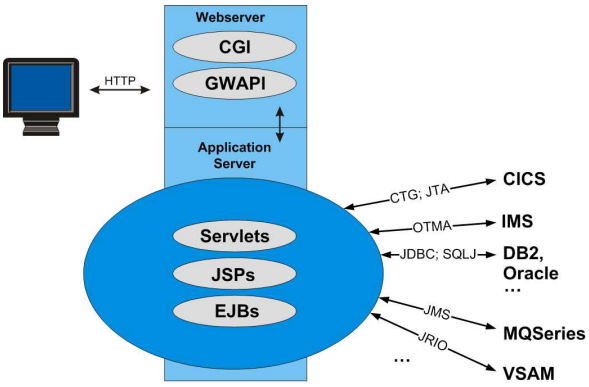


Abbildung 13.2: Application Server

13.3 J2EE und WebSphere

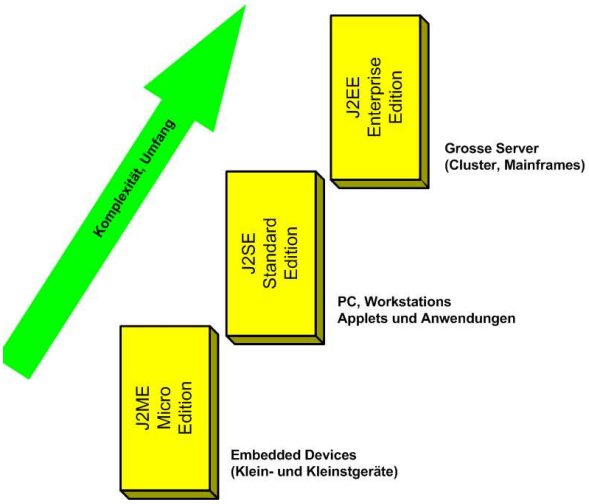
Java ist mehr als eine Programmiersprache

Hinter dem Java-Paradigma steckt der ernsthafte Versuch, aus Java mehr als eine Programmiersprache zu machen. Das Ziel ist es, mit Java eine Plattform für die unternehmensweite System- und Anwendungsentwicklung zur Verfügung zu stellen. Hinter Java steht nicht nur die Syntax einer Programmiersprache und eine Sammlung von APIs, sondern Java definiert auch eine Laufzeitumgebung, für

die herstellerunabhängig entwickelt werden kann. Dies betrifft sowohl die Client-Seite in Form von Anwendungen und Applets als auch die Server-Seite für die Unterstützung von Servlets, Java Server Pages und Enterprise Java Beans. Basis ist immer eine Java Virtual Machine (JVM), die plattformunabhängigen Bytecode in den jeweiligen Maschinencode der Zielmaschine umsetzt.

J2EE setzt auf der Serverseite beispielsweise Connection Pooling, Security und Transaktionsmechanismen voraus. Damit sollen die Entwickler von vielen mühsamen Routinearbeiten befreit werden, um sich auf die eigentliche Business-Logik konzentrieren zu können.

Abbildung 13.3: Die unterschiedlichen Java-Editionen



Mit Java Version 2 wurden drei unterschiedliche Editionen von Java eingeführt:

Für die *Micro Edition* wurde der Sprach- und Funktionsumfang von Java reduziert. Gleichzeitig wurden zusätzliche Funktionen für eingebettete Systeme hinzugefügt, um Klein- und Kleinstgeräte zu unterstützen.

Die *Standard Edition* ist der Kern der Java-Plattform, der für eigenständige, in Java entwickelte Anwendungen und Applets gedacht ist.

Die Idee hinter der *Enterprise Edition* ist, einen einfachen, einheitlichen Standard für verteilte Anwendungen mit Hilfe eines komponentenbasierten Anwendungsmodells zur Verfügung zu stellen. Mit J2EE wird dem sehr divergenten Markt von Anwendungsservern eine einheitliche Plattform auf der Basis von Java angeboten.

Mit J2EE wird für moderne E-Business-Anwendungen auf standardisierte Weise das eingebracht, was der Mainframe aufgrund seiner evolutionären Entwicklung in der kommerziellen Datenverarbeitung schon lange beherrscht:

- Transaktionsverarbeitung
- Messaging und Queuing
- Sicherheit

Zugriff auf Enterprise Services

Verteilte Anwendungen benötigen Zugriff auf Enterprise Services. Typische Services sind beispielsweise Transaktionsverarbeitung, DB-Zugriffe, Messaging etc. Anstatt diese Services durch proprietäre oder nicht-standardisierte Schnittstellen realisieren zu müssen, bietet die J2EE-Architektur diese APIs über standardisierte Container-Services an.

Für die Enterprise Edition gibt es bereits mehrere Spezifikationen. So ist die Version 1.2 seit Dezember 1999 verfügbar. Die Version 1.3 gibt es seit Juli 2001, und die Version 1.4 ist inzwischen auch spezifiziert. Die WebSphere Version 4 setzt die Spezifikation von J2EE 1.2 um, die Version 5 die von J2EE 1.3 und die Version 6 die von J2EE 1.4.

Container

Eine typische J2EE-Plattform (ein J2EE Application Server) beinhaltet einen oder mehrere Container und realisiert über diese den Zugriff auf Enterprise APIs, die in der J2EE definiert sind.

Die Container werden benötigt, um den Zugriff auf die Implementierungen der Services mit Hilfe der J2EE APIs zu ermöglichen. So kann beispielsweise eine J2EE-Implementierung der Java Messaging Server API Calls eine kommerzielle Message Oriented Middleware Software nutzen. Allerdings verwendet der Entwickler ausschließlich das standardisierte API Java Message Services.

Ein Container verkörpert somit die Laufzeitumgebung und bildet die Abstraktion zur jeweiligen Systemplattform. Das Verhalten eines Containers ist deklarativ spezifizierbar mittels eines XML-basierten Deployment Descriptors.

J2EE Java APIs (Version 1.3)

Die Spezifikation von J2EE definiert einen Satz von Java-Erweiterungen, die eine J2EE-Plattform unterstützen muss. Wenn ein Hersteller eines Application Servers seinen Server als J2EE-zertifizierten Server für eine bestimmte Spezifikation vermarktet, kann sich ein Entwickler darauf verlassen, dass ihm die Schnittstellen in einer exakt definierten Form zur Verfügung stehen. Die folgenden Beispiele zeigen die Spezifikationen, die von einer J2EE 1.3-Version mindestens unterstützt werden müssen.

Java Database Connectivity (JDBC) 2.0 Extension
: Das API erweitert das JDBC 2.0 J2SE API beispielsweise um Connection Pooling und verteilte Transaktionen.

RMI-IIOP 1.0
: Implementation des RMI API over IIOP. Dadurch wird eine Brücke zwischen RMI und CORBA ermöglicht.

Enterprise Java Beans (EJBs) 2.0
: Komponenten-Framework für verteilte Multi-Tier-Applikationen

Java Servlets 2.3
: Java Servlet API für die Entwicklung und Steuerung dynamischer Webseiten

Java Server Pages (JSP) 1.2
: ermöglicht schablonengesteuerte Webseiten mit eingebettetem Java-Code

Java Message Services (JMS) 1.0
: JMS ist ein Java API für Message Queuing.

Java Naming and Directory Interface (JNDI) 1.2
: Dieses API standardisiert den Zugriff auf Verzeichnisdienste. J2EE definiert auch ein JNDI Service Provider Interface (SPI).

Java Transaction API (JTA) 1.0
: Dieses API unterstützt bei verteilten, transaktionsorientierten Anwendungen.

JavaMail 1.2
: Dieses API bietet ein plattform- und protokollunabhängiges Framework für javabasierte Mail-Anwendungen.

Java Authentication and Authorization Service (JAAS) 1.0
: Security API für Authentifizierung und Autorisierung

Java API for XML Parsing (JAXP)
: API für XML Parsing

Architektur von J2EE

Eine J2EE-Plattform schließt einen oder mehrere Container ein.

Ein J2EE Container ist eine Laufzeitumgebung, um Anwendungskomponenten zu verwalten und Zugriffe auf J2EE APIs zu ermöglichen. In der J2EE-Architektur unterscheidet man zunächst einmal zwischen einem Web Container (der unter anderem eine Servlet Engine enthält) und einem EJB-Container, der die Enterprise Java Beans verwaltet. Inzwischen gibt es auch eine JCA (J2EE Connection Architecture), und möglicherweise werden künftig noch weitere Container definiert.

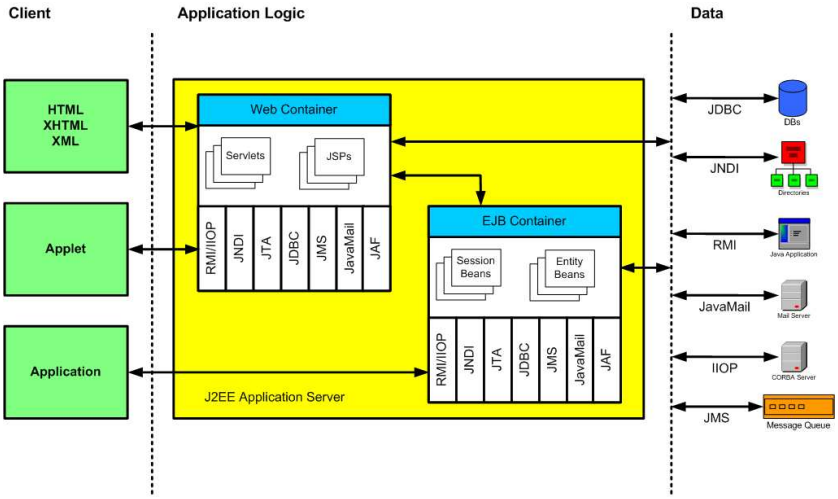


Abbildung 13.4: Die Java-2EE-Architektur

Rollenverteilung

Das J2EE-Modell definiert eine präzise Rollenverteilung für den Entwicklungszyklus von J2EE-Anwendungen. Diese Rollenverteilung dient dem besseren Verständnis der einzelnen Phasen. Natürlich können durchaus mehrere Rollen von einer Person oder einer Gruppe übernommen werden.

Hersteller von J2EE Servern (J2EE Product Provider)

Ein Server-Hersteller stellt Container, Dienste und Implementierungen der J2EE APIs über ein Programmpaket zur Verfügung. Sinnvollerweise bieten diese Hersteller auch Werkzeuge und Hilfsmittel für die Konfiguration und die Administration der Server an.

Hersteller von Werkzeugen (J2EE Tool Provider)

Ein Werkzeughersteller liefert Tools für die Entwicklung und das Zusammenpacken von Komponenten. Auch im Systemmanagement-Umfeld gibt es zahlreiche Hersteller, die Optimierungswerkzeuge zur Verfügung stellen.

Hersteller von Komponenten (Application Component Provider)

Anwendungskomponenten bilden die Bausteine einer J2EE-Anwendung. Hersteller von Komponenten entwerfen HTML- und JSP-Seiten und programmieren Tag-Bibliotheken, Servlets und EJBs. Außerdem müssen Konfigurationsdateien (*Deployment Descriptors*) für die Anpassung der Komponenten erstellt werden. Die Aufgaben erfordern keinerlei Kenntnisse über die

Zielplattform, in der die Komponenten dann zum Einsatz kommen. Komponenten können separat entwickelt oder eingekauft werden.

Zusammenfügen der Komponenten (Application Assembler)
Diese Rolle definiert das Zusammenfügen von Komponenten zu einsetzbaren Einheiten. Hierzu werden die Komponenten in Archive zusammengefasst, die als *Web Archive* (WAR) bezeichnet werden. Außerdem werden Abhängigkeiten zwischen den einzelnen Komponenten in Konfigurationsdateien definiert sowie eine Anleitung erstellt, wie Abhängigkeiten von externen Ressourcen vor dem eigentlichen Einsatz aufgelöst werden können. Die Kombination erfordert i. d. R. keine Kenntnisse über den Quellcode der Komponenten. Die Bestandteile werden am Ende in einem einzigen Archiv, dem *Enterprise Archiv* (EAR) zusammengefasst.

Inbetriebnahme der Komponenten (Application Deployer)
Hinter dieser Rolle steht die Anpassung von J2EE-Komponenten an eine spezifische Ziel- und Laufzeitumgebung. Hierzu zählt auch das Auflösen externer Ressourcen. Der Quellcode wird hierbei nicht angefasst, lediglich die Konfigurationsdateien werden mit geeigneten Werkzeugen editiert.

Systemadministrator
Hier geht es um den eigentlichen Betrieb. Die Funktion und das Zusammenspiel der Komponenten müssen überwacht werden. Hierfür werden Werkzeuge des Server-Herstellers oder auch Tools von Drittherstellern eingesetzt.

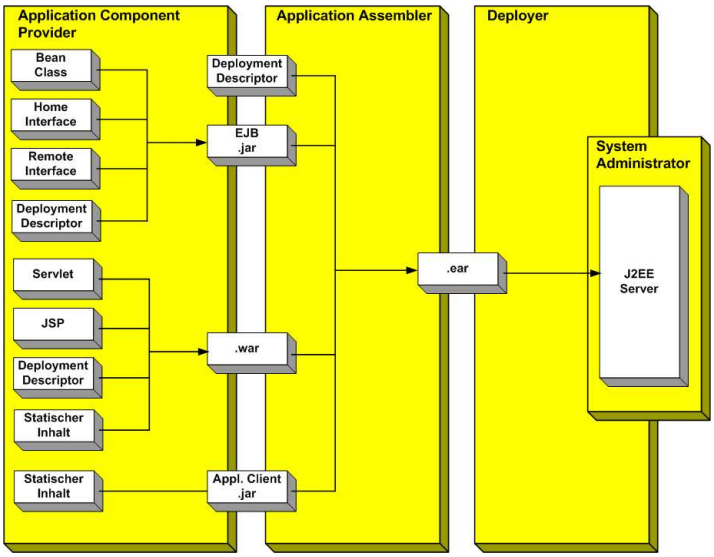


Abbildung 13.5: Rollenverteilung im J2EE-Modell

Eine der wichtigsten Schnittstellen zwischen den Rollen ist die zwischen dem Application Assembler und dem Deployer, da dies gleichzeitig auch die Schnittstelle zwischen Entwicklung und Betrieb ist. Beim Deployment wird eine Anwendung von einer plattformunabhängigen Entwicklung in eine spezifische Betriebsplattform integriert.

Ein Enterprise Archiv (ear-File) wird hierbei mit einem so genannten Deployment Tool in eine Betriebsumgebung gebracht. In J2EE wird gefordert, dass ein derartiges Tool zur Verfügung gestellt wird, sowie spezifiziert, was es können muss, nicht jedoch, wie es vom „Look and Feel" her aufgebaut ist.

13.4 WebSphere unter z/OS

WebSphere unter z/OS bietet den höchsten Grad an Quality of Service. In Verbindung mit WebSphere werden die klassischen Stärken des Mainframe genutzt:

- hohe Verfügbarkeit
- absolute Zuverlässigkeit
- höchste Security
- höchste Skalierbarkeit
- unterbrechungsfreie Anwendungsverfügbarkeit
- Transaktionsunterstützung inkl. Two-Phase-Commit
- Unterstützung durch System Management Tools
- Integrationsfähigkeit mit Backend-Systemen
- Tools für den Systembetrieb
- große I/O-Bandbreiten
- große Netzwerkbandbreiten

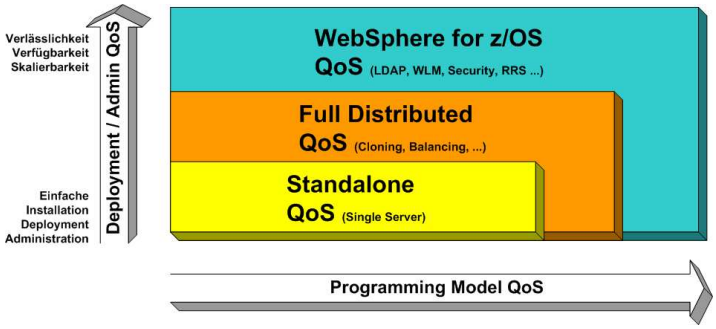
Bezüglich Einsatzkonzept und Installation gibt es einen großen Unterschied zwischen WebSphere auf dezentralen Systemen und WebSphere on z/OS. Für die dezentralen Plattformen (Windows, Linux, UNIX ...) bekommt man eine CD ausgeliefert; über einen Install-Button wird WebSphere installiert und kann gestartet und betrieben werden.

Bei WebSphere on z/OS sieht das Ganze etwas anders aus. Hier müssen vor der eigentlichen Installation einige Dinge vorbereitet und konfiguriert werden. Das hängt damit zusammen, dass in einigen Schlüsselbereichen auf die Funktionalität des

Mainframe gebaut wird. Dies bedeutet, dass diese Features nicht von WebSphere, sondern vom Betriebssystem zur Verfügung gestellt werden.

Typische Beispiele dafür sind die Bereiche Security, Transaktionsunterstützung, Workload Manager etc.

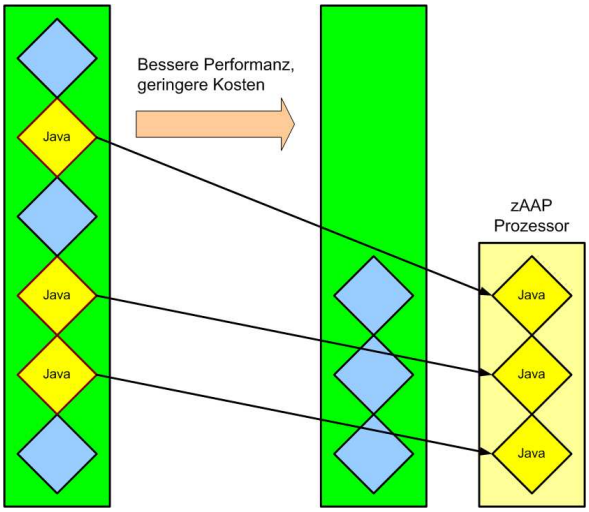
Abbildung 13.6: Quality of Service bei WebSphere



Der z/Series Application Assist Processor

Es gibt zwei Gründe für den Einsatz spezieller Java-Prozessoren, die IBM im April 2004 angekündigt hat: Java benötigt mehr Rechenleistung als andere Programmiersprachen. Damit diese Rechenleistung effizient und parallel zu „normaler" Verarbeitung genutzt werden kann, ist der Einsatz spezieller Prozessoren, die sich ausschließlich um Java kümmern, sinnvoll. IBM nennt diese Java-Prozessoren *z/Series Application Assist Processors* (zAAP).

Abbildung 13.7: Auswirkungen des Einsatzes von zAAPs



Die zAAPs sind billiger als die Standardprozessoren; der zweite Grund, der für ihren Einsatz spricht, ist die Tatsache, dass sie sich wie die Integrated Facility for Linux (IFL) nicht auf die Lizenzgebühren der Software auswirken.

Die Funktionalität des Java-Prozessors betrifft allerdings nicht nur WebSphere, sondern auch andere Subsysteme wie beispielsweise DB2, CICS und SAP, wenn sie Aktivitäten auf einer JVM auslösen.

Voraussetzungen für WebSphere Version 5

Möglicherweise verändern sich die Anforderungen für spätere WebSphere-Versionen. Die Grundvoraussetzungen werden jedoch auch künftig ähnlich sein.

Damit WebSphere Version 5 unter z/OS eingesetzt werden kann, müssen die folgenden Voraussetzungen erfüllt sein:

- OS/390 R10 oder z/OS
- für WebSphere angepasster Security Server (RACF)
- TCP/IP muss eingerichtet sein
- FTP Server muss eingerichtet sein
- UNIX System Services mit HFS muss verfügbar sein
- Resource Recovery Services (RRS) muss eingerichtet sein
- System Logger muss eingerichtet sein
- WLM (Workload Manager) muss im Goal Mode eingerichtet sein und für WebSphere angepasst werden

Optional je nach Einsatz und Anwendung muss der CICS Transaction Server 1.3 oder höher, IMS V8 oder höher und DB2 V7 oder höher zur Verfügung gestellt werden.

Der Java Message Server Integral Provider wird mit der WebSphere-Basis implementiert und besteht aus dem WebSphereMQ Queue Manager, einem JMS Broker für Public/Subscribe und einem JMS Client.

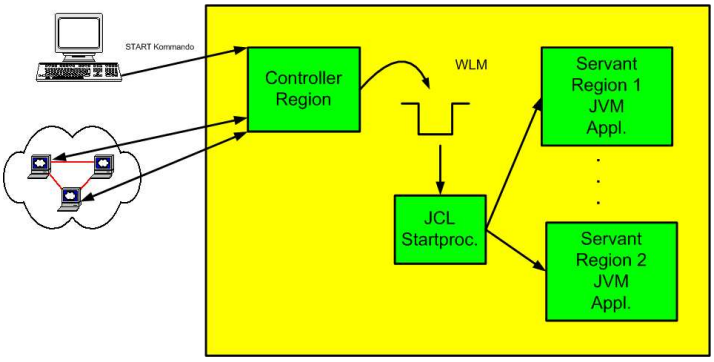
Für die volle MQ-Unterstützung ist eine separate WebSphereMQ-Lizenz notwendig.

WebSphere for z/OS Controller und Servants

WebSphere für z/OS besteht aus mehreren Adressräumen. Es gibt eine Control Region, welche die Requests entgegennimmt und in eine Work Queue stellt.

Über diese Work Queue werden die Requests an die Server Region(s) weitergegeben. Die Work Queue wird vom Workload Manager (WLM) überwacht. Wenn die Server Regions mit der Abarbeitung der Queue nicht mehr nachkommen, können weitere Server Regions gestartet werden.

Abbildung 13.8: WebSphere for z/OS

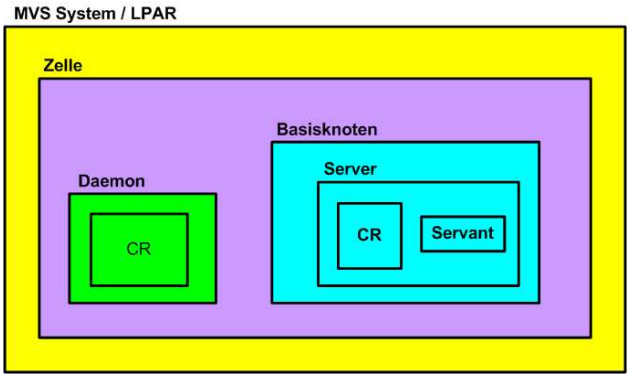


Diese Basisstruktur wird als „Server" bezeichnet. Der WLM muss entsprechend eingerichtet sein. Der Aufwand dafür ist abhängig von der eingesetzten Betriebssystem-Version. Sinnvoll ist der Einsatz von z/OS 1.2 oder höher, da dadurch zahlreiche PTFs nicht eingespielt werden müssen und der WLM für den Einsatz von WebSphere bereits vorbereitet ist.

Basisknoten

Ein Basisknoten besteht aus einer Controller Region und einem oder mehreren so genannten *Servants*. Hinter einem Servant stehen eine Java Virtual Machine und eine Instanz eines Application Servers.

Abbildung 13.9: WebSphere-Basisknoten



Pro Zelle/LPAR wird ein Daemon benötigt. Einschränkungen einer Basiskonfiguration: Es gibt keine System-/LPAR-übergreifende Zusammenarbeit. Damit ist die Skalierbarkeit und die Verfügbarkeit eingeschränkt. Das Starten/Stoppen von Servern ist nur über Operator-Kommandos und nicht über die Administrationskonsole möglich. Das Clustering von Servern ist nur eingeschränkt möglich.

Konfiguration in UNIX System Services

Die Konfiguration der Server ist im Hierarchical File System (HFS) von UNIX System Services unter z/OS untergebracht. Im Gegensatz zu früheren Versionen liegt das Konfigurations-Repository nicht mehr in einer Datenbank, sondern in Form von XML Files im Hierarchical File System von UNIX System Services.

Der Hauptvorteil dieser Änderung macht sich in höherer Verfügbarkeit und besserer Performanz bemerkbar. Es muss nicht mehr von vielen Servern parallel auf die zentrale Konfigurations-Datenbank zugegriffen werden.

Die Struktur der XML-Daten im Hierarchical File System ist identisch mit der Struktur in dezentralen Systemen.

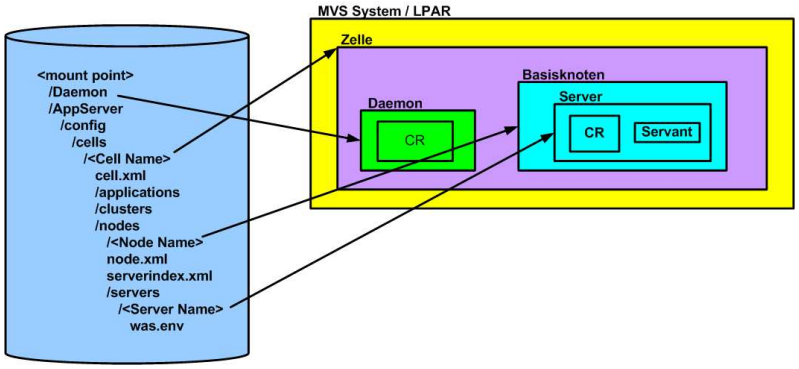


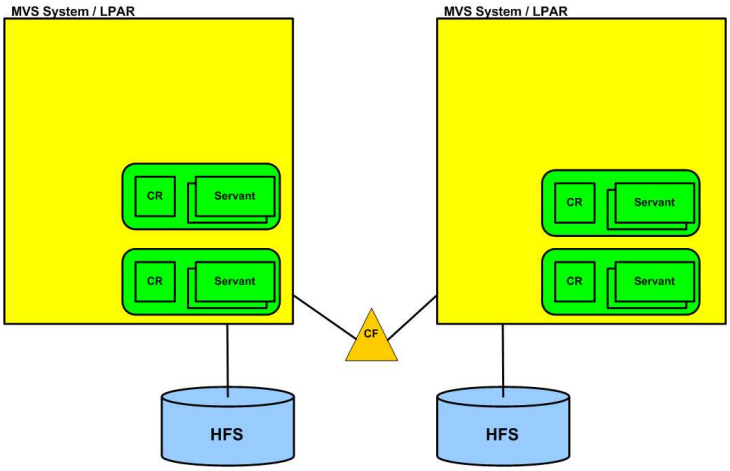
Abbildung 13.10: XML-Daten im HFS

Network Deployment-Konfiguration

Sehen wir uns zunächst eine Konfiguration an, in der mehrere Basisserver losgelöst voneinander in zwei Logical Partitions (LPARs) untergebracht sind.

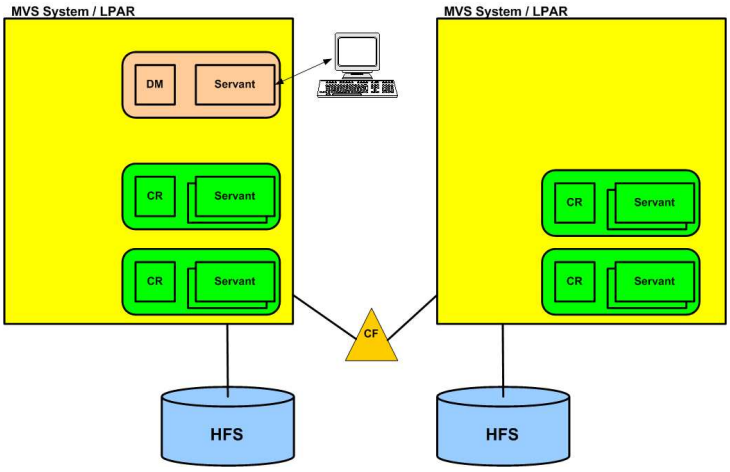
Das in der Abbildung 13.11 dargestellte Dreieckssymbol stellt eine so genannte *Coupling Facility* dar, die für eine Sysplex-Konfiguration benötigt wird. Eine Sysplex Coupling Facility kann als gemeinsam genutzter Hauptspeicher zwischen den beteiligten Betriebssystemen einer Sysplex-Konfiguration gesehen werden (vgl. dazu Kapitel 2).

Abbildung 13.11: Basiskonfiguration in mehreren Systemen



Damit die Möglichkeiten des Clustering und Workload Balancing über Systemgrenzen möglich sind, muss zunächst ein Deployment Manager eingerichtet werden. Ein Deployment Manager besteht aus einer Control Region und einem Servant, wobei dieser Servant nicht für „normale" Anwendungen, sondern ausschließlich für administrative Funktionen zur Verfügung steht.

Abbildung 13.12: Einrichten des Deployment Managers



Der Deployment Manager ist die Administrationszentrale einer Zelle. Die Konfiguration mit dem Deployment Manager muss nun in eine Zellenkonfiguration überführt werden, damit die Konfigurationsdaten und die Kommunikation auch Sysplex-konform funktionieren können.

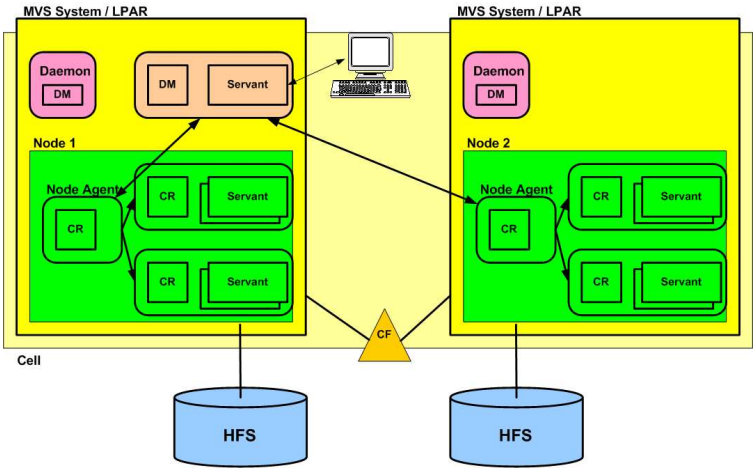


Abbildung 13.13: Aufbau einer Zelle

Über einen ISPF Configuration Dialog werden die Basisdaten einer ND-Konfiguration festgelegt und ein Deployment-Knoten erzeugt.

Danach können mit *addNode* Basiskonfigurationen in eine Network Deployment-Zelle überführt werden, wobei auch der Node Agent im jeweiligen Knoten eingerichtet wird. Dieser Vorgang wird auch als *Federation* bezeichnet.

WebSphere und Sysplex

Die Verteilung der Ressourcen funktioniert auch über Systemgrenzen hinweg, da WebSphere on z/OS auch Sysplex-fähig ist.

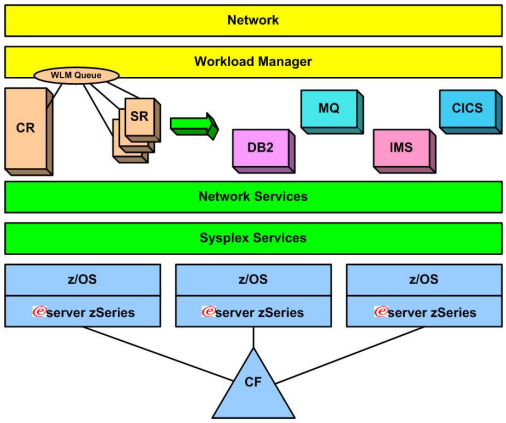


Abbildung 13.14: WebSphere und Sysplex

Hinter einer Sysplex-Konfiguration steckt das Clustering-Konzept der IBM Mainframes. Bis zu 32 Prozessorkomplexe können zu einem Verbund zusammengeschlossen werden (vgl. Abschnitt 2.1).

In einer Sysplex-Konfiguration können sowohl die WebSphere-Komponenten als auch die Subsysteme des z/OS wie DB2, CICS, WebSphereMQ etc. über mehrere physische Rechner verteilt sein.

Kapitel 14

Das Betriebssystem z/VM

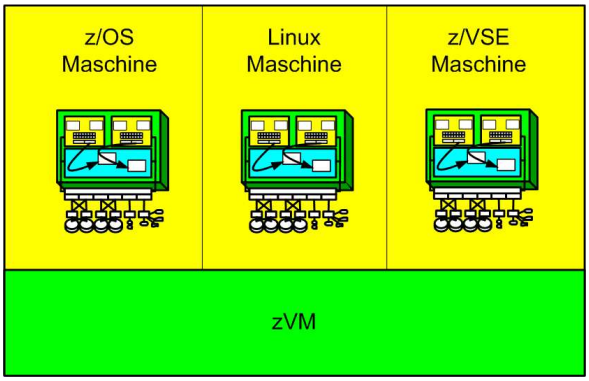
14.1 Warum z/VM?

Mit dem Betriebssystem z/VM wird auf Software-Basis eine Virtualisierungstechnik zur Verfügung gestellt, die es auf dem Mainframe bereits seit Anfang der 70-er Jahre gab. Sie hatte im Zuge der LPAR-Virtualisierung an Bedeutung verloren, erlebt jedoch nun in Zusammenhang mit Linux eine Renaissance. Dies ist einer der Gründe, warum im Folgenden der Einsatz von Linux in Verbindung mit z/VM den Schwerpunkt bildet.

Die klassische Mainframe-Architektur haben wir bereits in Kapitel 1 vorgestellt (vgl. auch die Abbildung auf Seite 23). Mit z/VM werden nun die physischen Systeme virtuell abgebildet (vgl. Abbildung 14.1). Es können Ressourcen eingerichtet und verwendet werden, die physisch gar nicht vorhanden sind. So wird beispielsweise oft das Spooling-System mit Lochkartenleser und Lochkartenstanzer verwendet, die man heute in einer physischen Konfiguration nicht mehr vorfindet. Virtuell können sie genutzt werden, um über einen Spool-Bereich Daten auszutauschen.

Mit z/VM als Hypervisor wird die Möglichkeit geboten, sehr viele Systeme bei geringem Verwaltungsaufwand auf einem bzw. wenigen physischen Rechnern betreiben zu können.

Abbildung 14.1: Die z/VM-Architektur



Es werden zwar zusätzliche Skills benötigt, der entsprechende Lernaufwand und die Einarbeitung lohnen sich jedoch, da danach eine leistungsfähige Umgebung mit hunderten oder gar tausenden von Servern betrieben und effizient verwaltet werden kann.

Die wichtigsten Vorteile von z/VM

- Ressourcen können zwischen mehreren Betriebssystemen geteilt werden. Dies betrifft beispielsweise CPU-Kapazität, Speicher, Plattenplatz, Netzwerkadapter etc.
- Neue „Gäste" können schnell und problemlos eingerichtet werden, ohne dass exklusive Ressourcen benötigt werden.
- Für die Kommunikation zwischen den Systemen können extrem schnelle Netzwerkverbindungen eingerichtet werden.
- Storage Management kann zentralisiert eingerichtet werden (Backup/Recovery).
- Umfassendes Workload Monitoring und Steuerungsmechanismen stehen zur Verfügung.

Der Einsatz von Linux auf dem Mainframe hat zahlreiche Vorteile hinsichtlich der Kosteneinsparung in den Bereichen Personal und Administration. Dies heißt jedoch nicht, dass es trivial wäre, hunderte oder tausende von Servern zu betreiben. Vielmehr stellt dies eine große Herausforderung dar und setzt qualifiziertes Personal mit Wissen und Erfahrungen aus den unterschiedlichsten Bereichen voraus.

14.2 Planung und Installation

Dies ist zweifellos die kritischste Phase beim Einrichten einer Umgebung. Die Entscheidungen, die während dieser Phase getroffen werden, haben großen Einfluss auf den künftigen Betrieb und die Performance des Gesamtsystems. Gerade in einem Bereich, wo die Basis für eine Infrastruktur gelegt wird, mit der sehr viele Server betrieben werden sollen, muss selbstverständlich besonders sorgfältig vorgegangen werden.

Folgende Rollen müssen zur Verfügung stehen, wobei von einer einzelnen Person mehrere Rollen ausgefüllt werden können.

- Linux-Systemadministrator
- Netzwerkadministrator
- z/VM-Systemprogrammierer
- Mainframe-Operator
- ggf. z/OS-Systemprogrammierer

In der Praxis ist es derzeit meist am schwierigsten, die Rolle des z/VM-Systemprogrammierers zu besetzen, da entsprechendes Know-how am Markt rar ist. Sehr viele Dinge müssen gemäß den Vorgaben konfiguriert werden und setzen teilweise tiefes VM-Wissen voraus. Da eine Umgebung in Abhängigkeit von den jeweiligen Anforderungen aufgesetzt werden muss, versagen in der Praxis allgemeingültige „Kochrezepte".

14.3 Komponenten von VM

VM hat zwei wesentliche Komponenten: das *Control Program* (CP) und das *Conversational Monitor System* (CMS).

Das CP ist ein so genannter Hypervisor und wird benötigt, um die Hardware-Ressourcen zu virtualisieren. Dabei werden die Ressourcen entweder dediziert aufgeteilt, gemeinsam genutzt oder emuliert.

Das CMS ist ein kleines Betriebssystem, das als Time Sharing Monitor mit dem TSO unter z/OS oder mit einer UNIX/Linux-Telnet-Session verglichen werden kann.

Das CP implementiert virtuelle z/Series-Maschinen. Auf einer einzigen physischen Hardware-Box können zahlreiche Kopien des gleichen Betriebssystems oder auch von unterschiedlichen Betriebssystemen gestartet werden. Die Benutzer sehen und

merken von dieser Virtualisierung nichts. Jede virtuelle Maschine hat ihren eigenen virtuellen Speicher, virtuelle Prozessoren, virtuelle Geräte etc. Ein CP Directory beinhaltet die Definition einer virtuellen Maschine.

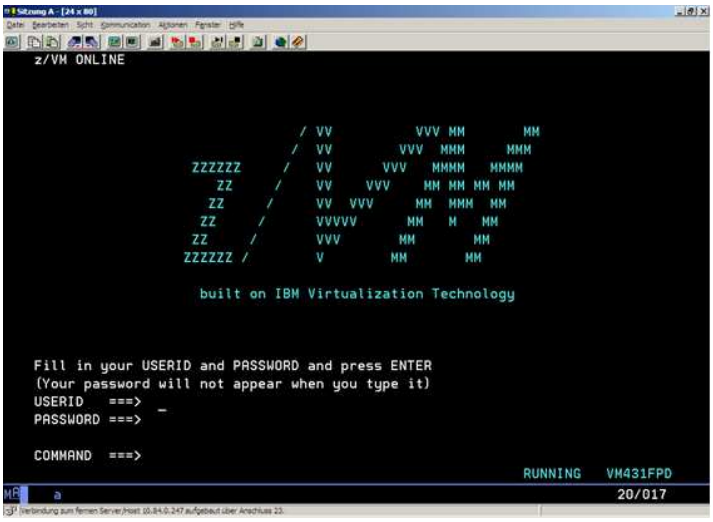
Ein Betriebssystem in einer virtuellen z/Series-Umgebung kommuniziert mit virtuellen Geräten. Das Mapping von den virtuellen Geräten auf die realen Geräte wird transparent vom Control Program vorgenommen. Da diese Virtualisierungstechnik schon relativ alt ist, werden auch Geräte wie Lochkartenstanzer und Lochkartengeräte abgebildet. Das mag zwar zunächst etwas antiquiert erscheinen, ist in Verbindung mit Virtualisierung und Spooling jedoch extrem hilfreich.

14.4 Umgang mit z/VM

Die Schnittstelle zum VM wird über eine 3270-Session aufgerufen. Diese unterscheidet sich in einigen wesentlichen Punkten grundlegend von einer UNIX/Linux Telnet-Session.

Eine 3270-Session beruht auf der ursprünglichen Idee eines so genannten „dummen Terminals" ohne Verarbeitungsmöglichkeit, sondern lediglich mit einem Terminal-Puffer. Eingaben werden in diesen Terminal-Puffer geschrieben und danach (zeilenweise) mit Auslösung der Enter-Taste über das 3270-Protokoll an den Mainframe geschickt.

Abbildung 14.2: z/VM-Login



Eine Telnet-Session dagegen arbeitet zeichenweise, d. h. jedes Zeichen wird direkt nach der Eingabe an den Zielrechner geschickt.

Heute arbeitet man meist nicht mehr mit dummen Terminals, sondern mit PCs und Workstations. Um dann mit einer 3270-Schnittstelle arbeiten zu können, benötigt man eine Terminal-Emulation. Im Gegensatz zu Telnet steht die jedoch nicht in jeder Umgebung per Default zur Verfügung, sondern muss zuvor installiert werden.

Nach Eingabe von UserID und Passwort wird eine Prozedur aufgerufen, die installationsspezifisch ist. Ein Beispiel ist in den Abbildungen 14.3 und 14.4 zu sehen.

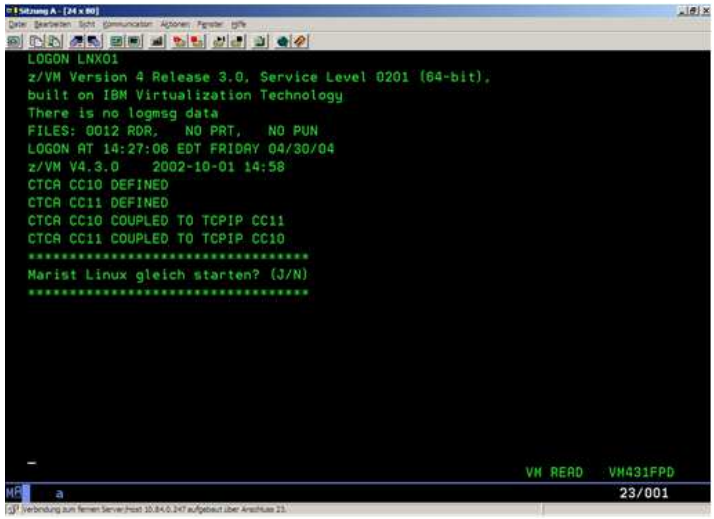


Abbildung 14.3: Aufruf einer Prozedur

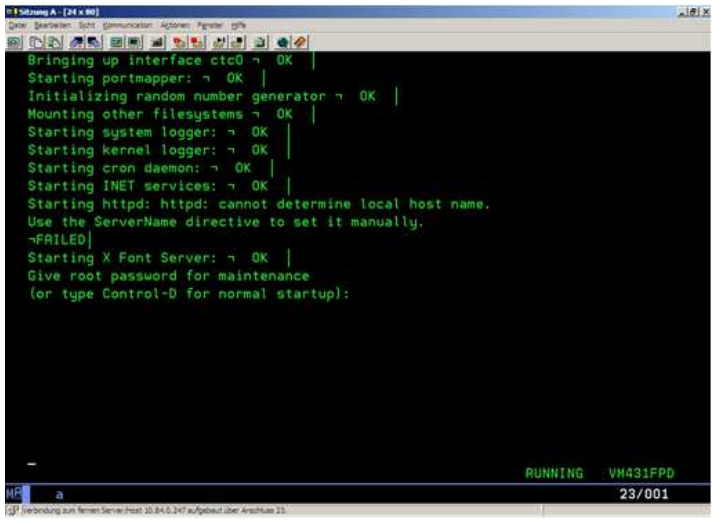
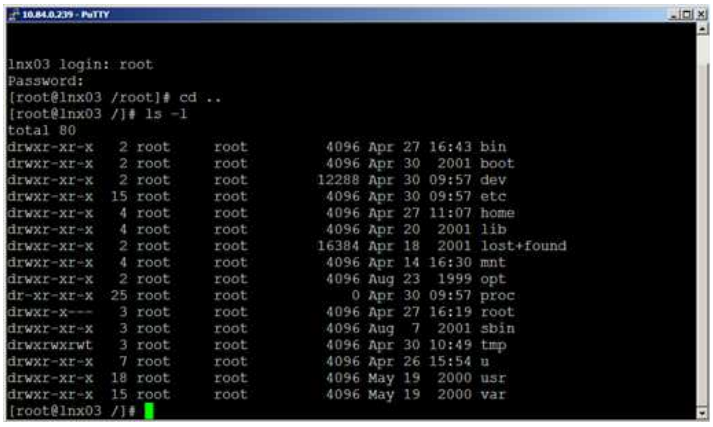


Abbildung 14.4: Systemstart

Wenn das System gestartet wurde, befindet man sich in diesem Fall in der CMS-Umgebung. Man könnte dann hier auch entsprechende Linux-Kommandos aufrufen. In der Praxis meldet man sich zuvor jedoch über eine standardisierte Telnet-Schnittstelle am Linux-System an.

Abbildung 14.5: Linux-Login unter PuTTY



14.5 Einstieg und Ausstieg

Das Layout des Einstiegsbildschirms variiert in der Regel von Installation zu Installation, jedoch wird man immer einen Eingabebereich für UserID, Passwort und Kommando vorfinden. Um sich anzumelden, werden eine UserID und ein Passwort eingegeben. Optional kann ein Kommando oder Script mitgegeben werden, das beim Logon-Vorgang dann automatisch ausgeführt wird.

Nach dem Logon werden diverse Nachrichten ausgegeben, die wiederum abhängig sind von der Art und Weise, wie das System installiert und aufgesetzt wurde.

Es erscheint der READY-Prompt. Unten rechts auf dem Bildschirm sollte die Nachricht **CP READ** zu lesen sein; dies signalisiert, dass das Control Program auf Eingaben wartet.

Andere Möglichkeiten in diesem Nachrichtenbereich sind:

VM READ
: Das CMS wartet auf Eingaben.

RUNNING
: Das System ist beschäftigt und bereitet beispielsweise gerade eine Ausgabe vor.

MORE...:
Es stehen weitere Informationen zur Ausgabe an. Um diese anzusehen, gibt es die **Clear**-Taste, auf manchen Tastaturen auch **Esc** oder **Break**.

HOLDING
Das System wartet auf das Drücken der **Clear**- bzw. **Esc**-Taste, um Informationen anzuzeigen.

NOT ACCEPTED
Die Nachricht erscheint, wenn ein weiteres Kommando eingegeben wurde, während das System noch beschäftigt war.

Mit **IPL CMS** kann das Conversational Monitoring System gestartet werden, das unter VM das Time Sharing ermöglicht.

Im Eingabebereich befindet sich dann die CMS-Kommandozeile, über die CP- oder CMS-Kommandos eingegeben werden können.

Wichtig ist zudem die Möglichkeit, das System wieder zu verlassen. Dies geschieht mit dem Kommando **logoff** in der Kommandozeile. Danach sollte der Eingangsbildschirm wieder erscheinen.

Ein weiterer wichtiger Punkt ist das „Disconnecten" von einer Konsole. Das Kommando hierzu: **DISConnect**

Die virtuelle Maschine läuft dann weiter. Ein „Reconnect" ist durch ein neues LOGON und die Eingabe von **Begin** möglich.

14.6 VM-Kommandos

Allgemeine Kommandostruktur

Ein VM-Kommando besteht aus einem Kommandonamen, dem meist Operanden folgen, die in Verbindung mit dem jeweiligen Kommando mitgegeben werden können. Einem Kommando kann das Schlüsselwort **CP** vorangestellt werden, damit es sofort vom Control Program verarbeitet wird, bzw. um dem CMS zu signalisieren, dass das Kommando an das CP weitergegeben werden soll.

Die Kommandos sind nicht Case-sensitiv, d. h. sie können in Groß- oder Kleinbuchstaben eingegeben werden.

Beispiele:

```
cp query time
CP HELP
```

Um Hilfe für ein bestimmtes Kommando zu erhalten, wird das jeweilige Kommando als Argument mitgegeben: **cp help query**

Kommandos müssen nicht in voller Länge, sondern können „abgeschnitten" oder als Kürzel eingegeben werden, solange das Kommando eindeutig bleibt. Das QUERY-Kommando beispielsweise lässt dadurch auf fünf verschiedene Arten schreiben:

```
QUERY
QUER
QUE
QU
Q
```

Und hier ein Beispiel für ein Kürzel:

```
MESSAGE
MSG
```

Privilegierungsklassen

CP-Kommandos sind in Privilegierungsklassen eingeteilt, die die notwendige Autorisierung kennzeichnen, die benötigt wird, damit das Kommando aufgerufen werden kann. Die Autorisierung wird vom Systemprogrammierer im VM Directory definiert. Es werden sieben Klassen von A bis G unterschieden.

Neben der Berechtigung, Kommandos ausführen zu können, steuern die Privilegierungsklassen in manchen Fällen auch den Output des Kommandos in Abhängigkeit von der Klasse.

Program Function (PF) Keys

Eine nützliche Einrichtung, um Tastatureingaben zu sparen und effizienter zu arbeiten, sind die Program Function (PF) Keys. Mit einem CP-Kommando können diese nach eigenen Wünschen gesetzt werden.

```
CP SET PFxx [options] command
```

Hierbei gilt: **xx** ist die Nummer der PF-Taste zwischen 1 und 24. **options** ist ein optionaler Kommando-Modifier. **command** ist das Kommando, das mit der PF-Taste verknüpft werden soll.

Die Modifier haben die folgende Bedeutung:

Delayed
: Das entsprechende Kommando wird im Eingabebereich angezeigt, wenn die PF-Taste gedrückt wird. Das Kommando kann dann modifiziert werden, bevor es mit der Enter-Taste ausgelöst wird.

IMMED
: Das Kommando wird sofort ausgeführt, wenn die PF-Taste gedrückt wird. Das Kommando wird nach der Ausführung in der Kommandozeile angezeigt.

NODISP
: Wie IMMED, nur dass das Kommando nach der Ausführung nicht angezeigt wird.

Mit CP QUERY PF kann die PF-Tastenbelegung abgefragt werden.

Beispiele:

```
CP SET PF1 NODISP HELP
CP SET PF3 IMMED QUERY TIME
CP SET PF12 RETRIEVE
```

14.7 Umgang mit Files

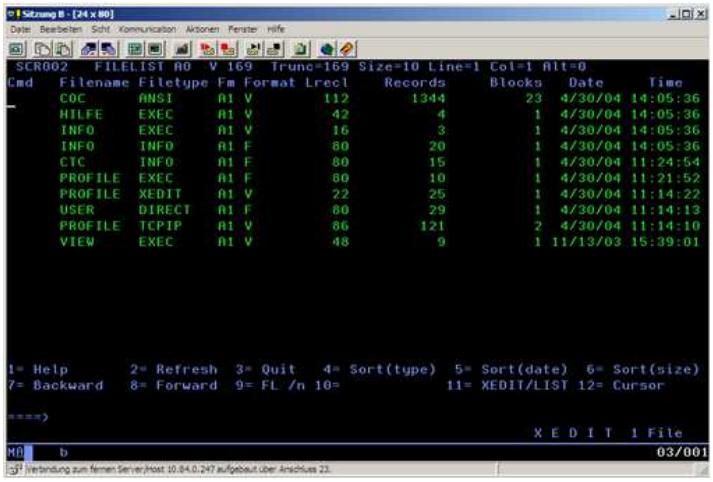
Daten unter VM werden entweder in einem Shared File System (SFS) oder auf einer Minidisk gespeichert. Als dritte Alternative steht auch noch das Byte File System (BFS) zur Verfügung. Um nachvollziehen zu können, wo bereits bestehende Daten gespeichert sind, gibt es ein entsprechendes Kommando – geben Sie in der Kommandozeile einfach q accessed ein:

```
q accessed
Mode  Stat     Files  Vdev  Label/Directory
A      R/W        10  191   R02191
D      R/O        27  192   SCO192
M      R/O       974  592   TCM592
S      R/O       687  190   MNT190
Y/S    R/O      1792  19E   MNT19E
Z/Z    R/O       492  19D   MNT19D
Ready; T=0.01/0.01 14:44:50
```

Das q steht für *query*, und mit accessed werden die Medien angezeigt, auf die zugegriffen werden kann. Aus der Vdev-Spalte ist ersichtlich, ob ein SFS oder eine Minidisk dahinter steckt. Im Falle von SFS wird in der Spalte DIR angezeigt, im Falle von Minidisks eine Gerätenummer, die die Adresse der Minidisk darstellt. In unserem Fall sind es ausschließlich Minidisks.

Mit dem **filelist**-Kommando können die verfügbaren Files aufgelistet werden.

Abbildung 14.6: Ausgabe des filelist*-Kommandos*



Die oberste Zeile des Bildschirms gibt allgemeine Informationen über die Anzeige aus. Die nächste Zeile bezeichnet die Spalten. In der Spalte **CMD** können Zeilenkommandos eingegeben werden. In den übrigen Spalten werden Informationen über die angezeigten Files ausgegeben, wie beispielsweise Format, Satzlänge, Blöcke etc., sowie Informationen, wann die letzten Änderungen vorgenommen wurden.

Ganz unten in der Anzeige werden die Program Function Keys (PF-Keys) angezeigt.

In der **CMD**-Spalte können Kommandos eingegeben werden, um beispielsweise Inhalte eines Files anzeigen zu lassen oder um Files zu löschen oder umzubenennen. Wenn im Anzeigebereich nicht alle anzuzeigenden Daten Platz finden, kann mit **PF8** (nach unten) und mit **PF7** (nach oben) geblättert werden.

Umbenannt wird ein File beispielsweise, indem in der **CMD**-Spalte **rename / newfile** eingegeben wird, wobei der Schrägstrich für das ausgewählte File steht, das in der Zeile angezeigt wird.

Mit **PF3** kann die File-Liste beendet werden.

Das Kommando **dirlist** dient dazu, detaillierte Informationen über ein Verzeichnis auszugeben. Als Argument wird der Name des Verzeichnisses oder der File Mode mitgegeben. Ohne Argument wird der Default File Pool angezeigt. Der Bildschirm ist vom Layout her dem von **filelist** sehr ähnlich.

Um eine Liste der verfügbaren Kommandos zu erhalten, geben Sie in der **CMD**-Spalte einfach das Kommando **help** ein.

14.8 Virtuelle Netzwerke mit z/VM

Virtuelle Netzwerke sind ein hervorragendes Mittel, um Systeme, die unter z/VM betrieben werden, miteinander in Verbindung zu bringen. Diese Verbindungen können ohne real vorhandene Hardware eingerichtet werden.

Es gibt in Verbindung mit Linux-Systemen hierzu drei unterschiedliche Möglichkeiten.

1. Virtual Channel-to-Channel (vCTC)

   Mit vCTC werden Punkt-zu-Punkt-Verbindungen zwischen Gastsystemen ohne reale Kanalzuweisungen ermöglicht. Eine reale Channel-to-Channel-Verbindung wird benötigt, wenn zwei physische Maschinen auf effiziente Art mit Hilfe des Channel-to-Channel Adapter-Protokolls verbunden werden sollen. Mit z/VM können virtuelle Channel-to-Channel Adapter definiert werden, über die virtuelle Maschinen dann ebenfalls über das CTCA-Protokoll kommunizieren können. Dies macht vor allem dann Sinn, wenn virtuelle Linux-Systeme mit anderen Systemen zusammengebracht werden sollen, die das Inter-User Communication Vehicle (IUCV) nicht unterstützen. Hierzu gehören beispielsweise das VSE/ESA oder z/OS.

2. Inter-User Communication Vehicle (IUCV)

   Eine Punkt-zu-Punkt-Verbindung mit TCP/IP zwischen Linux-Gastsystemen kann mit Hilfe von IUCV eingerichtet werden. Dies ist ein VM-spezifisches Protokoll für eine Maschinen-Maschinen-Kommunikation. Die Kernels von Linux for S/390 und Linux on z/Series beinhalten einen IUCV-Driver, mit dem virtuelle Linux-Systeme zusammengebracht werden können. Linux behandelt IUCV-Verbindungen wie jede andere TCP/IP-Verbindung. IUCV wird auch vom VM-eigenen TCP/IP Stack unterstützt. Somit kann mit dem IUCV auch eine Verbindung zwischen Linux und VM realisiert werden.

3. VM Guest LAN

   Mit dieser Art von Netzwerk wird eine LAN-Verbindung zwischen z/VM-Gastsystemen ermöglicht, ohne dass ein reales LAN benötigt wird.

   VM Guest LAN wurde mit z/VM Version 4.2 eingeführt und mit z/VM 4.3 erweitert und verbessert.

   Es gibt zwei Typen:

   - HiperSockets wurden mit z/VM 4.2 eingeführt und emulieren HiperSocket Networking in einem z/VM Image.
   - OSA (Open Systems Adapter) wurde mit z/VM 4.3 eingeführt und emuliert OSA Networking in einem z/VM Image.

Kapitel 15

Linux auf dem Mainframe

15.1 Was ist Open Source Software?

In den Anfangszeiten der Computer wurde Software von einer recht überschaubaren Anzahl von Programmierern und Entwicklern geschrieben. Der Austausch des zugrundeliegenden Quellcodes dieser Software unter den Entwicklern war völlig selbstverständlich. Als die kommerzielle Bedeutung des Softwaresektors jedoch erkannt wurde und verstärkt auch Manager aus anderen Branchen mit den dort vorherrschenden Geschäftsmodellen in den IT-Sektor wechselten, veränderte sich die Situation, und es entwickelte sich das, was heute unter dem Begriff ***proprietäre Software*** bekannt ist, also Software, auf die jemand Besitz- und Kontrollansprüche erhebt.

Nicht alle waren glücklich über diese Entwicklung, beispielsweise auch ein Entwickler namens Richard Stallman, der seit 1971 in den MIT Artificial Intelligence Labs tätig war und dort erfahren hatte, wie wichtig und nutzbringend die Freiheit im Umgang mit Information und eben auch Quellcode war. So wurde Stallman, der unter anderem den Editor Emacs entwickelte, zum bekennenden Gegner pro-

prietärer Software, verfasste zahlreiche Texte zum Thema Freie Software, gründete 1984 das GNU-Projekt[1] und später die gemeinnützige *Free Software Foundation* (FSF)[2].

Das freie GNU-System musste von Grund auf neu geschrieben werden. Der erste Schritt war die Entwicklung eines C-Compilers zur Implementierung der Systemkomponenten, und bis zum Jahr 1990 hatte Richard Stallman mit Hilfe einiger eifriger Mitstreiter ein nahezu vollständiges UNIX-System geschrieben. Allerdings war der Kernel dieses Systems zu komplex konzipiert und konnte nicht zügig genug fertiggestellt werden. Im Jahr 1991 startete dann Linus Torvalds, damals Student in Helsinki, über das Internet eine Initiative, mit der er Gleichgesinnte zur gemeinsamen Entwicklung eines „(free) operating system" suchte, das schon bald den Namen „Linux" erhielt. Dieser so entstehende Kernel wurde dann nach einigem Hin und Her unter die GNU General Public License (GPL) gestellt. Eine optimale Kombination, denn der Kernel bildete zusammen mit den GNU-Programmen ein komplettes Betriebssystem. Es dauerte jedoch noch weitere drei Jahre, bis die Version 1.0 von Linux freigegeben wurde. Die Kombination aus den GNU-Komponenten und dem Linux-Kernel ist heute das, was als „GNU/Linux" bezeichnet wird und was für so gut wie alle wichtigen Prozessorplattformen zur Verfügung steht.

Die Grundidee hinter *Open Source Software* (OSS)[3] ist relativ einfach: Wenn Programmierer Source Code lesen, verändern und verbreiten dürfen, entwickelt sich die Software automatisch weiter. Eine Software hingegen, die nicht aktiv weiterentwickelt wird, stirbt. Beispiele gibt es genügend.

Das Internet und die verfügbaren Tools und Technologien, die die Verteilung, den Zugriff auf den Code und die Kommunikation zwischen den Entwicklern ermöglichen, haben „Open Source" zu dem Phänomen werden lassen, das es heute ist.

OSS-Lizenzmodelle

„Quelloffen" (Open Source) bedeutet aber eben nicht nur freien Zugang zum Quellcode. Auch Open Source Software kennt (inzwischen zahlreiche) Lizenzbestimmungen und -modelle.

[1] „The GNU Project was launched in 1984 to develop a complete UNIX style operating system which is free software: the GNU system. (GNU is a recursive acronym for „GNU's Not UNIX"; it is pronounced „guh-noo."); vgl. **http://www.gnu.org**

[2] „The Free Software Foundation (FSF), established in 1985, is dedicated to promoting computer users' rights to use, study, copy, modify, and redistribute computer programs. The FSF promotes the development and use of Free Software, particularly the GNU operating system, used widely in its GNU/Linux variant." Vgl. **http://www.fsf.org**

[3] Der Terminus „Open Source Software" hat sich zur allgemeinen Bezeichnung „quelloffener Software" durchgesetzt. Teilweise stehen die dem zugrunde liegenden Auffassungen (**http://www.opensource.org**) aber auch im Widerspruch zu der in Lizenzfragen deutlich rigideren Auslegung von „Free Software". Weitere Details sollen hier nicht ausgeführt werden und sind, ausgehend von den genannten Websites, leicht nachzulesen.

Die GNU General Public License (GPL) spielt in der Praxis zweifellos die größte Rolle. Sie definiert eine ganze Reihe von Rechten und Pflichten insbesondere in Bezug darauf, was mit dem Source Code gemacht werden darf und was nicht. Dies ist ein deutlicher Unterschied zu den traditionellen Lizenzregelungen, wo es eher darum geht, die Rechte des Herstellers zu schützen und zu definieren, was ein Anwender mit der Software nicht machen darf.

So darf – und das ist der zentrale Punkt – „GPLte" Software im Source Code verändert werden, solange die Modifikationen wiederum offengelegt und der Community zur Verfügung gestellt werden. Durch dieses Konzept kann eine Software schnell und effektiv erweitert und verbessert werden, selbst dann, wenn der ursprüngliche Entwickler seine Arbeit nicht mehr weiterführen kann oder will.

Der Begriff „frei" meint darum auch nicht „kostenlos", sondern ist vielmehr im Sinne von „freiheitlich" zu verstehen. In den Worten Richard Stallmans:

> "Free software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech," not as in "free beer."[4]

Der Schlüssel zur Portabilität und zur Verfügbarkeit von Open Source Software ist die GNU Compiler Collection, kurz *gcc*. Übrigens: Über 90 Prozent des MVS-Codes ist inzwischen in *C* geschrieben und wird mit gcc übersetzt. Das heißt aber natürlich nicht, dass MVS ein offenes Betriebssystem ist.

Kommerzielle Unternehmen und OSS

IBM unterstützt die Idee der freien Software als Hersteller in großem Umfang. So wurde beispielsweise *Eclipse* als ursprünglich interne Entwicklung einer gemeinnützigen Stiftung übergeben. Der Domino Go Webserver wurde zu Gunsten des Apache Webservers aufgegeben, der seit dem Jahr 2000 als *IBM HTTP Server* mit einigen Security-Erweiterungen von IBM vermarktet wird.

Intel kümmert sich intensiv um die Linux-Portierung auf den ia64 Mikroprozessor. Die Portierung hat höchste Priorität in der Unternehmensstrategie. Sun hat sein StarOffice unter dem Namen OpenOffice.org unter der GPL freigegeben. Ziel ist es, damit auch im Zusammenhang mit Linux auf dem Desktop etwas zu bewirken. Medienwirksame Projekte wie beispielsweise die Umstellung der Stadtverwaltung München auf ein Linux-System zeigen, dass sich vieles bewegt.

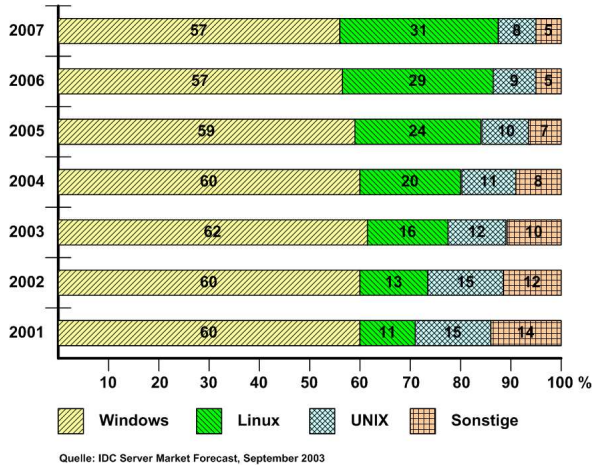
[4] http://www.gnu.org/philosophy/free-sw.html

15.2 Die Linux-Philosophie

Linux, ursprünglich im akademischen Bereich angesiedelt, breitet sich inzwischen auch in kommerziellen Umgebungen aus. Mit gutem Grund!

Linux bietet genau die auf Standards basierende Systemarchitektur, die UNIX versprochen hatte, allerdings aufgrund der divergierenden Derivate nicht halten konnte. Das ist einer der Hauptgründe, warum die Akzeptanz von Linux so rasch zugenommen hat. Diese Akzeptanz zeigt sich darin, dass Linux innerhalb kürzester Zeit zum am schnellsten wachsenden Serverbetriebssystem avancierte.

Abbildung 15.1: Verbreitung von Server-Betriebssystemen



Was die Anzahl installierter Serversysteme angeht, steht Linux bereits an zweiter Stelle hinter Windows NT, und es wäre nicht verwunderlich, wenn Linux in absehbarer Zeit die Führung übernimmt. Kein anderes Betriebssystem konnte in der Vergangenheit dieses enorme Wachstum vorweisen. Von einem Außenseiter zu einem ernst zu nehmenden Betriebssystem ist es tatsächlich eine beeindruckende Entwicklung.

Vorteile von Linux als Betriebssystem

Linux wird inzwischen in zahlreichen Umgebungen als Alternative zu anderen Systemen eingesetzt oder evaluiert. Warum ist das so? Was sind die besonderen Merkmale, die für einen Einsatz von Linux sprechen?

Unabhängigkeit von einer Hardware-Architektur

Durch eine spezielle Abstraktionsebene zwischen Hardware und Betriebssystem ist Linux portierbar wie kein anderes System zuvor. Das System kann

auf den unterschiedlichsten Plattformen eingesetzt werden, und der Source Code bleibt dennoch derselbe. So mussten auch für den Einsatz auf dem IBM Mainframe nur wenige Anpassungen vorgenommen werden.

Linux läuft bereits heute auf so gut wie allen verfügbaren Plattformen. Und dass es auf vereinzelte Plattformen (noch) nicht portiert wurde, hat grundsätzlich keine technischen Gründe.

So lässt die bisher nachgewiesene und realisierte Portierbarkeit den Schluss zu, dass Linux auch künftig auf Plattformen laufen wird, die es heute noch gar nicht gibt.

Funktionalität

Die Funktionalität von Linux wird ständig erweitert. Komponenten für das System Management, die Security und die Performance stehen in großem Umfang bereits zur Verfügung. Die Geschwindigkeit, mit der fehlende Funktionalitäten entwickelt oder von anderen Systemen übernommen und angepasst wurden, ist beeindruckend. Auch hier ist zu erwarten, dass Funktionen, die heute noch fehlen, innerhalb kürzester Zeit entwickelt bzw. verfügbar gemacht werden.

Unterstützung neuer Technologien

Dank der Anzahl aktiver Entwickler und zahlreicher renommierter Unternehmen, die ihre Strategien auf Linux aufbauen, werden auch neueste Entwicklungen sehr schnell unterstützt. Linux war beispielsweise eines der ersten Betriebssysteme, das die 64-Bit-Adressierung unterstützt hat. Damit können große Datenmengen im Hauptspeicher gehalten werden, ohne dass sie auf externe Geräte auslagert werden müssen.

Aufbau und Struktur von Linux

Zwei Begriffe spielen eine große Rolle, wenn es um die Struktur von Linux geht. Das ist zum einen die GNU Compiler Collection (*gcc*) als „Werkbank und Werkzeugpalette" für die Entwickler und der Kernel als Herz der Betriebsumgebung.

Kernel

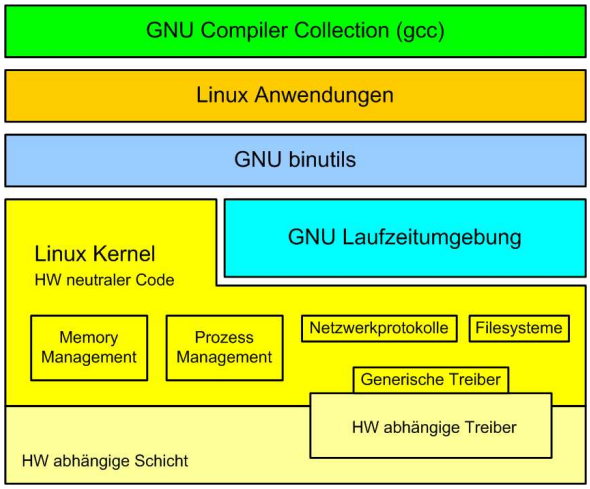
Der Kernel ist das Herz eines Linux-Systems; er kümmert sich vor allem um die Ressourcenzuordnung zu anderen Programmen, die auf dem System laufen. Er enthält außerdem die Basisfunktionen wie Multitasking, virtuelle Speicherverwaltung, TCP/IP Networking etc.

Die wichtigsten Komponenten sind das Memory- und Prozessmanagement sowie die Unterstützung der Netzwerkprotokolle und Filesysteme.

GNU Compiler Collection (gcc)

Hinter der GNU Compiler Collection steht eine Sammlung von Werkzeugen und Hilfsmitteln, die es ermöglichen, Programme aus unterschiedlichen Umgebungen, die in unterschiedlichen Programmiersprachen geschrieben sein können, auf zahlreichen Plattformen ablauffähig zu machen. Zu den unterstützten Programmiersprachen gehören derzeit C, C++, Objective-C, Ada, Fortran und Java.

Abbildung 15.2: Aufbau und Struktur von Linux



Anwendungen

Zahlreiche Linux-Anwendungen werden über sog. „Distributionen" gebündelt und verbreitet. Zusammenstellung und Abstimmung der oft in die Tausende gehenden Software-Pakete charakterisieren diese Distributionen, die im Rahmen freier Projekte oder von kommerziellen Anbietern entwickelt werden. Natürlich können auch eigene Anwendungen geschrieben und integriert werden.

GNU Binary Utilities

Die GNU Binary Utilies sind wiederum Werkzeuge, die als Teil einer Distribution ausgeliefert werden. Hierzu gehören beispielsweise ein Linkage Editor (ld) und ein Assembler (as).

Treiber

Ein Treiber oder Gerätetreiber ist ein File oder ein Programm, das die Kommunikation eines Systems mit einem spezifischen Gerät ermöglicht. Wenn Linux mit einem Gerät kommunizieren soll, muss ein entsprechender Gerätetreiber mit dem Linux-Kernel verknüpft werden. Dies wird erreicht, indem der Treiber entweder mit dem Kernel zusammen kompiliert wird oder indem er als so genanntes Modul integriert wird.

Laufzeitumgebung

Um eine bereits kompilierte Anwendung zum Ablauf zu bringen, werden zahlreiche Programme und Files benötigt, die in der Summe als *Laufzeitumgebung* (*Runtime Environment*) bezeichnet werden. Es handelt sich um ein Verzeichnis, in dem die entsprechenden Files untergebracht sind. Beispiele dafür sind **libgcj** für Anwendungen, die mit GCJ for Java geschrieben, oder **libg2c** für Anwendungen, die mit dem G77 Fortran Compiler übersetzt wurden.

Für in Java geschriebene Programme muss außerdem eine Java Virtual Machine (JVM) zur Verfügung stehen, die eine Laufzeitumgebung für den plattformunabhängigen Bytecode eines Java-Programms bildet.

Der Linux-Kernel

Grundbaustein eines jeden Linux-Systems ist der Betriebssystemkernel. Der Kernel wurde ursprünglich von Linus Torvalds alleine entwickelt und wird inzwischen von einer Gruppe, die auch als „Innerer Kreis" (Inner Circle) bezeichnet wird, weiterentwickelt. Diese Gruppe bestimmt letztendlich, welcher Code in den Kernel aufgenommen wird. Das letzte Wort hat dabei nach wie vor Linus Torvalds selbst. Bei der Entwicklung des Kernel hat die technische Qualität die oberste Priorität, da durch einen sicheren und stabilen Kernel eine solide Grundlage für das gesamte System geschaffen wird.

Die Kernel-Versionen folgen einem Nummerierungsschema: Die ungeraden Versionen (z. B. 1.9.x, 2.5.x) sind Entwicklerversionen, die schnell und häufig freigegeben werden, damit Entwickler damit arbeiten können und schnelles Feedback zu Features, möglichen Bugs, fehlenden Device-Treibern etc. geben. Diese Versionen sollten nicht unbedingt in einer produktiven Umgebung eingesetzt werden.

Die geraden Nummern (z. B. 1.8.x, 2.6.x) sind stabilere Versionen mit fest definierten Features, weniger Bugs und getesteten Gerätetreibern. Es gibt weniger häufig Freigaben. Diese Versionen sind auch für produktive Umgebungen geeignet.

Per Definition ist Linux ein Multiuser- und Multitasking-Betriebssystem. Das heißt, dass verschiedene Benutzer zur gleichen Zeit unterschiedliche Prozesse in einem Linux-System ausführen können, ohne dass diese Prozesse dabei kollidieren.

Neuere Versionen des Kernels sind modular aufgebaut. Dies bietet die Möglichkeit, bestimmte Funktionen in Module auszulagern und bei Bedarf im laufenden Betrieb nachzuladen.

Dieser modulare Aufbau bewirkt eine effiziente Nutzung des Systems und spart Ressourcen, da der Kernel während des Betriebs auf fast jedes Gerät angepasst werden kann. Dies ist unter anderem auch der Grund, warum Linux auf so vielen verschiedenen Plattformen verfügbar ist, von der Armbanduhr über PDAs und PCs bis hin zum Mainframe.

Der aktuelle Kernel 2.6 bringt einige Neuerungen, die gerade für den Einsatz in großen Umgebungen und damit für den Mainframe sehr interessant sind: optimiertes Speichermanagement, unterbrechbare (preemptive) Prozesse, eine POSIX-konforme Threading-Bibliothek, asynchrone I/Os und Completion Events sowie die Unterstützung von Plattengeräten mit mehr als 2 Terabyte.

Prozesse unter Linux

Die Aufteilung ausführbarer Programme in Prozesse ist ebenfalls eine wichtige Eigenschaft von Linux, welche die Stabilität des Systems deutlich erhöht. Alle Prozesse sind über eine Process ID (PID) eindeutig einem bestimmten Benutzer zugeordnet. Selbst wenn Programme von verschiedenen Usern gleichzeitig genutzt werden, können sie zu jeder Zeit dem entsprechenden Benutzer zugeordnet werden.

Ein Administrator kann somit feststellen, welche Programme von welchem Benutzer zu einem bestimmten Zeitpunkt benutzt werden und wo genau wie viele Systemressourcen verbraucht werden. Dadurch sind systemkritische Anwendungen sehr schnell und sicher zu ermitteln, und das System kann optimal an die verfügbaren Ressourcen angepasst werden.

Ein weiterer wichtiger Punkt hierbei ist, dass Prozesse, welche die Stabilität des Systems gefährden oder bereits abgestürzt sind, sofort beendet und bei Bedarf neu gestartet werden können, ohne dass der Betrieb des gesamten Systems beeinträchtigt wird. Dies ist eine wichtige Eigenschaft bei der Beurteilung der Ausfallzeiten und der Wartungsarbeiten eines Linux-Systems.

Konfigurationsfiles

Die Konfigurationsdateien unter Linux sind alle textbasiert aufgebaut. Das heißt, dass sämtliche Änderungen an der Konfiguration mit einem einfachen Texteditor vorgenommen werden können. Da bei den ersten Linux-Versionen noch keine Konfigurationstools vorhanden waren, war das Bearbeiten von Textdateien die einzige Möglichkeit, das System anzupassen.

Die Arbeit mit textbasierten Konfigurationsdateien hat sich allerdings auch in der späteren Entwicklung als sehr vorteilhaft erwiesen, denn der Systemadministrator hat dadurch die komplette Kontrolle über die Konfiguration des Systems oder einzelner Programme. Mit einem Blick in die Konfigurationsdatei sind alle Einstellungen auf einen (geübten) Blick sichtbar, und es können keine versteckten oder unbeabsichtigten Funktionen versehentlich aktiviert oder deaktiviert werden.

Zusätzlich ist es jedoch möglich und vor allem für Einsteiger auch wünschenswert, dass grafische Administrationsprogramme für die Konfiguration zur Verfügung gestellt werden. Linux ist in diesem Bereich zweifellos eines der flexibelsten Systeme, die es gibt.

X Window und Motif

Linux ist wie die verschiedenen UNIX-Derivate auch zunächst einmal ein textbasiertes System ohne grafische Benutzeroberfläche. In den 80-er Jahren wurde jedoch für die verschiedenen UNIX Systeme am Massachusetts Institute for Technology (MIT) mit dem *X-Protokoll* und *Motif* die Grundlage für grafische Oberflächen geschaffen. Das X-System stellt dabei praktisch den Unterbau für eine grafische Entwicklungsumgebung dar. Es übernimmt die Aufgabe eines Servers, der grafische Dienste zur Verfügung stellt. Um die grafischen Dienste auf dem Client sichtbar zu machen, kommt eine X-Oberfläche zum Einsatz.

Auf Basis dieses Systems wurden daher verschiedene grafische Oberflächen aufgesetzt. Unter Linux war es das Ziel, ein freies, herstellerunabhängiges X-System zu entwickeln. Daher wurde von einer Gruppe von Entwicklern ein neues System mit dem Namen *XFree86* entwickelt. Als grafische Oberflächen auf dem Desktop haben sich heute hauptsächlich KDE und GNOME durchgesetzt.

Shells unter Linux

Eine Shell ist die Time-Sharing-Schnittstelle zu einem UNIX/Linux-System, vergleichbar mit dem TSO unter z/OS. Der Unterschied zu TSO ist jedoch, dass in UNIX/Linux-Systemen in der Regel mehrere Shells zur Auswahl stehen. Beispiele für Shells, die unter Linux genutzt werden können, sind: Bourne Shell (/bin/sh), Bourne Again Shell (/bin/bash), C Shell (/bin/csh), erweiterte C Shell (/bin/tcsh), Korn Shell (/bin/ksh), Z Shell (/bin/zsh).

Um festzustellen, welche Shell in einem laufenden System als Default verwendet wird, kann man die Variable SHELL abfragen:

```
echo $SHELL
```

Beim Login wird die Shell aufgerufen, die im /etc/passwd-File für den jeweiligen User eingetragen ist. Mit dem chsh-Kommando kann die Shell interaktiv geändert werden.

Editoren unter Linux

Es stehen unter Linux zahlreiche, um nicht zu sagen: unzählige Editoren zur Verfügung. vi/vim/elvis, emacs/xemacs, joe, jed, jove, ed/sed, nano, pico, ne, Nedit, THE (The Hessling Editor) …

In Verbindung mit Shells und den Editoren werden auch oft Scriptingsprachen eingesetzt, beispielsweise ash/bash/csh/tcsh/ksh/ksh93/zsh, Perl, Python, Tcl.

Unter VM (und auch den anderen Großrechner-Systemen wie z/OS und z/VSE) wird vorwiegend der ***Restructured Extended Executer*** (REXX) verwendet, der ursprünglich einmal in einer VM-Umgebung entstanden ist.

15.3 Linux im Server-Bereich

Eingesetzt wird Linux zunächst einmal im so genannten „Infrastruktur-Bereich", d. h. als Router, Firewall, Webserver, Fileserver, Mailserver, Proxy, Datenbankserver und Printserver.

Einige Beispiele für die Bedeutung von Linux in diesem Bereich sollen hier aufgezeigt werden.

Linux als Webserver

Der am weitesten verbreitete Webserver der am Internet angeschlossenen Systeme ist der Apache Webserver, der ebenfalls auf Open Source basiert. Er hat inzwischen einen Marktanteil von deutlich über 60 Prozent. Die aktuellen Zahlen sind jederzeit von der Website von Netcraft abrufbar.[5] Apache Webserver wiederum werden zum überwiegenden Teil auf Linux-Systemen betrieben.

Linux als Firewall

Die Sicherheit und die Stabilität eines Linux-Systems sind Eigenschaften, die Linux auch als Firewall-Server auszeichnen. Allerdings gibt es keine allgemeingültige Lösung in diesem Bereich. Das Konzept für das Netzwerk muss genau auf die Bedürfnisse des Unternehmens abgestimmt werden. Im Linux-Kernel selbst sind jedoch bereits einige Sicherungen eingebaut. So beispielsweise ein Paketfilter, der als grundlegende Firewall-Funktionalität überprüft, ob eine korrekte Verbindung initiiert worden ist. Ab der Kernel-Version 2.4 gibt es inzwischen auch ein Intrusion-Detection-Modul zur automatischen Einbruchserkennung. Zahlreiche weitere Produkte und Programme bezüglich Sicherheit stehen als Open Source Software oder auch als kommerziell vermarktete Produkte zur Verfügung.

Linux als Print- und Fileserver

Ein Samba-Server kann in einem Netzwerk einen Windows-Server „vortäuschen". So kann ein Windows-Benutzer von einem Desktop aus beispielsweise auf einen Samba-Server zugreifen, der für ihn sämtliche File- und Print-Operationen erledigt.

Die Aufgaben des Servermanagements können mit Samba weitgehend durch das grafische Webtool SWAT durchgeführt werden. Neben erweiterten Kon-

[5] http://news.netcraft.com

figurationstools gibt es ab Samba 2.2 auch eine eigene Datenbank, in der Druckjobs abgelegt und verwaltet werden können.

Dies ist vor allem aus Kostensicht für ein Unternehmen interessant, da keine teuren Windows-Serverlizenzen anfallen und durch den Einsatz von Linux nebenbei die Stabilität des Systems verbessert wird.

Linux als Datenbankserver

Der sichere und stabile Betrieb macht Linux auch als Datenbankserver zu einer Alternative zu den gängigen UNIX-Systemen oder Windows Server-Plattformen. Diesen Umstand haben die führenden Datenbankhersteller erkannt und ihre Datenbanken unter Linux verfügbar gemacht.

Die Marktführer in diesem Bereich, Oracle und IBM mit DB2, spielen da ganz vorne mit. Neben diesen kommerziellen Datenbanken existieren auch freie Datenbanksysteme, die als Open-Source-Datenbanken zur Verfügung stehen.

Somit kann Linux auch als Datenbankserver im Unternehmensumfeld als Alternative zu den bisherigen UNIX- und Windows-Server-Systemen eingesetzt werden.

15.4 Commitment von IBM zu Linux

Allgemein hat IBM die Vorteile von Open Source Software erkannt. So wurde bereits Anfang 2000 die eigene Webserver-Reihe, der Domino Go Webserver, zugunsten des Apache Webservers aufgegeben, der als IBM HTTP Server mit einigen Security-Erweiterungen vermarktet wird.

IBM sieht die Betriebssysteme auch nicht mehr als die strategische Komponente im Portfolio, die sie früher einmal waren. Bereits im Mai 2000 äußerte sich der damalige CEO von IBM, Lou Gerstner, vor Journalisten:

„The first is host or operating system software, which is now and always will be a continually shrinking percentage of our revenue. We're not fretting about this because, frankly, operating systems no longer hold the strategic importance they once had in our industry. In a world of open standards, which is where the world is going, the operating system platforms – ours or anyone else's in the open world – are not going to be control points anymore. They won't be where the growth is, and they won't even command the margins they once did."

IBM bekennt sich voll zu Linux und vertritt die Auffassung, dass Linux die Schlüsselplattform für künftige E-Business-Anwendungen sein wird.

Seit dem Jahr 2001 investiert IBM jährlich eine Milliarde Dollar in Linux-Aktivitäten. Rund 2.000 Linux-Entwickler sind bei IBM im Einsatz. Irving Wladawsky-Berger (Vice President, Technology and Strategy, IBM) sagt dazu Folgendes:

„Now in IBM, we are committed to embracing Linux across everything that we do. Linux runs on our Intel servers, on our Power-based servers, on our iSeries. It runs on our mainframes, on our OEM technology, on our storage servers. Linux, as we know, is the only operating system of which you can safely say: It will run on architectures that have not yet been invented."

Viele Beiträge für Linux stammen von IBM, beispielsweise in vollem Umfang das Journaling File System (JFS) sowie der Logical Volume Manager (LVM).

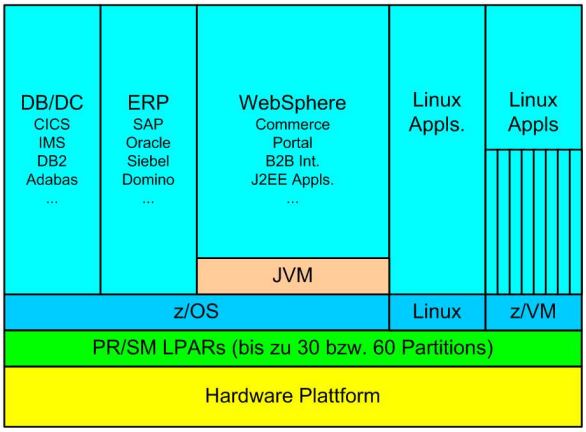
15.5 Neue Anwendungen auf dem Mainframe

Zahlreiche Open Source Tools und weitere Anwendungen, die für Linux entwickelt wurden, sind nun für den Mainframe verfügbar. Wenn eine Anwendung vernünftig entworfen ist, so dass sie verteilt werden kann, kann auch hier Linux als Plattform eine wichtige Rolle spielen. Anwendungen unter Linux sind oft flexibler und können schneller entwickelt werden.

Viele Open Source-Anwendungen wurden speziell für Linux entwickelt. Diese Anwendungen stehen nun auch auf dem Mainframe zur Verfügung. Ein klassisches Beispiel dafür wird durch das Kürzel LAMP gekennzeichnet. Es steht für die Möglichkeit, eine komplette Internet-Infrastruktur auf der Basis von Open Source aufbauen zu können.[6]

Das Portfolio der möglichen Workloads auf dem Mainframe wird dadurch deutlich erweitert.

Abbildung 15.3: Unterschiedliche Workloads auf einer homogenen HW-Plattform



[6] LAMP steht für die Komponenten *L*inux (Betriebssystem), *A*pache (Webserver), *M*ySQL (Datenbank) und *P*erl oder *P*HP (Programmiersprache).

Integration mit Backend-Systemen

Moderne Anwendungen basieren auf einer 3-Tier-Architektur. Backend-Systeme werden hierbei zumeist auf dem Mainframe betrieben. Linux auf dem Mainframe ermöglicht es, die mittlere Ebene, die oft aus unterschiedlichen Gründen auf UNIX/Linux-Systemen betrieben wird, nah an die Backend-Systeme heranzubringen. Dabei können die Vorteile besserer Performance und geringerer Komplexität ausgenutzt werden.

VSE/ESA hat keine eigene Java-Umgebung. Für VSE-Shops bietet Linux die Möglichkeit, Java-Anwendungen dicht bei den VSE/ESA-Systemen anzusiedeln und zu integrieren.

Virtualisierung

Auf dem Mainframe können mehrere logische Server auf einem physischen Server betrieben werden. Möglichkeiten in diesem Zusammenhang sind entweder Logical Partitions (LPARs) auf der Basis von HW/Microcode (vgl. Seite 25) oder Virtual Machine (VM) als Software-Lösung (vgl. Seite 24).

15.6 Linux in einer LPAR

Der wichtigste Unterschied bei der Vorgehensweise im Vergleich zu VM ist, dass ein IPL-fähiges Band erstellt werden muss und die IPLs von der Hardware Management Console (HMC) aus erfolgen müssen.

Wir gehen als Beispiel bei den folgenden Ausführungen von einer Red Hat-Version des Marist College aus. Eine Distribution, die auch heute noch kostenlos von der Website des Marist College[7] heruntergeladen werden kann. Die Vorgehensweise ist jedoch unabhängig von einer Distribution.

Um Linux native in einer Logical Partition (LPAR) zum Laufen zu bringen, sind folgende Schritte notwendig:

1. Besorgen des Codes (Download oder CD/DVD)
2. Aufbau eines IPL-fähigen Bandes
3. Format einer Platte und Erzeugen eines Filesystems
4. Entkomprimieren der Files in das Filesystem auf die formatierte Platte
5. Anpassen der Files im Filesystem

[7] http://www.marist.edu

6. Erzeugen und Aktivieren von Swap Space
7. Herunterfahren des Starter-Systems
8. ReIPL des Linux-Systems von der Platte
9. Nacharbeiten und Anpassungen

15.7 Linux unter z/VM

Wenn die Umgebung einmal eingerichtet und konfiguriert ist, ist das Einrichten eines zusätzlichen virtuellen Servers relativ einfach, wenn entsprechende Automatisationsroutinen eingerichtet sind. Der Netzwerk-Administrator ist für die Verwaltung der IP-Adressen und der Netzwerk-Architektur verantwortlich, der Linux-Administrator für die Konfiguration der jeweiligen Server. Es ist jedoch ratsam, dass sich sowohl ein Linux-Administrator als auch ein Netzwerk-Administrator ein gewisses Maß an z/VM-Skills aneignen, um die Effizienz des Systems bestmöglich auszunutzen. Es ist möglich, sehr viele Aktivitäten weitgehend zu automatisieren.

Bei Überwachung und Betrieb sind die Operatoren und Administratoren gefordert. Viele Dinge im Bereich Serverüberwachung und Performance-Monitoring sind von UNIX/Linux auf die zSeries-Umgebung übertragbar und umgekehrt, so dass mit etwas spezifischer Schulung im Bereich z/VM Monitoring bzw. Linux-Werkzeuge viel erreicht werden kann.

Es gibt mehrere Möglichkeiten, Linux in einer VM-Umgebung zu installieren. Die erste Frage, die sich stellt, ist die, ob der erste IPL von einem Band oder über den Internal Reader erfolgen soll. Wenn ein Band benutzt wird, ist der Vorgang dem in einer LPAR-Installation sehr ähnlich. Einfacher ist es jedoch meistens, den Internal Reader zu nutzen, da man dann kein Magnetbandgerät benötigt.

Ein weiterer Punkt betrifft die Frage, welche Gerätetreiber für Platten eingesetzt werden. Es gibt zwei Möglichkeiten: DASD-Treiber oder Minidisk-Treiber. Beim Einsatz von Linux in einer LPAR bleibt keine Wahl, da hier keine Minidisks zur Verfügung stehen. In VM-Umgebungen werden meist Minidisks eingesetzt.

Für die Vorbereitung und die Installation von Linux unter z/VM wird folgende Vorgehensweise empfohlen:

1. Installationsmethode festlegen
2. Vorbereiten der virtuellen Maschine für Linux
3. Vorbereiten der Netzwerk-Umgebung
4. Besorgen der Binärfiles

5. Kopieren der Files in das VM-System
6. Erzeugen des Kernel Options File
7. Hochfahren des Linux-Startsystems
8. Installation des Root File Systems tarball
9. Anpassen der Konfiguration
10. IPL des fertigen Systems

Bevor Linux on zSeries in einer virtuellen Maschine installiert werden kann, muss diese virtuelle Maschine vorab definiert werden. Dies geschieht über einen CP-Verzeichniseintrag (*CP Directory*).

Beispiel für einen derartigen Eintrag:

```
*************************************************************
*  z/VM 4.3.0  SYSTEM DIRECTORY                             *
*************************************************************
* Marist Linux
USER LNX01 ##### 200M 256M G
 IPL CMS PARM AUTOCR
 MACHINE ESA 4
 CONSOLE 0009 3215
 SPOOL 000C 3505 A
 SPOOL 000D 3525 A
 SPOOL 000E 1403 A
 LINK MAINT 0190 0190 RR
 LINK MAINT 019E 019E RR
 LINK MAINT 019D 019D RR
 LINK TCPMAINT 592 592 RR
* CMS Minidisk
MDISK 0191 3390 0001 0050 MARIST MR
* Boot-Partition
 MDISK 0200 3390 0051 0020 MARIST MR
* Root-Partition
 MDISK 0201 3390 0071 1000 MARIST MR
* User-Partition
 MDISK 0202 3390 1071 2000 MARIST MR
* Swap-Partition
 MDISK 0203 3390 3071 0200 MARIST MR
```

- Die Maschinenkennung ist LNX01. Ein Hauptspeicher von 200 MB wird zugewiesen, der bis zu 250 MB erweitert werden kann. Eine Minimalgröße von 64 MB wird empfohlen.
- Wenn man sich neu an eine virtuelle Maschine anmeldet, wird meistens automatisch CMS hochgefahren. Man könnte hier zwar auch direkt Linux hochfahren,

jedoch ist der Weg über CMS flexibler. Mit CMS kann ein Profil aufgerufen werden, das die Umgebung der virtuellen Maschine beschreibt, bevor der IPL des Linux-Systems vorgenommen wird.

- Die nächste Zeile beschreibt die Architektur (ESA) und die maximale Anzahl der Prozessoren.
- Mit der CONSOLE-Anweisung wird die Konsole definiert. Auch wenn mit einer 3270-Emulation gearbeitet wird, wird hier ein 3215-Gerät definiert.
- Die nächsten drei SPOOL-Anweisungen definieren einen Reader, einen Puncher und einen Printer für die virtuelle Maschine.
- Es folgen Read-only-Links auf CMS Minidisks, die anderen virtuellen Maschinen zugeordnet sind.
- Schließlich werden Minidisks definiert, auf denen das Linux-Filesystem und Daten abgelegt sind.

Umgang mit Geräten

Ein wichtiges Konzept von UNIX/Linux lautet: Alles ist ein File!

Die Hardware (Platten, Drucker, Netzwerkadapter usw.), die an das System angeschlossen ist, sowie viele Software-Mechanismen (wie z. B. der virtuelle Papierkorb /dev/null) werden als File repräsentiert. Die Files befinden sich normalerweise im Verzeichnis /dev.

Mit diesem Konzept ist es möglich, die gesamte Kommunikation mit Geräten auf File-Operationen aufzubauen.

Gerätetypen

Es gibt zwei Arten von Geräten: blockorientierte und zeichen-orientierte. Blockorientierte (z. B. Platten) arbeiten mit blockweiser Datenübertragung und zeichenorientierte (z. B. Printer, Terminals) mit byteweiser Datenübertragung.

Geräte haben eine Major- und eine Minor-Nummer. Die Major-Nummer identifiziert einen Gerätetyp und die Minor-Nummer die Einheit bzw. Instanz auf dem entsprechenden Gerät.

Über das File /proc/devices kann man sich anzeigen lassen, welche Gerätetypen der Kernel unterstützt. Die Ausgabe ist eine Liste von Major-Nummern, gefolgt von dem Basisnamen des entsprechenden Geräte-Files.

Um mit einem Gerät unter Linux arbeiten zu können, müssen die folgenden drei Bedingungen erfüllt sein:

- Es handelt sich letztendlich um Hardware, die angeschlossen (*attached*) sein muss.
- Der Linux-Kernel muss das Gerät unterstützen. Die Kernel-Gerätetreiber können entweder direkt in den Kernel „hineinkompiliert" werden oder als Module nach dem Start und somit auch im laufenden Betrieb nachgeladen werden.
- Es muss ein entsprechendes Geräte-File vorhanden sein. Der Gerätetyp wird über die Major-Nummer bestimmt. Die Minor-Nummer bestimmt ein bestimmtes Gerät aus mehreren identischen Geräten oder einen bestimmten Teil eines Geräts wie beispielsweise eine Partition auf einer Platte.

Die Major- und Minor-Nummern sind in dem File **/usr/src/linux/Documentation/devices.txt** abgelegt. Sie werden auch angezeigt, wenn der Inhalt des Verzeichnisses **/dev** mit dem Kommando **ls -l** abgefragt wird.

Folgende Übersicht fasst die Eigenschaften der zSeries-Blockgeräte zusammen:

Native oder HW-emulierte DASD
: Formatierung: Linux dasdfmt
Device Node: **/dev/dasd<n>**
partitionierbar: ja
Gerät muss partitioniert werden, um eine Boot Record, Kernel-Parameter oder autodetected zu unterstützen

VM Minidisk DASD-Treiber
: Formatierung: CMS FORMAT und RESERVE oder Linux dasdfmt
Device Node: **/dev/dasd<n>** partitionierbar: ja
Kernel muss rekompiliert werden, um **dasd_force_diag** zu unterstützen.

VM Minidisk Minidisk-Treiber
: Formatierung: CMS FORMAT und RESERVE
Device Node: **/dev/mnd<n>** partitionierbar: nein
Default für die Marist Distribution

XPRAM
: Formatierung: keine Formatierung notwendig
Device Node: **/dev/xpram<n>** partitionierbar: nein
Das Filesystem muss eine Blockgröße von 4 K (oder Vielfaches davon) haben.

Direct Access Storage Device (DASD)

Die Linux on zSeries-Gerätetreiber verwalten alle zSeries-Platten als Blockgeräte mit einer Major-Nummer 94. Ein Gerät wird unterteilt in vier adressierbare Be-

reiche, denen eine Minor-Nummer zugeordnet wird. Die erste Minor-Nummer repräsentiert das physische Volume und die anderen drei die Partitions.

Minor 0 = Adresse des physischen Geräts
Minor 1 = erste Partition
Minor 2 = zweite Partition
Minor 3 = dritte Partition

Die zweite Platte hat ebenfalls die Major-Nummer 94 und startet dann mit der nächsten freien Minor-Nummer, wobei eine eindeutige Major/Minor-Nummerierung erreicht wird.

Minor 4 = Adresse des physischen Gerätes
Minor 5 = erste Partition
Minor 6 = zweite Partition
Minor 7 = dritte Partition

Linux greift auf die Geräte nach dem Schema **/dev/dasd<n>** zu, auf die Partition auf dem Gerät über **/dev/dasd<n>1**.

VM Minidisk

Unter VM gibt es die Möglichkeit, eine physische Platte in so genannte Minidisks aufzuteilen. Dadurch erhält man bezüglich der Gerätekonfigurationen und vor allem bezüglich der Größe der Geräte eine enorme Flexibilität. Wie der Übersicht von Seite 275 zu entnehmen, können Minidisks, die von Linux on zSeries verwendet werden, auf zwei Arten formatiert werden:

- Mit dem Linux **dasdfmt**-Kommando, was allerdings nicht empfehlenswert ist: Auf die Minidisk kann dann von CMS aus nicht mehr zugegriffen werden, da durch die Formatierung die CMS Label/Volid-Information zerstört wird. Gerade der Zugriff von CMS auf Minidisks, die von Linux verwendet werden, ist einer der bedeutendsten Vorteile beim Betrieb von Linux unter VM.

- Mit dem CMS FORMAT- und RESERVE-Kommando. Dies ist die empfohlene Variante.

Es werden zwei Minidisk-Treiber unterschieden: ein alter (vor Kernel 2.2.15) und ein neuerer (ab Kernel 2.2.15). Wann immer möglich, sollte der neuere verwendet werden.

XPRAM

Die S/390- und zSeries-Hardware unterstützt Expanded Storage. Dies ist ein Konzept aus der Zeit vor zSeries, als noch mit einer Adressbreite von 31 Bits gearbeitet wurde (vgl. Kapitel 1). Das Konzept wurde in erster Linie für das unter MVS übliche Paging verwendet. Das Paging entspricht dem Swapping unter UNIX/Linux. Somit ist der Expanded Storage auch gut für Swap Files unter Linux geeignet. Der xpram-Treiber mappt ein Filesystem oder Swap Space auf Expanded Storage. Ein xpram-Gerät hat eine Major-Nummer von 35 und kann in bis zu 32 Partitions aufgeteilt werden. Der Device Node ist **/dev/xpram<n>**.

Filesysteme

Linux unterstützt diverse Arten von Filesystemen. Dazu gehören:

ext2
: das Second Extended Filesystem, das speziell für Linux entwickelt wurde; es gilt als De-facto-Standard.

reiserfs
: ein „journaling" Filesystem; „journaling" bedeutet, dass alle Änderungen betreffend des Filesystems mitprotokolliert werden.

Journaled File System (JFS)
: ermöglicht einen schnellen Restart nach einem Systemausfall, da ein Filesystem sehr schnell wiederhergestellt werden kann.

nfs
: Das Network File System ist der De-facto-Standard für Remote File Systems, d. h. für Filesysteme, die über System- und Plattformgrenzen hinausgehen.

swap
: Dies ist ein spezielles Filesystem für Paging.

procfs
: Ein virtuelles Filesystem (im Hauptspeicher), in dem der Kernel Systeminformationen abspeichern kann, die für File Operations benötigt werden.

smbfs
: Das Samba Filesystem, das File Sharing mit Windows-Systemen ermöglicht.

Mit Linux on zSeries wird meistens *ext2* oder *reiserfs* eingesetzt. Eine recht gute Einführung in das Filesystem *ext2* findet man unter: http://web.mit.edu/tytso/www/linux/ext2intro.html

Synchronisation eines Filesystems

In Verbindung mit einem Filesystem wird meistens gepuffert gearbeitet. Mit dem sync-Kommando kann man das physische Schreiben der Daten auf die Platte erzwingen. Dies passiert automatisch periodisch oder beispielsweise bei einem unmount oder bei einem shutdown. Das ist auch einer der Gründe, warum grundsätzlich immer ein Shutdown durchgeführt werden sollte.

Blockgrößen und Filesysteme

Beim Formatieren eines Geräts wird eine Blockgröße angegeben. Beispiel:

```
dasdfmt -b 4096 -f /dev/dasdf
```

Die Blockgröße, die dann mit *mke2fs* bei der Definition eines Filesystems angegeben wird, sollte gleich oder größer der Blockgröße sein, die bei der Formatierung angegeben wurde. Wenn beispielsweise mit einer Blockgröße von 4096 (4 K) formatiert wurde und ein Filesystem mit einer Blockgröße von 1 K erzeugt wird, sind 3 K pro Block nicht adressierbar.

Die Filesystem-Tabelle (/etc/fstab)

Das File /etc/fstab enthält Informationen über den Filesystem-Typ, die Gerätezuordnungen und wo diese gemountet sein sollen. Alle Einträge, die nicht mit einer noauto-Option gekennzeichnet sind, werden beim Boot-Vorgang gemountet.

```
#<device> <mountpoint><fs type><mount opts><to-be-dumped><fsck order>
/dev/dasdf1 / ext2 defaults,errors=remount-ro 0 1
none /proc proc defaults 0 0
/dev/dasdg1 /mnt ext2 defaults 0 2
/dev/dasdi1 swap swap defaults 0 0
```

Die Spalte <device> spezifiziert die Partition des Blockgeräts oder eines Files, das ein Filesystem beinhaltet. Der Mountpoint ist normalerweise ein leeres Verzeichnis, an das ein Filesystem angehängt (gemountet) wird. Mit <fs type> wird der Filesystem-Typ gekennzeichnet. Die <mount opts>-Spalte listet die Optionen auf, die für ein Filesystem gesetzt sind. Das Feld <to-be-dumped> hat mit Dump-Verarbeitung zu tun. Es teilt dem Kernel mit, ob das Filesystem gedumpt (1) oder

nicht gedumpt (0) werden soll. Die Spalte <fsck order> spezifiziert die Reihenfolge, in der die Filesysteme beim Boot-Vorgang überprüft werden. Das Root-Filesystem hat normalerweise eine 1 in diesem Feld, alle anderen eine 2. Filesysteme, die nicht überprüft werden müssen (wie z. B. ein Swap-Filesystem), haben den Wert 0.

Der Kernel mountet die Filesysteme beim Boot-Vorgang in der Reihenfolge, in der sie im fstab-File definiert sind. Das Root-Filesystem sollte deshalb immer am Anfang stehen.

Nach dem Boot-Vorgang können mit mount -a Filesysteme gemountet werden, die nicht die noauto-Option gesetzt haben. Das ist beispielsweise sinnvoll, wenn Änderungen im /etc/fstab- File gemacht worden sind. Man sieht dann auch gleich, ob die Definitionen im fstab-File richtig sind.

Prüfen und Reparieren eines ext2-Filesystems

Ein ext2-Filesystem kann nach einem fehlerhaften Shutdown, oder wenn das System nicht ordnungsgemäß heruntergefahren wurde, in einen inkonsistenten Zustand geraten. Beim Boot-Vorgang wird ein Check für Filesysteme durchgeführt, die in der fstab als inkonsistent gekennzeichnet sind. Der Check kann mit dem Utility *e2fsck* auch manuell aufgerufen werden. Mit dem Utility werden Inkonsistenzen festgestellt und beseitigt.

15.8 Connectivity

Es gibt mehrere Möglichkeiten, eine virtuelle Linux-Maschine mit einem physischen Netz oder mit einer anderen virtuellen Maschine zusammenzubringen. In Verbindung mit dem Mainframe werden drei Netzwerk-Treiber unterstützt:

- LAN Channel Station (LCS)
- Channel-to-Channel (CTC)
- Inter-User Communications Vehicle (IUCV)

Die Grundsatzentscheidung muss gefällt werden, wer die „Ownership" der physischen Netzwerkschnittstelle hat. Owner (Eigentümer, Verwalter) können sein:

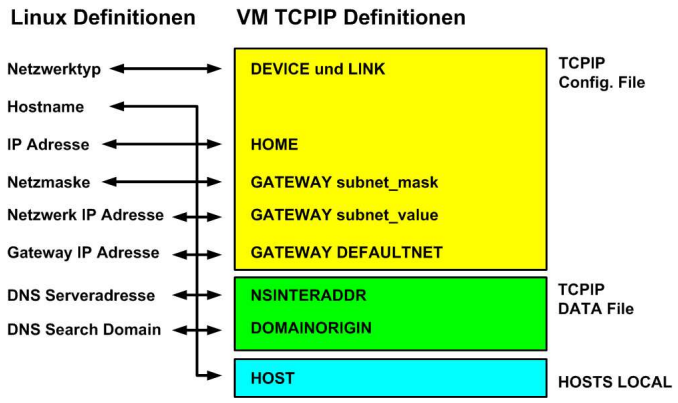
- Linux on zSeries in einer LPAR oder unter VM
- VM TCP/IP Stack
- ein anderes Betriebssystem (z. B. z/OS)

Wenn Linux on zSeries in einer virtuellen Maschine läuft und der VM TCP/IP Stack das Netzwerk-Interface verwaltet, muss eine Punkt-zu-Punkt-Verbindung zwischen jedem Linux on zSeries-System und dem VM TCP/IP Stack eingerichtet werden.

Wenn Linux on zSeries das Netzwerk-Interface verwaltet, muss eine Punkt-zu-Punkt-Verbindung von jedem anderen System, das auf das physische Netzwerk zugreifen will, zu diesem Linux-System eingerichtet werden. Das Gleiche gilt, wenn ein anderes System Owner ist. Das heißt, Verbindungen müssen immer zum Owner des Netzwerk-Interface eingerichtet werden, wenn auf das Netzwerk zugegriffen werden soll.

Wenn der VM TCP/IP Stack als Netzwerk-Gateway genutzt wird, müssen die Linux-Definitionen und die VM-Definitionen in Einklang gebracht werden.

Abbildung 15.4: Linux- und TCP/IP-Definitionen



Das HOSTS LOCAL File ist optional, wenn der VM TCP/IP Stack einen Domain Name Server (DNS) nutzt. Sowohl für Linux als auch für das VM TCP/IP wird eine Subnetz-Maske und ein Subnetz-Wert definiert, wenn ein Router benutzt wird. Für eine Punkt-zu-Punkt-Verbindung wird der HOST-Parameter anstelle der Subnetz-Definitionen im GATEWAY-Statement des TCP/IP-Konfigurationsfiles eingetragen.

Wenn ein CTC Link definiert wird, fragt der Kernel eine Peer-IP-Adresse anstelle einer Gateway-IP-Adresse ab.

LAN Channel Station (LCS)

Ein LCS-Treiber arbeitet in Verbindung mit einem Open System Adapter (OSA) und einigen anderen Geräten wie 3172, 2216 und LAN Adapter, die an eine P/390 oder R/390 angeschlossen sind. Er verwaltet Geräte, die sich dann verhalten wie eine LAN Channel Station.

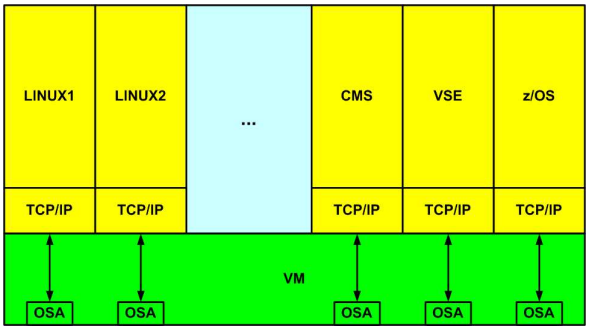


Abbildung 15.5: Lan Channel Station

Channel to Channel (CTC)

Ein CTC-Gerätetreiber wird benötigt, um eine Verbindung zwischen einem virtuellen Linux-System und dem TCP/IP Stack von VM, einem z/OS-System oder einer anderen virtuellen Linux-Maschine herzustellen.

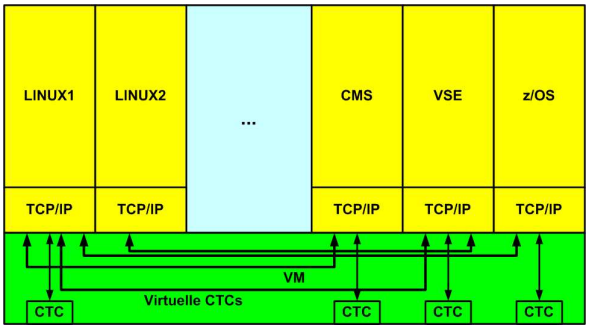


Abbildung 15.6: Channel to Channel

Inter-User Communication Vehicle (IUCV)

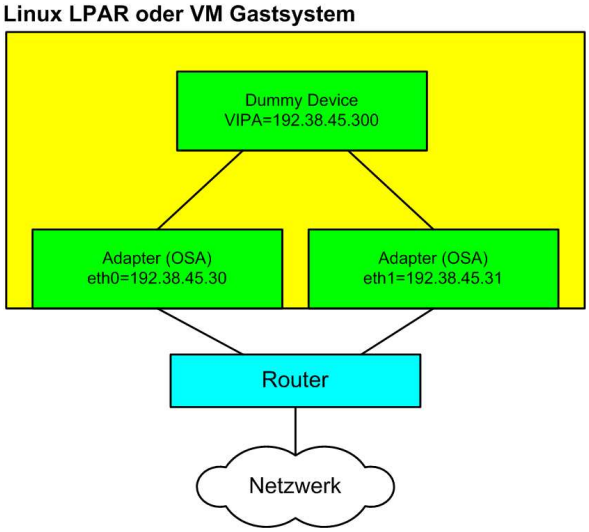
Der IUCV-Treiber verwaltet keine physische Netzwerkschnittstelle, sondern richtet eine High-Speed-Kommunikation zwischen virtuellen Linux on zSeries-Maschinen und dem VM TCP/IP Stack ein. Dies ist vergleichbar der Inter-Prozess-Kommunikation innerhalb eines UNIX/Linux-Systems, nur dass es eben nicht um die Kommunikation zwischen Prozessen innerhalb eines Systems geht, sondern um die Kommunikation zwischen virtuellen Maschinen.

Damit ein IUCV-Treiber genutzt werden kann, muss die virtuelle Maschine entsprechend autorisiert sein. Dies geschieht über Definitionen im CP Directory. Der ALLOW-Parameter legt fest, dass eine andere virtuelle Maschine einen Kommunikationspfad zu dieser Maschine einrichten darf. Der ANY-Parameter legt fest, dass diese Maschine einen Kommunikationspfad zu anderen Maschinen einrichten kann.

Virtual IP Address (VIPA)

Mit VIPA wird die Verfügbarkeit erhöht, da eine Adresse dem System und nicht mehr einem spezifischen Adapter zugeordnet wird.

Abbildung 15.7: VIPA-Funktion



Voraussetzung ist, dass der Kernel mit eingeschaltetem CONFIG_DUMMY aufgebaut wurde.

Das Aufsetzen der Umgebung:

1. Erzeugen eines Dummy-Geräts

   insmod dummy

2. Zuordnen einer virtuellen IP-Adresse

   ifconfig dummy0 9.164.188.100

3. Einschalten von VIPA auf den Netzgeräten

   echo add_vipa4 09A4BC64:eth0 > /proc/geth_ipa_takeover

   echo add_vipa4 09A4BC64:eth1 > /proc/geth_ipa_takeover

4. Setzen der Routen zu den virtuellen IP-Adressen

   route add -host 9.164.188.100 gw 9.164.188.10

   route add -host 9.164.188.100 gw 9.164.188.11

   oder dynamisch mit einem Routing Daemon wie **zebra** oder **gated**.

15.9 Hochverfügbarkeit und Clustering

Die Rolle der Verfügbarkeit

Verfügbarkeit ist ein Attribut für ein System, das aus der Sichtweise der Endanwender betrachtet werden muss. Unter einem Hochverfügbarkeitssystem versteht man ein System, das den Endbenutzern praktisch unterbrechungsfrei zur Verfügung steht. In Prozentzahlen ausgedrückt, beginnt die Hochverfügbarkeit bei 99,99 Prozent, das entspricht einer jährlichen Ausfallzeit von unter einer Stunde (vgl. dazu auch die Ausführungen zu Sysplex in Kapitel 2).

Verfügbarkeit von Linux auf dem Mainframe

Hier geht es nun hauptsächlich darum, was Linux und der Mainframe zu einer Hochverfügbarkeitslösung beitragen kann.

Sowohl zSeries als auch z/VM haben sich über Jahre hinweg entwickelt und verfügen über eine große Anzahl an geeigneten Werkzeugen, um die Verfügbarkeit sicherzustellen. Linux gibt es zwar erst seit einem Jahrzehnt, es hat sich jedoch in dieser Zeit bereits den Ruf erworben, robust zu sein, und es bringt einen stetig verbesserten Werkzeugkasten für die Unterstützung der Hochverfügbarkeit mit. Die Kombination beider Welten ermöglicht es so auch Linux, in dieser Liga mitzuspielen. So können beispielsweise Anwendungsserver für E-Business-Anwendungen konfiguriert werden, die den Ausfall eines Prozessors, eines Plattengeräts und auch eines Linux-Betriebssystems verkraften, ohne dass die Anwender dies bemerken.

Die Kosten einer Konfiguration für Hochverfügbarkeit sind eng verknüpft mit den Fehlersitutationen, gegen die ein System gewappnet werden soll. Das Designteam einer Hochverfügbarkeitslösung muss vor allem Abwägungen hinsichtlich Risiko versus Kosten vornehmen.

Single Points of Failure und Redundanz

Single Points of Failure werden hauptsächlich durch Doppelführung von Komponenten eliminiert. Redundanz gibt es auf diversen Ebenen: Ausgehend von den

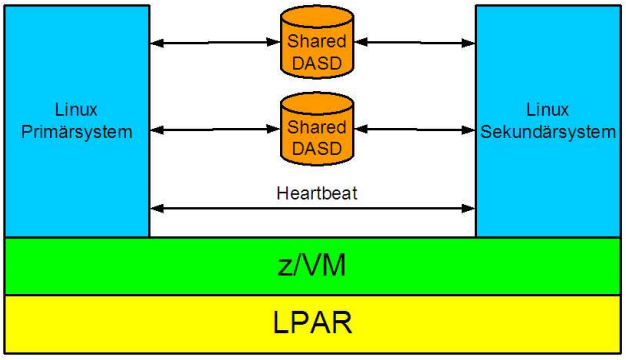
Prozessoren kann die Redundanz weitergeführt werden bis zu redundanten Application Servern, die auf unterschiedlichen physischen Maschinen eingerichtet werden.

Es ist möglich, redundante Linux-Gastsysteme zu konfigurieren, wobei die Verfügbarkeit gesteigert wird, indem ein sekundäres Linux-System darauf wartet, dass ein primäres Linux-System ausfällt, um dessen Workload(s) zu übernehmen. Es gibt zwei Ansätze, dies zu erreichen: ***Hot Standby*** oder ***Clustering***.

Hot Standby mit Linux

Im Falle von Hot Standby gibt es ein alternatives, schon gestartetes Linux-System, das nur darauf wartet, ein ausgefallenes Linux-System zu ersetzen. Unter z/VM ist dies einfach ein gestartetes, aber ruhiggestelltes System, das ausgeswappt wird und dadurch keinerlei Ressourcen verbraucht. Wenn das Hauptsystem ausfällt, kann das Hot-Standby-System sofort die Workloads des ausgefallenen Systems übernehmen. Die Kosten für dieses sehr effiziente Szenario sind extrem niedrig. Vergleicht man dieses Szenario von den Kosten her mit einer heute typischen Serverfarm, wo ein komplettes physisches System bereitgestellt werden muss, um den gleichen Effekt zu erreichen, so schneidet die Lösung mit z/VM logischerweise um ein Vielfaches besser ab.

Abbildung 15.8: Hot Standby-Lösung im Schema



Die Konfiguration eines derartigen Hot-Standby-Systems ist sehr einfach. Es wird ein Cloning-Prozess eingerichtet, um das entsprechende Image zu konfigurieren. Der ***Heartbeat***-Prozess ist eine einfache Kommunikation zwischen den zwei Images, die „Ich lebe"-Nachrichten austauschen. Die Geschwindigkeit des Recovery hängt davon ab, wie schnell die Anwendungsdaten in einen konsistenten Zustand gebracht werden können.

Cluster-Lösung mit Linux

Eine Cluster-Konfiguration basiert darauf, dass mehrere Server parallel aktiv sind. Einige (typischerweise zwei) Server in dieser Konfiguration agieren als Workload Manager. Dies bedeutet, dass sie die hereinkommenden Requests entgegennehmen und an einen von mehreren Anwendungsservern weitergeben.

Vorteil der Cluster-Konfiguration: Der Ausfall eines Servers beeinträchtigt nicht die Verfügbarkeit der Anwendung. Es gibt somit überhaupt keine Recovery-Zeit. Ein weiterer Vorteil besteht darin, dass die Kapazität einer Cluster-Konfiguration durch Hinzufügen bzw. Wegnehmen von Servern dynamisch verändert werden kann. Der Workload Manager wird jeweils darüber informiert und reagiert entsprechend mit der effizienten Verteilung der Requests.

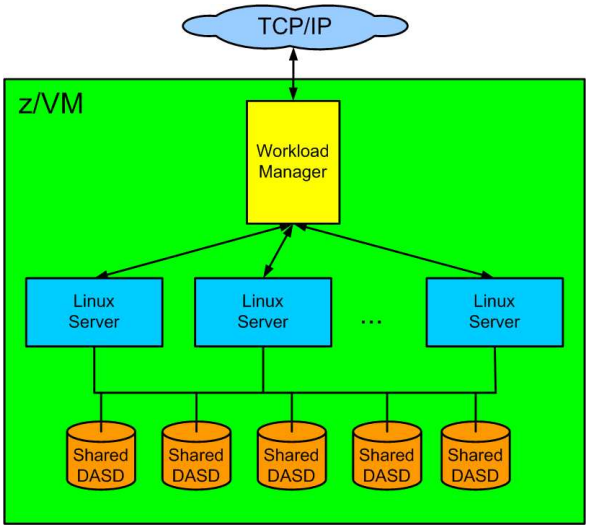


Abbildung 15.9: Aufbau eines Linux-Clusters

Ein Linux Cluster kann mit z/VM sehr einfach aufgebaut werden. Es gibt dafür ein Open-Source-Projekt unter dem Namen *Linux Virtual Server* (LVS). Der Workload Manager wird hierbei über zwei Linux Images implementiert, wobei einer als Hot Standby konfiguriert wird. Das Open-Source-Modul FAKE wird verwendet, um die IP-Adresse des ausgefallenen Servers zu übernehmen. Die Anwender merken nichts davon, dass sie von einem anderen Server bedient werden.

Der Cluster besteht aus mehreren geclonten Linux-Systemen mit einem Filesystem, das es ermöglicht, auf gemeinsame Platten zuzugreifen. Die Verbindung zwischen den Cluster-Elementen wird über ein z/VM Guest-LAN realisiert, das auf Hiper-Sockets basiert. Nur die beiden Workload Manager haben redundante Verbindungen nach außen über reale Netzwerk-Adapter. Auch die Möglichkeiten des *Capacity Upgrade On Demand* (CUoD) lassen sich bei dieser Konfiguration voll ausnutzen.

Redundanz und z/VM

Das z/VM nutzt in vollem Umfang die Vorteile der zSeries-Hardware und kann so konfiguriert werden, dass davon auch die Gastsysteme sehr stark profitieren.

Das Hardware-Konfigurationsfile, das die Umgebung steuert, in der das z/VM ausgeführt wird, sollte immer so ausgelegt sein, dass mindestens jeweils zwei Hardware-Ressourcen definiert sind. Das bedeutet: mindestens zwei CPUs, mindestens zwei Pfade zu den Plattengeräten und mindestens zwei LAN-Verbindungen.

Nun kann natürlich auch das z/VM ausfallen. Es handelt sich dabei zwar um ein äußerst stabiles Betriebssystem, allerdings spielt hier der Faktor Mensch eine entscheidende Rolle, so dass auch ein Ausfall des z/VM berücksichtigt werden muss.

Der Restart des z/VM kann weitgehend automatisiert werden. Dennoch dauert es natürlich eine gewisse Zeit, bis das z/VM und die darunter betriebenen Gastsysteme neu hochgefahren sind, so dass damit kein Hochverfügbarkeitsszenario realisiert werden kann. Wenn die Ausfallzeit verkürzt werden muss, ist es unumgänglich, ein zweites z/VM einzurichten.

Dieses zweite z/VM kann in einer anderen LPAR auf derselben Maschine eingerichtet werden, wobei vorausgesetzt wird, dass die zSeries-Hardware nicht für den Ausfall verantwortlich ist. Ein solcher Hardware-Ausfall ist zwar sehr unwahrscheinlich, wenn jedoch auch dieser Fall berücksichtigt werden muss, muss das z/VM auf einer anderen physischen Maschine untergebracht werden – möglicherweise gar an einem anderen Ort, wenn zusätzlich auch ein Katastrophenfall berücksichtigt werden soll.

15.10 Anwendungen unter Linux auf dem Mainframe

Welche Anwendungen eignen sich nun für die Verarbeitung unter Linux auf dem Mainframe?

Wenn es darum geht zu selektieren, für welche Anwendung eine Implementation bzw. eine Portierung zu Linux auf dem Mainframe lohnt, sollte man sich Gedanken machen über das Profil einer Anwendung und das dazu passende Profil einer Hardware-Plattform.

In Abbildung 15.10 wird ein zSeries-Profil mit einem pSeries-Profil verglichen, das auf anderen RISC-Systemen meist sehr ähnlich aussieht. Der Mainframe weist ein recht ausgeglichenes Profil aus und bietet vor allem Vorteile, wenn es um Anwendungen mit hohen I/O-Raten geht. Wenn diese Anwendungen dann noch mit anderen Servern und Betriebssystemen auf dem gleichen Rechner kommunizieren, werden die Vorteile noch größer.

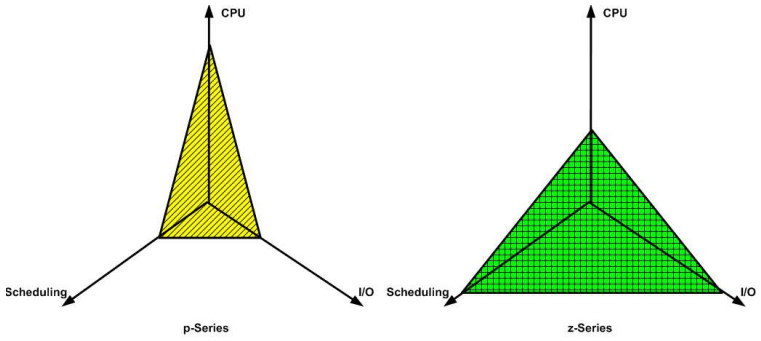


Abbildung 15.10: Performance verschiedener Hardware-Plattformen

Weniger geeignet für den Mainframe sind erfahrungsgemäß Anwendungen, die sehr CPU-intensiv sind, da dann die Vorteile des Mainframe weniger ausgenutzt werden können. Man sollte eben immer berücksichtigen, wie das Profil einer Anwendung aussieht und wie es mit dem entsprechenden Profil einer Hardware-Plattform übereinstimmt.

Es gibt unterschiedliche Ansätze in diesem Zusammenhang. Geeignet sind in den meisten Fällen Anwendungen, die auf die Backend-Systeme zugreifen, die sich auf dem Mainframe befinden.

Ein äußerst interessantes Szenario ist beispielsweise der Einsatz von SAP. Der SAP-Datenbankserver wird unter z/OS in einer DB2-Umgebung implementiert. In diesem Zusammenhang kann die höchste Verfügbarkeit über Parallel Sysplex erreicht werden.

Die Application Server werden dann unter Linux auf dem gleichen physischen Rechner eingerichtet, entweder in einer LPAR (Logical Partition) oder unter z/VM. Der Vorteil einer derartigen Konfiguration liegt darin, dass die jeweiligen Application Server losgelöst in einer jeweils separaten Umgebung eingerichtet werden und über die Bordmittel des Mainframe wie Internal Resource Director und HiperSockets die Flexibilität und die hohen Übertragungsraten ausnützen können, ohne sich gegenseitig zu stören. Ein gewaltiger Pluspunkt gerade in Verbindung mit dem Einsatz von z/VM sind die Automatisierungsmöglichkeiten. Ein großer IBM-Kunde aus Österreich hat erfolgreich demonstriert, dass die Einrichtung einer komplett neuen SAP-Umgebung innerhalb von 10 Minuten vonstatten gehen kann.

Mitte 2004 waren bereits rund 500 Anwendungen von ca. 300 Software-Unternehmen auf Linux for zSeries portiert, Tendenz stark wachsend, da immer mehr Unternehmen die Vorteile von Linux auf dem Mainframe erkennen.

Kapitel 16

Performance- und Workload-Management

16.1 Workload-Management

Die Prioritätensteuerung und das ausgereifte Workload-Management gehören zu den Funktionalitäten des IBM Mainframe, die im Vergleich zu anderen Systemen deutliche Vorteile aufweisen. So ist es problemlos möglich, den Mainframe in den *Peak-Hours* (das sind die Zeiten der Hauptbelastung eines Systems) bis an die Leistungsgrenze auszureizen und dabei dennoch die Ziele für die Endbenutzer zu berücksichtigen und einzuhalten.

Der *Workload Manager* (WLM) setzt einen völlig neuen Fokus hinsichtlich der Systemvorgaben für die Prioritätensteuerung. Wie allein schon durch die Bezeichnungen zum Ausdruck kommt, lag bei der ursprünglichen Komponente *System Resource Manager* (SRM) der Schwerpunkt auf den Systemressourcen, beim Workload Manager dagegen auf den Workloads.

Eine Workload ist die Zusammenfassung gleichartiger Arbeiten wie beispielsweise Batch, TSO oder Datenbanktransaktionen, die installationsspezifisch nach diversen Kriterien (bei Batchjobs zum Beispiel aufgrund von Namenskonventionen oder von Jobklassen) weiter unterteilt werden können.

Mit dem *Goal Mode* wird die SRM-Steuerung durch eine Workload-Steuerung auf der Basis von Zieldefinitionen, die von *Service Level Agreements* (SLAs) abgeleitet werden können, ergänzt. Der *Compatibility Mode* ermöglicht die herkömmliche Steuerung über die SRM-Parameter für eine Übergangszeit, die mit der z/OS-Version 1.3 beendet wurde.

Service Level Agreements sind die Basis für die Zieldefinitionen des Workload-Managements. Das macht die Schnittstelle zum WLM deutlich verständlicher, da die Service Level Agreements nicht mehr in Parameter für den System Resource Manager übersetzt werden müssen, sondern direkt in die WLM-Definitionen eingebracht werden.

In den SLAs stehen keine Zahlen bezüglich der Auslastung der Ressourcen, sondern Zahlen und Definitionen, die aus Endbenutzer-Sicht relevant sind, wie Verfügbarkeit, Antwortzeiten und Verweilzeiten.

SLAs sind Verträge zwischen einem Rechenzentrum und den Benutzern, wobei es keine Rolle spielt, ob die Benutzer Mitarbeiter der eigenen Firma oder externe Kunden sind. Selbst wenn keine festgeschriebenen SLAs vorhanden sind, gibt es zumeist doch mindestens eine Vorstellung davon, was von dem System (bzw. den Systemen) erwartet wird. SLAs sind neutraler als die Benutzervorstellungen, und es gibt eine ganz wichtige Vorgabe für SLAs: Sie müssen messbar sein!

Abbildung 16.1: Beispiel für Service Level Agreements

Die WLM-Definitionen werden nicht mehr wie beim SRM in Parametersätzen der PARMLLIB abgelegt, sondern über einen ISPF-Dialog verwaltet. Die entsprechend resultierenden Daten werden in einem Sysplex Couple Dataset abgelegt. Selbst wenn ansonsten nicht mit Sysplex gearbeitet wird, muss die entsprechende Um-

gebung mit den Couple Datasets eingerichtet werden. Man spricht in diesem Fall von einem *Monoplex*.

Die Parameter für eine Umgebung, die mit dem WLM verwaltet wird, werden als *Service Definition* bezeichnet.

WLM Service Definition

Pro Sysplex (auch wenn es sich um eine Monoplex-Umgebung handelt) gibt es eine Sysplex-Definition.

Eine Service Definition besteht aus:

Service Policies
: Es kann eine oder mehrere Service Policies geben, welche die Zielvorgaben für einen bestimmten Zeitpunkt festlegen. Es besteht die Möglichkeit, alternative Policies zu definieren und diese Zielvorgaben für Benutzergruppen zeitlich abhängig zu ändern. Dieser Mechanismus wird *Policy Override* genannt. Man versucht dies in der Regel zu vermeiden und einen universellen Parametersatz zu definieren, der allgemeingültig verwendet werden kann.

Resource Groups
: Eine oder mehrere Resource Groups legen die minimale und maximale Prozessorkapazität für Service Classes fest.

Classification Rules
: Mit Classification Rules wird die Zuordnung der ankommenden Arbeit in Service Classes festgelegt. Zur Klassifizierung der Arbeit können LU-Name, Subsystem ID, Transaction Class, Subsystem Instance oder UserID verwendet werden.

Workloads
: Mit Workloads wird gleichartige Arbeit in einer Installation abgebildet und gegeneinander abgegrenzt.

Service Classes
: Mit Service Classes werden die Performance-Ziele und Betriebsmittel-Anforderungen der Workloads festgelegt. Service Classes können in mehrere Perioden unterteilt werden, wodurch sich die Zielvorgaben abhängig von der Laufzeit verändern. Für jede Service Class Period wird definiert, wie wichtig sie ist (*Importance Definition*). Auf der Basis dieser Angaben entscheidet der SRM, welche Service Classes welche Betriebsmittel zugeteilt bekommen und welche gegebenenfalls Betriebsmittel abgeben müssen.

Zieldefinitionen

Das Wichtigste bei einer Service Policy sind die Ziele, die für eine Service Class definiert werden. Diese können in Gruppen eingeteilt werden:

System
: Dies sind durch das Betriebssystem vorgegebene Gruppen, die für Systemadressräume und andere Adressräume höchster Priorität vorgesehen sind. Es gibt dafür die Service Classes *System* und *SYSSTC*. Diese sind nicht beeinflussbar und mit einer festen Priorität verankert. Allerdings ist es möglich, Adressräume der Gruppe *SYSSTS* zuzuordnen und Systemadressräume aus der Gruppe *System* herauszunehmen. Alle Adressräume in der Klasse *System* laufen mit einer Dispatching Priority (DP) von 255 (hexadezimal FF) und alle Adressräume in der Klasse *SYSSTC* mit einer DP von 253 (hexadezimal FD). Alle anderen Adressräume haben eine niedrigere Priorität.

Antwortzeiten
: Antwortzeiten werden verwendet, um Gruppen von Transaktionen zu steuern. Sie können als *Average Response Time Goal* und als *Percentile Response Time Goal* angegeben werden. Damit sollen die SLAs direkt in die Definition einfließen können. Das System sorgt dann dafür, dass die definierten Antwortzeiten erreicht werden.

Execution Velocity
: Execution Velocity definiert den Grad an akzeptierbaren Verzögerungen (Delays). Diese Vorgabe wird dann gemacht, wenn keine Antwortzeiten definiert werden können. Die Definition erfolgt durch einen Zahlenwert zwischen 1 und 99, wobei 99 die obere Grenze definiert, wo so gut wie keine Delays zugelassen werden.

Discretionary
: Discretionary bedeutet, dass keine expliziten Vorgaben gemacht werden. Diese Workloads haben niedrigste Priorität und werden immer dann bedient, wenn keine andere Arbeit ansteht.

Importance
: Neben der Definition von Zielen ist die Festlegung der Wichtigkeit von Workloads untereinander das Mittel, um die Systemsteuerung zu kontrollieren. Für alle Service Class Periods, für die eine Antwortzeit oder Execution Velocity definiert wurde, kann eine Importance definiert werden.

  Darüber wird definiert, wie wichtig es ist, dass die Vorgaben auch tatsächlich eingehalten werden. Die Service Classes *System* und *SYSSTC* erhalten aufgrund ihrer expliziten Dispatching-Priorität immer den Vorrang.

Arbeitsweise des WLM

Basierend auf den Definitionen einer Service Policy beobachtet der WLM das System, indem er Daten über Betriebsmittel, Adressräume und Transaktionen sammelt. In regelmäßigen Abständen (alle 10 Sekunden) vergleicht er die gesammelten Daten mit den Zielvorgaben, um zu ermitteln, ob die Workloads den Vorgaben entsprechend verarbeitet werden.

Stellt er fest, dass eine Anpassung der Betriebsmittelvergabe angebracht ist, muss er zunächst ermitteln, ob dies überhaupt möglich ist. Der WLM prüft hierzu nach, ob es Workloads gibt, die niedrigere Priorität haben als die Workload, die ihre Vorgaben nicht erreicht.

Der nächste Schritt ist zu berechnen, ob eine Anpassung einen positiven Effekt erzielen würde. Eine höhere Dispatching-Priorität bereinigt zum Beispiel keinen Speicher-Engpass! Sollte die Umverteilung einen positiven Effekt erwarten lassen, werden noch die Auswirkungen auf die anderen Workloads untersucht. Wenn auch diese Überprüfung zur Zufriedenheit ausfällt, wird die Anpassung tatsächlich vorgenommen. Danach wird überwacht, wie sich die Anpassung auswirkt, und der Regelmechanismus beginnt von vorne.

16.2 System Management Facility (SMF)

Die Basis für eine Vielzahl von Überwachungs- und Analysetools bildet die System Management Facility (SMF). Das Betriebssystem und einige Subsysteme erstellen Datensätze (*SMF-Records*), die über einen SMF-Adressraum in SMF-Dateien geschrieben werden. Das Sicherstellen der kontinuierlichen Verfügbarkeit der Datensammlung ist eine der wichtigsten Aufgaben beim Einrichten eines produktiven Systems.

Die Daten werden zunächst in ein SMF Dataset (VSAM-organisiert) geschrieben. Es werden mehrere SMF Datasets bereitgestellt, allerdings kann nur eines aktiv benutzt werden. Ist dieses voll, wird automatisch auf das nächste Dataset gewechselt, und das volle Dataset muss gesichert werden, was normalerweise automatisiert eingerichtet wird, um sicherzustellen, dass keine Daten verloren gehen.

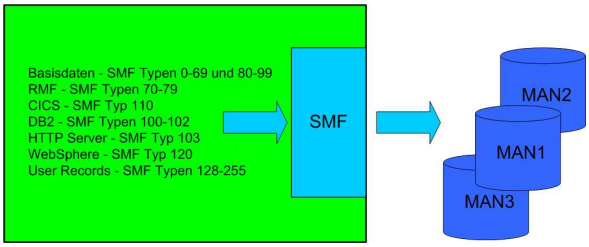


Abbildung 16.2: Die System Management Facility (SMF)

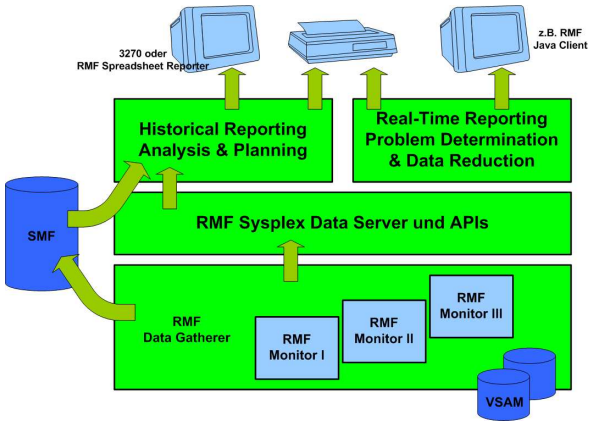
Jedem SMF Record ist eine eindeutige Nummer zugeordnet, die als *SMF Type* bezeichnet wird. Ein SMF Type kann Subtypes haben.

Die Daten sind mit einem 18 oder 24 Byte langen Header versehen. Neben dem Satztyp bzw. Subtyp werden hier unter anderem auch Datum, Uhrzeit und System-ID mitgeführt.

16.3 Resource Measurement Facility (RMF)

RMF ist ein Produkt für das Mainframe Performance-Management. Es handelt sich dabei um ein Basisprodukt für das Sammeln von Performance-Daten; RMF bietet darüber hinaus ein Berichtswesen für die Sysplex-weite Überwachung, Performance-Analyse und Kapazitätsplanung.

Abbildung 16.3: RMF Funktionsüberblick



Der RMF Data Gatherer bildet die Grundlage für die Performance-Analyse und Kapazitätsplanung. Er liefert die Daten für die diversen Monitore, die mit RMF mitgeliefert werden oder auch von anderen Herstellern wie BMC, CA oder Candle angeboten werden.

Die RMF-Monitore

Monitor I

ist ein Langzeit-Datensammler für alle Ressourcetypen und Workloads. Der Monitor I schreibt SMF-Sätze, die dann für Performance-Analysen und Kapazitätsplanung verwendet werden. Es gibt zahlreiche Produkte (auch von anderen Herstellern), die auf den RMF-Sätzen aufbauen.

Monitor II
: ist ein Schnappschuss-Datensammler für die Anzeige von Zuständen von Adressräumen und Ressourcen-Auslastungen.

Monitor III
: ist ein Kurzzeit-Datensammler für Problemeingrenzungen, Anzeige von Verzögerungen und die Überwachung der Performance-Ziele.

Index