Einführung in die Technische Informatik-Zusammenfassung

Computersysteme

Nun wollen wir die Zusammenarbeit der einzelnen Komponenten einer Zentraleinheit bestehend aus:

- Rechenwerk (ALU)
- Speicherbausteinen
- Ein-/Ausgabeeinheiten

betrachten.

Prozessoren

Maschinen-Code

Wie der Mikro Code handelt es sich auch beim Maschinen Code um Bitkombinationen. Der Maschinen Code eines Prozessors ist die Menge aller definierten Datenwörter, die ein Interpreter decodieren kann.
Da der Interpreter die Maschinenbefehle aus dem Speicher (RAM oder ROM) liest, kann das Maschinenprogramm sehr viel umfangreicher als das Micro-Programm sein, da der Adressraum um ein Vielfaches größer ist.
Damit teilen sich Daten und Instruktionen den Adressraum. RAM und ROM werden wegen der direkten Bindung an den Prozessor und zur Unterscheidung von anderen Speichermedien als Hauptspeicher bezeichnet.

Vom Interpreter (Mikro Programm) sind folgende Schritte durchzuführen:

1. Einen Maschinenbefehl aus dem Speicher in das dafür bestimmte Register einlesen.
2. Befehl decodieren (d.h. analysieren), um zu analysieren, welche Operation auf welche Operanden anzuwenden ist.
3. Die Operation durchführen
4. Diesen Arbeitszyklus beim Punkt 1 fortsetzen

Als nächstes stellt sich nun die Frage, unter welcher Adresse die zu interpretierenden Maschinenbefehle gespeichert sind. Es wird dazu ein Register verwendet, das diese Adresse enthält. Es wird Programmzähler, Program Counter (PC), Instruction Counter (IC) oder Instruction Pointer (IP) genannt.
Sind alle Micro-Befehle, aus denen sich ein Maschinenbefehl zusammensetzt, abgearbeitet, erhöht der Interpreter den Program Counter und liest aus dem Speicher den nächsten Befehl n das Instruction Register ein.

Transfer-Operationen

Übertragen die Maschinen Daten von einer Quelle zu einem Ziel, ohne diese dabeizu verändern. Transfer-Operationen die Daten im Hauptspeicher bearbeiten, nennt man

oftmals Load- (Wort in ein Register laden) bzw. Store-(Registerinhalt speichern) Operationen.

Input/Output-Operationen

Input/Output-Operationen (I/O-Befehle) sind ähnlich den Transfer-Befehlen, allerdings kommt es hier zu einem Datenaustausch mit Peripheriebausteinen (externen Geräten). Zur Durchführung dieser Funktionen haben die Rechner eine Anzahl von Ports (ein solcher Port entspricht einem Register), die Steuerung der Peripheriegeräte erfolgt über Steuerworte. Zur Auswahl des korrekten externen Registers gibt man in der I/O-Operation seine eindeutige Adresse an.
Bei der Vergabe der Adressen können zwei Methoden eingesetzt werden:

1. independent I/O (isolated I/O)-System: der Hauptspeicher und die Ports weisen völlig unabhängige Adressen auf. Für I/O- und Transferoperation werden unterschiedliche Befehle benutzt. Der Vorteil dieser Methode liegt darin, dass der Adressraum in vollem Umfang zur Verfügung steht.
2. Memory-mapped-I/Os: Die Ports werden so behandelt, als wären sie gewöhnliche Speicherstellen. Ob eine Adresse zu einem Speicherwort oder zu einem Port gehört, ist allein durch die Verdrahtung festgelegt. Da es keinen Unterschied bei der Bearbeitung von einem Wort im Hauptspeicher oder einem I/O-Register gibt, werden auch dieselben Transferbefehle verwendet.

Arithmetische Operationen

Prozessoren beherrschen zumindest die Grundrechnungsarten, Addition, Subtraktion, Multiplikation und Division. Spezielle Operationen für Integer sind increment (zahl+1), decrement (zahl-1). Bitweise Inversion ist genauso schon ein Standartkommando einer CPU.
Floating Point Operation ist die Verknüpfung zweier reeller Zahlen.
Bei der Ausführung von Arithmetischen Operation können auch Fehler auftreten, z.B. Überlauf. Die meisten Prozessoren besitzen ein PSW (Program Status Word) um eine Überlauf oder Übertrag anzuzeigen.
Flag- oder Bit.Kommandos sind Operationen die Bits manipulieren.

Logische Operationen

Diese sind bereits in der Register-Transfer-Ebene implementiert. Solche Operationen sind z.B. Set-Bit(bit auf 1), clear-Bit(bit auf 0), complement Bit oder clear Register (eine ganze Folge von Bits auf 0). Zusätzlich gibt es noch die binären boolschen Funktionen AND, OR oder XOR die jedes Bit des Operanden einzeln verarbeiten.

Shift-Operationen

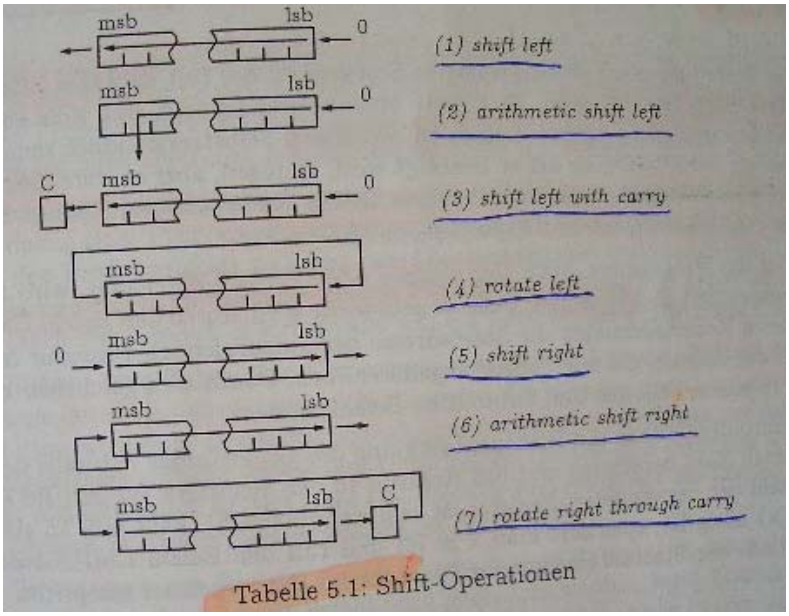


Tabelle 5.1: Shift-Operationen

Flow-Control

Es wird ermöglicht, den sequenziellen Ablauf eines Programms zu unterbrechen. Das können z.B. auch Subroutine-Calls und Interrupts.

Sprünge

Es gibt bedingte und unbedingte Sprünge. Unbedingte Sprünge werden im Maschinen Code Jump-operations und bedingte Sprünge Branch-operations genannt. Es gibt hier eine größere Auswahl an Bedingungen: Bit-Tests (Jump-on-Bit-set, Jump-on-Bit-not-set), Word-Tests (Jump-if-zero, Jump-if-negative, o.ä.), Vergleichsoperation von zwei Wörtern (Jump-if-equal, jump-if-greater, usw.).

Subroutine Calls

Bei umfangreichen Programmen ist es oft der Fall das ein Folge von Befehlen an mehreren Stellen benutzt wird. Anstatt diesen Code an jeder Stelle einzufügen ist es rationeller ihn nur einmal zu Programmieren und von verschiedenen Stellen aus zu benutzen. Das ermögliche die Funktionen Call-Subroutine und Return-from-Subroutine. Dieser Teil des Programms wird Subroutine oder Prozedur genannt. Durch Call-Subroutine wird dieser Programmteil aktiviert und durch Return-from-Subroutine wird wieder zum normalen Programmablauf zurückgekehrt.

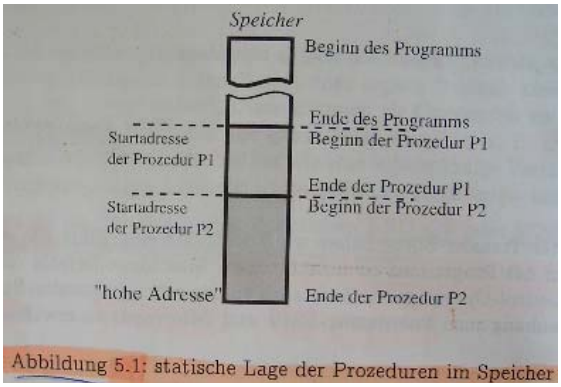


Abbildung 5.1: statische Lage der Prozeduren im Speicher

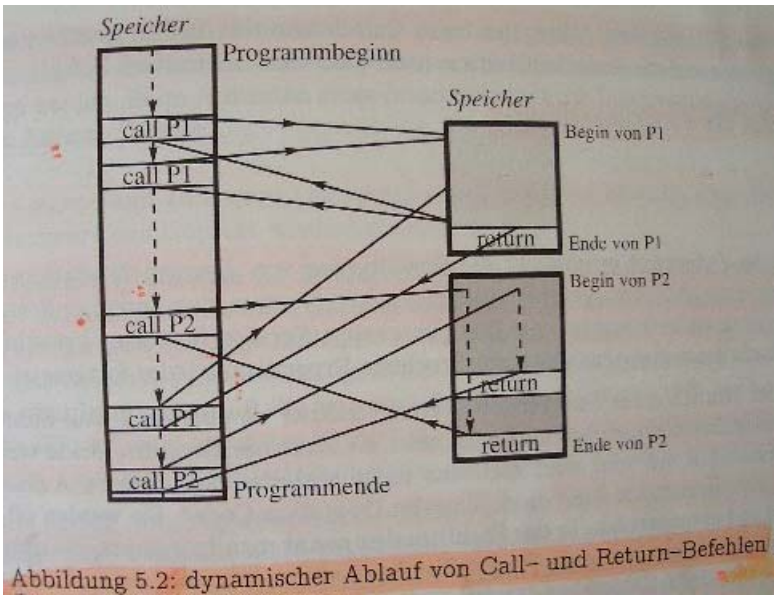


Abbildung 5.2: dynamischer Ablauf von Call- und Return-Befehlen

Damit das Return-from-Subroutine-Kommando die korrekte Adresse (Return Address) dem Program Counter zuweisen kann, muss sie zwischengespeichert werden. Bei geschachtelter Aufruffolge ist die Verwaltung der Rücksprungadressen ein sehr schwieriges Unterfangen. Ein oftmals verwendetes Konzept in diesem Zusammenhang ist der Stack.

Subroutine-Befehle müssen folgende Kommandos ausführen:

Call-Subroutine

1. Retten des PSW, der Registerinhalte und des Program-Counters
2. Laden des PC mit der Prozedur-Startadresse
3. Abarbeitung der Prozedur durch den Interpreter

Return-from-Subroutine:

1. Wiederherstellen des PSW und der Registerinhalte
2. Laden des PC mit dem Wert, der beim Call-Subroutine-Befehl gesichert worden ist
3. Fortsetzen des Programmablaufs

Interrupts

Sie bewältigen unvorhergesehen Situationen im Programmablauf, dafür wird der Programmablauf unterbrochen und es wird in eine bestimmte ISR (Interrupt Service Routine) Service Routine gesprungen. Der Auslöser dafür kann ein prozessinterner Ausnahmefall oder ein externes Ereignis sein. Interne Ausnahmefälle sind z.B. division durch Null oder illegaler Operation-Code, sie werden als Traps bezeichnet. Beim Shakehand-Verfahren können mit Hilfe des Power-Fail-Interrupts manche Prozessoren noch bestimmte abschließende Befehle durchführen bevor die Energieversorgung aussetzt.

Nach dem Autreten eines Interrupts übernimmt spezielle hardware – die Interrupt Logik – die Steuerung der CPU. Sie stellt fest, ob ein externer oder interner Ausnahmefall vorliegt. Sofern möglich wird der laufende Befehl ausgeführt und der Programm-status (alle Register, PC, und das Statusregister) gerettet.

Zur Feststellung der externen IR-Quelle gibt es mehrere Konzepte:

- Es wird stehts die gleiche ISR-Startadresse im Speicher angesprungen. Die Routine erkennt dann selbstständig, welche Quelle den Interrupt hervorgerufen hat.
- Der Prozessor besitzt (hardwaremässig) genügend Interrupteingänge. Jede Leitung ist eindeutig einem bestimmten Ereignis zugeordnet (uncodierte Interruts).
- Jede Quelle besitzt eine eindeutige Identifikations-Nummer, die sie gleichzeitig mit dem Interruptsignal anlegt.

Die Zuordnung einer Interruptquelle zur Adresse der ISR kann auf zweierlei Arten geschehen:

- **Fixe Zuordnung**: jeder Quelle ist eine bestimmte ISR und somit eine fixe Adresse zugeordnet.
- **Interruptvektor**: Die Interrupts-Tabelle, die an einer definierten Stelle im Speicher liegt, gibt die Zuordnung der Quellen zu den Startadressen der ISRs an.

Der Befehl Return-from-Interrupt bewirkt, dass die ISR beendet wird und die Hardware den Context wiederherstellt. Während kritischer Abschnitte ist es manchmal nötig, alle bzw. bestimmte Interrupts zu sperren. Eine Interrupt-Mask bestimmt, welche der Quellen zugelassen bzw. gesperrt (maskiert) sind. Non-Maksable-Interrupts sind Interrupts die immer bearbeitet werden müssen.

Sollte der Fall auftreten das während eines Interrupts ein weiterer Interrupt auftritt gibt es eine Interrupt Priorität die das regelt.

<u>Stack-Operationen</u>

Sie werden auch Kellerspeicher oder Stapelspeicher genannt, es dient u.a. zum Speichern des PSWs. Das Prinzip besteht darin das die letzte eingetragene Information als Erste wieder gelesen oder entfernt wird (Last-In-First-Out).
Die Push-Operation legt ein oder mehrere Wörter auf den Stapel, die Pop-Operation entnimmt ein oder mehrere Wörter, die zuoberst auf dem Stapel liegen.

Der Vorteil dieses Konzepts besteht darin, dass die Adressverwaltung durch das System übernommen wird, Der Benutzer muss die Adresse des Elements nicht kennen das er auf den Stack legt. Das System muss nur jene Adresse um Hauptspeicher verwalten, an der sich das oberste Element des Stacks befindet. Dazu verwendet es ein spezielles Register, den SP (Stackpointer).
Wenn der Stapel in Bereiche gerät, in denen sich Programme bzw. Daten befinden (stack size overflow), treten Probleme auf.

Realisierung der beiden Befehle push_16 und pop_16:

```
push_16(reg)

1.          memory[SP] ← reg     # Das Register reg unter der im SP ange-
                                 # gebenen Adresse im Speicher ablegen.
2.          SP←SP-1              # SP um 1 erniedrigen
```

```
pop_16(reg)

1.          if (¬SP=0) goto 3            # Falls im SP nicht (FFFF)16
                                         # gespeichert ist, springe zu 3
2.          Interrupt Stack-underflow    # Interrupt Stack-underflow auslösen
3.          SP ← SP+1                    # SP um 1 erhöhen
4.          reg ← memory[SP]             # Das Element, das unter der im SP
                                         # angegebenen Adresse im Speicher
                                         # abgelegt ist, im Register
                                         # reg abspeichern
```

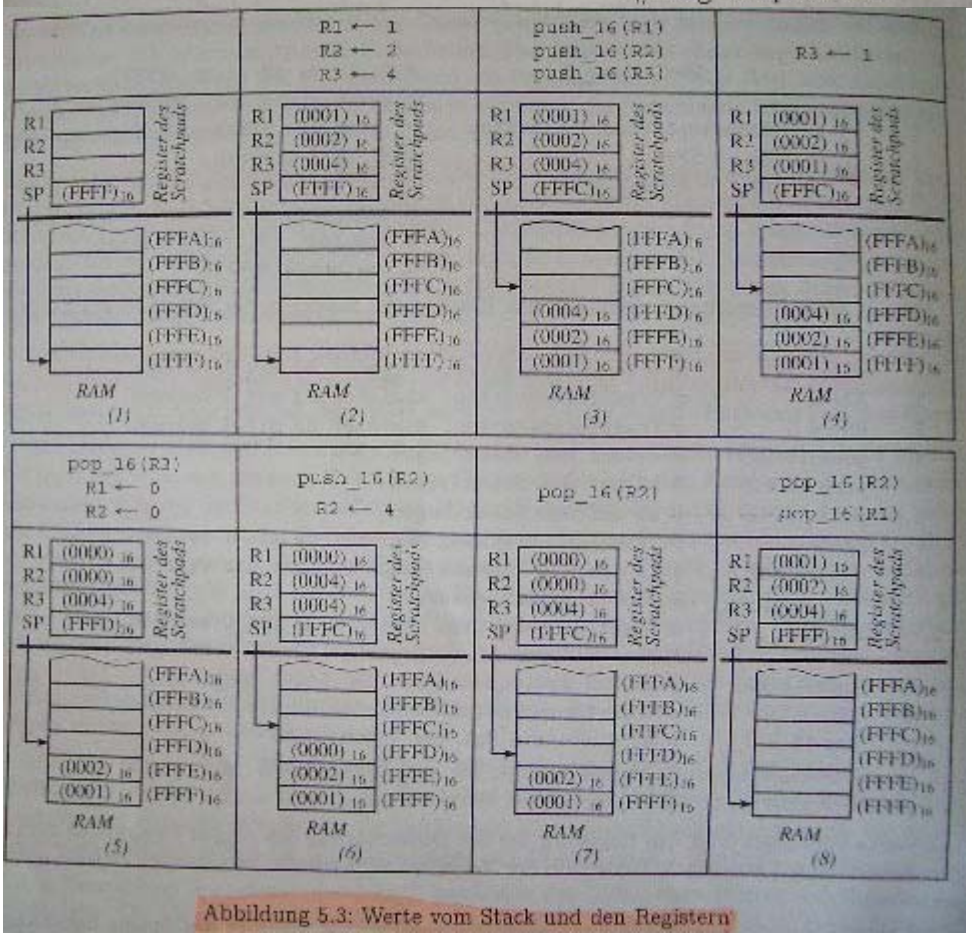


Abbildung 5.3: Werte vom Stack und den Registern

Beim wiederherstellen des Programm Status nach Subroutinen besteht eine große Fehlerquelle durch das ausführen von Sprungbefehlen.

<u>Adressierungsarten</u>

Eine Maschinen-Instruktion enthält meist Befehl, der auf einen oder mehrere Operanden angewandt werden muss. Dazu ist es erforderlich, ihre Adressen zu

kennen. Am einfachsten ist es an dieser Stelle direkt die Adresse der Speicherzelle anzugeben.
Dies hat aber folgende Nachteile:

- Die Adressen müssen bereits zur Zeit der Programmerstellung bekannt sein, d.h. , es wird festgelegt, wo die Software zur Ausführungszeit im Speicher liegt.
- Es ist besonders günstig, Adressierungsmechanismen zu besitzen, die bei einer geringeren Anzahl von Bits einen gleich großen Adressraum ansprechen können.
- Die absolute Adressangabe ist eine eher unkomfortable Methode.

Daher ist man dazu übergegangen, die Adressen der Operanden aus den Inhalten von Registern, Speicherzellen und Konstanten, die im Maschinen-Code angegeben werden, zu berechnen (dynamische Adressberechnung).

Adressierungsarten:

Implied Mode (implizite Adressierung)

Die Gruppe von Befehlen, die dieses Verfahren verwendet, ist nur für einen bestimmten Operanden definiert, darum muss dieser nicht eigens in einem Adressfeld angegeben werden. Ein Beispiel für ein solches Kommando ist die Instruktion „Enable Interrupts“.

Register Mode

Die Register werden über die ihnen zugeordneten Adressen angesprochen.

Immediate Mode

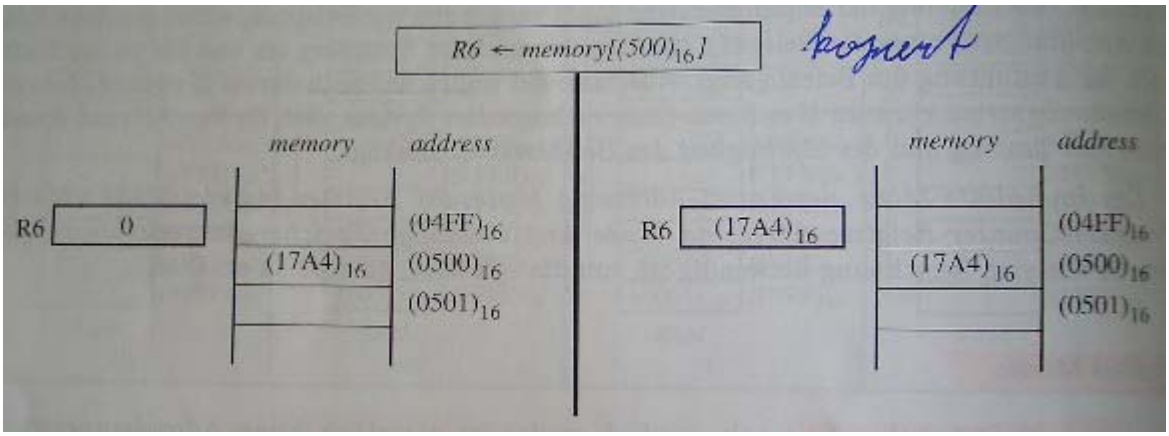
Der Wert des Operanden ist direkt im Maschinenbefehl enthalten. Es befindet sich somit statt einer Adresse direkt die Konstante, die verarbeitet werden soll, im Operandenfeld.

Direct-Addressing Mode

Der Wert im Operandenfeld entspricht der Speicheradresse des gewünschten Operanden.

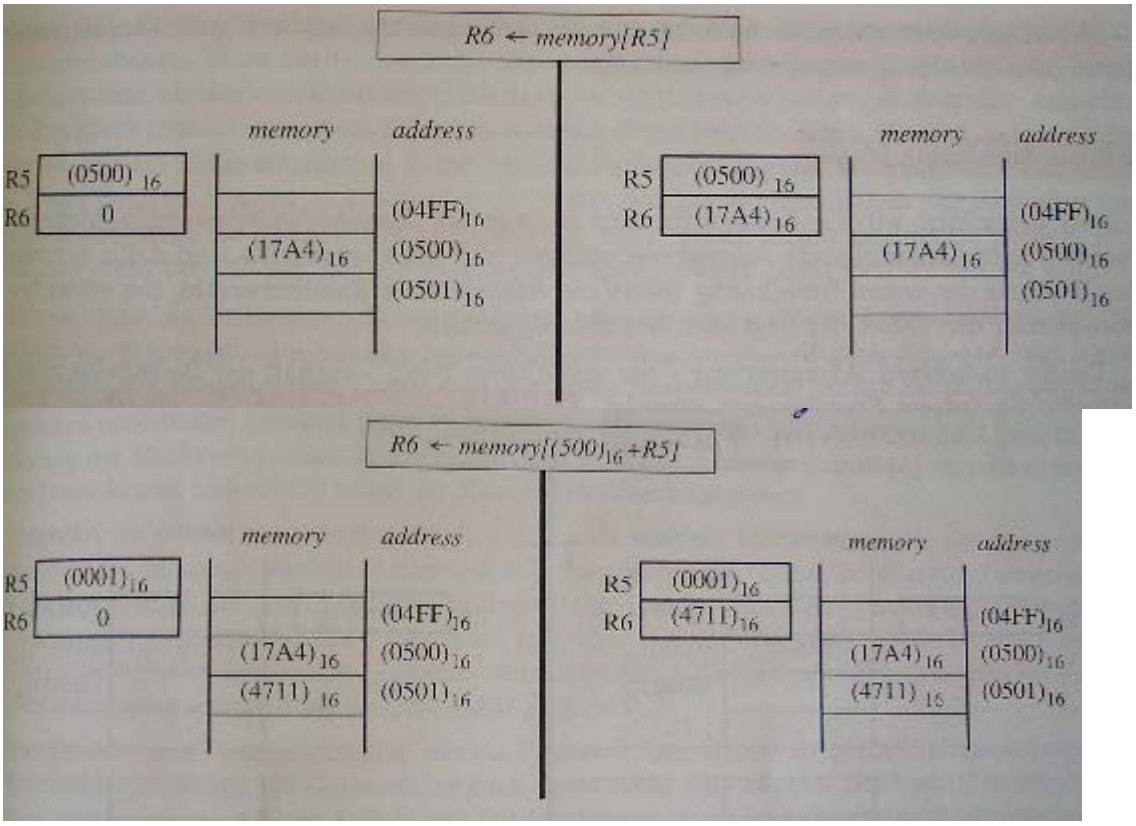
Beispiele:

- Goto $(220)_{16}$ Lädt den PC mit der Adresse $(220)_{16}$
- R6 ← memory[$(500)_{16}$] Das Wort, das sich an der Adresse $(500)_{16}$ befindet, wird in das Register R6 geladen. Das Operandenfeld des Befehls enthält daher die Adresse $(500)_{16}$



Register-Indirect Mode

Das Register, welches im Maschinen-Befehl als Operand angegeben ist, enhält die effektive Adresse des gewünschten Datenwortes. Das Register wird daher auch pointer genannt.
Eine weitere Variante ist die Base-Register-Addressierung, dabei enthält der Maschinen Befehl noch ein Displacement- oder Offset Feld. Um die effektive Adresse zu errechnen, wird zum Wert des im Operandenfeld adressierten Registers das Displacement addiert.
Den gleichen Effekt erzielt man durch indizierte Adressierung. Das Register beinhaltet bei dieser Adressierung das Displacement, während die Basisadresse im Befehl selbst gespeichert ist.



Zwei weiter Varianten sind die Register-Indirect-With-Postincrement und Register-Indirect-With-Predecrement. Bei diesen beiden Befehlen wird nach jedem

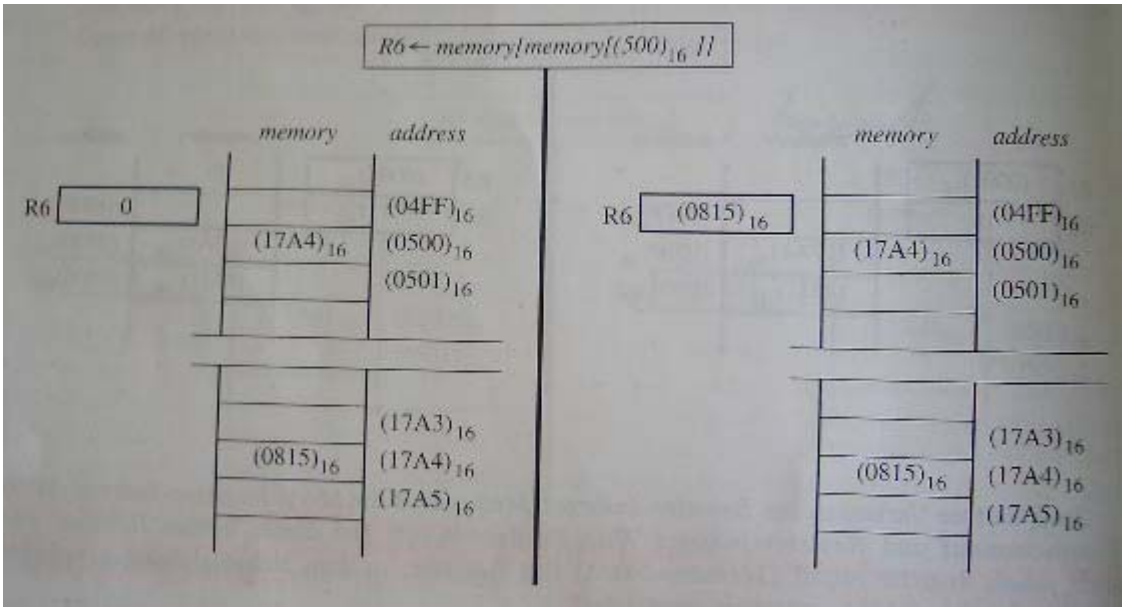
Speicherzugriff (memory fetch) das Register, in dem sich die Adresse befindet, erhöht oder vor der Leseoperation vermindert.

Program-Counter-Relative-Addressing Mode

Hier berechnet sich die effektive Adresse durch die Addition eines im Befehl angegebenen Offsets zum aktuellen Programmzählerstand. Das gestattet das Erstellen von Programmen, die an einer beliebigen Stelle des Arbeitsspeichers lauffähig sind. In einigen Prozessoren ist dieser Modus nur in Verbindung mit Sprüngen, Verzweigungsbefehlen oder Unterprogrammaufrufen zulässig.

Indirect-Addressing Mode

Hierbei handelt es sich um eine Zweistufige Speicheradressierung. Der Zellenwert entspricht der effektiven Adresse des gesuchten Operanden.
Für ein indirekt adressiertes Wort schreiben wir memory[memory[address]].



Architekturen

Heutige Prozessoren besitzen Komponente, die beispielsweise die Speicherverwaltung unterstützen, im einfachsten Fall eine eigene ALU für Adressberechnungen oder sogar eine Memory Management Unit zur Realisierung von virtuellen Speichersystemen und zum Schutz gegen missbräuchliche Zugriffe.
Die Performance eines Prozessors hängt sowohl von der Größe als auch von der Anzahl der vorhandenen Register und ihrer Funktionalität ab.
Leistungsschwache Prozessoren besitzen wenigstens zwei allgemeine Datenregister, einen Accumulator und einen B-Register zum Speichern der Operanden für die ALU-Funktionen.
An speziellen Registern müssen immer ein Memory Address Register, ein Stack Pointer, ein Program Counter und ein Program Status Word für die Funktionalität des Prozessors vorhanden sein.
Solche einfachen Prozessoren haben natürlich ein einfaches Instruktionsformat. Sie benutzen zur Adressierung meist den Implied-Addressing Mode.

Register die sowohl zur Speicherung von Daten als auch zur Modifikation von Adressen eingesetzt werden können, bezeichnet man als Mehrzweckregister. Dabei ist zwischen zwei Architekturströmungen zu unterscheiden: CISC(Complex Instruction Set Computer) und RISC(Reduced Instruction Set Computer). RISC Architekturen verfügen über große Registerfiles (128 und mehr Register). CISC-Prozessoren enthalten im Gegensatz dazu meist nur 16 oder ein paar mehr Register. Weiters haben die Adress- und Datenwortbreite sowie die Taktfrequenz einfluss auf die Performance des Prozessors.

Parallelverarbeitung innerhalb eines Rechners

Bei der Durchführung eines Maschinenbefehls wird der Befehl aus dem Hauptspeicher eingelesen und dann von einem Interpreter verarbeitet.
Dabei sind folgende Arbeitsschritte zu bewältigen:

1. Instruktion holen
2. Befehl decodieren
3. Operanden holen
4. Befehl exekutieren
5. Ergebnis speichern

Durch Varianten der klassischen Architektur kann man die Leistung des Prozessors steigern:

Vektorverarbeitung

Es kann vorkommen, dass ein und derselbe Befehl auf mehrere gleichartige Datenelemente zweimal hintereinander angewandt wird – wobei jedesmal alle Schritte der Abarbeitung eines Maschinenbefehls sequenziell ausgeführt werden müssen. Um Zeit einzusparen, lädt man bei diesem Konzept alle Operanden gleichzeitig in die Register und führt dann die Operation an allen vorhandenen Datenelementen parallel aus. Damit diese Variante funktioniert muss eine entsprechend große Anzahl von Registern und ALUs existieren.

Superskalare Verarbeitung

Superskalare Verarbeitung steigert die interne Paralleliät, indem alle Bestandteilezur Abarbeitung eines Befehls mehrfach in Form weitgehend autonomer Funktionseinheiten zur Verfügung stehen. Superskalare Prozessoren besitzen mehrere skalare Verarbeitungseinheiten.
Da mehrere Verarbeitungseinheiten zur Verfügung stehen, können weder Instruktionen mit längerer Ausführungszeit noch Cache Misses die Pipeline behindern.
Ein Nachteil dieses Konzepts liegt in der Tatsache, dass es durch die Parallelität zu einer Steigerung der internen Konflikte kommt.
Die Leistungssteigerung sehr stark von der jeweiligen Aufgabenstellung abhängt, ist eine effektive Nutzung nicht garantiert.
Ein weiteres Feature ist der Einsatz von Co-Prozessoren für Spezialfunktionen, dies kann auf verschiedene Arten erfolgen:

Vollständig sichtbare Anbindung: Der CPU-Instruktionssatz beinhaltet auch Instruktionen für die Co-Prozessoren. So erfolgt eine explizite Übergabe der Kontrolle von der CPU an den Co-Prozessor.

Partiell sichtbare Anbindung: Der CPU-Instruktionssatz enthält ebenso wie im ersten Fall Instruktionen für die Co-Prozessoren, aber CPU und Co-Prozessor arbeiten partiell unabhängig.

Transparente Anbindung: Die CPU weiß nichts von den Co-Prozessoren. Die Co-Prozessoren erkennen bestimmte Speicheradressen der CPU als Co-Prozessor Instruktionen. Die Co-Prozessoren funktionieren in diesem Fall weitgehend unabhängig von der CPU.

Intruction-Pipeline

Beim Pipelining wird der Hardware-Aufwand unter Umständen nicht im vollen Umfang vervielfacht, sondern die bestehenden Teile werden effizienter genutzt. Diese Konzept ist mit Fließbandarbeit vergleichbar, der Herstellungsprozess wird in einzelne teil zerlegt.

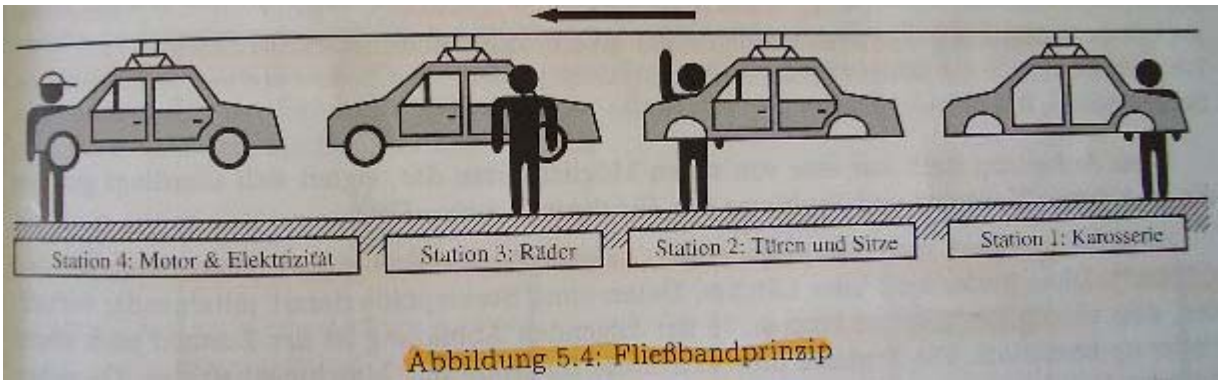
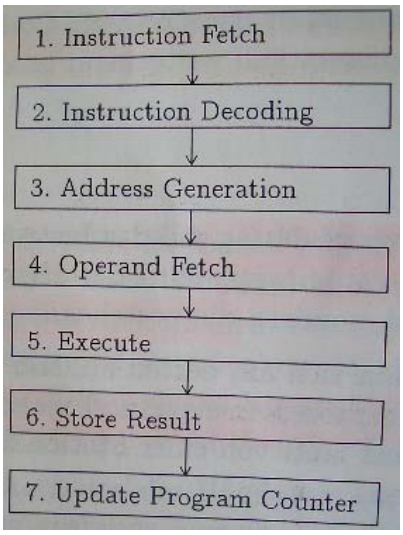


Abbildung 5.4: Fließbandprinzip

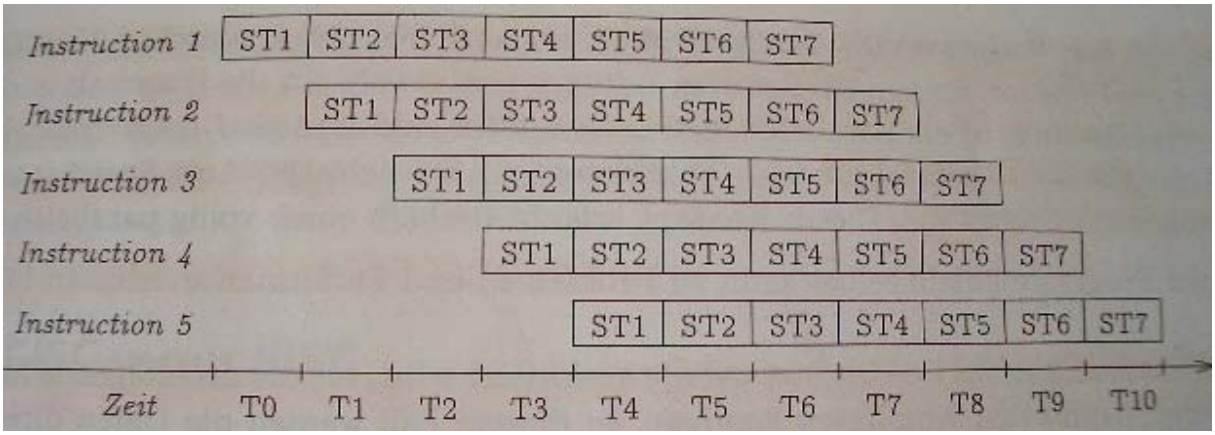
Aus der Sicht des Autos: Es macht keinen Unterschied, da es stets alle Produktionsschritte durchlaufen muss.
Aus dem Blickwinkel einer Person am Ende des Fließbandes: Für ihn ist das Intervall, das zwischen der Fertigstellung zweier aufeinander folgender Autos liegt, entscheidend kleiner als die Gesamtherstellungsdauer eines Kraftfahrzeugs.

Es ist ersichtlich, dass sich das Förderband nur dann weiterbewegen kann, wenn die zeitaufwendigste Tätigkeit am aktuellen Auto abgeschlossen ist.
Die Gesamtperformance richtet sich daher nach der zeitaufwendigsten Fertigungseinheit.
Beim Pipelining-Prinzip ist es deshalb wichtig, dass Bearbeitungszeiten der einzelnen Produktionseinheiten ungefähr gleich sind, damit kein Leerlauf an einzelnen Stationen entsteht.

Um dieses Prinzip auf einen Mikroprozessor umzulegen, unterteilen wir zunächst den Arbeitsvorgang „Maschinenbefehl ausführen“ noch detaillierter in die folgenden Schritte:



Zunächst realisiert man für jeden dieser Arbeitsschritte eine eigene Verarbeitungseinheit (Stufe). Diese Stufen sind über Latches, Daten- und Steuerpfade so miteinander verbunden, dass sie parallel arbeiten können.



In der Prozessorarchitektur sind die langsamsten Stufen für gewöhnlich die Speicheroperationen, sie bestimmten damit das Weiterschalten der Befehle und die Performance. Die Gesamtproduktivität beim Pipelining ist allgemein dennoch besser als bei sequenzieller Ausführung der Einzelschritte.

Folgende Voraussetzungen müssen für das Pipelining-Konzept erfüllt sein:
Falls die Befehlsteile keine Hardware-Betriebsmittel gemeinsam benutzen, genügt es, diese einfach auszuführen.
Damit sich die einzelnen Stufen aber möglichste wenig gegenseitig beeinflussen, benötigt jede Einheit eigene Latches zur Aufnahme des aktuellen Datenwortes.
Der Kontrollmechanismus, der nötig ist, um die Teilergebnisse von einer Stufe in die nächste zu transferieren, erhöht natürlich die Komplexität der Hardware.
Es muss auch eine Anpassung des Maschinen-Codes erfolgen, damit eine effiziente Nutzung der Pipeline garantiert werden kann.
Da ein Befehlsformat mit fixer Länge das sequenzielle Laden der Instruktionen erleichtert, findet ein solches Verwendung.

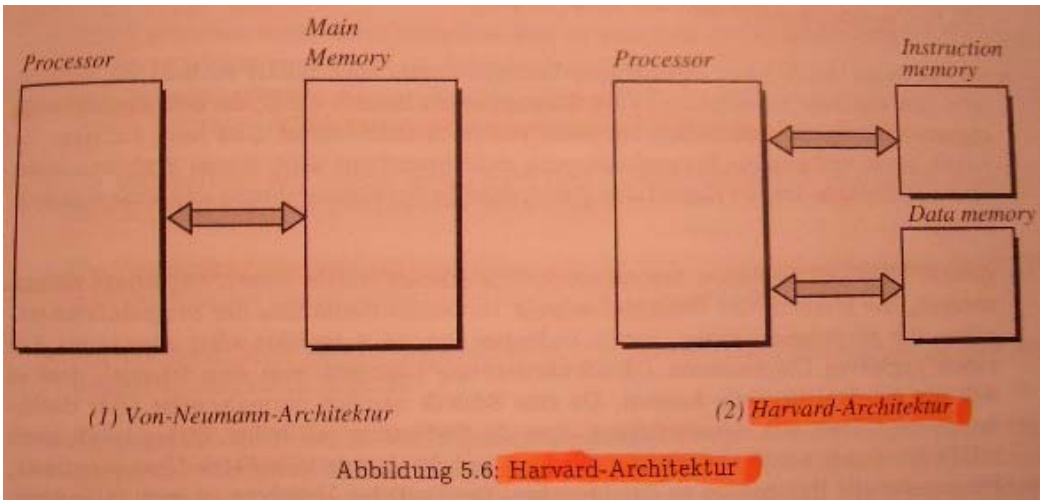


Abbildung 5.6: Harvard-Architektur

Wenn bei einer vollständig gefüllten Pipeline drei Stufen gleichzeitig versuchen auf den Hauptspeicher zuzugreifen, ergibt sich bei der klassischen Von-Neumann-Architektur ein Problem, da nur ein Speicher für und ein Bus sowohl für die Daten als auch für die Befehle vorhanden sind, ist ein paralleler Instruction-fetch und Operand-fetch unmöglich.
Eine Lösung dafür ist die Havard-Architektur, sie sieht getrennte Speicher und Busse für Programme und Daten vor.

Sogar der Programmablauf selbst kann zu Problemen und Performanceverlusten führen:

- Wird das Resultat des Befehls, der gerade exekutiert wird, für die nachfolgende Instruktion benötigt, kann es zu Konflikten kommen. In diesem Fall werden die Daten direkt an das jeweilige Rechenregister übergeben (data forwarding).
- Wenn das Ergebnis der aktuellen Operation ein Register zerstört, das von der nächsten Instruktion schon eingelesen wurde, ergibt sich daraus ein fehlerhafter Programmablauf. Durch geschicktes Umstellen der Befehle kann dieses Problem behoben werden (interferring instructions).
- Nach bedingten und unbedingte Sprüngen und Subroutine Calls muss die gefüllte Pipeline für ungültig erklärt (flush-pipe) und daher neu geladen werden.

Da Sprünge sehr häufig vorkommen sind bessere Methoden entwickelt worden um dieses Problem zu bewältigen.

Unbedingte Sprünge lassen sich frühzeitig erkennen, so dass die Instruction Fetch Unit an der neuen Stelle im Programm fortsetzen kann, noch bevor die Stufe ST7 den PC modifiziert.
Bei bedingten Sprüngen existieren größere Probleme, da ihr Ziel erst in der Execution-Unit durch die Auswertung der Bedingung ermittelt werden kann, ist es möglich das der gesamte Inhalt der Pipeline ungültig ist.
Dadurch wird der Aufwand nochmals erhöht, es bieten sich folgende Lösungsansätze an:

- Eine besonders einfache Maßnahme besteht darin, den Pipeline-Mechanismus zu stoppen, sobald die Decoding-Unit einen Sprungbefehl erkennt. Die Freigabe erfolgt erst, sobald die Zieladresse des Sprunges

ermittelt bzw. der PC erneuert wurde. Dieses Vorgehen bezeichnet man als Interlocking.
- Ein weiterer Lösungsansatz besteht darin, die dem Sprungbefehl sequenziell folgende Instruktion auf jeden Fall noch auszuführen. Man nennt dieses Verfahren Delayed Branch.
- Andere Verfahren versuchen, sobald ein Sprung erkannt wurde, dessen Zieladresse vorauszusagen (Predicted Branch).
- Die Prognose kann durch die Verwendung eines Sprungziel-Cache weiter verbessert werden. Dabei handelt es sich um eine Tabelle, welche die Zieladressen der schon einmal ausgeführten Sprüngeenthält (Branch History).

CISC versus RISC

Bei der CISC-Architektur werden auf der Maschinenbefehlsebene viele komplexe Konstrukte zur Verfügung gestellt. Dies geht jedoch auf Kosten der Performance. Softwareseiteige Ansätze zur Verbesserung der Performance sind:

- Beschränkung auf wenige (100 bis 200) Instruktionen, die durch die Compiler aktiv genutzt werden, die Emulation der anderen erfolgen durch den existierenden Befehlssatz.
- Möglichst optimale Implementierung dieser geringen Anzahl von Befehlen auf dem Prozessor

Die aus diesen Aspekten hervorgegangenen Architekturen nennt man RISC. Durch die zweite Forderung kann ein RISC in etwas der gleichen Zeit, die ein CISC benötigt um einen Befehl abzuarbeiten, mehrere Befehle ausführen.

Zur Performance-Steigerung tragen mehrere Punkte bei:

- RISCs brauchen wegen ihrer einfachen Befehle praktisch keinen Micro-Code. Durch die einfachere Hardware sind auch höhere Taktraten verwendbar.
- Die Verarbeitung der Befehle erfolgt im Pipeline-Prinzip, so dass in der Regel pro Maschinentakt ein Befehl beendet werden kann.
- Die einheitliche Länge aller Befehle ermöglicht einen hohen Wirkungsgrad der Pipeline und erleichtert eine effizientere Organisation
- Nur die Load/Store-Befehle kommunizieren mit dem Speicher, alle anderen Befehle verwendet ausschließlich Register.
- Es stehen eine große Anzahl von allgemein benutzbaren Registern zum Speichern von Operanden zur Verfügung (Registerbänke).
- Mehrere Pipelines werden implementiert.
- Um die durchschnittliche Dauer der Lade- und Speicherbefehle zu reduzieren, wird ein On-Chip-Cache für Daten eingerichtet.
- Verwendung von speziellen Compilern die sogar Pipelinekonflikte weitgehend automatisch lösen.

RISC-Weiterentwicklungen:

SPARC-Architektur (Scalabre Processor ARChitecture): Konventioneller RISC-Prozessor mit vierstufiger Befehlspipeline (Instruction Fetch, Instruction Decoding, Execute, Store Result).

MIPS-Architketur (Microprocessor without Interlocking Pipelining Stages): Die Performance-Steigerung wird durch eine feinstufige Befehlspipeline und die realisierte Speicherhierarchie umgesetzt.

Speicher

Da Zugriffe auf Speicherbausteine Langsam sind und bei der klassischen Rechnerarchitektur sowohl Daten als auch Programme sich im Hauptspeicher befinden ist dies Problematisch. Man spricht hierbei vom „von Neumannschen Flaschenhals“. Wir wollen nun Maßnahmen vorstellen die den Transfer zwischen Hauptspeicher und Prozessor deutlich beschleunigen. Unter Bandbreite versteht man die Anzahl der Bits auf die pro Sekunde zugegriffen werden kann.
Dabei muss erwähnt werden das schnellere Speicher (SRAM) viel teurer sind als langsamere (DRAM).

Ansätze zu Steigerung der Performance sind:
- Eine geschickte Anordnung der vorhandenen Speicherbausteine
- Der Einsatz von Caches
- Harvard-Architektur

Interleaved Memory

Dieses Konzept geht davon aus, dass meistens sequenziell auf den Speicher zugegriffen wird. Der Speicher wird in gleich große Bereiche (Speicherbänke) unterteilt, aufeinanderfolgende Adressen liegen immer in einer anderen Bank.

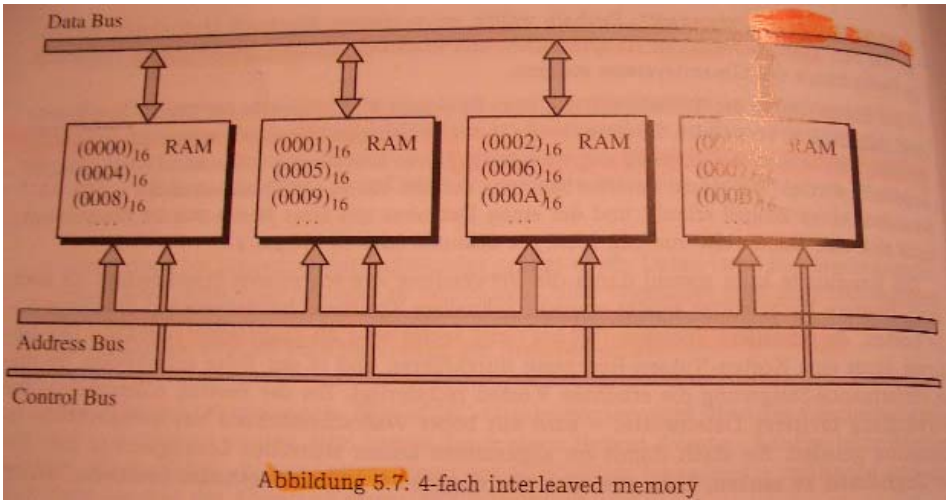


Abbildung 5.7: 4-fach interleaved memory

Man erkennt, dass der Adressbereich in den Bausteinen nicht mehr geschlossen ist. Damit der ganze Chip trotzdem vollständig ausgelastet werden kann, benutzt man eine geschickte Anordnung der Adressleitungen, um diese Struktur zu erreichen.
Die einfachste Form des Interleavings ist eine Aufsplittung des Speicherbereichs in zwei Teile. In einem befinden sich die Datenwörter mit den geraden, im anderen die mit den ungeraden Adressen.

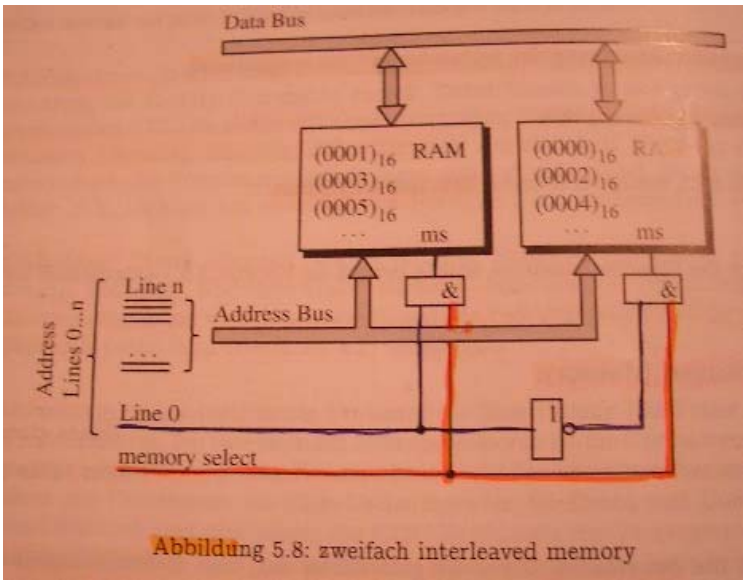


Abbildung 5.8: zweifach interleaved memory

Das Interleaved Memory überträgt sozusagen das Pipeline-Prinzip auf Speicherzugriffe.

<u>Caches</u>

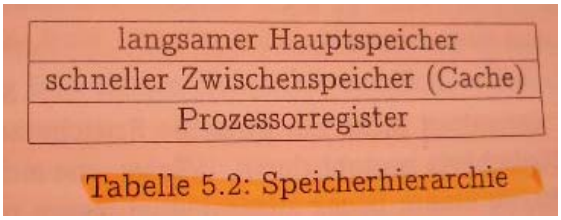


Tabelle 5.2: Speicherhierarchie

Ein Cache setzt sich aus SRAMs zusammen. Er ist allgemein kleiner als der Hauptspeicher, aber größer als das Register-File. Er besitzt eine eigene Hardware für das Adressieren, Laden und Auslagern von Speicherinhalten.
Die Cache-Speicher sollen dem Prozessor während der Ausführung eines Programms die benötigten Daten entweder aus dem eigenen Speicherbereich (Cache Hit) oder dem Hauptspeicher (Cache Miss) zur Verfügung stellen. Die Wirksamkeit eines Cache hängt von seiner Trefferrate (hit rate) ab.

Die durchschnittliche (effektive) Speicher-Zugriffszeit t_{eff} kann wie folgt berechnet werden:

$t_{eff} = h*t_{cache} + (1 – h) * t_{main}$

t_{cache} … Zugriffszeit auf den Cache
t_{main} … Zugriffszeit auf den Hauptspeicher
h … Trefferrate im Cachespeicher
Die effektive Zugriffszeit hängt rein von der Trefferrate ab.

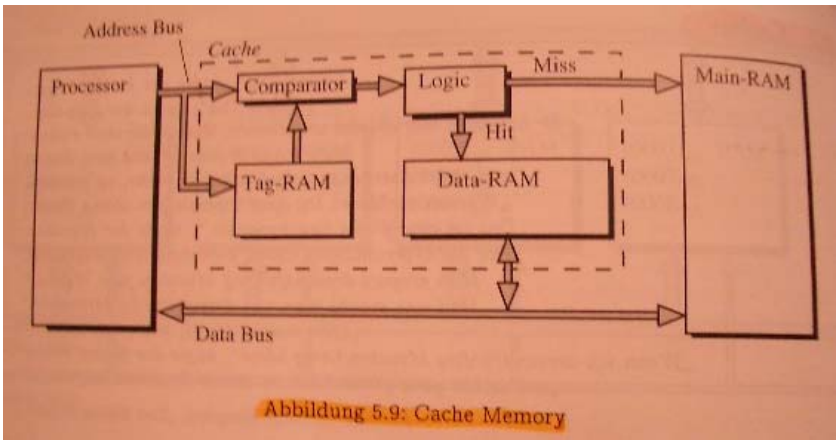


Abbildung 5.9: Cache Memory

Das Tag-RAM beinhaltet ein Verzeichnis von den Adressen der Speicherplätze, die sich derzeit im Cache befinden. Bei einer Lese- oder Schreiboperation wird deshalb die Adresse der benötigten Informationen mit den im Tag-RAM gespeicherten verglichen. Bei einer Übereinstimmung wird das Hit-Signal aktiviert und das Daten RAM legt die gewünschte Bitkombination auf den Bus. Ansonsten muss das Datum aus dem Hauptspeicher geladen werden. Um zukünftige Misses zu vermeiden, wird jedes Datum, das aus dem Hauptspeicher geladen wird, auch in den Cache-Speicher aufgenommen.

Voll assoziative Caches verwalten nicht nur einzelne Speicherzellen, sondern stets zusammenhängende Blöcke. Eine Möglichkeit besteht darin, Wörter, die sich nur durch die drei letzten Bits der Adresse unterscheiden, zu einem Block zusammenzufassen und sie in eine Zeile des Caches zu laden.

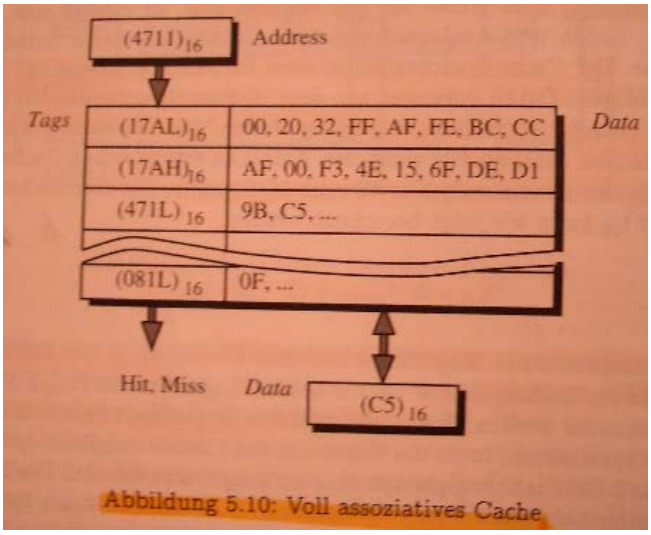


Abbildung 5.10: Voll assoziatives Cache

L = 0 …7 (LOW-Byte)
H = 8 … F (HIGH-Byte)

Wegen des begrenzten Speicherraums muss man nach einiger Zeit Zeilen freigeben, um neue nachladen zu können. Verschiedene Replacement-Strategien sind dabei einsetzbar:

LRU (Least Recently Used): Im Fall mehrerer Kandidaten wird jener ausgewechselt, dessen letzter Aufruf am längsten zurückliegt.
LFU (Least Frequently Used): Die Zeile die am seltensten Verwendet wurde wird ausgetauscht.

RANDOM : Von den mögliche Kandidaten wird einer zufällig selektiert

Die genannten Strategien sind alle etwas gleich gut.

Eine andere Realisierungsvariante von Caches besteht darin, bestimmte Adressen nur in einer vorgeschriebenen Cache-Zeile zu speichern. Das bedeutet, dass ein Wort abhängig von den niedrigsten Stellen seiner Adresse zwingend einem genau festgelegten Fach zugewiesen wird (Direct Mapping). Im Tag-RAM werden nur mehr die höherwertigen Stellen der Adresse vermerkt.

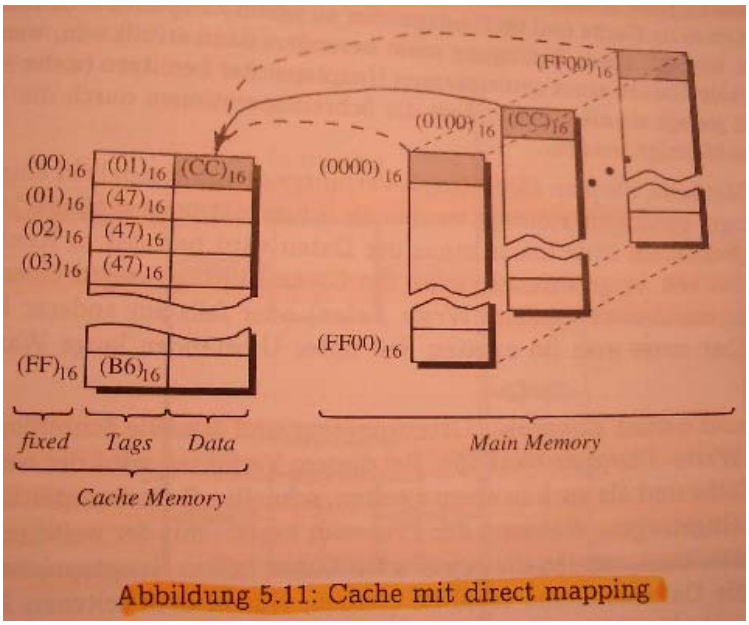


Abbildung 5.11: Cache mit direct mapping

Assoziativer Zweiwege-Cache ist eine verbesserung des Direct Mapping

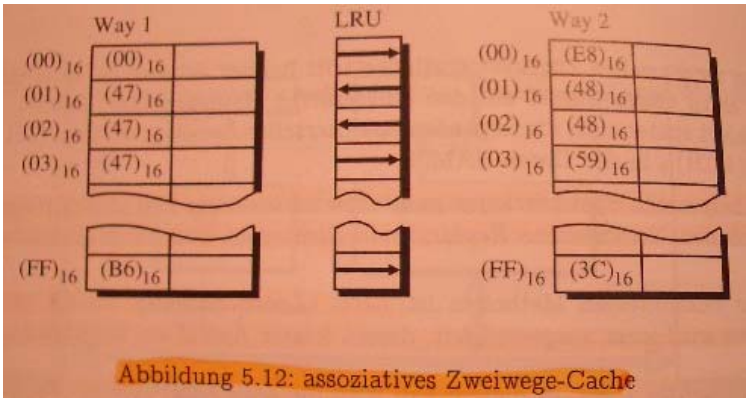


Abbildung 5.12: assoziatives Zweiwege-Cache

Die LRU-Methode kann man hier besonders leicht implementieren. Der Hardware-Aufwand ist beim Direct Mapping bzw. bei den Mehrwege-Caches recht gering.

Nun wollen wir die Besonderheiten der Schreibprozedur näher betrachten. Die Art und Weise wie diese gehandhabt werden, hat entscheidende Auswirkungen auf die Performance. Es ist am einfachsten, die Daten sowohl im Cache als auch im Hauptspeicher einzutragen, sie sozusagen durch den Zwischenspeicher „hindurchzuschieben“. Daher nennt man diese Verfahren auch Write Through.

Ein weiteres Verfahren ist das Copy-Back-Verfahren, hier werden die Schreibaktionen zunächst lediglich im Cache ausgeführt und gesammelt. Die

Aktualisierung wird erst vorgenommen, wenn die Cache-Einträge wegen eines Cache Miss an den Hauptspeicher retourniert werden (Write Later).

Die Vorteile dieser beiden Konzepte (Datenkohärenz und schnelle Schreiboperationen) vereinigt die Buffered-Write-Through-Methode. Bei diesem Verfahren wird der neue gleichzeitig sowohl in das Cache als auch in einen zweiten, schnellen Zwischenspeicher (Pufferspeicher) eingetragen. Während der Prozessor bereits mit der weiteren Abarbeitung des Programms fortfahren kann, werdend die gepufferten Daten in den Hauptspeicher übertragen.

Zur Entschärfung des Neumannschen Flaschenhals besitzen moderne Prozessoren oft getrennte Daten- und Intruktions-Caches (split Cache).

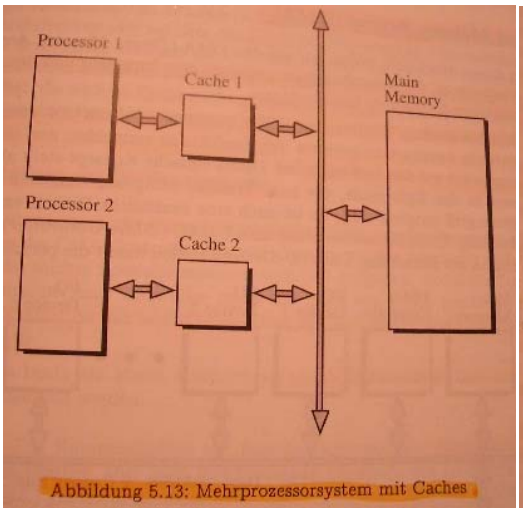


Abbildung 5.13: Mehrprozessorsystem mit Caches

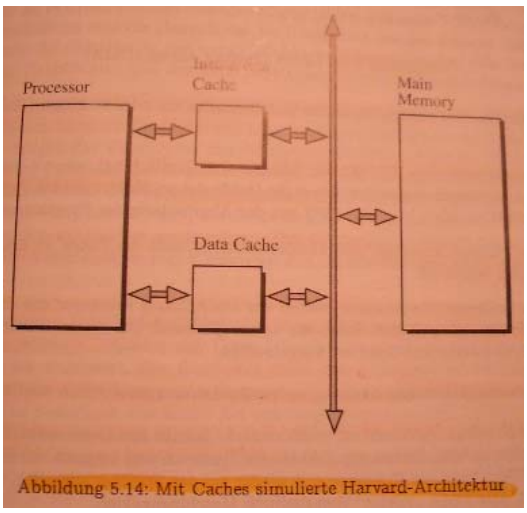


Abbildung 5.14: Mit Caches simulierte Harvard-Architektur

Direct Memory Access (DMA)

Dieses Konzept erlaubt es, die Kommunikation zwischen dem Prozessor und den meist sehr viel langsameren peripheren Geräten zu beschleunigen.
Ein DMA dient zur direkten Übertragung großer Datenmengen vom bzw. zum Speicher, ohne dabei die CPU in Anspruch zu nehmen. Um Konflikte zu vermeiden, darf die CPU während das DMA-Vorgangs nicht auf den Bus zugreifen.

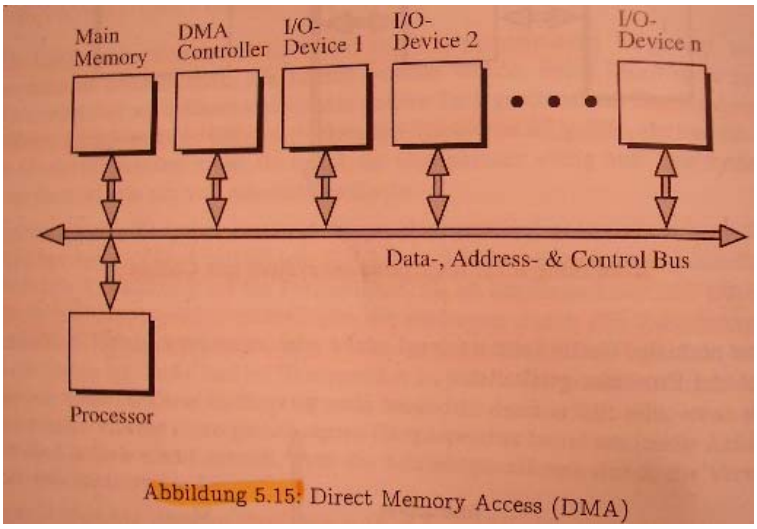


Abbildung 5.15: Direct Memory Access (DMA)

Folgende Schritte müssen beim Datentransfer durchgeführt werden:

1. Der Prozessor teilt dem DMAC die Adresse der Quelle, die des Ziels sowie die Größe der zu übertragenden Daten mit.
2. Der DMAC fordert zunächst vom entsprechenden Gerät die Daten an und wartet, bis es zum Transfer bereit ist.
3. Nach dem Ende der Übertragung meldet der DMAC dem Prozessor den erfolgreichen Abschluss der Aktion meistens durch ein Interruptsignal. Die Daten werden dann direkt zwischen I/O-Device und Speicher ausgetauscht.

Wegen der unterschiedlichen Buszuteilung unterscheidet man zwei Arten von DMAs:

1. Das Cycle-Stealing-Verfahren, sobald das Gerät seine Bereitschaft zum Datentransfer meldet, fordert der DMAC die Busse an und beginnt mit der Übertragung. Der Prozessor muss warten.
2. Verbesserte DMAs beobachten die Busse und transferieren die Daten während der Zyklen, in denen der Prozessor nicht den Bus benutzt.

Controller und Co-Prozessoren

Controller sind Prozessoren, die spezielle Aufgaben erfüllen, um den Prozessor zu entlasten. I/O-Controller, welche die Kommunikation mit bestimmten Arten von peripheren Geräten durchführen, nennt man auch Channels.

Die wichtigsten (Mikro-)Controller:

Externspeicher: Harddisksm Floppys, Tapes oder optische Platten
Graphik I/O: übernehmen die Steuerung von Bildschirmen
Serial I/O: Anbindung der CPU an die Peripherie mit Hilfe einer seriellen Datenleitung. Der Controller führt die Umwandlung zwischen parallelen und seriellen Darstellungen der Daten durch.
Netzwerke: Die Kommunikation in Netzwerken wird mit Spezial-Prozessoren (communication processors) durchgeführt.
Mathematik-Co-Prozessoren: Führt beispielsweise Gleitkomma-Berechnungen durch.
Graphik-Co-Prozessoren: Entlastet die CPU von komplexen graphischen Operationen.
Signalprozessoren: Die Synthese und Analyse von analogen Signalen ist ebenfalls sehr rechenintensiv und benötigt häufig spezielle Reihenentwicklungen. Diese DSPs (Digitale Siganprozessoren) haben meiste mehrere analoge Eingänge (A0, A1, … , A7), eine analoges Signal wird in ein digitales umgewandelt, das in der Folge von der CPU des Signalprozessors verarbeitet wird. Ein DSP besitzt häufig getrennte Busse für die Zugriffe auf RAM und ROM. Die Ergebnisse des DSP werden über einen Digital/Analog-Converter als analoges Signal ausgegeben.

Die Hauptanwendungsgebiete von DSPs sind:

- Pulscodemodulation (PCM)
- Sprach- und Bildverarbeitung
- Regelungstechnik

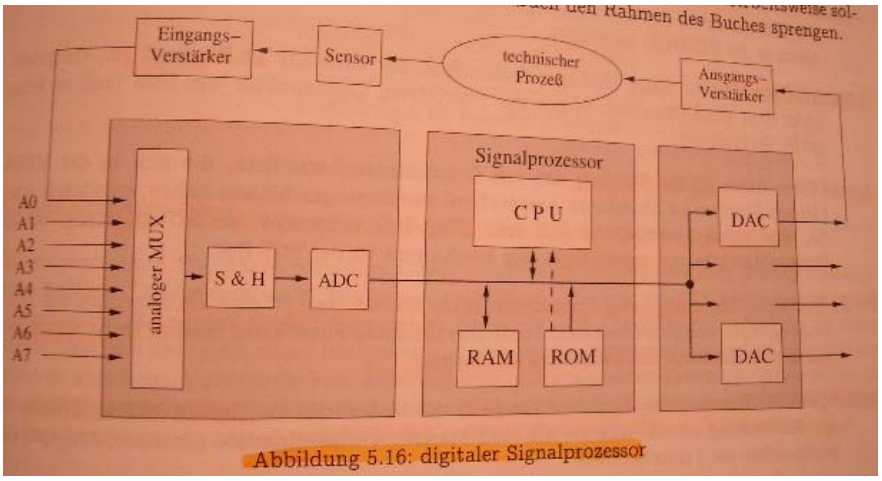


Abbildung 5.16: digitaler Signalprozessor

Multimediaprozessoren: Sind eine Weiterentwicklung der Graphik-Controller, er komprimiert und dekomprimiert Bilddaten, erzeugt schnelle Graphik- bzw. Spezialeffekte, erzeugt 2D und 3D Objekte, usw.

Interconnection

Die Bestandteile eines Computersystems müssen stets miteinander kommunizieren können. Dies wird durch die Interconnection (Switch), einen Teil der Architektur möglich. Bisher fanden für die Interconnection stets Bussysteme Einsatz. Dabei lassen sich zwei Arten unterscheiden:

Paralleler Bus: Dieses Verfahren verwendet zur Übertragung von jedem Bit jeweils eine Leitung, das kann allerdings zu Problemen bei der Anzahl der benötigten Leitungen führen.
Serieller Bus: Hier werden alle Bits nacheinander über eine Leitung übertragen, wodurch es zu sehr hohen Übertragungszeiten kommen kann.

Bussysteme die im Laufe der Zeit verbreitet eingesetzt wurden:

ISA (Industrial Standard Architecture): Datenwortbreite 16 Bit
EISA (Extended Industrial Standard Architecture): kompatible Weiterentwicklung des ISA auf 32 Bit
MICROCHANNEL: 32 Bit breit
Local Bus: 32 Bit PC Bus, Er ist streng an der Prozessorarchitektur orientiert.
PCI: Der PCI Bus (Peripheral Component Interconnect Bus), verbesserung des Local Bus, da er viel mehr Funktionen wie dieser besitzt. Er ist ausserdem von der Architektur des Prozessors unabhängig.
SCSI (Small Computer System Interface): High-Level Schnittstelle für die Anbindung sowohl interner als auch externer Peripheriegeräte.
PCMCIA (PC Memory Card International Association): Entwicklung für Busse, die mittels eines schkartengroßen Bausteins mit dem Computer verbunden werden. Hauptsächlich für Notebooks, 16 Bit

6.Netzwerke

Aufbau

Mehrere Rechner sind über eine Medium miteinander verbunden. Jeder Rechner enthält eine eindeutige Kennzeichnung, damit Daten spezifisch an ihn adressiert werden können.

Netzwerktypen

LAN = Local Area Network
MAN = Metropolitan Area Network
WAN=Wide Area Network

Ein Subnet besteht aus mehreren Übertragungsstrecken und aus Schaltstellen die zwei oder mehr Übertragungsstrecken miteinander verbinden.
Schaltstellen sind spezialisierte Rechner die wir als Interface Message Processors (IMPs) bezeichnen.
Ihre Aufgabe ist es über Input Channels hereinkommende Daten an den richtigen Output Channel weiterzuleiten.

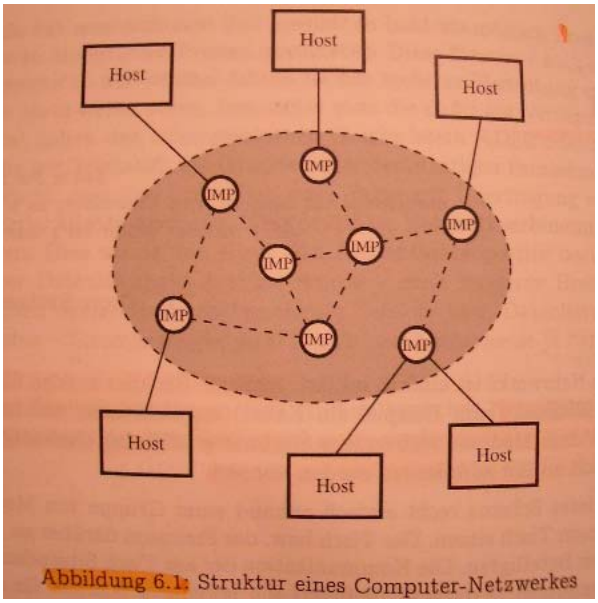


Abbildung 6.1: Struktur eines Computer-Netzwerkes

Circuit- und Packet Switching

Circuit- und Packet Switching sind zwei unterschiedliche Techniken zur Informationsübertragung innerhalb eines Subnets.
Beim Circuit Switching wird vor dem Datenaustausch eine genau festgelegte Verbindung zwischen den Kommunikationspartnern hergestellt.
Wesentlich adäquater ist das Packet Switiching. Die zu übermittelnden Daten werden dabei in Blöcke gewisser Größe zerteilt. Jedes solche Packet wird mit der Adresse des Empfängerhosts versehen und unabhängig von den anderen auf die Reise geschickt.

Es gibt zwei verschiedene Strukturen für ein Communication Subnet:

Point-to-Point Subnets: Eine einzelne Übertragungsstrecke verbindet hier genau zwei IMPs. Die Übermittlung von Paketen zwischen zwei nicht direkt verbundenen IMPs ist nur im Umweg über andere IMPs möglich. Ein solcher zwischen-IMP hat die Aufgabe, ein hereinkommendes Packet zu empfangen, bis zum Freiwerden der richtigen Output Channels zu speichern und schließlich weiterzuschicken.

Broadcast Subnets: Subnets dieser Art zeichnen sich durch eine einzelne Übertragungsstrecke aus, die alle IMPs verbindet. Die für die Ankopplung des Hosts zuständigen IMPs sind in der Praxis auf die Network-Controller der einzelnen Computer reduziert, die Notwendigkeit von „Zwischen-IMPs“ entfällt.

Es gibt die Möglichkeit, die Kapazität eines Broadcast-Mediums (statisch) auf viele, logisch getrennte Übertragungskanäle aufzuteilen. Geeignete Techniken dafür sind FDM (Frequency –Division Multiplexing) und TDM (Time-Division Multiplexing).

Standartdisierung

Computer Netzwerke waren relativ frühzeitig von Standartdisierungsbestrebungen geprägt. Eine wichtige Organisation auf diesem Gebiet ist die ISO (International Organization of Standardization) die auch mit dem OSI Reference Model (Open Systems Interconnection) einen wichtigen Beitrag zur Standardisierung von Computer Netzwerken geleistet hat.

ISO	*International Organisation for Standardization*
IEEE	*Institute of Electrical and Electronics Engineers*
NBS	*National Bureau of Standards*
IEC	*International Electrotechnical Commission*
ECMA	*European Computer Manufacturers Association*
IFIP	*International Federation for Information Processing*

Tabelle 6.2: Internationale Standardisierungsbehörden

7.Architekturen

Die Mechanismen, die ein Host (bzw. ein IMP) zur Verfügung haben muss, um an einem Netzwerk partizipieren zu können, werden in der Praxis durch eine Anzahl aufeinander aufbauender Schichten (Layer) implementiert. Der Layer n einer Maschine kommuniziert dabei unter Einhaltung eines genau festgelegten Satzes von Regeln und Konventionen (das sogenannte Layer n Protokoll) mit dem Layer n einer anderen Maschine.

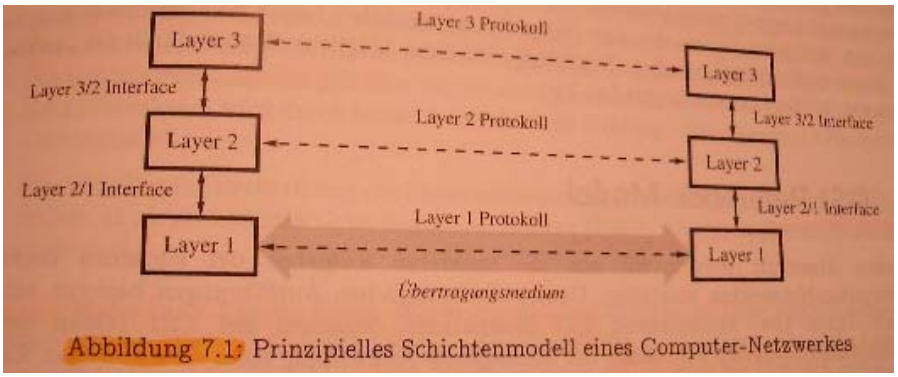


Abbildung 7.1: Prinzipielles Schichtenmodell eines Computer-Netzwerkes

OSI Reference Model

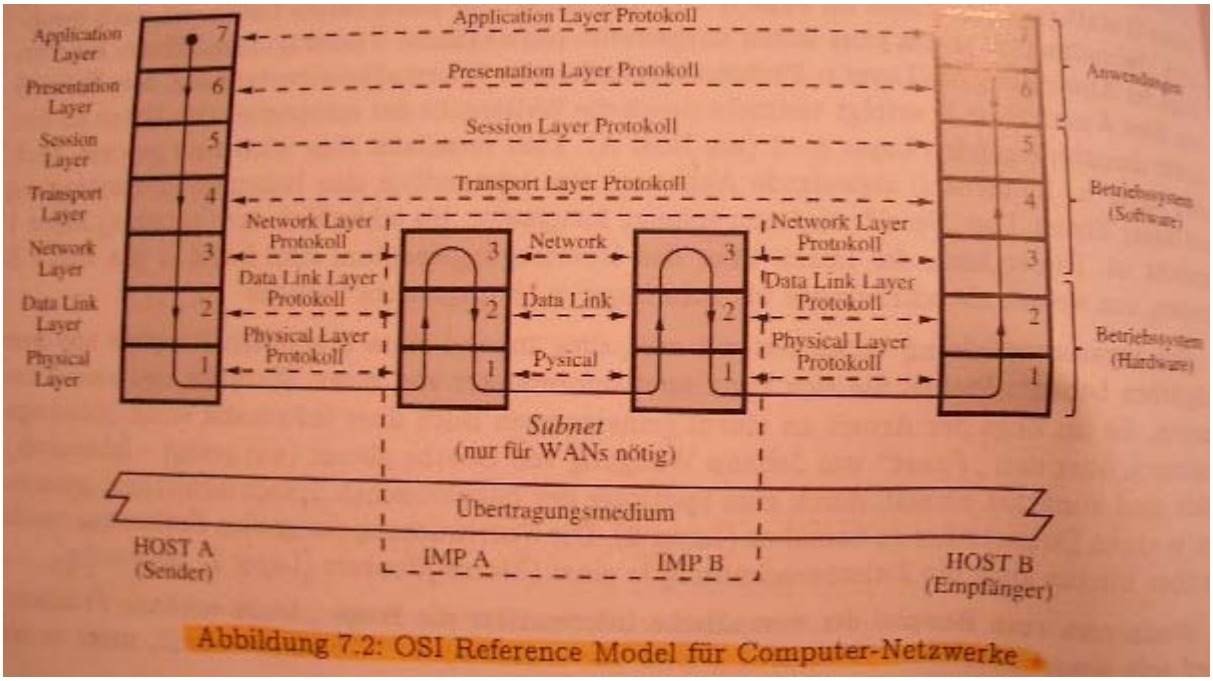


Abbildung 7.2: OSI Reference Model für Computer-Netzwerke

Layer 1 – Physical Layer: Die Aufgabe dieses Layers kann grob mit der Übertragung einzelner Bits umrissen werden. Das Layer 1 Protokoll umfasst also grob gesagt die Konventionen und Regeln, nach denen „einzelne“ Bits zu übertragen sind. Die angebotenen Services bieten die Möglichkeit, einen Strom von Bits über das jeweilige physikalische Medium zu senden beziehungsweise zu empfangen. Bittaktregeneration wird dadurch bewirkt, dass das empfangene (verzerrte) Empfangssignal durch einen Amplitudenentscheider bewertet wird. Wortsynchronisation erfolgt dadurch, dass beim Sender periodisch am Anfang jedes Nachrichtenblocks ein Synchronisationswort in den seriellen Bitstrom eingefügt wird. Neben den periodisch eingeblendeten Synchronisationsworten treten statistisch verteilte, scheinbare Synchronisationsworte als Bestandteil der Nutzdaten auf, die es jedoch aufgrund fehlender Periodizität auszublenden gilt.

Layer 2 – Data Link Layer: Dieser Layer bildet den Grundstein zur fehlerfreien Datenübertragung. Die vom Network Layer kommenden Daten werden in „mundgerechte“ Stücke portioniert und mit einem Header und einem Trailer versehen. Der Header kennzeichnet den Beginn des Frames, der Trailer enthält üblicherweise eine CRC-Checksumme und markiert das Ende des Frames. Die Frames werden nun nacheinander an den jeweiligen Empfänger IMP übermittelt. Eine weitere Aufgabe des Data Link Layers ist es in Verlust geratene oder beschädigte Frames erneut zu senden, dies wird als Error Control bezeichnet.

Layer 3 – Network Layer: Dieser Layer ist für den eigentlichen Betrieb des Communication Subnets zuständig. Damit ist jene Grenze erreicht, die den Zustandsbereich des Betreibers eines Subnets von dem der einzelnen Netzwerkbenutzer trennt. Die konkreten Schnittstellen zum Transport Layer heißen in der ISO-Terminologie Network Service Access Points (NSAPs).
Das Ziel eines Packets ist durch die eindeutigen NSAP-Adressen festgelegt.
Die Hauptaufgabe der Network Layer ist das sogenannte Routing, also die Lösung der Frage, über welche IMPs ein Paket am effizientesten zu seinem Ziel geschickt

werden kann. Dazu bietet der Network Layer in der Regel sowohl connection-oriented als auch connection-less services an. Bei den connection-oriented services wird vor dem eigentlichen Datenaustausch eine dedizierte (logische) Verbindung zwischen den Kommunikationspartnern hergestellt. Im Gegensatz dazu wird bei einem connection-less service jedes Daten Packet unabhängig von allen anderen durch das Subnet geschleust.

Layer 4 – Transport Layer: Die Aufgabe dieses Layers kann grob mit der Abschirmung der höheren Layer von gewissen Eigenheiten des Network Layers umrissen werden. Der Transport Layer stellt Methoden zur „sicheren“ Übertragung von Transport-Paketen bereit.
Die konkreten Schnittstellen zu Session Layer werden Transport Service Access Points (TSAPs) genannt. Sie repräsentieren die Endpunkte der sogenannten Transport Connections und werden durch netzwerkweit eindeutige Adressen identifiziert. Die Bereitstellung eines einheitlichen Schemas für TSAP-Adressen ist eine der ganz wesentlichen Aufgaben des Transport Layers.
Hinsichtlich Services geht es hauptsächlich um die Bereitstellung von connection-oriented services.

Layer 5 – Session Layer: Grob umrissen ist dieser Layer für die Kommunikation zwischen Prozessen auf verschiedenen Hosts zuständig.
Im Prinzip bietet der Session Layer etwas erweiterte Transport Layer Services an, unter anderem auch ein Analogon zu den Atomic Actions, und zwar für Messages. Eine Atomic Action kann nicht in mehrere Einzelaktionen geteilt werden, sie muss garantiert entweder komplett erfolgreich durchgeführt werden oder im Fehlerfall in den Urzustand vor Beginn der Ausführung zurückkehren. Im Fall der Message wird sichergestellt, dass entweder alle oder aber gar keine der zur Activity gehörenden Messages beim Empfänger ankommen.

Layer 6 – Presentation Layer: Dieser Layer ist im wesentlichsten mit der Syntax und der Semantik der übertragenen Information befasst. Da verschiedene Computer intern unterschiedliche Datenformate aufweisen können, muss bei einer Datenübertragung eine entsprechende Konversion stattfinden.

Layer 7 – Application Layer: Dieser Layer enthält die eigentlich Applikationen, für welche die ganzen Services des Netzwerkes eigentlich gedacht sind (z.B. Electronic Mail Services oder File Server).

LAN und WAN

Ethernet

Unter der Bezeichnung Ethernet ist die bekannte Technologie Carrier Sense Multiple Access with Collision Detection (CSMA/CD() am weitesten verbreitet.

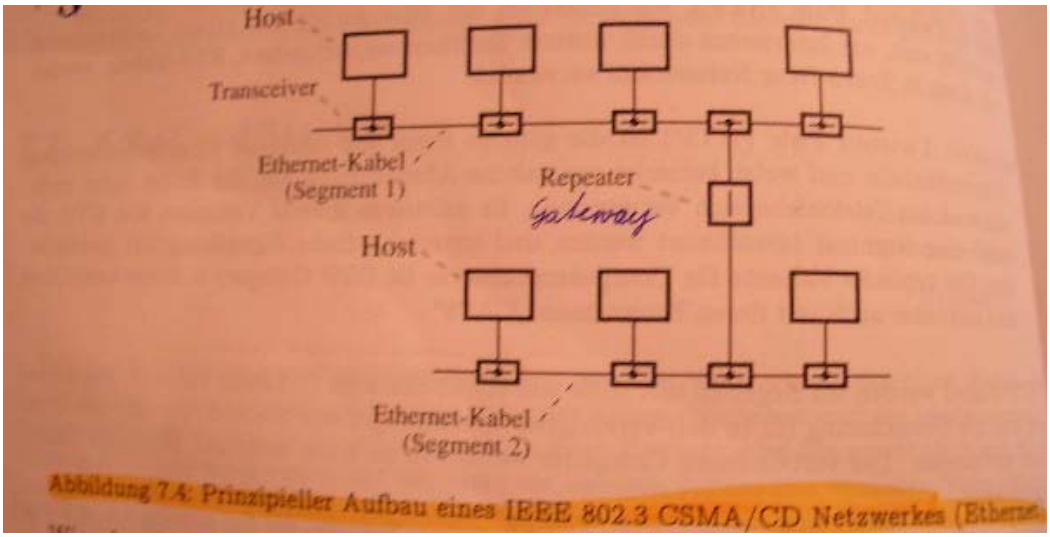


Abbildung 7.4: Prinzipieller Aufbau eines IEEE 802.3 CSMA/CD Netzwerkes (Ethernet

In einem Ethernet-Netzwerk ist ein Mechanismus zu Koordination gleichzeitiger Sendeversuche erforderlich. Dazu wird CSMA/CD verwendet. Das Prinzip ist ganz einfach: Ehe ein „sendewilliger“ Host einen Frame abschickt, stellt er fest, ob gerade eine Datenübertragung stattfindet. Ist das der Fall, wird noch deren Beendigung abgewartet, andernfalls sendet er seinen Frame sofort. Sofern nicht allzu viele Teilnehmer gleichzeitig ihre Frames loswerden wollen, funktioniert dieses Verfahren problemlos und schnell.
Als häufigstes Übertragungsmedium findet in heutigen Netzwerken die Twisted-Pair-Verkabelung Anwendung, über welche die Computer sternförmig mit einem sogenannten Hub oder Switch verbunden sind. Ein Hub besitzt die Eigenschaft, die an einem Eingang eingehenden Signale auf alle Ausgänge zu verteilen.
Mit BNC können ca. 500 m mit Twisted Pair verkabelung nur mehr 200m überwunden werden.

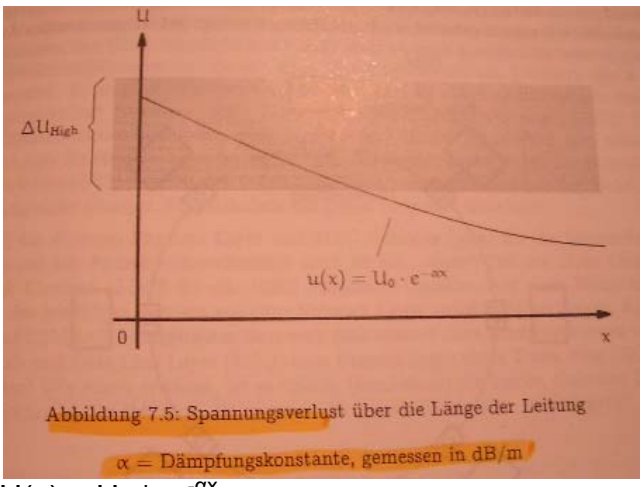


Abbildung 7.5: Spannungsverlust über die Länge der Leitung

α = Dämpfungskonstante, gemessen in dB/m

$U(x) = U_0 * e^{-\alpha x}$
α = Dämpfungskonstante, gemessen in dB/m

<u>Fast Ethernet</u>

Hier sind Übertragungsraten von 100 MBit/s möglich.
Im Gegensatz zu gewöhnliches Hubs besitzen Switches die Fähigkeit, zwei Hosts miteinander kommunizieren lassen zu können ohne dass dies die Kommunikation des restlichen Netzwerks betrifft,´.

Token Ring

Die Hosts werden hier durch eine „Kette“ einzelner Koaxialkabel derart verbunden, dass die Topologie eines geschlossenen Ringes entsteht.

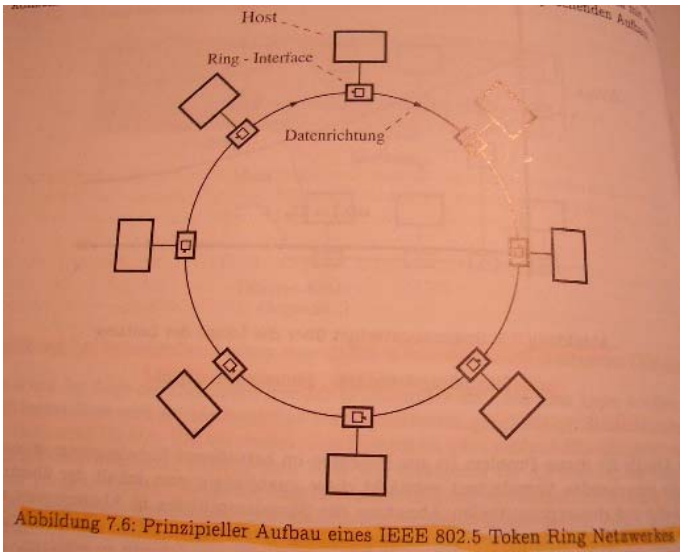


Abbildung 7.6: Prinzipieller Aufbau eines IEEE 802.5 Token Ring Netzwerkes

Im Normalzustand rotiert nun auf dem Ring ein spezielles Bitmuster, der sogenannte Token, mit dessen Hilfe gleichzeitige Schreibzugriffe koordiniert werden: Wenn eine Teilnehmer einen Frame übertragen will, so wartet er zunächst einmal darauf, dass der Token an seinem Ring-Interface vorbeikommt. Ist das der Fall, so invertiert er das letzte Bit des Tokens und wandelt ihn so in jenes Bitmuster um, das den Start eines Frames kennzeichnet.
Die eigentliche Datenübertragung erfolgt dadurch, dass der Host den Ring am Interface auftrennt und seinen Frame bitseriell in den Ring einspielt. Da auf diese Weise nur jener Teilnehmer einen Frame senden darf, der unmittelbar zuvor den Token verändert hat, kann zu jedem Zeitpunkt höchstens ein Host senden.
Der Hauptvorteil des Token Rings liegt zweifellos darin, auch in Hochlastfällen optimale Performance gewährleisten zu können. Es ist gegenüber Ethernetvarianten definitiv kollisionsfrei.

WaveLAN

Wi-Fi Standards (Wireless Fidelity) erlauben die Vernetzung von Computern per Funk und wurden unterm IEEE Standart 802.11 exakt definiert.
Die 802.11-Familie umfasst aktuell fünf Protokolltypen: 802.11-legacy, 802.11a, 802.11b, 802.11g und 802.11n, wobei letzterer sich noch in der Planungsphase befindet.

Wi-Fi Netzwerktrennung SSID:

Es existieren die diversen Wi-Fi Standards Kanäle, die helfen sollen, lokal nebeneinander betriebene Funknetze voneinander getrennt zu halten.
Der Service Set Identifier (SSID) legt für jedes Wi-Fi Netzwerk einen eindeutigen Netzwerknamen fest. Damit zwei Geräte miteinander kommunizieren können, muss deren SSID übereinstimmen.

Betriebsmodi

Infrastructure Modus: in diesem Modus existiert im Netz ein Access Point, welcher die Übertragung von Signalen im Netzwerk reguliert.

ad hoc Modus: im ad hoc Modus besteht das Funknetz nur aus Client-Systemen ohne Access Point.

Digital Subscriber Line (DSL)

DSL repräsentiert eine Technik, digitale Daten über eine Leitung des öffentlichen Telefonnetzes zu senden und zu empfangen. Die Daten werden über das ungenutzte Frequenzspektrum der Telefonleitungen geschickt.

Funktionsweise

Die menschliche Stimme reich von 0 bis 15 kHz. Um sie allerdings per Telefonleitung zu übermitteln, reicht es, sie auf 300 Hz bis 3,3 kHz frequenzmässig zu begrenzen. Die meisten heutigen Telefonleitungen können allerdings Frequenzen bis hinauf zu 200 kHz bis 800 kHz übermitteln. Das frei bleibende Band kann nun mit Hilfe eines DSL-Modems für die Übermittlung von Daten genutzt werden.

Bluetooth

Bluetooth ist eine Funktechnik, die im Frequenzbereich zwischen 2,402 und 2,480 GHz sendet.
Bluetooth unterteilt den Frequenzbereich in 79 Kanäle und springt 1600-mal pro Sekunde zwischen diesen hin und her. Diese Verfahren heißt Frequency-Hopping, das Verfahren senkt die Wahrscheinlichkeit, dass sich Bluetooth-Verbindungen gegenseitig behindern.

ADSL und SDSL

ADSL steht für Asynchronous Digital Subscriber Line und bedeutet das es einen Unterschied zwischen Download- und Upload-Rate der Leitung gibt.
SDSL ist die Abkürzung für Synchronous Digital Subscriber Line und bedeutet, dass die Down- und Upload-Rate der Leitung gleich groß sind.
Sie sind die bekanntesten Formen von DSL.

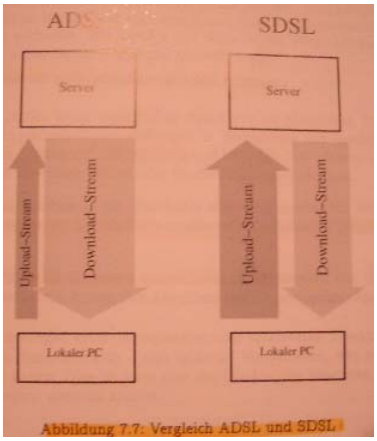


Abbildung 7.7: Vergleich ADSL und SDSL

8.Protokolle

Das Internet besteht heute grob betrachtet aus drei Ebenen:

Backbones: Sie repräsentieren das „Rückrad“ des Internets
Midlevel Networks: diese stellen die von ISPs an deren Endkunden zur Verfügung gestellten Netzbereiche dar.
LANs: die an Midlevel Networks angeschlossenen Netzwerke

Internet Protocol (IP)

Die Kommunikation zwischen den einzelnen Hosts erfolgt mit Hilfe des Internet Protocol (IP), dieses arbeitet im OSI Reference Layer 3 (Networks Layer).
IP liefert nun die Infrastruktur, um Datagrams von einer Quelle zu einem Ziel zu befördern, unabhängig davon, ob sich beide im gleichen Netzwerk befinden. Der Transport Layer zerlegt die zu übermittelnden Datenströme in mehrere kleine Datagrams – im deutschen auch als Pakete bezeichnet –, die danach übermittelt und am Ziel durch den Networks Layer wieder in richtiger Reihenfolge zusammengesetzt werden.
Ein solches Datagram darf bis zu 64 Kilobyte gross sein, nimmt allerdings im Durchschnitt um die 1500 Bytes ein.

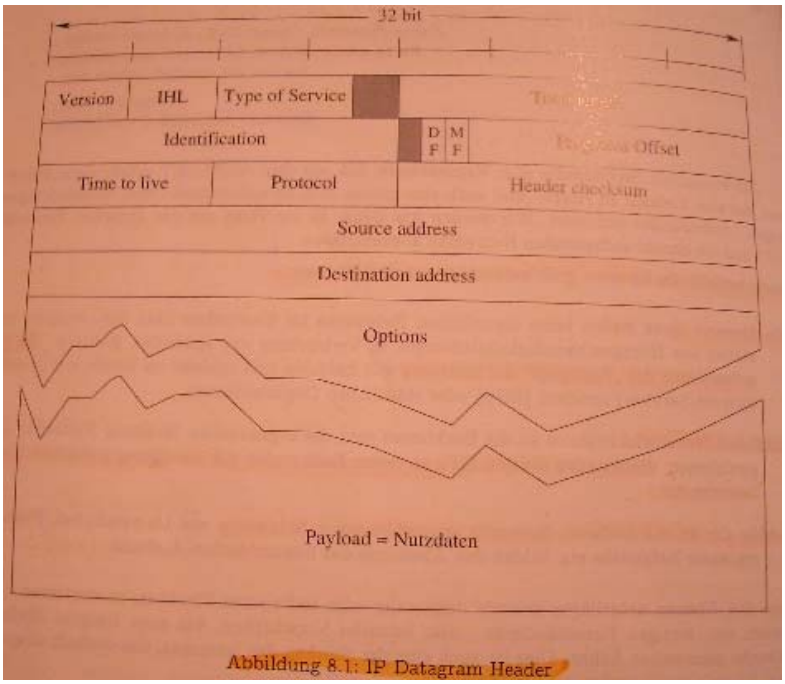


Abbildung 8.1: IP Datagram Header

Version: IP-Protokoll Versionsnummer
IHL: tatsächliche Headerlänge
Type Of Service: Art der Übertragung
Total Length: Gesamtlänge
Identification: Alle zu einem Datenstrom gehörenden Datagrams besitzen dieselbe Identifikationsnummer
DF: „Don’t Fragment“ weist Router an, das Datagram nicht in weitere kleinere Datagrams zu zerlegen.

MF: „More Fragements“ Es zeigt dem Empfänger-Host an, wann die Fragmentserie endet und er mit dem Zusammenfügen zum ursprünglichen Datagram beginnen kann.
Fragment Offset: gibt die Stelle in einer Serie von zu einem Datenstrom gehörenden Datagrams an.
Time to Live: dieser Wert (0 bis 255) wird bei jedem HOP um 1 dekrementiert. Erreicht der Wert 0, wird das Datagram vom betroffenen Router fallen gelassen.
Protocol: gibt an, welchem Prozess im Transport Layer der Datenstrom übergeben werden soll
Header checksum: Checksum zur Überprüfung der Korrektheit des Headers
Source address: Quelladresse des Datagrams
Destination address: Zieladresse des Datagrams
Options: Platz für Optionale Flags zur Erweiterung

Eine Firewall besitzt eine Tabelle mit Regeln, die exakt festlegt, welche Art von IP-Paketen von einem Netzwerk-Interface zum anderen wechseln dürfen.
Jeder Router kann selbständig über die bei ihm eingehenden Datagrams „regieren“, anhand seiner Routingtabelle kann er optimal entscheiden, wie er das Paket unter den gerade bestehenden Bedingungen weiterleitet.

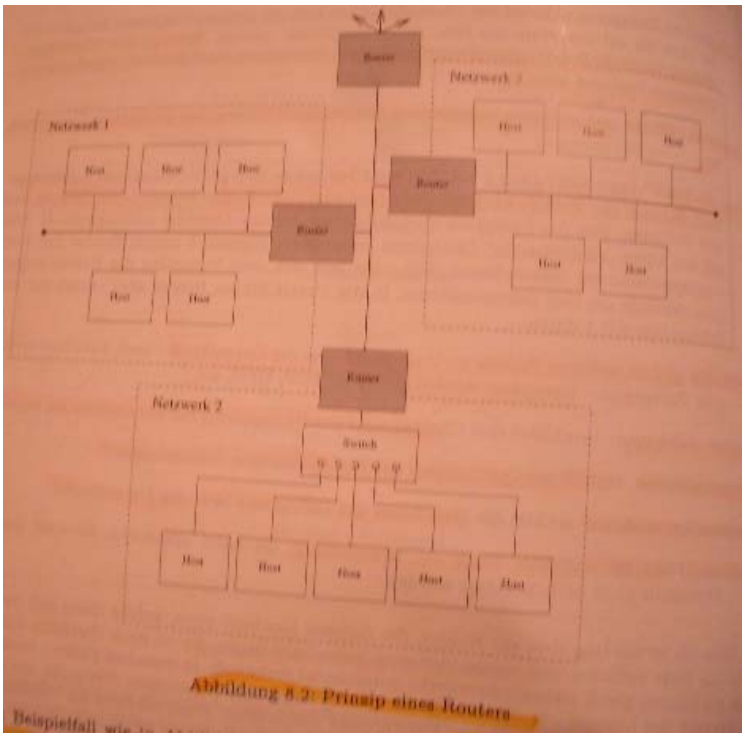
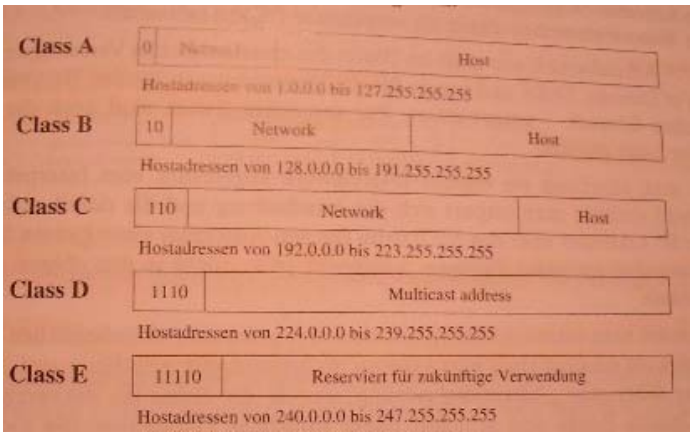


Abbildung 8.2: Prinzip eines Routers

Hosts aus „Netzwerk 1“ können beliebig Pakete zu anderen Hosts im lokalen Netz (Netzwerk 1), Netzwerk 2 oder Netzwerk 3 senden.
Grundsätzlich besitzt jeder Host, ebenso wie jeder Router, mindestens eine eigene, eindeutige IP Adresse.

IP Adressen werden in folgende Größenkategorien unterteilt:



Klasse A: 126 Netzwerke, 16 Millionen Hosts
Klasse B: 16.382 Netzwerke, 65.536 Hosts
Klasse C: 2 Millionen Netzwerke, 254 Hosts

Das Internet Protocol erlaubt das Adressieren von mehreren Hosts gleichzeitig, diesen Vorgang nennt man Multicasting.

Es gibt sogenannte private Adressen die rein für den internen Gebrauch gedacht sind:

- 127.*.*.*
- 10.*.*.*
- 192.168.*.*
- 172.16.0.0 bis 172.31.255.255

Für den Anschluss eines ganzen IP-Netzwerks an das Internet bedarf es nur einer einzigen „externen“ (d.h., nicht in den obigen Adressbereichen liegenden) Adresse. Dazu verwendet man intern Adressen aus den drei privaten Adressbereichen und ein spezielles Gateway, das als einziger Host eine externe IP-Adresse mit Anschluss zum Internet besitzt.
Ein weiterer wichtiger Aspekt des Internet Protokolls ist das Subnetting. Es erlaubt die interne Aufteilung eines Netzwerks in mehrere Subnetze.
Jedem Host wird eine Subnet Mask spezifisch zugewiesen, die eigentliche Subnetz-Nummer errechnet man durch AND-Verknüpfung der Host-Adresse mit der Subnet-Mask.

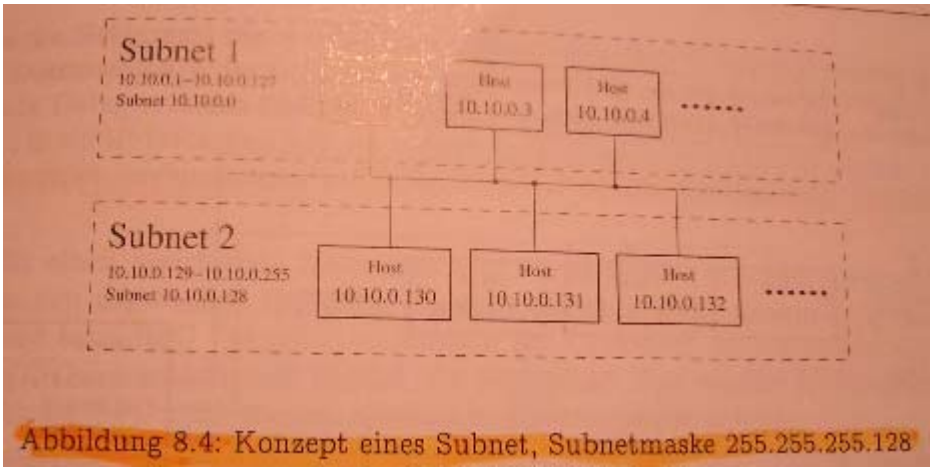


Abbildung 8.4: Konzept eines Subnet, Subnetmaske 255.255.255.128

Für beide Subnets gilt die Subnet Maske 255.255.255.128. Führt man eine AND-Verknüpfung einer Adresse durch, erhält man so die beiden Subnetz-Nummern 10.10.0.0 und 10.10.0.128

10.10.0.2 AND 255.255.255.128 = 10.10.0.0
10.10.0.129 AND 255.255.255.128 = 10.10.0.128

Möchte man zwei weitere Subnetze unterbringen müsste nur die Subnetmaske der Rechner auf 255.255.255.192 geändert werden.

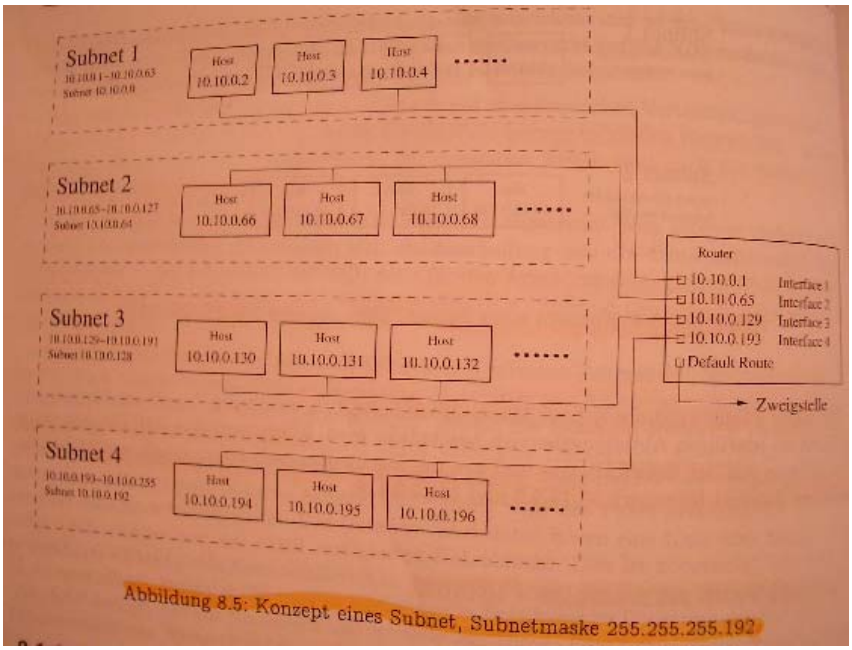


Abbildung 8.5: Konzept eines Subnet, Subnetmaske 255.255.255.192

Der Abgebildete Router besitzt für jeden der vier Netzsegmente eine eigene IP-Adresse. Dieser muss die Pakete der vier – physikalisch voneinander getrennten – Netzsegmente korrekt routen und braucht sich dafür nur die folgenden vier Routing-Einträge merken:

10.10.0.0 → Interface 1
10.10.0.64 → Interface 2
10.10.0.128 → Interface 3
10.10.0.192 → Interface 4

TCP

Im Internet rufen User Daten von Servern ab. Dieser Abruf erfolgt in zwei Schritten: zuerst sendet der Computer des Benutzers (Client) eine Anfrage an den Server. Daraufhin antwortet der Server, indem er die genannte Datei oder eine Fehlermeldung zurücksendet. Diese Form der Kommunikation entspricht einer Client-Server-Architektur.

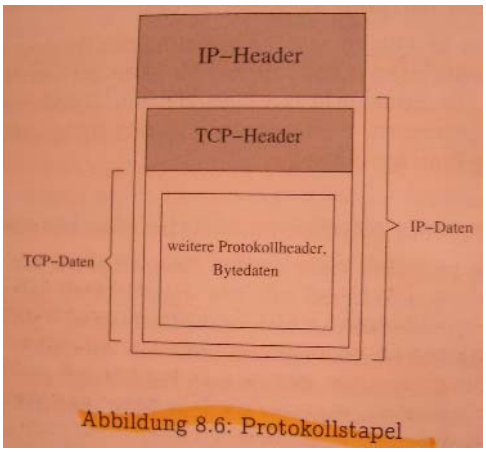


Abbildung 8.6: Protokollstapel

Ein solches Vorgehen ist rein auf IP-Basis nicht einfach implementieren. Das TCP (Transmission Control Protocol) ist ein auf IP aufbauendes, verbindungsorientiertes Protokoll, es ermöglich eine Punkt-zu-Punkt Verbindung.
Die Quell- und Zielpunkte, von denen TCP-Verbindungen ausgehen, nennt man Ports. Diese sind wie Hausnummern durchnummeriert. Die Ports von 1 bis 1024 sind sogenannte well known ports, die für vordefinierte Programmzwecke reserviert sind.

Port	Zweck
21	File Transfer Protocol
22	Secure Shell
23	Telnet
25	Simple Mail Transfer Protocol
110	POP3

Tabelle 8.1: Auszug aus der Portliste, die durch die IANA definiert wurde

UDP

Das Gegenstück zu TCP bildet das User Datagram Protocol (UDP). Manchmal sind initialisierte Verbindungen nicht unbedingt wünschenswert, da es mitunter reicht, einen einfachen Request an einen Server zu senden und dann eine einfache Antwort abzuwarten. Das reduziert den für eine Anfrage notwendigen Nachrichtenaufwand und –umfang und entlastet dadurch gleichzeitig das Netzwerk.

IPv6

Ein immanentes technisches Designproblem das IPv4 anhaftet, ist die Tatsache, dass über die Zeit die Adressen ausgehen.

Die von IP Version 6 gesetzten Ziele sind:

- Unterstützung von Milliarden von Hosts
- Reduktion des Umfangs der Routingtabellen
- Vereinfachung des Protokolls in Hinsicht auf Router
- Sicherheit im Hinblick auf Authentifizierung und Datenschutz
- Stärkere Betonung des Type of Service
- Bezüglich der Funktion des Multicasting soll man den Umfang der angesprochenen Hosts einschränken können

- Protokollerweiterungen sollen auch nach der fertigen Spezifikation noch problemlos möglich sein
- Die Koexistenz des neuen und vorhergehender alter Protokolle soll gewährleistet sein

IPv6 Adressen haben eine Länge von 16 Bit:

A000:0000:0000:0000:0A14:4512:ABCD:CDEF

Um die schreibweise etwas zu vereinfachen, wurden die folgenden drei Optimierungsn eingeführt:

- Führende Nullen können weggelassen werden
- Eine oder mehrere Gruppen von Nullen können durch eine leergelassene Gruppenstelle zwischen zwei Doppelpunkten angeschrieben werden
- IPv4-Adressen kann man im Format 192.168.0.1 anschreiben

Somit lässt sich die obige Adresse etwas als:
A000::A14:4512:ABCD:CDEF notieren.

Die Vereinfachung des Protokoll Header in IPv6 ermöglicht eine schnellere Bearbeitung durch Router.
Weiter relevante Punkte sind:

- Eine verbesserte Unterstützung von Optionen
- Die zuvor obligatorischen Flags sind nun optional
- Router dürfen Optionsfelder, die sie nicht betreffen, ignorieren

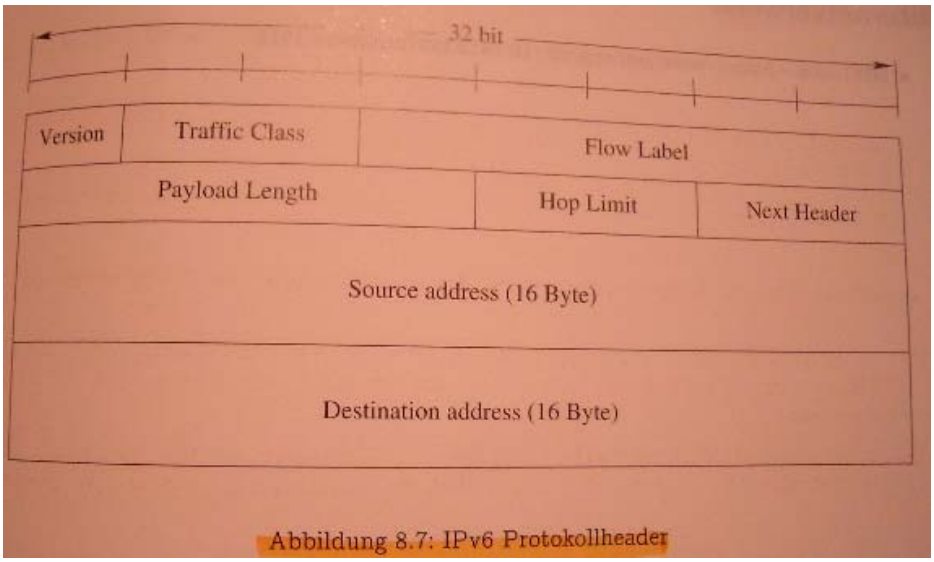


Abbildung 8.7: IPv6 Protokollheader

Version: bei IPv6 den Wert 6
Traffic Class: dieses Feld unterscheidet herkömmliche Pakete von Datagrams mit Echzeitinhalten
Flow Labels: ermöglicht eine Form der „Bandbreitenreservierung“
Payload length: Länge des angefügten Datenblocks hinter dem Header

Next Header: das zuvor erwähnte Feld, das angibt welcher Erweiterungsheader dem Primärheader folgt
Hop Limit: früher als „Time To Live“ bezeichnet
Source address: Quelladresse
Destination address: Zieladresse

9.Betriebssysteme und Systemsoftware

Übersicht

Bei der Entwicklung von Betriebssystemen gibt es fünf grundsätzliche Bereiche:

- Prozesse
- Speicherverwaltung
- Ablaufplanung und Ressourcenverwaltung
- Datenschutz und Datensicherheit
- Systemstruktur

Ziele und Funktionen von Betriebssystemen

Ein Betriebssystem ist ein Programmsystem, das die Ausführung von Anwenderprogrammen unterstützte und zugleich eine Software-Schnittstelle zwischen der Applikationssoftware und der Computerhardware darstellt. Die Leistungsmerkmale eines Betriebssystems sind:

Mensch-Maschine-Schnittstelle: Benutzerfreundlichkeit eines Betriebssystems
Effizienz: effiziente Nutzung der vorhandenen Computer-System-Ressourcen
Aspekte der Weiterentwicklung von Betriebssystemen: Ein Betriebssystem sollte Modular aufgebaut sein, so dass neue Systemfunktionen relativ problemlos integriert werden können. Dabei sind die Wartbarkeit, die Kompatibilität, sowie die Portierbarkeit von besonderer Bedeutung.

Betriebssystemschnittstelle zwischen Benutzer und Computersystem

Für die Darstellung der Funktionalität eines Betriebssystems innerhalb eines Computersystems eignet sich zunächst ein relativ einfaches Layer-Modell:

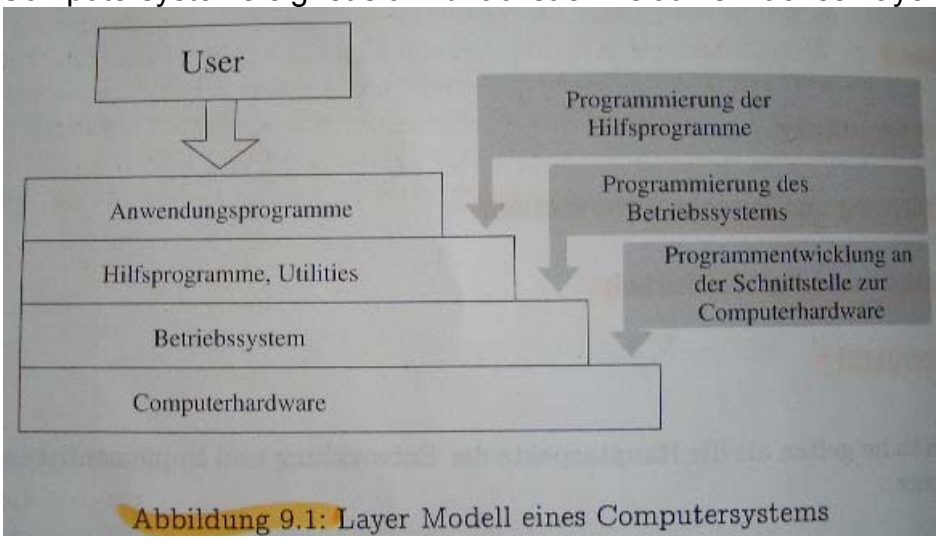


Abbildung 9.1: Layer Modell eines Computersystems

Ein Betriebssystem bietet üblicherweise folgende Dienste an:

- Prozessmanagement: Anwenderprogramme müssen gestartet und wieder beendet werden
- Interprozess-Kommunikation: Prozesse müssen miteinander kommunizieren
- Speichermanagement: Der Hauptspeicher muss fair unter den laufenden Prozessen aufgeteilt werden
- Zugriff auf E-/A-Geräte: Das Betriebssystem stellt eine einheitliche Schnittstelle zur Verfügung, so dass mit einfachen Lese- und Schreiboperationen auf diese Geräte zugegriffen werden kann
- Zugriff auf Dateien: Schutzmechanismen für die Kontrolle des Zugriffs auf Dateien müssen zur Verfügung gestellt werden
- Fehlerbehandlung: Das Betriebssystem muss so reagieren, dass der Fehlerzustand mit einer möglichst geringen Auswirkung auf die Anwenderprogramme behoben werden kann
- Accounting: Führung von Nutzungsstatistiken für die verschiedenen Ressourcen im Computersystem sowie die Überwachung von Leistungsdaten

Betriebssystemaufrufe

System Calls stellen das Interface zwischen laufenden Programmen und dem Betriebssystem dar. Bei System Calls wird ein Software-Interrupt Befehl (Trap) verwendet. Ein solcher Trap hat in etwa die gleiche Wirkung wie ein Interrupt, d.h., die normale Exekution des Prozessors wird unterbrochen.
Setzt nun ein User oder ein Programmprozess einen solchen System Call ab, so wird in der beschriebenen Weise eine eindeutig zugeordnete Prozedur vom Betriebssystem gestartet. Diese Schnittstelle wird Application Programmers Interface (API) genannt.

Beispiel:

Ein Teil des Betriebssystems ist das sogenannte Filesystem. In diesem werden nun System Calls zur Verfügung gestellt, welche die Manipulation von Files erlauben. Will z.B. ein Programmporzess ein File bearbeiten, so muss er es zuerst mittels F_OPEN(filename, attributes) vom Betriebssystem anfordern, um die Datei zu öffnen. Mit den Attributen wird die Zugriffsart angegeben, es gibt lesende, schreibende und exekutierende Zugriffe.
Das Betriebssystem soll nun überprüfen, ob der anfordernde Programmprozess die benötigten Zugriffsrechte besitzt. Wenn dies der Fall ist, liefert der System Call eine das benötigte File-Objekt repräsentierende File-ID, andernfalls eine Fehlermeldung zurück. Alle weiteren gleichartigen System Calls verwenden nicht mehr den File-Namen sonder die File-ID als Parameter.
F_READ(file-ID, element) dient dazu, ein Element von einem File zu lesen, mittels F_WRITE(file-ID, element) kann ein Element auf ein File geschrieben werden.

Ähnliche System Calls finden wir unter UNIX für Dateizugriffe. Die wichtigsten System Calls für den Umgang mit dem Dateisystem sind:

- open: Dateien öffnen
- creat: Dateien anlegen
- close: schließen

- read: Lesen
- write: Schreiben
- unlink: Entfernen und ggf. löschen von Dateien

Betriebssystem-Struktur

In der Praxis eröffnen sich zwei Möglichkeiten, die Gliederung in Schichten vorzunehmen: die konsistente und die quasikonsistente Schichtung.

Konsistente Schichtung:
Konsistente Schichtung bedeutet, dass jede Schicht n nur Funktionen und Objekte der darunter liegenden Schicht(n – 1) benutzt. Daher gilt für jede Schicht:

- es sind nur Funktionen der darunter liegenden Schicht benutzbar
- ihre Objekte sind nur für die unmittelbar darüber liegenden Schicht (n + 1) sichtbar

Quasikonsistente Schichtung:
Bei einer quasikonsistenten Schichtung können die Dienste aller darunter liegenden Schichten benutzt werden. In den meisten Fällen erweist sich die quasikonsistente Schichtung als effektiver.

Schichtenmodell:

Zwischen Anwendungsprogramm und Hardware liegt das Betriebssystem. Die Schnittstelle zu den Anwenderprogrammen heißt Application Programmers Interface (API), zur Seite der Hardware besteht das Betriebssystem aus Hardware-Treibern, auch Hardware Abstraction Layer (HAL) oder kurz Hardware Layer (HL) genannt.

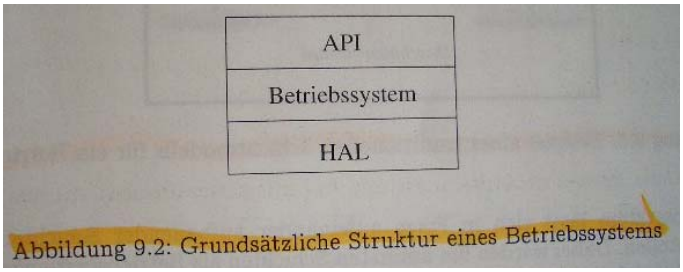


Abbildung 9.2: Grundsätzliche Struktur eines Betriebssystems

Ausgehend vom Aufruf einer Interprozesskommunikation gehörenden System Calls (etwa S_P) pflanzt sich die Exekution durch die nicht zuständigen Layer hindurch fort, bis die zuständige Schicht erreicht ist. Dort erfolgt die eigentliche Bearbeitung des System Calls.

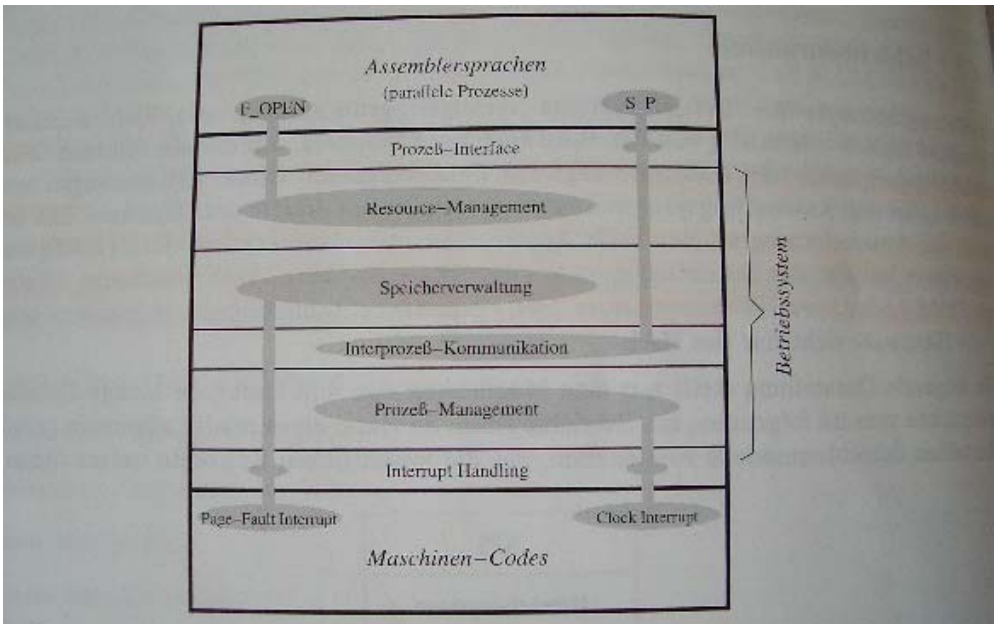


Abbildung 9.3: Beispiel eines traditionellen Schichtmodells für ein Betriebssystem

10.Prozesse

Im Laufe der Zeit lässt sich eine sehr deutlich Zunahme der „gleichzeitig“ zu erledigenden Tätigkeiten eines Computers feststellen. Bis in die Ära des Batch-Betriebs arbeitete ein Rechner noch zu jedem Zeitpunkt an genau einer Aufgabe.

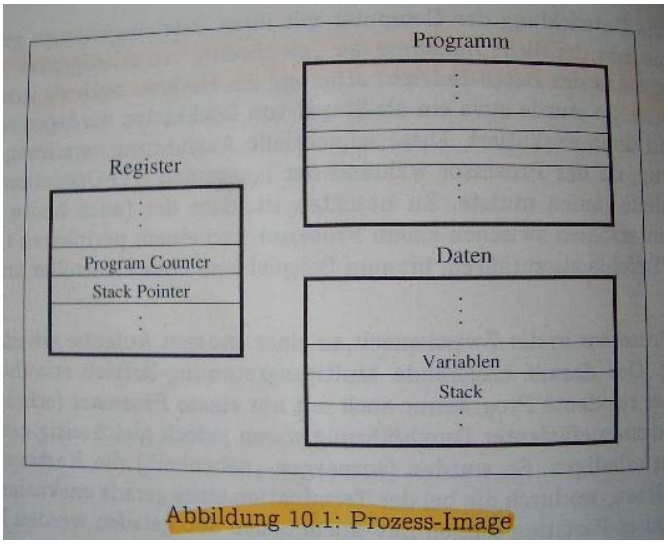


Abbildung 10.1: Prozess-Image

Parallelität

Es ist eine altbekannte Tatsache, dass die Lösung eines komplexen Problems durch mehrere, voneinander relativ unabhängige Teile meist wesentlich einfacher als eine integrierte Gesamtlösung ist.
An sich kann ja jeder Prozessor nur ein Programm exekutieren, auf einem Computer mit n Prozessoren wären daher eigentlich nur n (echt) gleichzeitige Prozesse möglich. Es ist aber eine der ganz zentralen Aufgaben eines Betriebssystems, ein virtuelles „10000000-Prozessorsystem“ zu realisieren.

Dabei treten jedoch auch Probleme wie z.B. Deadlocks auf.

Prozesshierarchien

Jede Programmausführung erzeugt eine Menge von Prozessen, die wiederum Nachfolgeprozesse auslösen können. Der Prozess der einen oder mehrere Prozesse auslöst wird als Parent Prozess bezeichnet, seine Nachfolgeprozesse selbst als Child Prozesse.

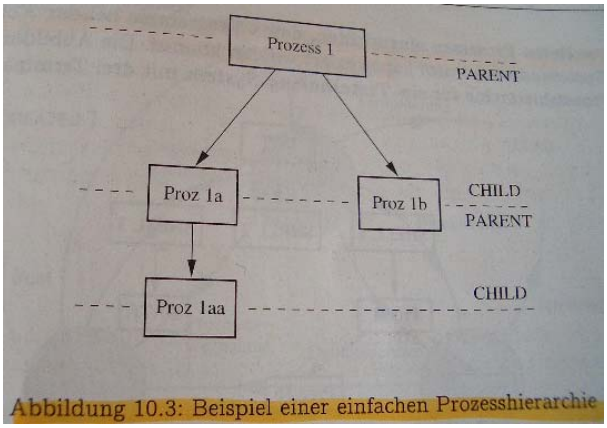


Abbildung 10.3: Beispiel einer einfachen Prozesshierarchie

Für jeden Prozess kann eindeutig bestimmt werden, wer sein unmittelbarer Vorgänger (Parent Prozess) ist und welche Knoten seine unmittelbaren Nachfolger (Child Prozesse) sind. Eine logisch zusammengehörige Menge von Prozessen wird gern als Job bezeichnet.
Folgenden zwei Eigenschaften liegen der Parent-Child Beziehungen zugrunde:

Einheitliche Kontrollmöglichkeit eine Jobs (Unterbaumes): es wäre denkbar, die Terminierung oder das Anhalten des gesamten Jobs schlicht durch eine geeignete Signalisierung des initialen Parent Prozesses zu erwirken.

Vererbung der Prozess-Umgebung: Einem Prozess ist eine mehr oder minder umfangreiche Umgebung zugeordnet, die als Menge von Ressourcen angesehen werden kann. Um nun die Child Generierung möglichst einfach zu halten, bietet sich natürlich an, die bereits beim Parent verwendete Umgebung (Speicherbereiche, angeforderte Objekte, …) auch beim Child einzusetzen.
Das mit einem gehörigen Aufwand verbundene Kompieren der Prozess-Umgebung kann oft durch die Verwendung spezieller System Calls oder durch den Einsatz von Threads vermieden werden.
Im Zuge der Startup-Sequenz eines Computers wird ein meist mit Init oder Root bezeichneter Prozess erzeugt. Dieser hat die Aufgabe, eine Reihe von weiteren Prozessen einzurichten deren Programme bei der Konfiguration des Systems spezifiziert werden können.

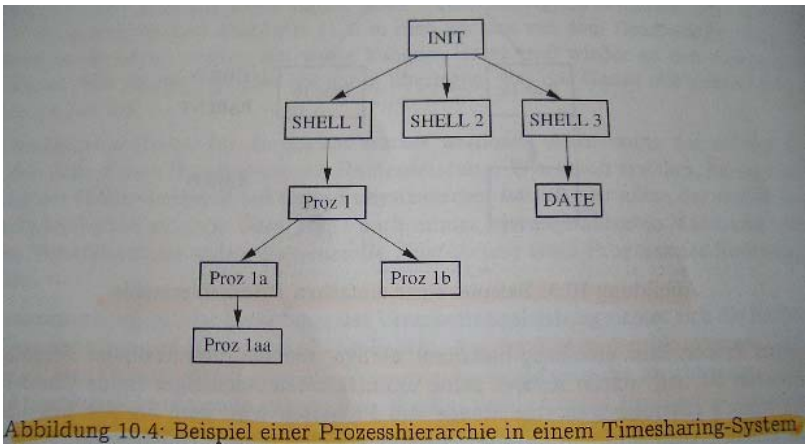


Abbildung 10.4: Beispiel einer Prozesshierarchie in einem Timesharing-System

Prozesszustände

Jeder Prozess befindet sich zu jedem Zeitpunkt in einem der folgenden Prozesszustände: CREATED, RUNNING, READY, BLOCKED, SUSPENDED, DEAD.

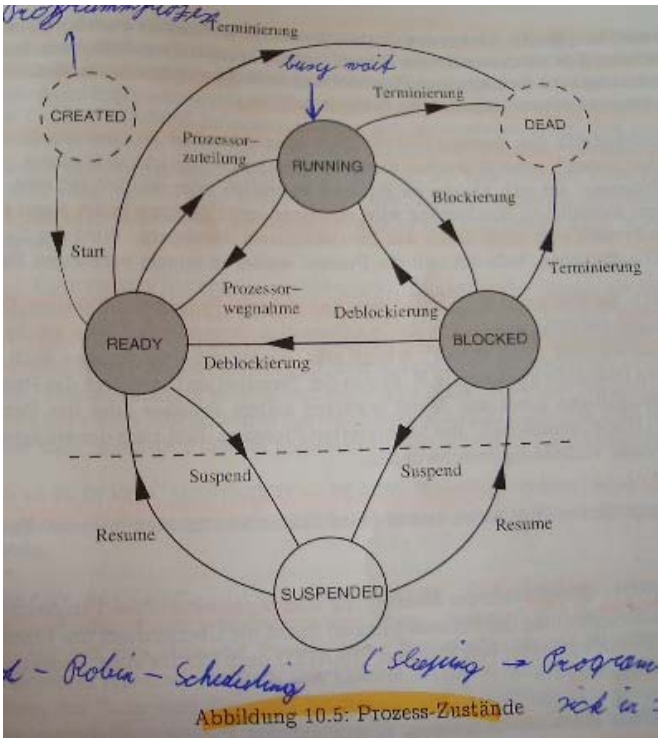


Abbildung 10.5: Prozess-Zustände

CREATED: Das Erzeugen eines Prozesses bewirkt das Anlegen einer entsprechenden Datenstruktur.
READY: Prozess ist bereit zur Ausführung
RUNNING: Prozess ist im Besitz des Prozessors
BLOCKED: Der Zustand Blocked wird eingenommen, wenn ein Prozess bei seiner Exekution einen Punkt erreicht, an dem er auf den Eintritt eines externen Ereignisses warten muss. Dies kann z.B. die Eingabe eines Zeichens vom Terminal oder die Termination eines anderen Prozesses sein. Wichtig ist, dass ein Prozess nur durch

eine von sich selbst ausgehende Aktion den Zustandswechsel nach Blocked verursachen kann.
Das Verlassem vpm Blocked ist nur durch ein „von aussen“ kommendes Ereignis möglich.
SUSPENDED: entspricht einer Blockierung in die der Prozess „von aussen“ versetzt werden kann. Ein derartiger Übergang wird in der Regel von einem anderen Prozess oder auch durch das Betriebssystem verursacht.
DEAD: Der Prozess hat sich vom aktiven Dasein gelöst. Die dafür notwendige Terminierung kann ein Prozess selbst initiiert haben oder „von aussen“ erfolgt sein.

Zustandsübergänge:

CREATED → READY: Start, geburt eines Prozesses
READY → RUNNING: Prozessorzuteilung
RUNNING →READY: Prozessorwegnahme
RUNNING → BLOCKED: Blockierung. Der gerade laufende Prozess ist auf das Eintreten eines Ereignisses angewiesen und wird bis zu diesem Eintritt blockiert.
BLOCKED → READY, BLOCKED → RUNNING: Deblockierung. Sobald die zuvor angeforderte Ressource verfügbar ist oder eine entsprechende Zeitbedingung zum Abbruch des Wartens führt wird der Prozess wieder aktiviert.
BLOCKED → SUSPENDED, READY → SUSPENDED: Im Gegensatz zu Blocked wird Suspended explizit angeordnet.
SUSPENDED → BLOCKED, SUSPENDED → READY: Resume. Der Suspendierungszustand wird wieder aufgehoben.
RUNNING, READY, BLOCKED → DEAD: Terminierung. Das Beenden eines Prozesses geschieht in der Regel durch Eigeninitiative.
SUSPENDED → DEAD: Terminierung. Auch aus dem Suspended Zustand ist eine Terminierung durchaus möglich. Sie weist allerdings stets einen gewaltsamen Charakter auf.

Die Existenz eines Prozesses kann nun entweder durch die eigene Terminierung oder aber durch den Abbruch „von aussen“ beendet werden. Dieses Killen eines Prozesses ist nicht ohne Probleme, da das Opfer von der Massnahme asynchron „überrascht“ wird, es gibt gewisse Abschnitte im Leben eines Prozesses, in denen ein derartiger Abbruch problematisch wäre.
Entsprechende Lösungsmöglichkeiten basieren meist auf sogenannten Atomic Actions. Bei einer solchen – auch als unteilbare Operation bezeichneten – Folge von Instruktionen wird durch geeignete Massnahmen sichergestellt, dass ihre Ausführung entweder zur Gänze oder gar nicht erfolgt

Übliche Betriebssysteme legen für jeden Prozess einen sogenannten Prozessdeskriptor an, in dem für die Verwaltung des Prozesses notwendigen Daten abgelegt werden. Ein solcher Deskriptor besteht aus einem Teil zur Identifikation des Prozesses, einem Teil zur Speicherung von Zustandsinformationen und einem Teil zur Speicherung von Kontrollinformationen.

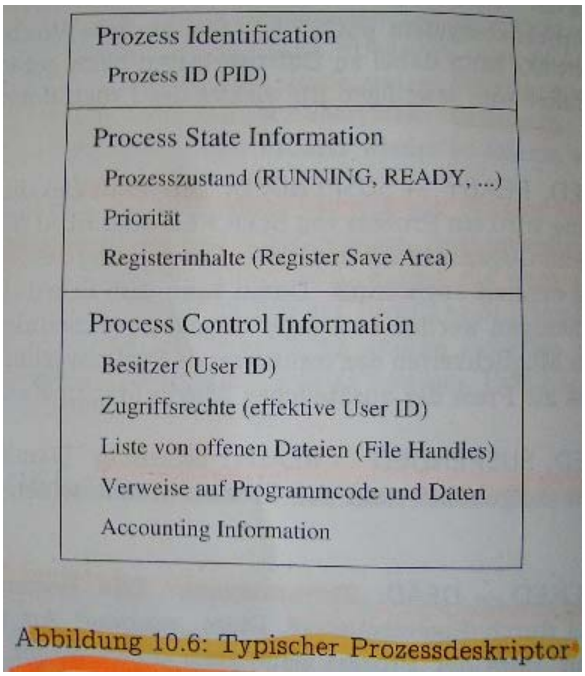


Abbildung 10.6: Typischer Prozessdeskriptor

Wie erreichen wir es, dass ein Prozess, nachdem ihm der Scheduler einaml den Prozessor entzogen hat, wieder fortsetzen kann, als ob nichts gewesen wäre? Dazu ist es notwendig, die Inhalte aller (relevanten) Register des Prozessors (den sogenannten Context) in der Register Save Area im Prozessdeskriptor abzuspeichern.
Da auch der Program Counter ein Teil des Contextes ist, setzt der Prozessor nach diesem Context Switch die Exekution mit dem „nächsten“ Befehl des neuen Prozesses, also dort, wo letzterer unterbrochen wurde fort.
Als wesentliches Charakteristikum von Betriebssystemen gilt in diesem Zusammenhang die so genannte Context Switch TIME (CST). Neben der reinen Context-Save und –Restore –Tätigkeit wird ausserdem auch die Zeit für die Findung einer Scheduling-Entscheidung eingerechnet.

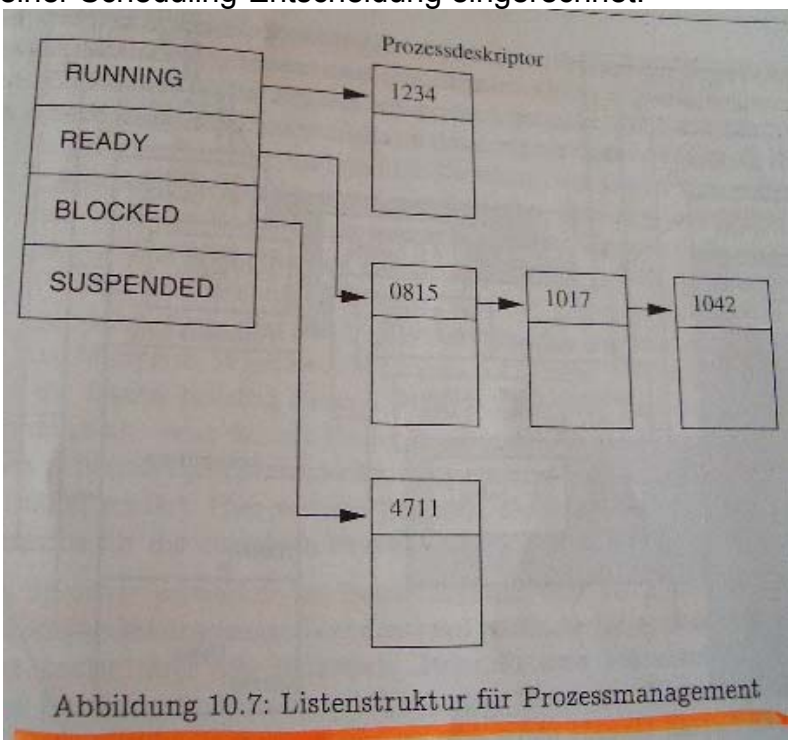


Abbildung 10.7: Listenstruktur für Prozessmanagement

Welche System Calls benötigt ein Betriebssystem nun für das Prozess-Management?
Durch den Aufruf P_CREATE(program, parameter, attributes) in einem Prozess wird ein Child Prozess erzeugt. Mittels P_WAIT() kann ein Parent-Prozess auf die Termination eines Childs warten. P_EXIT() hast die Aufgabe das Betriebssystem von der Terminierung eines Prozesses zu unterrichten. Dies ist natürlich jenes Ereignis, das ein P_WAIT des Parent-Prozesses beendet. Für den Fall, dass der Parent-Prozess (noch) nicht per P_WAIT wartet, verbleibt der Child-Prozess im Zustand DEAD, bis der Parent per P_WAIT – in diesem Fall nicht blockierend – den Resultatwert übernimmt (z.B. so auch in UNIX implementiert, wo dieser Zustand als ZOMBIE anstatt von DEAD bezeichnet wird).

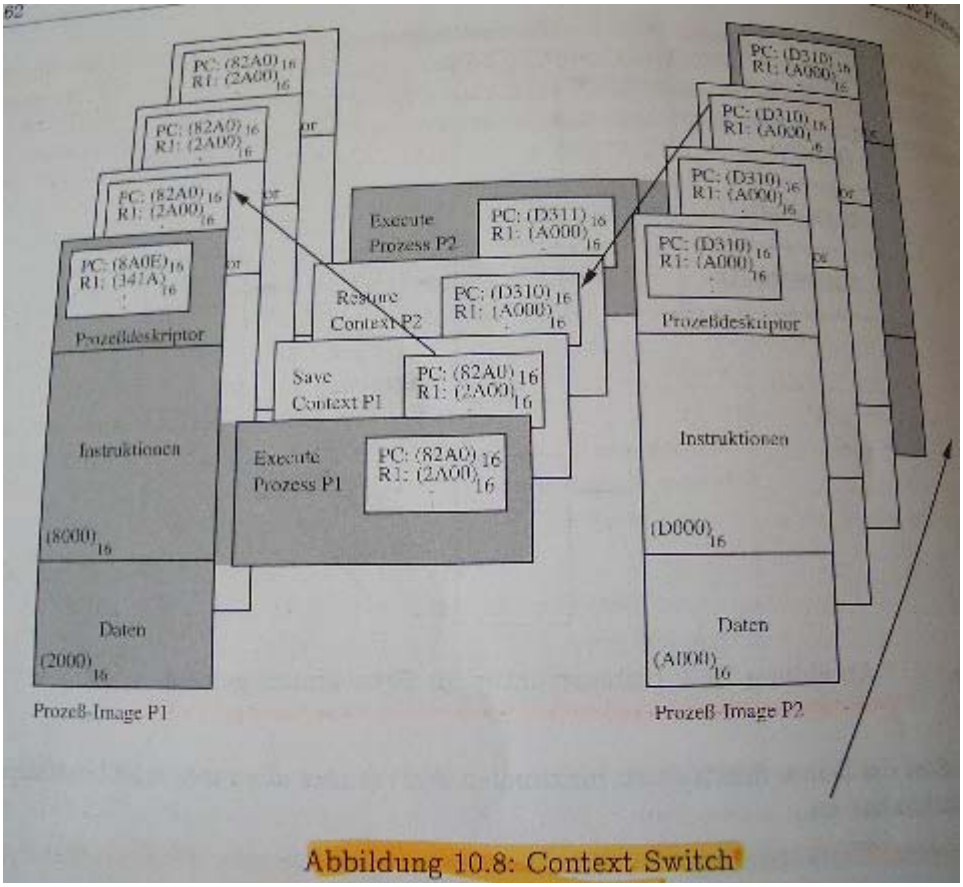


Abbildung 10.8: Context Switch

Mittels P_SLEEP(event) kann ein Prozess (sich selbst) von RUNNING nach BLOCKED überführen. P_SIGNAL(process-ID, event) schafft die Möglichkeit, dem durch process-ID identifizierenten Prozess den Eintritt des Ereignisses event zu signalisieren.

Threads

Wenn von einem Prozess die Rede war, haben wir immer zwei grundlegende Konzepte unter diesem Begriff zusammengefasst:

Ressourcenverwaltung: Jedem Prozess sind bestimmte Ressourcen zugeordnet, auf die der Prozess über Betriebssystemroutinen zugreifen kann.

Neben dem Speicher verwaltet das Betriebssystem auch noch andere Ressourcen, wie z.B. Zeiger auf die geöffnete Daten oder eine Liste von Sockets zum Datentransfer über das Netzwerk.

Programmausführung: Unter dem Context versteht man den Inhalt der Prozessorregister während der Ausführung. Für jeden Prozess muss man sich auch merken, welche Prozeduren ein Prozess aufgerufen hat, um zum aktuellen Punkt in der Ausführung zu kommen. Ein weiterer Punkt sind lokale Variablen, die für jede Prozedur neu angelegt werden.

Wir wollen uns nun einen Prozess vorstellen, dem eine einziger Satz an Ressourcen (und vor allem ein einziger privater Addressraum) zugeordnet ist, in dem aber mehrere Programmausführungen gleichzeitig ablaufen.
Für die Menge von Programmausführungen unter dem Mantel eines Prozesses hat sich der Begriff Lightweight Processes etabliert, für die auch noch der Ausdruck Thread Verbreitung gefunden hat. Die nunmehr aus nichts anderem als jeweils einem Context und Stack bestehenden Threads „laufen“ quasi parallel oder tatsächlich parallel auf mehreren Prozessoren verteilt. Der Context und der Stack werden übrigens auch zusammengefasst als thread-spezifische Daten bezeichnet.

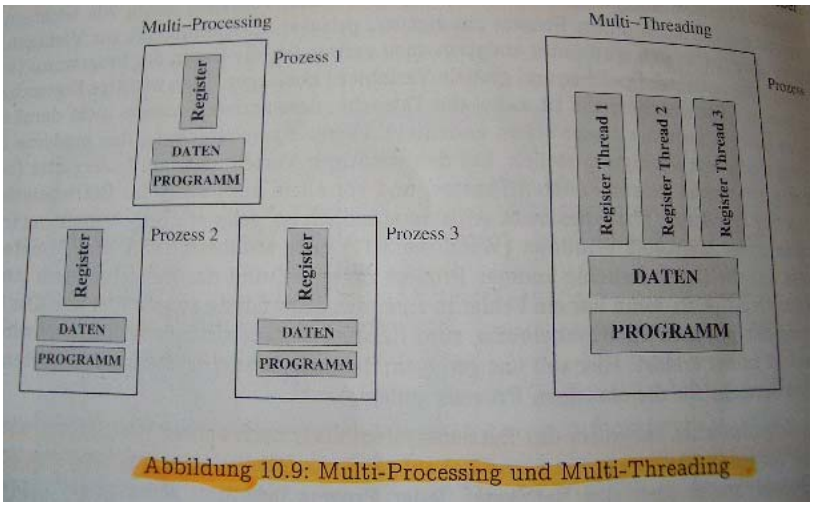


Abbildung 10.9: Multi-Processing und Multi-Threading

Der Grund für die Verwendung von Threads ist eine mögliche Leistungssteigerung des Gesamt Systems. Es bedeutet nämlich für das Betriebssystem einen recht grossen Speicher- und Rechenaufwand, voneinander getrennte Prozesse zu erstellen und zu verwalten.

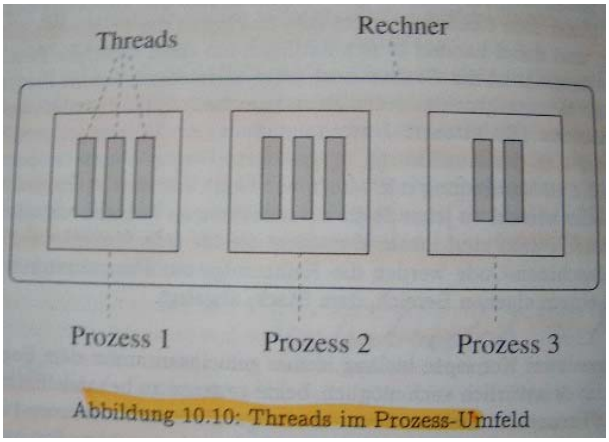


Abbildung 10.10: Threads im Prozess-Umfeld

Wird stattdessen ein zusätzlicher Thread angelegt, der diese Aufgabe übernimmt, muss das Betriebssystem nur einen extra Stack anlegen und Platz schaffen, wo der entsprechende Context gespeichert werden kann.

Die Aufteilung der Programmausführung auf mehrere Threads ist normalerweise nicht Sache des Betriebssystems und muss explizit vom Programmierer vorgenommen werden. Während die Unterteilung in Prozesse noch recht einfach zu bewerkstelligen ist, indem man grob jedes in sich abgeschlossene Programm einem Prozess zuordnet, ist bei den Threads eines Prozesses eine wesentlich engere Beziehung und Abhängigkeit untereinander gegeben.
Das Konzept der Threads hat demnach ganz ähnliche Eigenschaften wie Unterprogramme bzw. Subroutinen einer prozeduralen Programmiersprache. Sie verfügen einerseits über den Zugriff auf globale Variablen und andererseits sind lokale Variablen auf den jeweiligen Thread beschränkt.
Die genauen Bedingungen und Regeln für die Koordination der Threads gibt das sogenannte Thread-Management vor.

Eigenschaften von Threads im Vergleich zu herkömmlichen Prozessen:

- Ein Thread besteht lediglich aus einem Registersatz des Prozessors und den thread-spezifischen Daten.
- Alle Threads haben vollständigen Zugriff auf Programm- und Datenbereiche
- Feinere Realisierung der parallelen Abarbeitung durch die zusätzliche Thread-Ebene
- Die Neugenerierung eines Threads ist effizient
- Die Kommunikation zwischen Threads kann wegen des Sharings leicht über globale Variablen erfolgen.
- Thread-Funktionen werden über ein eigenes Thread-Interface angeboten
- Es gibt keine klare Parent-Child beziehung zwischen Threads so, wie sie zwischen Prozessen existieren

Es treten bei Threads aber gewisse Probleme auf die bislang bei Prozessen verhindert worden sind. Zunächst gibt es wegen des Sharings des gleichen Adressraums praktisch keine Schutzmechanismen zwischen Threads, weiter treten Schwierigkeiten bei der Verwendung von Bibliotheken auf.

Der Zugriff zur Thread-Funktionalität, in Form sogenannter Thread-Packages realisiert, stützt sich grundsätzlich auf zwei Ansätze der Thread-Integration in ein Betriebssystem.

Thread-Funktionen ausserhalb des Betriebssystems: Die Thread-Funktionalität wird dabei als eine Sammlung von diversen Funktionen bzw. Unterprogrammen realisiert, die ausserhalb des Betriebssystems angesiedelt sind.

Threads als integraler Bestandteil des Betriebssystems: Die für den Einsatz der Threads notwendigen Funktionen sind auf System Calls des Betriebssystems abgebildet.

Der Einsatz von Threads findet seinen Ursprung in zwei Bereichen: Einerseits wird dort, wo das Anlegen von Prozessen aufwendig ist, mit Threads gearbeitet. Weiter

werden Vorgänge innerhalb von Prozessen mit Hilfe von Threads effizient und unkompliziert entworfen.

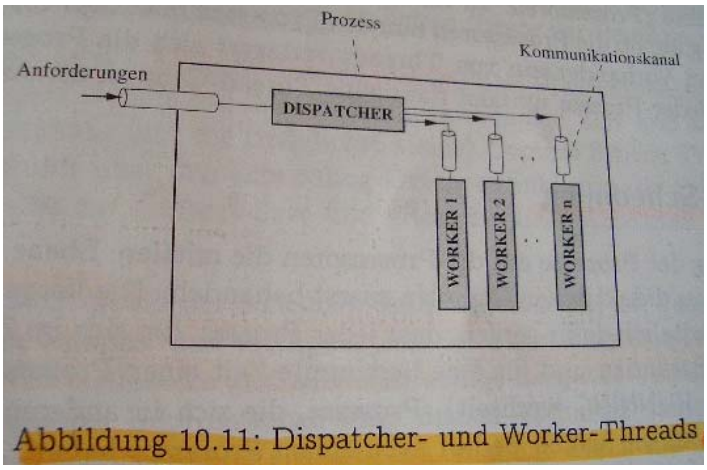


Abbildung 10.11: Dispatcher- und Worker-Threads

Mit der Aufteilung des Prozess-Geschehens in mehrere Threads entbindet der Anwender das Betriebssystem von der Entscheidung, wie eine Parallelisierung vorzunehmen ist bzw. wie sie effizient erfolgen kann.

Scheduling

Dies beschäftigt sich damit wie auf einer Maschine mit nur wenigen Prozessoren (meist nur einem) ein virtuelles 10000000-Prozessorsystem realisiert werden kann.

Prozess-Scheduling

Die Vorgehensweise ist eigentlich recht einfach: Wir müssen lediglich dafür sorgen, dass jeder Prozess, der sich im Zustand READY befindet, in bestimmten Abständen und für eine bestimmte Zeit einen Prozessor zugeteilt bekommt.

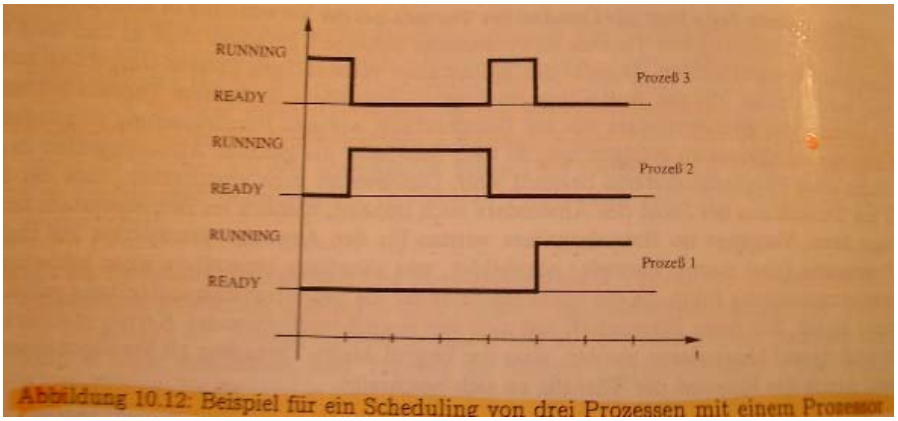


Abbildung 10.12: Beispiel für ein Scheduling von drei Prozessen mit einem Prozessor

Es existieren eine Menge verschiedener Strategien für das Prozess-Scheduling:

- Fairness: Die Verteilung der Prozessorkapazität soll gerecht sein
- Effizienz: Die Prozessoren sollten möglichst optimal ausgelastet werden
- Durchsatz: Die Anzahl der verarbeiteten Jobs sollte maximiert werden
- Antwortzeiten. Gerade bei interaktiven Prozessen ist es wichtig, raschen Response zu liefern
- Prozessorwechselzeit: (Context Switch Time) soll minimal sein

Wichtige Scheduling Verfahren:

First Come First Served (FCFS): Ist eine non preemptive Scheduling-Technik. Hierbei wird einem Prozess der einmal zugeteilte Prozessor bis zu seiner Termination nicht mehr entzogen.

Round Robin Scheduling (RRS): Hier bekommt nacheinander jeder READY Prozess für ein Zeitintervall einen Prozessor zugeteilt. Läuft das Zeitquantum eines Prozesses im Zustand RUNNING ab, ohne dass der Prozess BLOCKED wurde, wird am Ende der Ready-Liste eingetragen, eine vor dem Ablauf des Time Slices stattfindende Blockierung bewirkt natürlich den Eintrag in die Blocked-Liste.
Kritisch für das Round Robin Scheduling ist die Länge der Time Slices, ist sie zu klein, reduziert der Scheduling Overhead die nutzbare Prozessorleistung unzulässig, ist sie zu groß, sind die Antwortzeiten unbefriedigend.

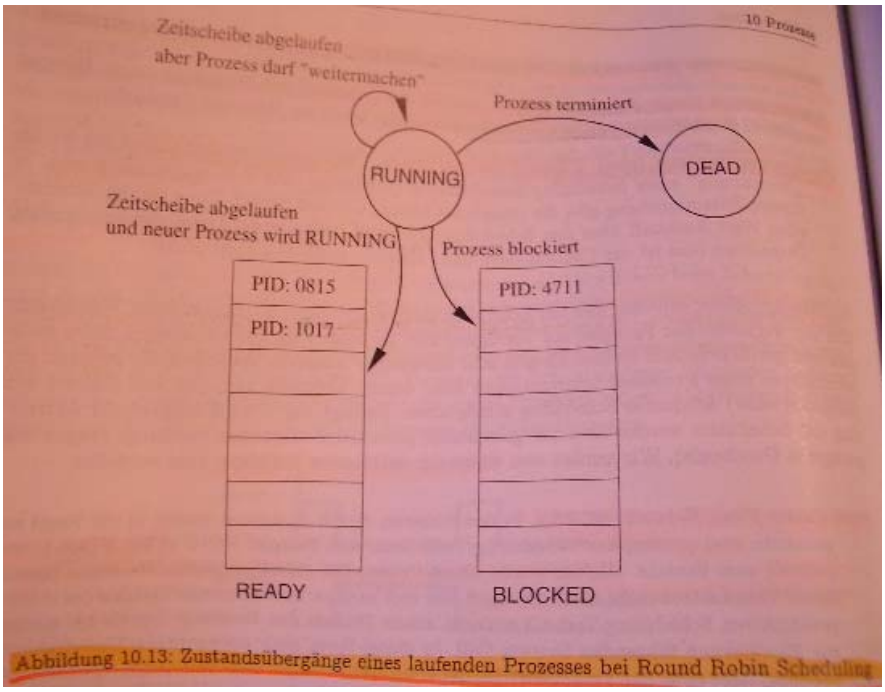


Abbildung 10.13: Zustandsübergänge eines laufenden Prozesses bei Round Robin Scheduling

Static Priority Scheduling (SPS): In einem Computer gibt es wichtigere und weniger wichtige Aufgaben. Es ist daher natürlich, Prozessen eine Priorität zuzuordnen und die Prozessorkapazität entsprechend aufzuteilen.
RUNNING ist immer der Prozess, der die höchste Priorität hat. Immer wenn ein anderer Prozess in den Zustand READY wechselt, der eine höhere Priorität als der aktuell laufende Prozess hat, nimmt der Scheduler dem laufenden Prozess den Prozessor weg und bringt den neuen Prozess zur Ausführung. Hierbei handelt es sich um eine preemptive Strategie.
Es kann hierbei jedoch auch passieren, dass ein niedrigpriorer Prozess nie einen Prozessor zugeteilt bekommt, ein Effekt der Starvation genannt wird.

Dynamic Priority Scheduling (DPS): Schwierigkeiten dieser Art können durch die dynamische Änderung von Prioritäten gelöst werden. Durch eine geschickte Vergabe derselben kann sowohl eine Anpassung an verschiedene Betriebssituationen als auch eine adäquate Aufteilung der Prozessorkapazität erfolgen.

Konzeptuell kann man sich vorstellen, dass die Priorität eines RUNNING Prozesses in regelmäßigen Abständen um einen gewissen Wert b, die Priorität eines READY Prozesses aber um einen Wert a erhöht wird. Für b = -1 für a = 0 erhalten wir RRS, die oben erwähnten regelmässigen Intervalle entsprechen klarerweise dem Quantum.
Dadurch kann die Monopolisierung der Prozessoren durch einzelne Prozesse verhindert werden.

Es gibt noch andere Verfahren, wie etwa das Shortest Job First (SJF) oder Shortest Remaining Time (SRT) Scheduling.

Thread-Scheduling

Mit dem Thread Scheduling wird das Prozess-Scheduling um eine hierarchisch untergeordnete Ebene des Schedulings ergänzt.
Ein Problem tritt dann auf wenn ein Thread blockiert, obwohl noch andere Threads im Zustand READY warten. Ein geschickter Thread-Scheduler nimmt darauf Rücksicht und gibt den Prozessor erst dann ab, wenn alle Threads eines Prozesses blockieren.

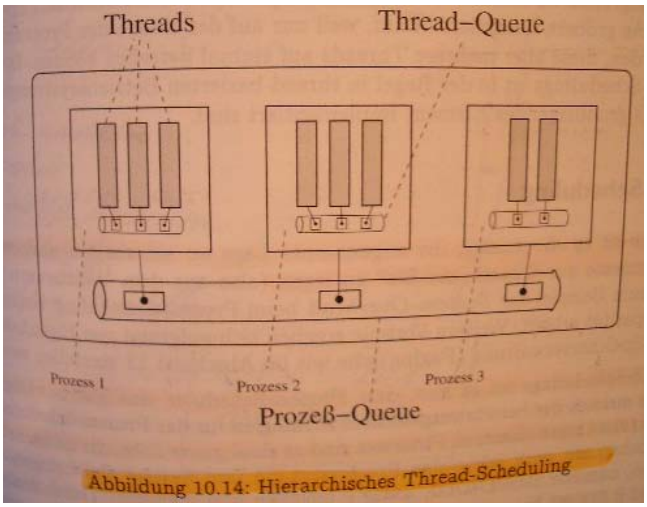


Abbildung 10.14: Hierarchisches Thread-Scheduling

Durch Einführung systemweiter Prioritätsebenen ist es damit sogar möglich, die Ausführungsabfolge zu einem höherprioren Thread eines anderen Prozesses wechseln zu lassen, obwohl der gerade noch aktive Prozess durchaus über Threads im READY Zustand verfügt.

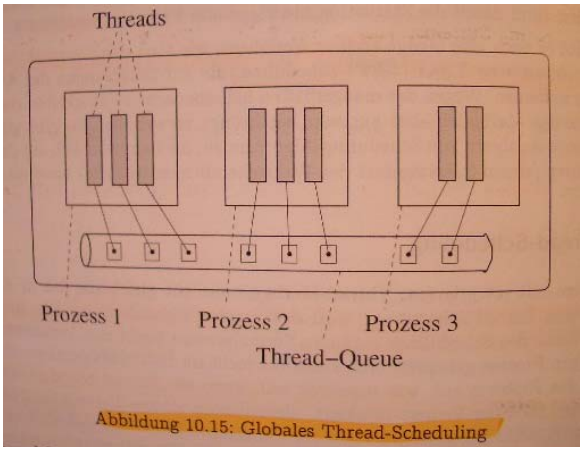


Abbildung 10.15: Globales Thread-Scheduling

Job-Scheduling

In Hochlastfällen ist es oft günstig, die angespannte Lage zu entschärfen und einige der konkurrierenden Prozesse zur Gänze „auf Eis“ zu legen.
Ziel des Job-Schedulings ist es also, dem Prozess-Scheduler eine gut bewäligbare Arbeit zuzuteilen. Hier sind statt der einzelnen Prozesse einfach ganze Jobs betroffen.

11.Interprozess-Kommunikation

Server-Prozesse

Server Prozesse haben die Aufgabe, anderen Prozessen gewisse Dienstleistungen zur Verfügung zu stellen. Um diese Dienste nutzen zu können, müssen die (User-)Prozesse (Clients) nun Requests über die Mechanismen der Interprozess-Kommunikation an die diversen Server-Prozesse schicken.

Der Vorteil eines Server-Prozesses soll anhand eines Printer Servers veranschaulicht werden. Hier wird das sogenannte Printer Spooling eingesetzt. Anstatt des Druckes wird mittels F_OPEN(„PRINTER“, attributes) sozusagen insgeheim statt des Druckers ein File auf der Disk geöffnet. Alles was nun der Prozess vermeintlich auf den Drucker schreibt geht in Wirklichkeit auf dieses Spool-File. Wenn der erzeugende Prozess den (imaginären) Drucker mittels F_CLOSE wieder schließt, wird ein Service Request an den Printer Server geschickt, der diesen veranlasst, das File auszudrucken.
Der Vorteil dieser Methode besteht darin das ein Prozess den Drucker nicht während seines gesamten Ablaufs blockiert und das somit mehrere Prozesse auch Druckbefehle schicken können.

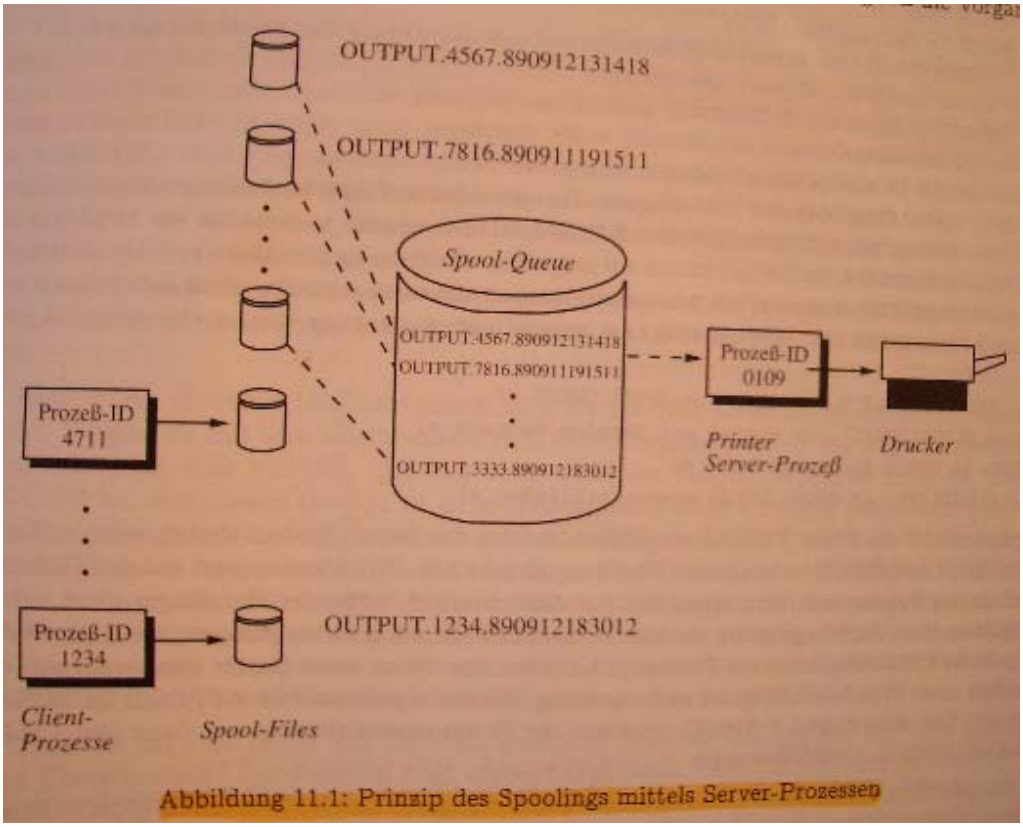


Abbildung 11.1: Prinzip des Spoolings mittels Server-Prozessen

Es können aber auch Probleme auftreten wenn unter bestimmten Umständen ein Prozess zufällig vor einem anderen fertig wird, der das fertig werden des vorher fertig werdenden Prozesses schon vor seinem eigenen fertig werden gebraucht hätte. Solche Probleme werden auch als Race Conditions bezeichnet.
Während der Ausführung einer Critical Section durch einen Client müsste also sichergestellt sein, dass kein anderer damit beginnen kann. Dieser sogenannte gegenseitige Ausschluss (mutual exclusion) ist eines der ganz wichtigen Probleme der Interprozess-Kommunikation.

Synchrone Methoden

Synchrone Methoden zur Interprozess-Kommunikation zeichnen sich dadurch aus, dass der Empfänger eine Nachricht durch eine eigene Aktivität abholen muss.

Semaphore

Semaphore sind ein geeignetes Konzept zur Vermeidung von Race Conditions. Ein Semaphor ist ein Objekt, bestehend aus einem auf 0 initialisierten Counter und einer zunächst leeren Liste für Prozess-IDs. Eine Reihe von System Calls erlaubt nun die Synchronisation paralleler Prozesse.

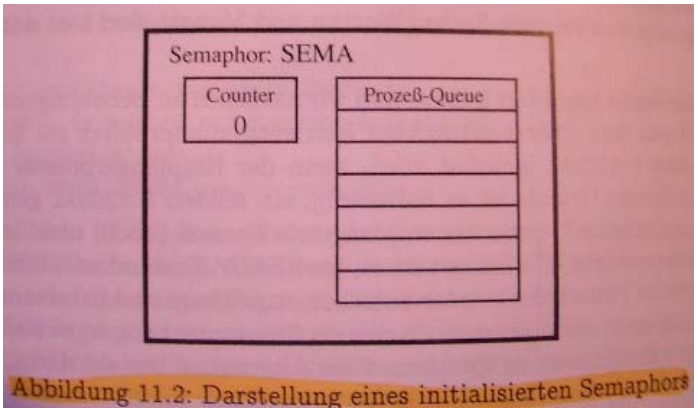


Abbildung 11.2: Darstellung eines initialisierten Semaphors

Es gibt zwei Operationen, S_P(semaphor-ID) und S_V(semaphor-ID), deren Wirkung davon abhängt, ob der Counter des Semaphors größer, kleiner oder gleich 0 ist.

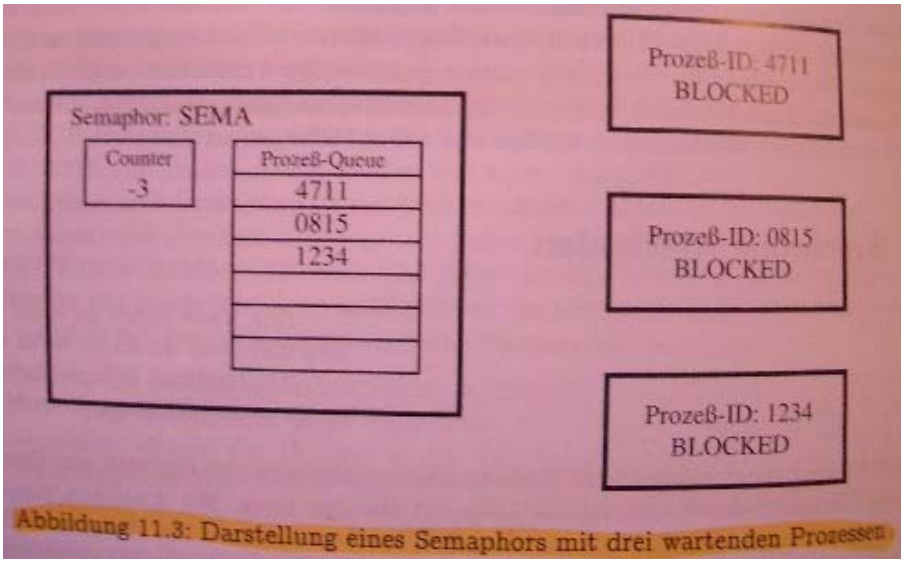


Abbildung 11.3: Darstellung eines Semaphors mit drei wartenden Prozessen

S_P: Counter > 0 → Counter = Counter -1
Counter <= 0 → Counter = Counter -1, der Aufrufende Prozess wird in die Prozess-Queue eingetragen und in den Zustand BLOCKED versetzt.

S_V: Counter => 0 → Counter = Counter +1
Counter < 0 → Counter = Counter +1, der erste Prozess wird aus der Prozess-Queue entfernt und in den Zustand READY versetzt.

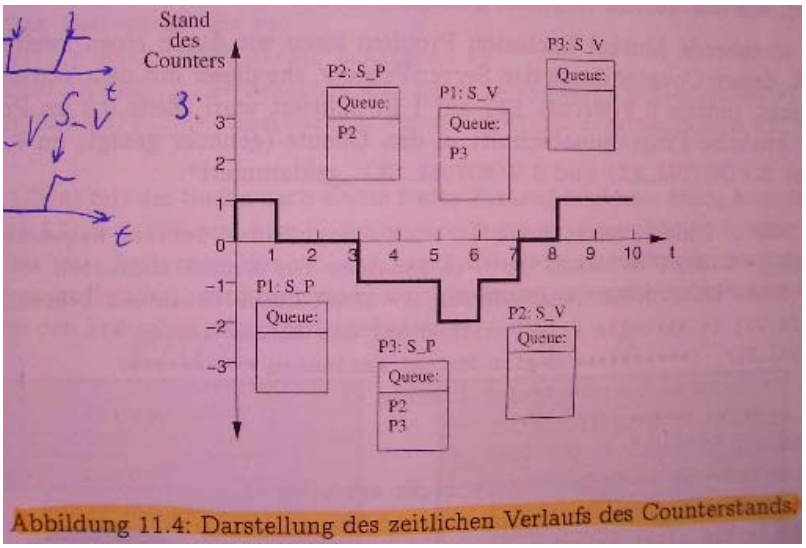


Abbildung 11.4: Darstellung des zeitlichen Verlaufs des Counterstands

Die Ordnung der Prozess-Queue muss nicht unbedingt FIFO sein, sondern eine z.B. nach der Priorität der Prozesse geordnete Liste ist ebenfalls möglich.

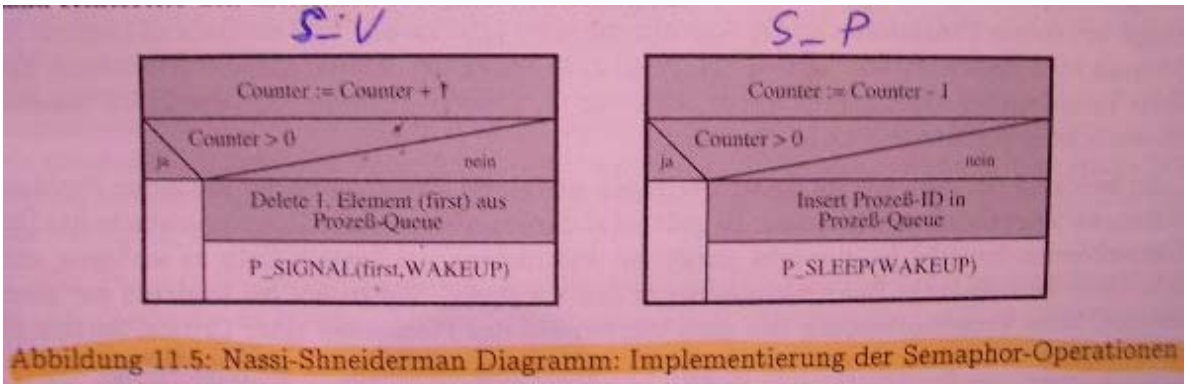


Abbildung 11.5: Nassi-Shneiderman Diagramm: Implementierung der Semaphor-Operationen

Der Mechanismus eines Semaphors, den wir zur Verhinderung von Race Conditions in den Prozessen vorgesehen haben, hat jedoch selbst mit Race Conditions auf der Betriebssystemebene zu kämpfen.
Deshalb sollten die grauen Bereiche in den Diagrammen unteilbar (atomic) gemacht werden.

Es gibt zwei grundsätzlich verschiedene Lösungsansätze für dieses Problem. Der erste ist sicherlich der einfachste: Wir erlauben keine gleichzeitige Ausführung kritischer System Calls, simultane Aufrufe werden in einer Warteschlange gesammelt und einer nach dem anderen ausgeführt. In diesem Zusammenhang ist es üblich, von Serialized Actions zu sprechen.
Der andere Weg ist der, echte oder quasi-parallele Ausführung zuzulassen, durch Hard- oder Softwaremaßnahmen aber eine Mutual Exclusion zu gewährleisten.

Message Passing

Im Gegensatz zu den Semaphoren handelt es sich beim Message Passing um einen Mechanismus, der eine Kommunikation im Sinne eines Datenaustausches zwischen Prozessen erlaubt. Üblicherweise deponiert dabei ein Senderprozess Nachrichten an einem Message Exchange, von wo sie ein Empfängerprozess abholen kann.
Praktisch wird dies durch eine Anzahl von System Calls ermöglicht.
Mittels der Operation E_SEND(exchange-ID, message) kann nun eine einzelne Message an dem angegebenen Exchange deponiert werden. Ein Prozess, der eine einzelne Message von einem Exchange abholen will, kann dies unter Zuhilfenahme von E_RECEIVE(exchange-ID, message) tun.

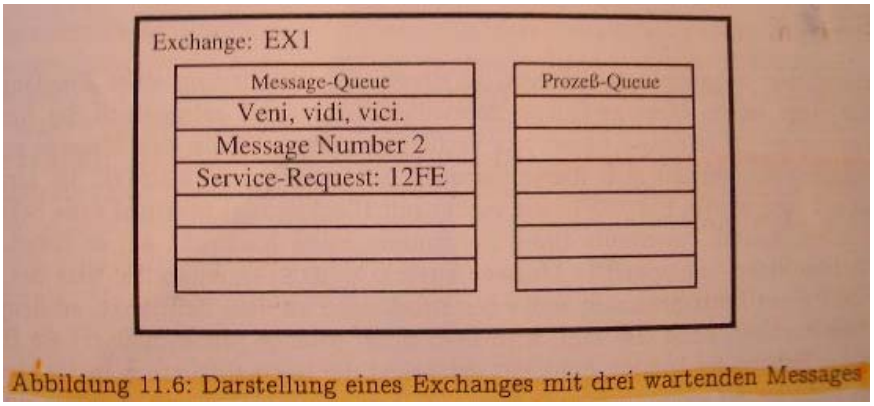


Abbildung 11.6: Darstellung eines Exchanges mit drei wartenden Messages

Der Inhalt der Messages ist für das Betriebssystem übrigens gleichgültig.
Die Prozess-Queue erfüllt wieder dieselbe Aufgabe, die sie auch bei den Semaphoren hatte. Wenn nun ein Empfänger sein E_RECEIVE executiert, bevor der sendende Prozess E_SEND rufen konnte, wird der Empfängerprozess in den Zustand BLOCKED versetzt und mittels seiner Prozess-ID in die Prozess-Queue eingetragen.
Wenn der Sender nun die Message Abschickt, wird diese nicht in die Message-Queue eingetragen, sondern dem ersten Prozess in der Prozess-Queue übergeben, das für dessen Zustandsübergang von BLOCKED nach READY nötige externe Ereignis ist eingetreten.

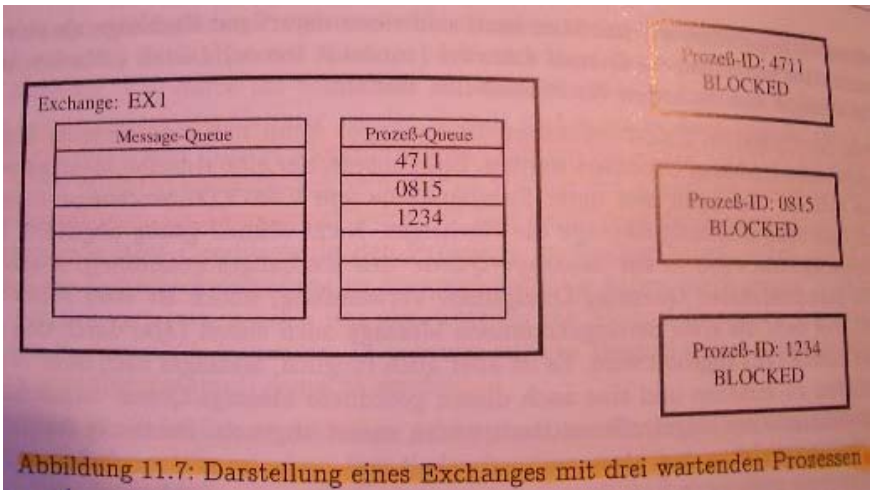


Abbildung 11.7: Darstellung eines Exchanges mit drei wartenden Prozessen

Mittels E_ACCEPT(exchange-ID, message, time) wird der Prozess nicht in den Zustand BLOCKED versetzt wenn noch keine Message da ist, sondern er schaut nur

von Zeit zu Zeit nach ob schon eine Message da ist und wenn nicht arbeitet er normal weiter.

Höhere Mechanismen

Hier ist z.B. das Rendezvous-Konzept zu erwähnen bei dem der Senderprozess auf den Empfänger warten muss.
Oder das Konzept des Monitors, bei dem ein Programmausschnitt automatisch durch den Einschluss in eine S_P und S_V geklammert wird.

Asynchrone Methoden

Hier wird vom Empfänger kein explizites Abholen der an ihn geschickten Nachrichten verlangt. Es handelt sich dabei um die sogenannten asynchronen Signale. Im Gegensatz zu synchronen Techniken unterbricht die Ankunft einer asynchronen Nachricht den Empfänger in seiner normalen Tätigkeit.

Durch einen System Call A_TRIGGER(signal, process-ID) wird dem spezifizierten Prozess das angegebene Signal signal geschickt. Mit Hilfe des Service Calls A_CATCH(signal, service-routine) kann spezifiziert werden, was bei Eintreffen von signal geschehen soll.
Asynchrone Signale entsprechen exakt den Interrupts, allerdings auf Prozessebene.

Deadlocks

Es genügt nicht, die einzelnen Aufträge in der Reihenfolge ihrer Ankunft (First Come First Serve) abzuarbeiten um Deadlocks zu verhindern.
Das Betriebssystem sieht sich mit einer Folge von Objekt-Anforderungen (und Freigaben) konfrontiert.

Eine Menge von Prozessoren ist nun im Zustand eines Deadlocks, wenn jeder einzelne von ihnen auf etwas wartet, was nur durch eine Aktivität eines anderen Prozesses aus dieser Menge hervorzubringen ist. Da diese Bedingung aber die Blockierung aller Prozesse impliziert, ist keine einzige „erlösende“ Aktion möglich. Es kann also kein Objekt freigegeben werden. Eine sorgfältige Analyse von Deadlock-Situationen zeigt nun vier notwendige Bedingungen für deren Entstehung:

Mutual Exclusion: Ein bestimmtes Objekt kann zu jedem Zeitpunkt von höchstens einem Prozess okkupiert sein.
Resource Waiting: Wenn ein beantragtes Objekt gerade besetzt ist, geht der anfordernde Prozess in den Zustand BLOCKED über, wartet also auf dessen Freiwerden.
Partial Allocation: Prozesse, die bereits im Besitz von Objekten sind, können die Zuteilung weiterer beantragen.
Nonpreemption: Ein einmal zugeteiltes Objekt muss explizit durch den die Ressource haltenden Prozess wieder freigegeben werden, kann ihm also nicht zwangsweise entzogen werden.

Das Problem eines Deadlocks kann in einem sogenannten Resource Allocation Graph dargestellt werden, wird in so einem Graph ein Zyklus gefunden, entspricht dieser einem Deadlock. Dadurch kann das Problem der Behandlung eines Deadlocks

überhaupt erst an das Betriebssystem delegiert werden. Dafür gibt es folgende prinzipielle Möglichkeiten:

Deadlock Detection and Recovery: Wir ein Zyklus im Resource Allocation Graph gefunden, muss dieser durch das Terminieren eines oder sogar mehrerer Prozesse aufgelöst werden.

Deadlock Prevention: Diese Methode basiert darauf, allein durch die Beachtung gewisser Kriterien beim Design eines Betriebssystems eine der notwendigen Bedingungen für einen Deadlock a priori zu „verletzen“, wodurch ein solcher gar nicht auftreten kann.

Deadlock Avoidance: Dieser Technik liegt eine sorgfältige Analyse der Objekt-Anforderungen zugrunde. Dabei wird bei jeder Anforderung versucht vorausschauend zu klären, ob die Zuteilung irgendwelche späteren Deadlocks nach sich ziehen kann.

12.Speicherverwaltung

Nun widmen wir uns der Frage, wie Prozesse eigentlich im Speicher eines Rechners „angelegt“ werden bzw. welche Voraussetzungen dafür eigentlich notwendig sind. Welche betriebssysteminternen Aktivitäten zieht also das P_CREATE(program, parameter, attributes), mit dem ein neuer Prozess erzeugt wird, nun tatsächlich nach sich.
Zuerst einmal ist es wichtig, Speicherplatz für den Prozess vorzusehen.

Bei alledem kommt der Speicherverwaltung eines Betriebssystems eine ganz wesentliche Rolle zu, der Vorstellung der hierbei relevanten Konzepte sind die folgenden Abschnitte gewidmet. Zwei zentrale Begriffe sind dabei die virtuellen und physikalischen Speicheradressen und Adressräume.

Ein Befehl, der etwa den Inhalt zweier Speicherzellen addieren soll, bekommt als Operanden deren virtuelle Adressen. Ein derartiges Maschinenprogramm könnte etwas folgenden Aufbau haben:

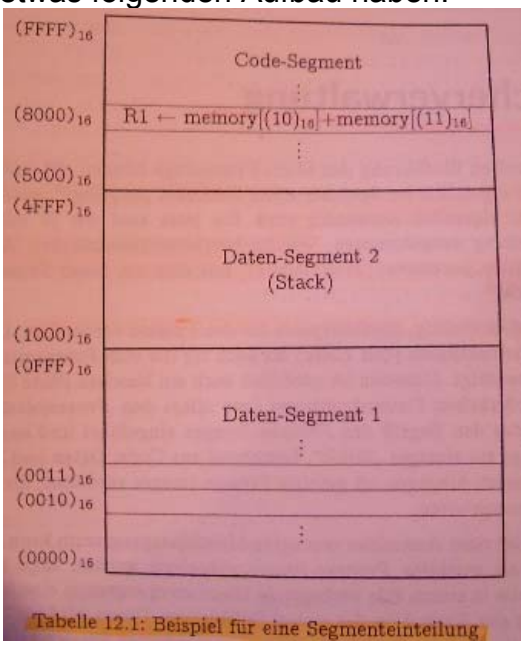


Tabelle 12.1: Beispiel für eine Segmenteinteilung

Es muss ein Weg gefunden werden, gleiche virtuelle Adressen verschiedener Prozesse auf eindeutige physikalische Adressen abzubilden, ein Vorgang, den wir Binding nennen wollen.

Virtuelle Adresszuordnung

Wenn eine größere Aufgabe gelöst werden soll, so ist es zweckmäßig, sie aufzuteilen und die Einzelteile getrennt zu behandeln.
Zur Durchführung dieser sogenannten Modularisierung müssen jedoch flankierende Maßnahmen von Seiten der Entwicklungswerkzeuge getroffen werden.

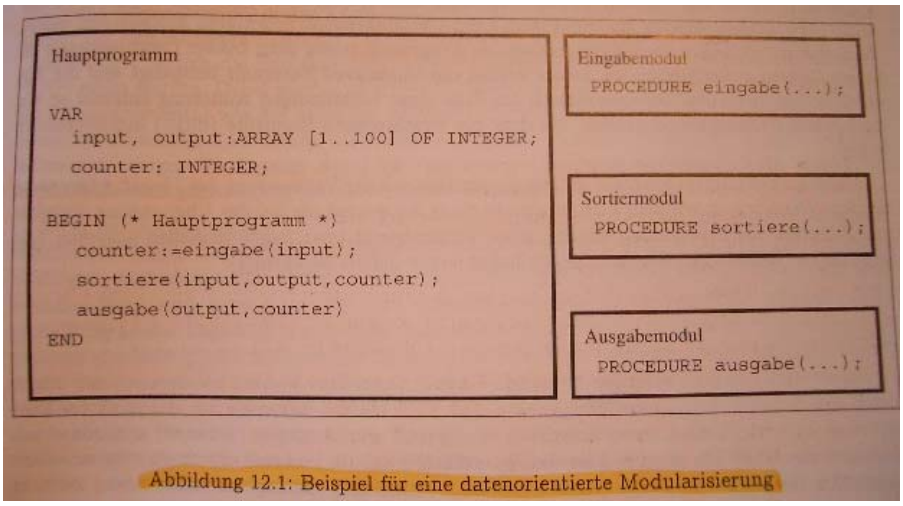


Abbildung 12.1: Beispiel für eine datenorientierte Modularisierung

Einer der Hauptvorteile dieser Modularisierung ist die weitgehende Unabhängigkeit von der konkreten Implementierung.
Das Zusammenfügen der benötigten Module zu einem ausführbaren Maschinenprogramm erfolgt durch den sogenannten Linker.
Für das Linken selbst gibt es prinzipiell zwei Möglichkeiten, das nach der Übersetzung erfolgende statische und das zur Laufzeit stattfindende dynamische Linken der Module. Beim statischen Linken werden alle zur Bildung des Gesamtsystems benötigten Module von einem zu den Entwicklungswerkzeugen gehörenden Programm segmentweise zu einem Ganzen zusammengefügt.
Dabei gibt es nun ein Problem im Zusammenhang mit der Verwendung des Direct Addressing Modes. Wenn wir annehmen, dass innerhalb eines Segmentes die bei der Übersetzung vergebenen Adressen bei 0 beginnen, könnte ein Sortiermodul etwa folgendermaßen aussehen:

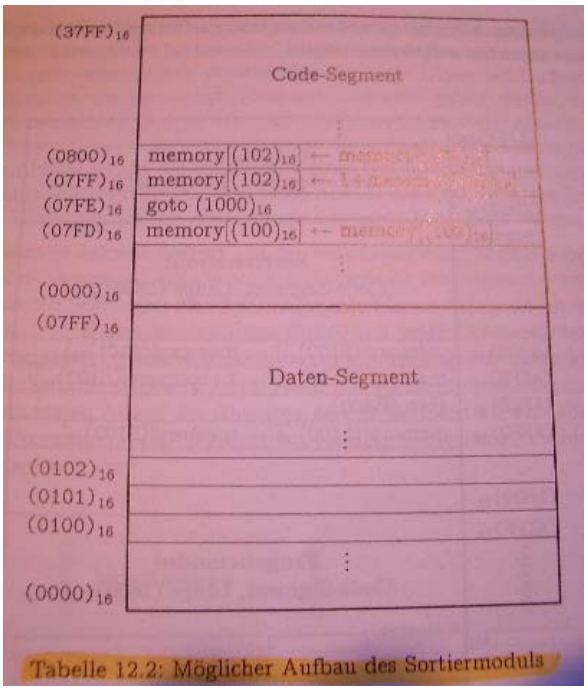


Tabelle 12.2: Möglicher Aufbau des Sortiermoduls

Wenn der Linker alle Module segmentweise zusammenfügt, ergibt sich für unser Sortiermodul folgenden in Abbildung 12.3 ersichtliche Verschiebung.

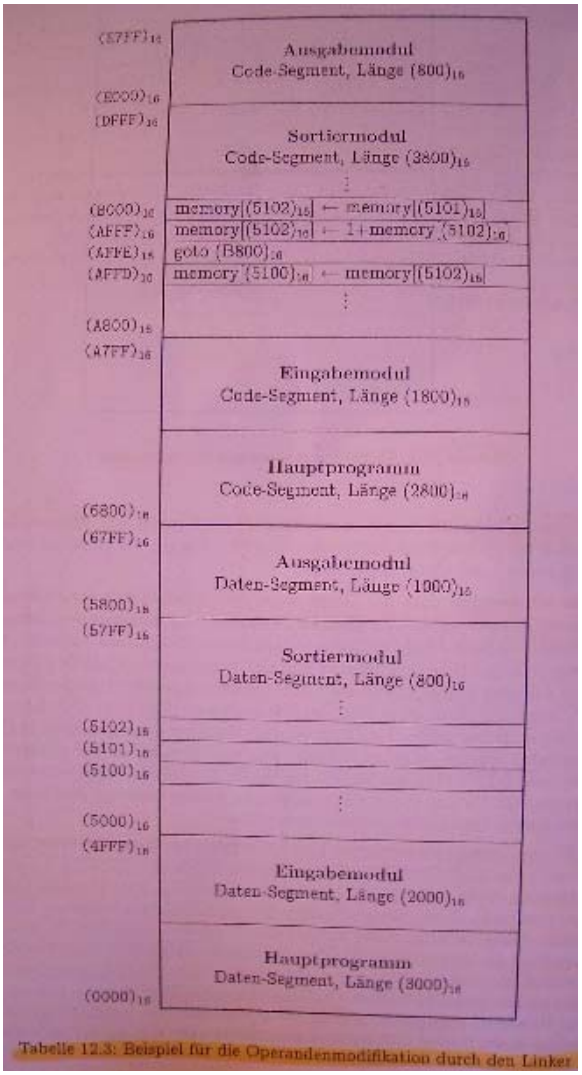


Tabelle 12.3: Beispiel für die Operandenmodifikation durch den Linker

Bei direkter Adressierung muss der Linker daher eine Modifikation der Operanden durchführen. In unserem Beispiel würde sonst (bei der Execution) das verschobene Sprungziel (Adresse $(B800)_{16}$ verfehlt werden.

Einer der Hauptnachteile des statischen Linkens ist, dass die Änderung eines universell verwendbaren Moduls das neuerliche Linken aller Programmsysteme prinzipiell alle konstituierenden Teile, auch wenn diese „fast nie“ aufgerufen werden, daraus entstehen natürlich recht große Programme. Damit ist es auch sehr wahrscheinlich, universelle Module als Teile vieler Programme gleichzeitig (also mehrfach) im Speicher der Maschine zu finden. Diese Probleme werden durch das dynamische Linken verhindert. Dabei wird im Maschinenprogramm statt eines Unresolved Externals ein spezieller System Call und eine Identifikation abgespeichert. Kommt der Prozessor (zur Laufzeit!) an eine derartige Stelle, so wird durch den Service Call der Runtime Linker aktiviert, der die benötigte Funktion in einer Runtime Library lokalisiert.
Alle bisher besprochenen Maßnahmen hatten den Zweck, die Ordnung der diversen Segmente in einem linearen Adressraum herzustellen. Unglücklicherweise hat dieser eindimensionale Adressraum einen ganz wesentlichen prinzipiellen Nachteil. Dieser wird offensichtlich, wenn wir Segmente betrachten, die dynamisch (also zur Laufzeit!) größer werden können. Bereits bei zwei derartigen Segmenten gibt es keine wie auch immer geartete Anordnung im virtuellen Adressraum, die ein „Ineinanderwachsen“ verhindern würde.

Eine zweidimensionale Adresse besteht aus der Verkettung einer Segment-Nummer und einer Adresse innerhalb des Segments. Idealerweise sollte sowohl der Segment Adressraum als auch der jeweilige virtuelle Adressraum sehr groß sein. Damit könnte jedes Code- und jedes Daten-Segment, ja sogar jedes einzelne Unterprogramm in ein eigenes Segment gesteckt werden.

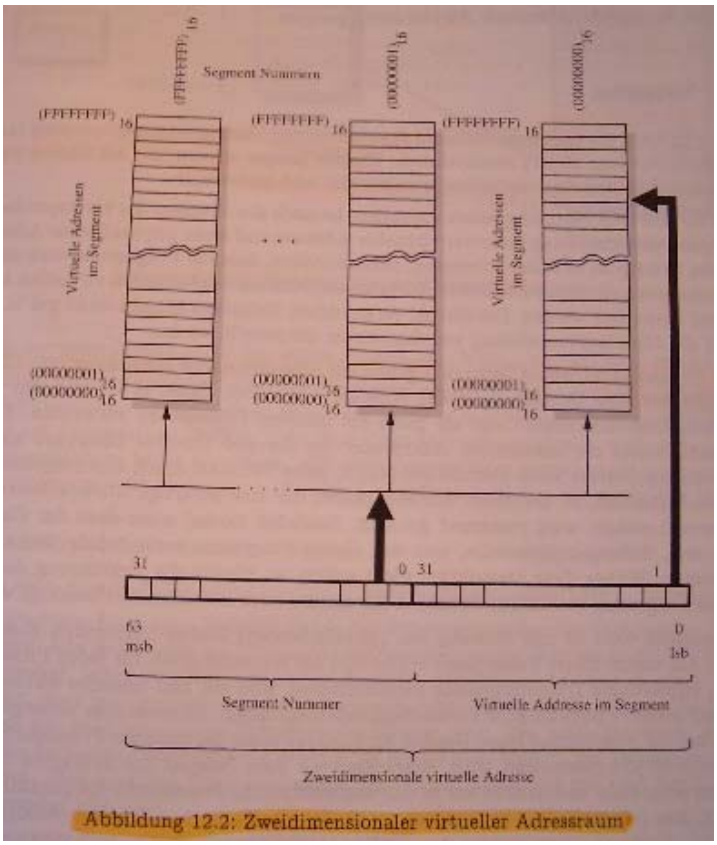


Abbildung 12.2: Zweidimensionaler virtueller Adressraum

Physikalische Adresszuordnung

Wir haben also jetzt eine ganze Anzahl von Prozessen mit ihren virtuellen Adressräumen vor uns, die mittels Multi-Processing gleichzeitig ausgeführt werden sollen. Nun wird aber der gesamte Speicherbedarf im allgemeinen wesentlich größer als der physikalische Speicher der Maschine sein. Da jedoch zu jedem Zeitpunkt nur soviele Prozesse gleichzeitig executiert werden können, wie Prozessoren vorhanden sind, braucht aber nur eine gewisse Anzahl von Prozess-Images gleichzeitig speicherresident zu sein.

Swapping

Ist die einfachste virtuelle Speichertechnik, Prozess-Images werden hier als Ganzes zwischen dem physikalischen und dem virtuellen Speicher hin- und herbewegt. Jedem Prozess ist zu jedem Zeitpunkt genau ein auszuführendes (physikalisches) Code-Segment zugeordnet.

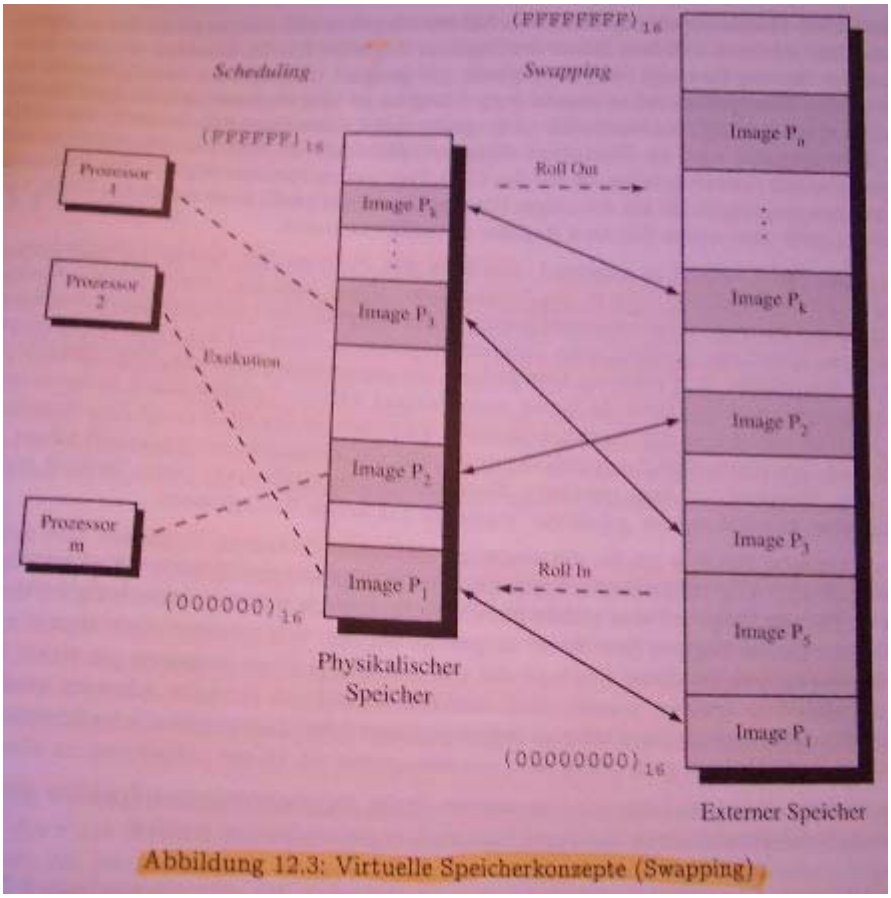


Abbildung 12.3: Virtuelle Speicherkonzepte (Swapping)

Wollen wir an dieser Stelle kurz das Problem der Memory Protection überlegen. Eine Möglichkeit wäre es, im Prozessor ein Upper und ein Lower Bound Register verzusehen, mit deren Hilfe ein aus einem Segment „hinausgreifender“ Speicherzugriff abgefangen werden kann.
Ein anderer Weg wäre die Verwendung von sogenannten Storage Keys. Es gibt hier spezielle Memory-Architekturen, die jedem großen Block aufeinanderfolgender Speicherwörter eine Art (Speicher-) Register zuordnen, in das ein Storage Key eingetragen werden kann. Vor einem Speicherzugriff überprüft nun der Prozessor, ob

der Inhalt seines Prozessor Key Registers mit dem jeweiligen Storage Key übereinstimmt.

Dieses einfache Binding ist für die eingangs erwähnten Swapping-Techniken ausreichend.
Wir möchten noch einmal explizit darauf hinweisen, dass bei dieser Methode das gesamte Image eines Prozesses „in einem Stück“ in den physikalischen Speicher geladen wird. Aufeinanderfolgende virtuelle Adressen werden daher auf aufeinanderfolgende physikalische Adressen abgebildet. Der physikalische Speicher einer Maschine muss demzufolge groß genug sein, um wenigstens ein Image aufnehmen zu können.

Ein wichtige Frage ist natürlich, an welcher Stelle im physikalischen Speicher ein Image untergebracht werden soll. Zwei Varianten können hier unterschieden werden:

fixe Partitionen: Der physikalische Speicher wird in n fixe Partitionen eingeteilt und je ein Prozess-Image in einen derartigen Bereich geladen.
variable Partitionierung: Ein Image kann ab jeder beliebigen physikalischen Adresse geladen werden, die den Beginn eines genügend großen, freien Speicherbereichs bezeichnet. Da normalerweise mehrere geeignete Kandidaten existieren werden, gibt es eine Reihe von Auswahlverfahren (z.B. First Fit, Best Fit, Buddy Verfahren)

Paging

Durch geeignete Maßnahmen wird (scheinbar) jedem Prozess tatsächlich sein gesamter virtueller Adressraum auch physikalisch zur Verfügung gestellt. Zu diesem Zweck wird dieser in (sehr viele) gleichgroße Abschnitte, sogenannte Pages eingeteilt. Jede Page ist eine logische Page-Nummer zugeordnet, die ihrem Platz im virtuellen Adressraum entspricht.

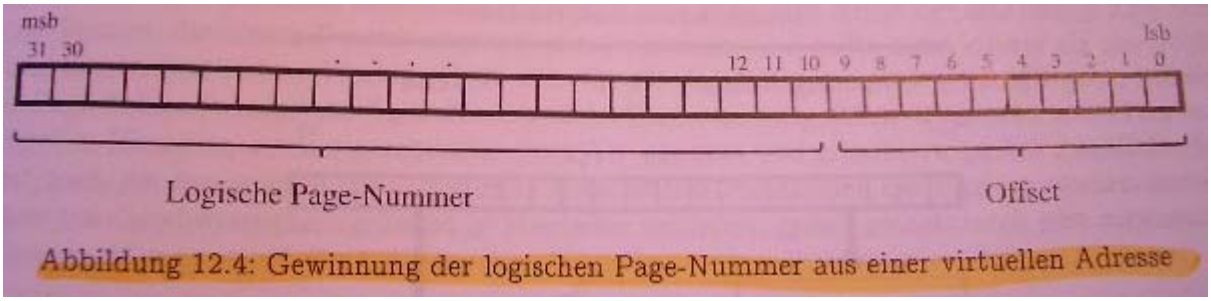


Abbildung 12.4: Gewinnung der logischen Page-Nummer aus einer virtuellen Adresse

Analog wird auch der physikalische Speicher der Maschine in Bereiche derselben Größe unterteilt, die Page Frames genannt werden. Jeder solche Frame wird durch eine logische Frame-Nummer, die seiner Position im physikalischen Speicher entspricht, identifiziert.
Der Trick beim Paging besteht nun darin, nur jene Pages im physikalischen Speicher zu halten, die tatsächlich „gebraucht“ werden.

Konzeptuell können wir uns vorstellen, dass das Betriebssystem für jeden Prozess eine Page Table führt, welche die Abbildung Pages → Page Frames enthält. Bei jeder Page-Nummer wird die Frame-Nummer desjenigen Page Frames eingetragen,

in dem die Page geladen ist. Diese Page Table wird nun wie folgt zur Umsetzung der virtuellen auf die physikalischen Adressen verwendet.

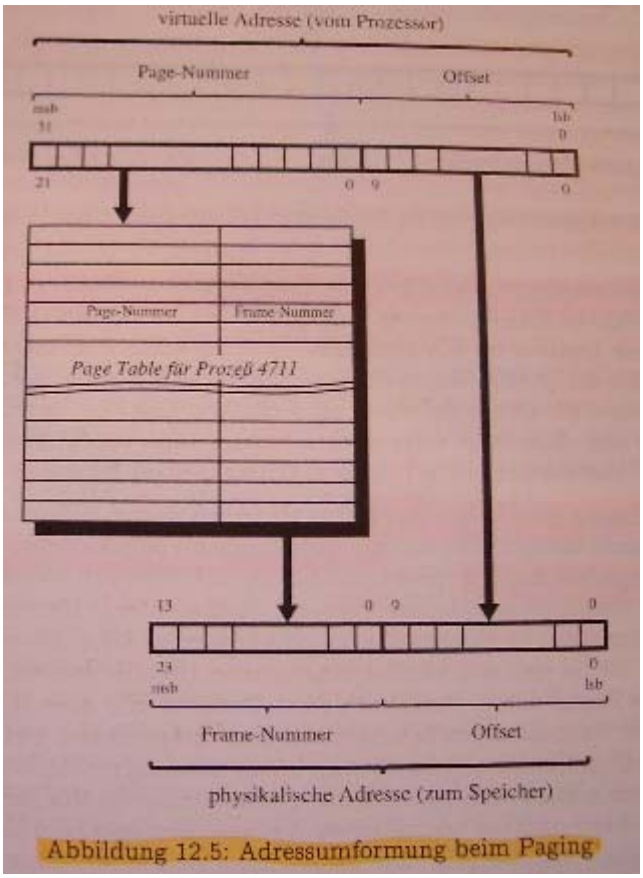


Abbildung 12.5: Adressumformung beim Paging

Durch die Eintragung ein und desselben Page Frames in die Page Tables mehrerer Prozesse ist auch das Sharing (etwas von Code-Segmenten) relativ einfach zu bewerkstelligen.

Sofern alle benötigten Pages geladen sind, funktioniert die Methode wunderbar. Gerade diese Voraussetzung wollten wir aber vermeiden, die meisten Einträge in der Page Table werden daher statt einer Frame-Nummer eine Kennung „Page nicht geladen“ beinhalten. Wenn der Zugriff auf eine derartige virtuelle Adresse erfolgt, ist ein sogenannter Page Fault die Folge. Der die Referenz verursachende Maschinenbefehl wird abgebrochen, da zuerst die benötigte Page in einen Page Frame geladen werden muss.

Bei einem Page-Fault stellt sich dann das Problem, einen belegten Page Frame auswählen und die darin gespeicherte Page austauschen zu müssen.
Hierfür gibt es verschiedenen Page Replacement Methoden:

- **First In First Out (FIFO)**: Jede Page bekommt zum Zeitpunkt ihres Ladens einen Zeitstempel, wenn ein Page Fault aufzulösen ist, wird die „älteste“ Page ersetzt.
- **Least Recently Used (LRU)**: Hier wird jene Page ersetzt, deren letzte Referenz am weitesten in der Vergangenheit liegt.
- **Least Frequently Used (LFU)**: Hierbei wird jene Page für einen Austausch herangezogen, die am wenigsten oft benutzt wurde.

- **Not Used Recently (NUR)**: Anstelle eines Reference Counters wird pro Page nur eine Kennung (Referenced/Not Referenced) eingetragen. Da aber im Laufe der Zeit praktische jede Page den Vermerk Referenced erhält, werden in gewissen Abständen die Kennungen aller Pages auf Not Referenced gesetzt, was dem Aging entspricht.

Segmentierung

Als letztes wollen wir noch Methoden zur Realisierung zweidimensionaler Adressräume vorstellen, und zwar die reine Segmentierung und die Segmentierung mit Paging. Erstere ist nichts anderes als ein segmentweises Swapping, die zweite Methode verwendet Paging, um die einzelnen linearen virtuellen Adressräume zur Verfügung stellen zu können.
Die endgültige Speicheradresse wird dann durch die Addition der gefundenen Startadresse mit dem Offset innerhalb des Segments gewonnen.
In Abbildung 12.6 haben wir 16 Bit Segment-Nummern, 16 Bit Offsets innerhalb eines Segmentes und 24 Bit physikalische Adressen angenommen. Zu beachten ist, dass die maximale Länge eines Segmentes dadurch 64 KByte beträgt.
Jedem Segment wird dabei eine eigenen Page Table zugeordnet, ein Eintrag in der Segment Table eines Prozesses zeigt nicht mehr auf das Segment selbst, sonder auf dessen Page Table.

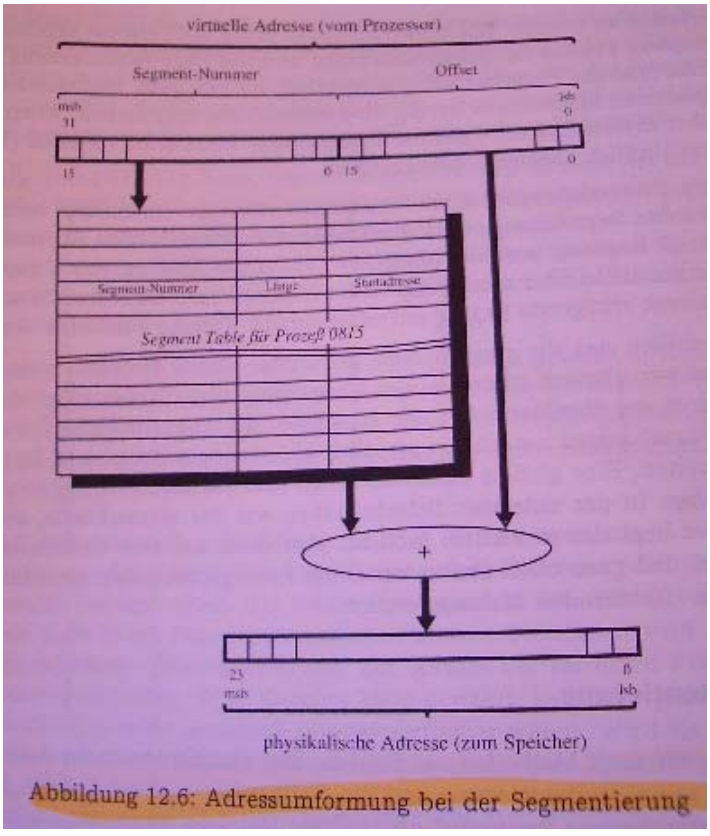


Abbildung 12.6: Adressumformung bei der Segmentierung

Die physikalische Adresse kann dann durch die Verkettung der Frame-Nummer mit dem Offset innerhalb der Page gewonnen werden.

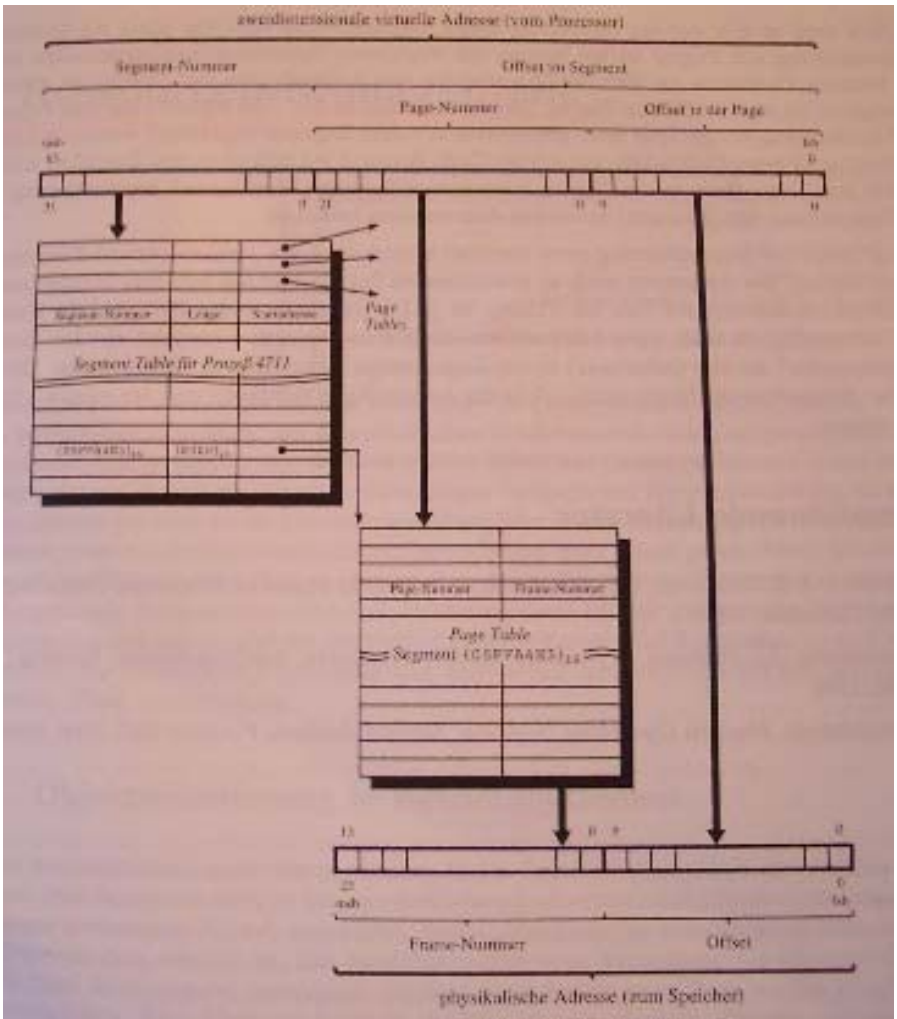


Abbildung 12.7: Adressumformung bei der Segmentierung mit Paging