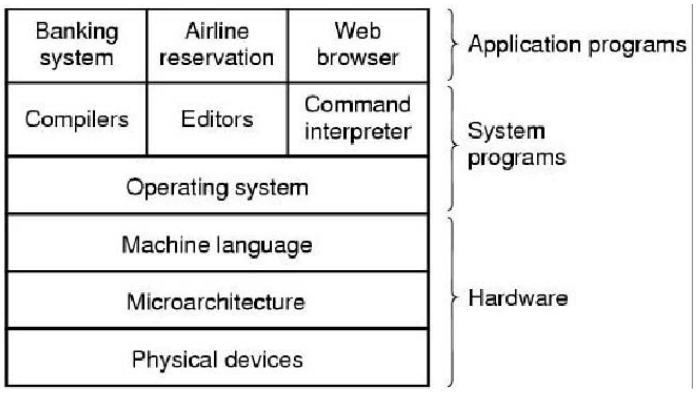
Kapitel 1

1. Wie kann man Software klassifizieren und was versteht man unter Systemsoftware?

Computer sind ohne Software nicht sinngemäß verwendbar.
Erst mit Software wird es möglich Informationen zu speichern, zu verarbeiten und wieder zu finden.
Software muss die abstrakte Anwendungssicht auf der Grundlage der realen Hardware realisieren. Man spricht hier auch von Systemsoftware, da diese meist direkt auf der Hardware des Rechensystems aufsetzt.
Die Software von Computern kann also grob in zwei Arten unterteilt werden:
Systemprogramme, die der Verwaltung des Betriebs eines Computers selber dienen, und
Anwendungsprogramme, die Probleme ihrer Benutzer lösen.
Das grundlegendste aller Systemprogramme ist das Betriebssystem, das alle Betriebsmittel verwaltet und eine Basis liefert, auf der Anwendungsprogramme programmiert werden können.
Die Erstellung von Systemsoftware ist keine triviale Aufgabe.
Wenn sich jeder Entwickler damit
beschäftigen müsste, wie ein Gerätetreiber für eine Festplatte oder eine Grafikkarte funktioniert, so wäre es sehr unwahrscheinlich, dass jemals so viele Programme geschrieben worden wären.

2. Nennen sie die grundlegenden Bestandteile (Ebenen) eines Rechensystems!



Erste Ebene ist die Hardware
- erste Schicht → physikalischen Geräte
- Zweite Schicht → Mikroprogramme wird im Allgemeinen in ROM Bausteinen abgelegt.

Zweite Ebene ist die Ebene der Systemprogramme. Deren Unterste Schicht ist das Betriebssystem. Auf dem Betriebssystem setzt der Rest der Systemsoftware auf z.B. Cpmpiler, Editoren usw..

Dritte Ebene sind die Anwendungsprogramme. Diese setzten auf der Ebene der Systemprogramme auf. Diese Programme werden von den Benutzern geschrieben, um ihrer speziellen Probleme zu lösen.

Zusammenfassend: Bestandteile eines Rechensytems sind:

- Hardware
- Systemprogramme
- Anwendungsprogramme

3. Welches sind die Beiden Hauptaufgaben von Betriebsystemen?

Aufgabe 1: Das Betriebssystem als eine erweiterte Maschine

Im Allgemeinen ist die Architektur von Computern (Instruktionssatz, Speicherorganisation, I/O-Struktur und Busstruktur) auf der Ebene der Maschineninstruktionen sehr einfach, aber schwierig zu programmieren. Insbesondere trifft dies auf die Ein-/Ausgabe zu. Damit nun ein Entwickler von der Komplexität der Programmierung für Hardware verschont bleibt, verwendet man eine Software. Diese Software, das Betriebssystem, verbirgt eine Menge unangenehmer Aufgaben (Speicherverwaltung, Uhren, Unterbrechungen etc.) vor dem Entwickler und liefert zugleich eine Vielzahl von Services (Dateisystem, Zugangskontrolle, Programmausführung etc.). In jedem Fall ist die Methode, die das Betriebssystem dem Benutzer zur Verfügung stellt, wesentlich einfacher und leichter zu benutzen als die darunterliegende Hardware. Unter diesem Blickwinkel ist es die Aufgabe eines Betriebssystems, dem Benutzer ein Äquivalent einer erweiterten Maschine bzw. einer virtuellen Maschine anzubieten, die leichter zu programmieren ist als die darunterliegende Hardware.

Aufgabe 2: **Das Betriebssystem als ein Betriebsmittelverwalter**

Die Sichtweise des Betriebssystems als eine erweiterte Maschine, die in erster Linie für den Benutzer eine angemessene Schnittstelle zur Verfügung stellt, entspricht einer Top-dow-Sicht. Eine alternative Betrachtung ist die Bottom-Up-Sicht, bei der das Betriebssystem die Aufgabe der Verwaltung aller Bestandteile eines komplexen Systems hat. Moderne Computer bestehen aus einer Vielzahl von Komponenten (Prozessoren, Speichern, Platten, Uhren, Netzwerkschnittstellen etc.). In der hier betrachteten alternativen Sichtweise besteht die Aufgabe eines Betriebssystems darin, eine geordnete und kontrollierte Zuteilung der Komponenten für die konkurrierenden Programme durchzusetzen. Man stelle sich das einfache Szenario vor, dass drei Programme auf demselben Computer versuchen würden, ihre Ausgabe gleichzeitig auf denselben Drucker zu leiten. Wird ein Computer von mehreren Benutzern verwendet, so wird die Notwendigkeit für die Verwaltung und den Schutz von Speichern, I/O-Geräten und anderen Betriebsmitteln immer offensichtlicher. Die Notwendigkeit dafür entsteht durch die gemeinsame Verwendung teurer Betriebsmittel.

4. Geben Sie eine Klassifizierung der Betriebsmittel nach diversen Kriterien an!

- Hardware- vs. Softwarebetriebsmittel
  - z.B. Prozessoren und Programme
- einmal vs. mehrmals benutzbare Betriebsmittel
  - z.B. Netzpakete und Speicherzellen
- exklusiv vs. nicht-exklusiv nutzbare Betriebsmittel
  - z.B. Drucker und Dateien (*read*)
- entziehbare vs. nicht-entziehbare Betriebsmittel
  - z.B. Prozessoren und Drucker

5. Welche Instanz in einem Rechnersystem kontrolliert das Betriebssystem?

Das Betriebssystem als Betriebsmittelverwalter unterliegt keiner Kontrollinstanz und verwendet auch selbst Betriebsmittel.

Die vorrangigen Aufgaben des Betriebssystems als Betriebsmittelverwalter:

- Wer benutzt welche Betriebsmittel?
- Erfüllung von Betriebsmittelanforderungen
- Benutzung von Betriebsmitteln abbrechen
- Anforderungen, die mit verschiedenen Programmen oder Benutzern im Konflikt stehen, vermitteln

---

Kapitel 2

1. Nennen Sie mindestens 4 Charaktere für Rechensysteme der ersten Generation!

- Programmierer = Operateur
- Eingabe über Lochkarten, Steckkarten usw.
- Sequentielle Nutzung
- Schlechte Ausnutzung der Ressourcen
- Kein Betriebssystem

2. Was versteht man unter einem Stapelverarbeitungssystem (Batch-System)?

Das Bach-System ist charakteristisch für die 2.Generation von Rechensystemen und verbessert die Auslastung der Ressourcen erheblich. Ohne dieses System würde ein großer Teil der Rechenzeit durch die Wegzeiten des Bedieners durch die Rechenräume verschenkt, denn der Bediener müsste die Lochkarten holen, die Programme zur Ausführung bringen, auf die Ausgabe warten und das Ergebnis vom Drucker abholen, erst dann könnte der nächste Lochkartenstapel abgearbeitet werden.
Das Batch-System sieht vor, dass ein hoher Anteil von Programmen („Jobs“) auf den Lochkarten gesammelt wird und zu einem kleinen Rechner gebracht wird, wo die Jobs auf Magnetband eingelesen werden. Dieses wird zurückgespult, zu einem anderen Rechner gebracht, und mit Hilfe eines Programms werden alle Job abgearbeitet. Anstatt die Ergebnisse gleich zu drucken, werden diese auf ein zweites Band geschrieben. Das fertige Ausgabeband wird herausgenommen und zu einem anderen kleinen Rechner gebracht, der das Ergebnis druckt. Das Eingabeband wird durch ein zweites Eingabeband, das vorher am ersten Rechner beschrieben wurde, ersetzt. Der Zyklus kann von vorne beginnen. Durch Trennung der Ein und Ausgabe erreichte man also eine wesentlich höhere Auslastung der Ressourcen.

3. Nennen Sie 3 Charaktere für Betriebssysteme der zweiten Generation!

Charaktere Betriebssysteme….

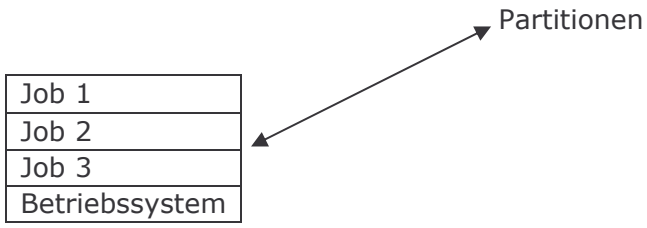
- Betriebssystem war meist FMS oder IBSYS
- Betriebssystem lief am Rechner, der die Jobs zur Ausführung brachte
- Betriebssystem wurde geladen und brachte Jobs zur Ausführung

Charaktere der Rechensysteme….

- Steigerung der Auslastung durch Vorbereitung eines Stapels von Jobs
- Trennung Programmierer/Bediener
- Verbesserung durch Trennung der Ein/Ausgabe

4. Was versteht man unter Multiprogramming, Spooling und Time-Sharing?

- Multiprogramming: Konzept, bei dem die CPU nicht auf Beendigung der Ein/Ausgabe warten muss, sondern auch während Ein/Ausgabe beschäftigt wird. Speicher wird in Teile zerlegt, sodass jeder Job eigene Partition bekommt. Wenn ein Job auf Beendigung der Ein/Ausgabe wartet, kann die CPU nächsten Job bearbeiten. Mehrer Jobs können quasi gleichzeitig behandelt werden. Das Betriebssystem teilt der CPU die Jobs zu.



- Spooling: Jobs werden auf Platten gespeichert. Nach Beendigung eines laufenden Jobs lädt das Betriebssystem einen neuen in die frei gewordene Platte und führt in aus.

- Variante des Multiprogrammings, bei der jeder User On Line Zugang zum System über ein Terminal hat. Die CPU kann den aktiven Usern einen interaktiven Dienst bieten, im Hintergrund aber Stapeljobs verarbeiten.

5. Nennen Sie vier Charaktere für Betriebssysteme der vierten Generation!

- Enorme Schritte bei Leistung/Preis/Abmessungen
- Enorme Entwicklung der PC-Betriebssysteme
- Dezentralisierung und Client-Server setzen sich durch
- Verteilte Betriebssysteme und solche mit Mehrprozessor-Fähigkeit kommen dazu
- Verteilte Systeme bekommen Software-Unterstützung für Middleware (‚third-party')
- Middleware-Komponenten gehören zum Standardlieferumfang von Betriebssystemen

---

Kapitel 3

1. Wie unterscheiden sich Mainframe Betriebssysteme von anderen?

Betriebssysteme für Großrechner, also Rechnersysteme die ganze Räume ausfüllen und heutzutage noch häufig in Rechenzentren großer Firmen und Behörden zu finden sind, unterscheiden sich ganz wesentlich von gängigen Personal Computer Betriebssystemen. Der
signifikanteste Unterschied besteht wohl in der Ein-/Ausgabe-Kapazität der Hardware und damit in der Ein-/Ausgabe-Verwaltung des Betriebssystems. Großrechner mit einer Kapazität
von mehreren hundert Festplatten und tausenden von Gigabyte an Daten sind eher die Regel
den die Ausnahme. In letzter Zeit erfahren Großrechner ein Come-back als High-End Webserver oder Business-to-Business (B2B) Server.
Betriebssysteme für Großrechner sind stark auf die gleichzeitige Bearbeitung mehrerer Jobs,
von denen die meisten ein erstaunlich hohes Ein-/Ausgabe-Verhalten aufweisen, ausgelegt.

2. Welche drei Dienste werden von Mainframe Betriebssystemen implementiert?

Typischerweise implementieren Mainframe Betriebssysteme drei unterschiedliche Arten von Services:

- Batch-Operating
  Abarbeiten von Routine-Jobs ohne irgendeinen interaktiven Eingriff.
- Transaction-Processing
  Handhabung einer großen Anzahl von sehr kleinen Anforderungen.
  Jede dieser Anforderungen ist sehr klein, das System muss aber z.B. Tausende pro
  Sekunde abarbeiten.
- Time-Sharing
  Erlaubt mehreren entfernten Benutzern das gleichzeitige Ausführen von Jobs auf dem
  selben Rechensystem.

Die genannten Services stehen oft in einer engen Beziehung zueinander und Betriebssysteme
für Großrechner führen meist alle parallel aus. Ein Bespiel für ein aktuelles Großrechner-Betriebssystem ist OS/390, ein Nachfolger von OS/360.

3. Was unterscheidet Server von Personal Computer Betriebssystemen?

3.2. Server Betriebssysteme
Server Betriebssysteme laufen auf Server Maschinen, welche entweder sehr leistungsfähige
Personal Computer, Workstations oder sogar Großrechner sind. Server bedienen mehrere Benutzer gleichzeitig über ein Netzwerk und erlauben diesen Hardware und Software-Ressourcen gemeinsam zu benutzen. Sie unterstützen meist Druck-, Datei- und Webdienste.

3.4. Personal Computer Betriebssysteme
Personal Computer Betriebssystem zeichnen sich durch eine sehr gute Benutzerschnittstelle
aus. Typische Aufgaben auf einem Personal Computer sind Textverarbeitung, Tabellenkalkulation,
Internet-Zugang etc. Es handelt sich hierbei um die bekannteste Form eines Betriebssystems. Technisch gesehen stellen Personal Computer Betriebssysteme meist eine
Teilmenge der Funktionen von Server Betriebssystemen zur Verfügung.

Unterschied:
Server Betriebssysteme sind dafür ausgelegt eine viel höhere Anzahl von Anforderungen gleichzeitig zu bearbeiten. Dies erfordert meist auch viel leistungsfähigere Hardware. PC-Betriebssysteme besitzen meist weniger Funktionen, sind allerdings benutzerfreundlicher.

4. Nennen Sie die wichtigsten Designaspekte für Multiprozessor Betriebssysteme!

- **Art der Implementierung**
- **Welche Ressourcen können gemeinsam genutzt werden**
- **Wie die Prozessoren verbunden werden**

Aufgaben des BS im Wesentlichen gleich wie für Ein-Prozessor-Systemen.Abweichung nur bei Scheduling und Synchronisation von Abläufen.

Funktionen derselben Instanz eines BS können prinzipiell auf allen Prozessoren gleichzeitig ausgeführt werden. Alle Funktionen arbeiten auf gemeinsamen Datenstrukturen (z.B. Prozess-Warteschlangen, Dateitabellen, Gerätetabellen) und müssen für eine korrekte Synchronisation (z.B. wechselseitiger Ausschluss) sorgen.

5. Was versteht man unter harter und was unter weicher Echtzeit?

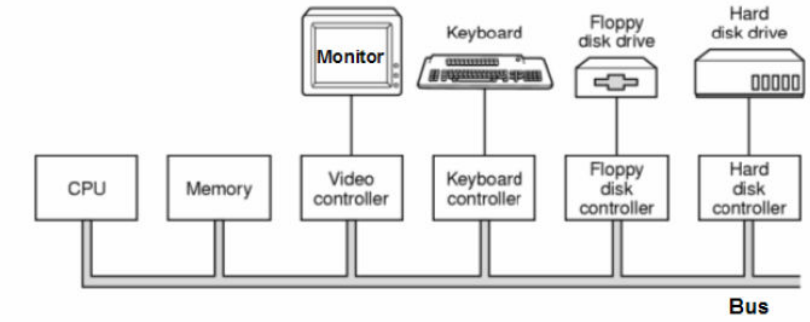
Bei harter Echtzeit muss eine Aktion in einer gewissen Zeit oder einem gewissen Zeitintervall absolut abgeschlossen sein (echte „Deadlines"). Bei weicher Echtzeit ist ein überschreiten der Zeitgrenzen zwar nicht erwünscht, aber akzeptabel. Echtzeit Betriebssysteme unterliegen daher im Gegensatz zu den bisher genannten Betriebssystemen grundsätzlich einem anderen Design in ihrer Architektur.

Kapitel 4

1. Nennen sie die Hauptkomponenten eines einfachen Rechnersystems!

Die Hauptkomponenten eines einfachen Rechensystems sind:

- CPU
- Speicher
- Video Controller
- Keyboard Controller
- Floppy Disk Controller
- Hard Disk Controller

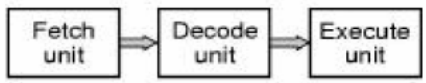


Der Prozessor, der Speicher und die Ein-/Ausgabemodule (Video-, Keyboard-, Floppy disk- und Hard Disk Controller) sind mittels eines Systembus miteinander verbunden. Dieser ermöglicht eine Kommunikation zwischen den Komponenten.

2. Wie ist der Grundzyklus eines Prozessors uns was ist eine Pipeline?

Der Grundzyklus eines jeden Prozessors ist das Holen einer Instruktion aus dem Speicher, das Dekodieren der Instruktion, um den Instruktionstyp und die Operanden festzustellen, das Ausführen der Instruktion und schließlich das Fortfahren mit der nächsten Instruktion. Auf diese Weise werden Programme ausgeführt.

Jeder Prozessortyp besitzt dabei seinen eigenen Befehlssatz, was heißt, dass Programme für einen bestimmten Prozessortyp im Allgemeinen nicht auf einem anderen Prozessortyp laufen. Da der Speicherzugriff, um eine Instruktion zu holen, wesentlich langsamer ist als die Prozessorverarbeitungszeit für diese Instruktion, besitzen alle Typen von Prozessoren Register in denen sie Schlüsselvariable und temporäre Ereignisse ablegen.
Moderne Prozessoren besitzen Einrichtungen um mehr als eine Instruktion gleichzeitig bearbeiten oder ausführen zu können. Bei dem so genannten Pipeline-Konzept besitzt der Prozessor mehrere separate Einheiten für das holen, Dekodieren und Ausführen.
Folgendes Bild zeigt die Funktionsweise einer einfachen Pipeline:



3. Erläutern sie die Begriffe Benutzermodus, Kernelmodus und Trapping!

Fast alle Prozessoren besitzen zwei Betriebsarten, nämlich den Benutzer**-** und den Kernelmodus. Ein ganz spezielles Register (PSW - Program Status Word) im Prozessor enthält spezielle Steuerbits welche die Betriebsart festlegen. Wenn der Prozessor im Kernelmodus betrieben wird, so kann dieser sämtliche Befehle des Instruktionssatzes benutzen und alle Eigenschaften der Hardware nutzen. Das Betriebssystem selbst wird im Kernelmodus ausgeführt und besitzt daher uneingeschränkten Zugriff auf die gesamte Hardware. Anwenderprogramme laufen jedoch im Benutzermodus. Diese können nur einen Teil des Befehlssatzes sowie der vorliegenden Hardwareeigenschaften verwenden. Im Allgemeinen sind alle Befehle für die Ein-/Ausgabe und die Speicherverwaltung im Benutzermodus nicht zugänglich. Das Umschalten in den Kernelmodus durch das Statusbit im PSW Register ist ebenso wenig erlaubt. Wenn Anwenderprogramme die Dienste des Betriebssystems in Anspruch nehmen wollen müssen sie einen Systemaufruf durchführen, welcher mit dem Kernel kommuniziert, und das Betriebssystem aufruft. Dieser Vorgang wird als „TRAPPING" bezeichnet. Der TRAP Befehl wechselt vom Anwender- in den Kernelmodus und startet das Betriebssystem. Nach dem Beenden des Systemaufrufes wird die Kontrolle an das Anwenderprogramm zurückgegeben.

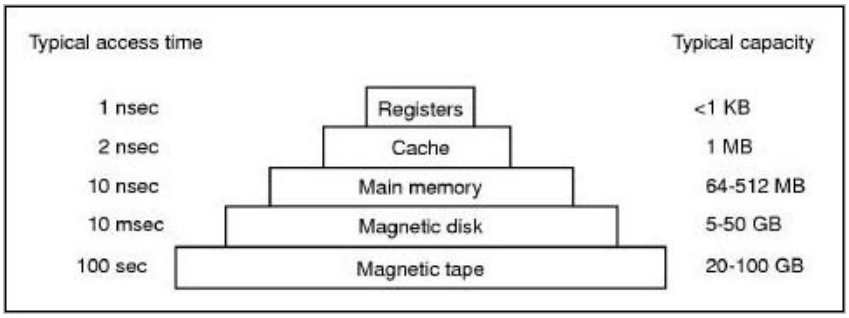
4. Beschreiben Sie die Speicherhierarchie eines Rechensystems!

- Speicher ist nach dem Prozessor die 2.wichtigste Komponente

Speicher sollte:
- schnell
- üppig vorhanden
- billig sein

Diese drei Anforderungen treffen auf keinen am Markt erhältlichen Speicher zu
- daher wird das Speichersystem eines Rechners hierarchisch ausgelegt.

Typische Speicherhierarchie für die meisten Computersysteme:



Charakteristisch für alle Systeme:

- umso langsamer die Zugriffszeit, desto höher die Kapazitäten und niedriger die Kosten (pro Bit).
- Bei schnellerer Zugriffszeit sinken die Kapazitäten und die Kosten steigen.

5. Was ist ein Gerätetreiber und auf welche Arten kann dieser installiert werden?

Da viele Ein/Ausgabe Geräte eine starke Wechselwirkung mit dem BS aufweisen, ist deren Steuerung eine nicht zu unterschätzende Angelegenheit, daher muss jeder Controller dem BS eine vereinfachte Schnittstelle bieten. Da die meisten Controllertypen unterschiedlich sind, wird dazu jeweils individuell angepasste Software benötigt. – Diese Software wird als Gerätetreiber bezeichnet. Ein Controller benötigt für jedes BS einen separaten Treiber.

- Ein Gerätetreiber ist eine für jeden Controller individuelle Software, die eine vereinfachte Schnittstelle zwischen Controller und Betriebsystem darstellt.
- Für jedes BS ein Gerätetreiber
- Der Gerätetreiber muss in den Kernelmodus eingebunden werden

Einbinden in den Kernelmodus:

- erneutes Linken des Kernels mit anschließendem Neustart
- Eintragen des Gerätes in eine Gerätedateiliste des BS mit anschließendem Neustart
- Dynamisches Laden und in Betrieb nehmen des Gerätetreibers (on-the-fly installation)

Geräte die *hot-plugging* unterstützen (zB.: USB, Firewire,..) erfordern *dynamisch ladbare Treiber*.

Kapitel 5

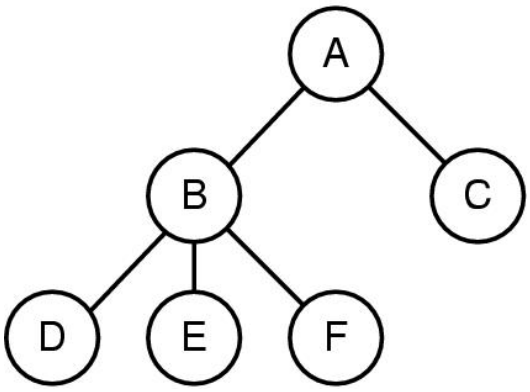
1. Beschreiben Sie kurz den Systemablauf bei einem Prozesswechsel!

Das Betriebssystem entscheidet periodisch einen Prozess zu stoppen und einen anderen Prozess zu starten.
Alle Informationen, eines Prozesses, welcher temporär suspendiert wird, werden in der sogenannten Prozesstabelle zwischengespeichert, um ihn zu einem späteren Zeitpunkt wieder starten zu können.

2. Was versteht man unter einer Prozesstabelle, einem Prozessstrukturbaum und Interprozess-Kommunikation?

Prozesstabelle: ist eine Tabelle des Betriebssystems, in der alle Informationen eines jeden Prozesses (bis auf den Inhalt des eigenen Adressraumes) gespeichert werden. Dies muss geschehen damit der Prozess genau an derselben Stelle wieder fortgesetzt werden kann, wo er gestoppt wurde.

Prozessstrukturbaum: entsteht, wenn ein Prozess einen oder mehrere andere Prozesse, so genannte Kindprozesse, erzeugen kann und diese Prozesse ihrerseits ebenfalls Kindprozesse erzeugen können.

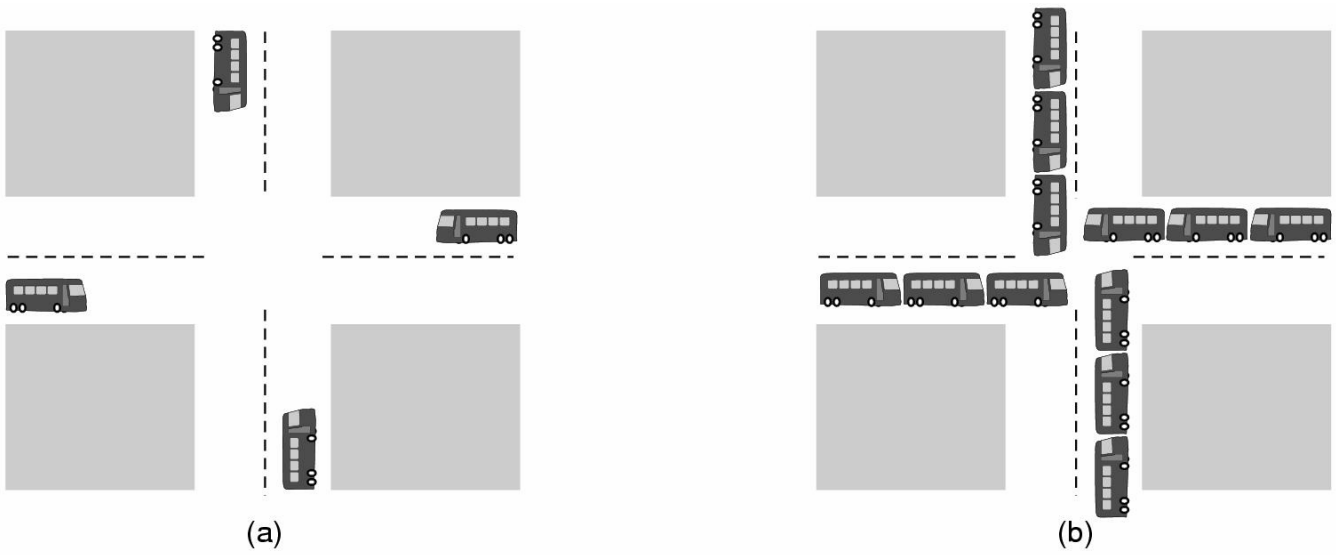


Interprozess-Kommunikation bezeichnet die Kommunikation zwischen verwandten Prozessen die untereinander kooperieren. Dies kann geschehen um eine gemeinsame Aufgabe zu lösen und um sich bei der Erledigung der Aufgabe abzustimmen.

3.Erläutern Sie den Begriff „Deadlock" anhand eines Beispiels!

Ein Deadlock (Verklemmung) ist eine Situation wenn mehrere Prozesse miteinander interagieren und es zu einem Kommunikationszustand kommt, aus dem es keinen sinnvollen Fortschritt gibt.

Beispiel: Vier Busse nähern sich einer gleichrangigen, ungeregelten Kreuzung. Hinter diesen folgen weitere Busse. Wenn alle Busse gleichzeitig die Kreuzung erreichen kommt es zu einer Blockade, da die Busse weder weiter- noch zurück fahren können. In diesem Fall ist kein Fortschritt mehr erzielbar.



4. Was versteht man unter dem Adressraum eines Prozesses? Nennen Sie die Unterschiede zwischen Prozess- und Dateisystemhierarchien!

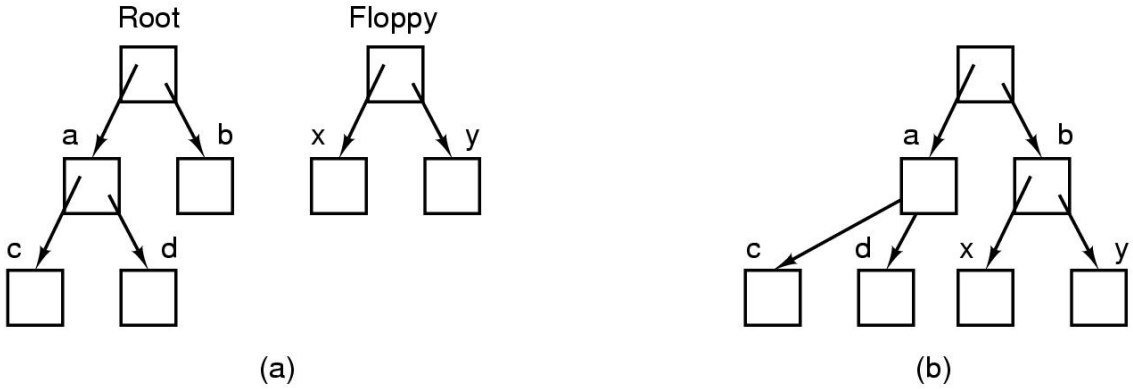
Prozesse verwenden normalerweise einen bestimmten Adressraum (0 bis MAX). Der Adressraum ist im einfachsten Fall kleiner als der physikalische Hauptspeicher, also wird der Prozess vollständig im Hauptspeicher gehalten damit der Adressraum gänzlich genützt werden kann. Wenn der Adressraum nicht in den physikalischen Hauptspeicher passt, dann wird ein Teil durch die Technik des „Virtual Memory" auf die Festplatte ausgelagert.

Prozesshierarchien: sind meist nicht sehr tief und bestehen nur relativ kurze Zeit
Dateisystemhierarchien: können eine sehr hohe Tiefe aufweisen können und meist lange bestehen (bis zu Jahren).

Benutzerrechte und Schutzmechanismen stellen weiter Unterschiede zwischen Prozessen und Dateien dar.

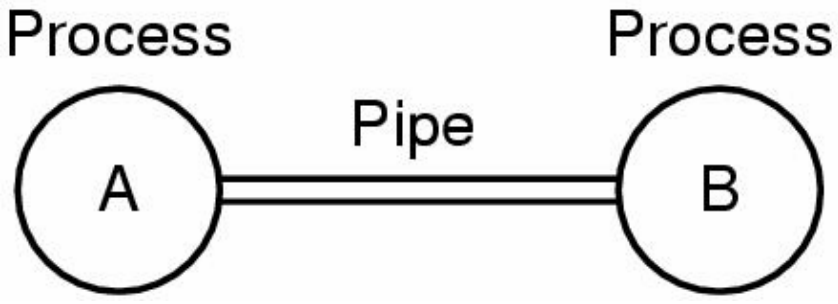
5. Erläutern Sie das Konzept des Dateisystemverbundes! Was ist eine Pipe?

1. Dateisystemverbund (=mounted file system)
   Zwei nicht in Relation stehende Dateisysteme können verbunden werden, indem ein Dateisystem (genauer: die Wurzel dieses Dateisystems) an ein anderes Dateisystem angebunden wird (z.B. Disketten- an Festplattenverzeichnis)



a) Vor dem „Mounten": Dateien vom Disketten-LW sind nicht über Root zugänglich
b) Nach dem „Mounten": Dateien sind nun Teil der Datenhierarchie

Pipe
Eine Pipe ist quasi eine Pseudo-Datei(erfüllt ähnliche Aufgaben wie eine Datei, ist aber in Wirklichkeit keine), die zum Verbinden von zwei Prozessen zum Zweck der Interprozess-Kommunikation (Kommunikation zwischen Prozessen) benutzt wird.



Möchten also Prozess A und B miteinander kommunizieren, so müssen sie vorher eine Pipe einrichten. Wenn Prozess A an B Daten sendet, so schreibt er diese in

die Pipe, als wäre es eine normale Ausgabe in eine Datei. Prozess B liest dann die Daten von der Pipe, als wäre es eine normale Eingabe von einer Datei.

Kapitel 6

1. Beschreiben Sie die Abläufe bei einem Systemaufruf!

Anwenderprogramme kommunizieren mit dem Betriebssystem und fordern von diesem Dienste an, indem sie Systemaufrufe tätigen. Jedem Systemaufruf entspricht eine Bibliotheksprozedur, die vom Anwenderprogramm aufgerufen werden kann. Diese Prozedur
schreibt die Parameter eines Systemaufrufes an angegebene Stellen (z.B. Maschinenregister),
und führt dann einen TRAP-Befehl aus, um das Betriebssystem zu starten.

2. Was passiert nach dem Aufruf eines TRAP-Befehls?

Nachdem das BS nach einen TRAP-Befehl die Kontrolle übernommen hat, prüft es die Parameter auf
ihre Gültigkeit und führt (bei Richtigkeit) die angeforderte Funktion aus. Nach erfolgreich beendeter Arbeit, schreibt das Betriebssystem einen Statuscode in ein Register, welches den
Erfolg oder Misserfolg der Ausführung dokumentiert, und führt den „return-from-TRAP"-Befehl aus, um die Kontrolle an die Bibliotheksprozedur zurückzugeben. Diese returniert dann in üblicher Weise an den Aufrufer den Statuscode in Form eines Funktionswertes.

3. Erklären Sie die Vorgänge bei einem „fork"-Systemaufruf!

Beim Aufruf erzeugt der Befehl eine exakte Kopie des Originalprozesses. Nach dem Aufruf gehen der Vater- und Kindprozess dann eigene Wege. Dies bedeutet, dass während der Ausführung des Befehls alle Variablen die gleichen Werte besitzen. Nach dem Kopieren der Daten des Vaterprozesses können allerdings in beiden Prozessen die Variablen verändert werden, ohne dass der andere Prozess davon betroffen ist.
Der „fork"-Systemaufruf returniert einen Wert, der für den Kindprozess null ist und für den Vaterprozess die Prozessidentifikationsnummer (PID, „process identifier") des Kindprozesses darstellt. Unter Verwendung der returnierten PID können die Prozesse nun feststellen, welcher der Vaterprozess und welcher der Kindprozess ist.

4. Erläutern Sie die wichtigsten Systemaufrufe für das Dateimanagement!

„open"-Systemaufruf (um eine Datei zu lesen oder zu beschreiben):
Der Aufruf spezifiziert im Allgemeinen den Namen der zu öffnenden Datei, im Weiteren ob es sich um einen absoluten oder relativen Pfadnamen handelt und einige Steuerflags wie etwa Lese- oder Schreibberechtigung oder beides.

„create"-Systemaufruf (um Dateien zu erzeugen):
Der returnierte Dateideskriptor wird zum Lesen und Beschreiben der Datei verwendet („read"- und „write"- Systemaufrufe).

„close"-Systemaufruf (um Dateien zu schließen):
Gibt den Dateideskriptor für ein Wiederöffnen frei.

„read"- und „write"-Systemaufruf (um Dateien zu lesen und zu schreiben)
Bei den meisten Programme nur sequentielle Lese- und Schreiboperationen.
Für diese Anforderung stehen Systemaufrufe zur Positionierung des Dateizeigers zur Verfügung. Ein Dateizeiger ist ein Zeiger, der mit einer Datei assoziiert ist und die aktuelle Position innerhalb einer Datei angibt. Bei Lese- und Schreiboperationen wird die Position des Dateizeigers automatisch mitverwaltet. Mit den Systemaufrufen zur Positionierung des Dateizeigers kann die Position des Dateizeigers direkt verändert werden, d.h., dass darauffolgende Lese- oder Schreiboperationen quasi irgendwo in der Datei stattfinden können.

5. Was versteht man unter „Linking"? Was bewirkt der „kill"-Systemaufruf?

Unter *Linking* versteht man einen Systemaufruf für Verzeichnismanagement. *Linking* bezweckt, dass Verzeichnisse oder Dateien unter mehreren Namen und aus unterschiedlichen Verzeichnissen heraus angesprochen werden können, obwohl sie physikalisch nur einmal vorhanden sind. Im Gegensatz zu Kopien werden Änderungen an einer gemeinsamen Datei sofort für alle sichtbar.

- Systemaufruf für Verzeichnismanagement
- Erzeugt echte Links
- Es existiert nur eine physikalische Datei / Verzeichnis
- Änderungen sofort für alle sichtbar

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Der *kill* – Systemaufruf ermöglicht Anwendern und Anwenderprozessen Signale zu senden. Ist ein Prozess auf den Empfang eines Signals vorbereitet, so kann er es bei Empfang auswerten. Ist er es nicht so terminiert dieses Signal den Prozess.

- der *kill* – Systemaufruf sendet Signale an Prozesse
- wird zum Beenden von Prozessen verwendet

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

1. Beschreiben Sie den Aufbau monolithischer Systeme! Wie kann eine einfache Strukturierung erfolgen?

Am häufigsten werden monolithische Systeme verwendet. Sie haben keine Struktur. Das BS besteht aus Prozeduren, von denen jede eine andere aufrufen kann. Parameter und Ergebnisse stellen die Schnittstellen dar. Alle Prozeduren werden zusammen zu einem ausführbaren Code gelinkt.
Jede Prozedur ist für jede andere sichtbar (kein „Information Hiding")

Eine Basisstruktur könnte wie folgt aussehen:

- Ein Hauptprogramm, welches die angeforderte Dienstprozedur aufruft
- Eine Menge von Dienstprozeduren, die die Systemaufrufe durchführen
- Eine Menge von Hilfsprozeduren, die die Dienstprozeduren unterstützen

2. Nennen Sie das wichtigste Architekturmerkmal bei geschichteten Systemen!

Das BS wird als eine Hierarchie von Schichten organisiert, von denen jede auf Basis der darrunterliegenden Schicht konstruiert ist (Bsp.: THE-System).

So muss sich die obere Schicht z.B. nicht um die Speicherverwaltung, den Mehrprozessorbetrieb kümmern.

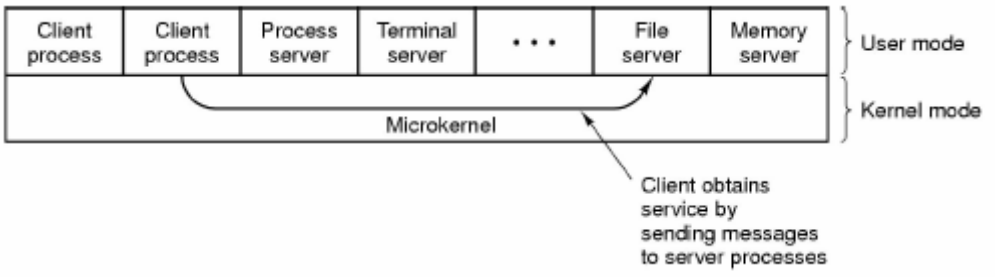
3. Erläutern Sie das Konzept der virtuellen Maschine!

Ziel war es, die Funktionen Mehrprogrammbetrieb und erweiterte Maschine strikt voneinander zu trennen.
Der Kern (Monitor) der virtuellen Maschine kommt direkt auf der Hardware zur Ausführung und stellt den Mehrprogrammbetrieb (mehrere virtuelle Maschinen für die nächst höhere Schicht) zur Verfügung.
Weil jeder dieser virtuellen Maschinen exakte Kopien der tatsächlichen Hardware zur Verfügung stehen, kann auf jeder VM jedes (auch verschiedene) Betriebssysteme arbeiten.
Bsp.: VM/370-System

4. Was ist ein Microkernel, wie funktioniert selbiger und welche Vorteile hat dieses Konzept:

In modernen Betriebssystemen wurde die Entwicklung aufgegriffen, den Code in höhere Schichten und weiter zu verlagern, um so viel wie möglich aus dem Betriebssystem herauszulösen bzw. zu entfernen, sodass ein minimaler Kern überbleibt. Die Aufgabe des Kerns (Microkernel) in diesem Modell ist es, die Kommunikation zwischen dem Client und dem Server durchzuführen.

Durch die Aufteilung des Betriebssystems in Teile, von denen sich jeder nur um einen einzelnen Bereich zu kümmern hat (Prozess-, Datei-, Terminal-, Speicher-Service .etc), wird jeder Teil kleiner und handhabbarer. Weil nun alle Prozesse im Benutzermodus ausgeführt werden können und nicht mehr im Kernelmodus laufen, haben sie keinen direkten Hardwarezugriff mehr. (Vorteil: Fehler im Dateiserver lässt zwar Dateiserver abstürzen betrifft das gesamte System jedoch nicht)
Skizze:



5. Welche Auswirkungen hat es für den Client, wenn der Server bei der Nachrichtenkom-munikation nicht lokal, sondern über ein Netzwerk entfernt läuft! Begründen sie ihre Aussage:

Ein weiterer Vorteil des Client-Server Modells ist seine Anpassungsfähigkeit bei der Verwendung für verteilte Systeme. Wenn ein Client mit einem Server durch versenden einer Nachricht kommuniziert, so muss der Client nicht wissen, ob die Nachricht lokal auf seinem eigenen System bearbeitet wird, oder ob diese zur Bearbeitung über ein Netzwerk an einen entfernten Server geschickt wird. Soweit es den Client betrifft, geschehen in beiden Fällen die gleichen Dinge: eine Anforderung wird versendet und eine Antwort kommt zurück.

Kapitel 1

1. Was sind die drei wichtigen Anforderungen für die langfristige Speicherung von Daten? Was bedeutet in diesem Zusammenhang „Persistenz"?

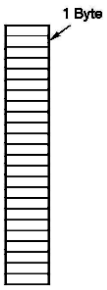
- Es muss möglich sein, sehr große Mengen von Informationen speichern.
- Die Informationen müssen über die Terminierung der Prozesse hinaus bestehen bleiben.
- Es muss für mehrere Prozesse möglich sein, gleichzeitig auf die Informationen zuzugreifen.

Persistenz:
Durch die Speicherung von Informationen auf externe Speichermedien (z.B.: Festplatten) in Einheiten (Dateien), wird die Information nicht von der Erzeugung oder Terminierung eines Prozesses beeinflusst – sie ist dann **persistent**. (*Speicherung von Infos unabhängig von einem Prozess*)

(Sind Informationen nur im –kleinen- Adressraum eines Prozesses abgelegt sind sie:

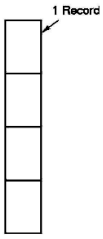
1. *verloren wenn der Prozess terminiert, unerwartet stirbt oder das Rechensystem ausfällt*
2. *nur für einen Prozess verfügbar)*

2. Welche drei Möglichkeiten für die Strukturierung von Dateien gibt es? Erklären Sie kurz die jeweilige Idee dahinter!



1. Die Strukturierung einer Datei durch eine Bytefolge *(unstrukt. Menge von Bytes*)

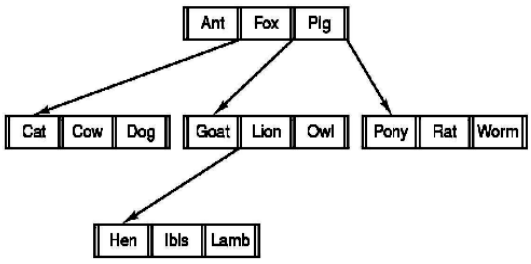
Das Betriebssystem weiß hierbei nichts über Inhalt einer Datei *(kümmert sich auch nicht darum*), sondern sieht nur die **Folge der Bytes** (*dadurch erreicht das BS maximale Flexibilität*). Das **Anwenderprogramm verleiht** der Bytefolge eine **Bedeutung** (*Anwend.progr. kann beliebig auf einer Datei operieren und sie frei benennen - Das BS unterstützt nicht, steht dem aber auch nicht im Weg.).*



2. Strukturierung durch Verwaltung von Byteblöcken (Records)

Die Datei besteht aus einer **Folge von Records** fester Länge *(jeder davon besitzt interne Struktur*). Grundidee: eine Leseoperation **liest** und eine Schreiboperation **überschreibt** einen **ganzen Record** oder fügt einen an.
(Diese Form der Strukt. war in der Zeit der Lochkarten-Speichermedien bedeutsam – heute gibt es dafür kein allgemein gebräuchliches System mehr.)

3. Strukturierung durch Verwaltung eines Baumes als Strukturform



Datei besteht hier als Baum von Records - müssen nicht alle gleich lang sein - von denen

jeder ein **Schlüsselfeld** an einer bestimmten Position enthält. Der Baum ist nach diesen Schlüsselfeldern sortiert → ermöglicht sehr **schnelles Suchen.** Die Grundoperation hierbei ist die Bearbeitung des Records der einen speziellen Schlüssel enthält und nicht des nächsten Records.
(*Außerdem können Records Dateien zugefügt werden → Position entscheidet BS und nicht der Anwender.)*
Verwendung: im Kontext der kommerzielle Detenverarbeitung auf Mainframes

3. Nennen Sie vier Arten von Dateitypen und charakterisieren Sie diese! Was ist der grundlegende Unterschied zwischen ASCII- und Binärdateien?

- Dateien: enthalten die Anwenderdaten
- Verzeichnisse: Systemdateien zur Verwaltung der Struktur eines Dateisystems
- Zeichenorientierte Spezialdateien: stehen in Verbindung mit der Ein/Ausgabe und werden benutzt um serielle Ein/Ausgabegeräte wie Terminals oder Drucker zu emulieren
- Blockorientierte Spezialdateien: werden zur Emulation von externen Speichermedien (z.B.: Festplatten) benutzt.

Grundlegender Unterschied zwischen ASCII und Binärdateien: Binärdateien sind im Gegensatz zu ASCII-Dateien nicht lesbar. Ihre Ausgabe auf einen Drucker liefert ein unverständliches Listing.

4. Beschreiben Sie die drei wesentlichen Eigenschaften des sequentiellen Zugriffs! Welche beiden Verfahren kennen Sie für den wahlfreien Lesezugriff?

Sequentieller Zugriff:

- Lesen der Bytes/Sets nur von Dateianfang an
- Eine beliebige Positionierung innerhalb der Datei ist nicht möglich
- Geeignet für sequentielle Speichertypen (z.B.: Magnetbandspeicher)

Verfahren für den wahlfreien Lesezugriff:

1. Jede Leseoperation gibt an, an welcher Stelle mit dem Lesen begonnen werden soll
2. Die aktuelle Position wird über einen speziellen Befehl festgesetzt → dann kann die Datei von der neuen Position aus sequentiell gelesen werden.

5 Wozu benötigt man Dateiattribute? Nennen Sie mindestens fünf Systemaufrufe für Dateioperationen und beschreiben Sie diese kurz!

1. Dateiattribute
   Jede Datei besitzt einen Namen und beinhaltet ihre Daten. Zusätzlich verbinden alle Betriebssysteme weiter Informationen mit jeder Datei (z.B. Datum, Uhrzeit der Erzeugung, Dateigröße, etc.). Diese zusätzlichen Informationen werden als Dateiattribute bezeichnet. Mögliche Attribute:

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Wozu benötigt man die Attribute jetzt?
Diese Frage beantwortet eigentlich fast ein Blick auf die Tabelle.
z.B. ‚Time'-Attribut: Durch unterschiedliche Zeiten wird festgehalten, wann eine Datei erzeugt, zuletzt genutzt, verändert worden ist. Hilfreich für unterschiedliche Zwecke, z.B. wenn beim C-Programmieren die Quellcodedatei (*.c) jünger als die Objektdatei(*.obj) ist, muss neu kompiliert werden, um zu den aktuellen Ergebnissen zu kommen.
Außerdem können auch Dateien nach Attributen gesucht, sortiert, etc. … werden.

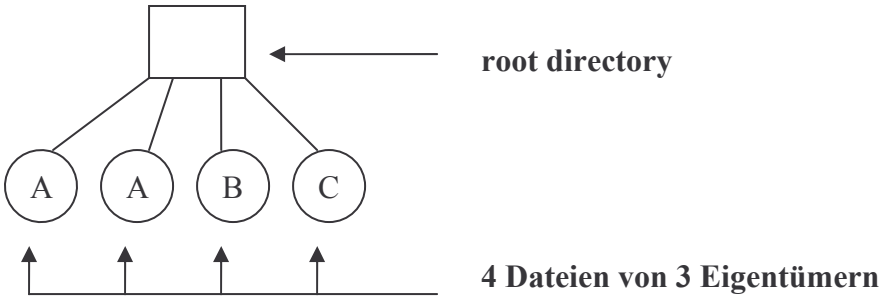
2. Systemaufrufe für Dateioperationen

CREATE	Eine (gültige) Datei ohne Daten wird erzeugt;
DELETE	Eine nicht mehr benötigte Datei wird gelöscht;
OPEN	Bevor eine Datei benutzt werden kann, muss diese geöffnet werden;
CLOSE	Schließt eine nicht mehr benötigte Datei;
READ	Systemaufruf zum Lesen von Daten aus einer Datei;
WRITE	Systemaufruf zum Schreiben von Daten in eine Datei;
APPEND	Variante von WRITE; Daten werden am Ende einer Datei angefügt;
SEEK	Positioniert den Dateizeiger in einer Datei an die Leseposition;
GET ATTRIBUTES	Liefert die Dateiattribute einer Datei;
SET ATTRIBUTES	Setzt die Dateiattribute einer Datei;
RENAME	Ermöglicht eine Umbenennen der Datei;

Kapitel 2

1. Beschreiben Sie das Schema eines einstufigen Verzeichnissystems mit Skizze! Welche Vor- und Nachteile hat dieses?

Skizze: einstufiges Verzeichnissystem:



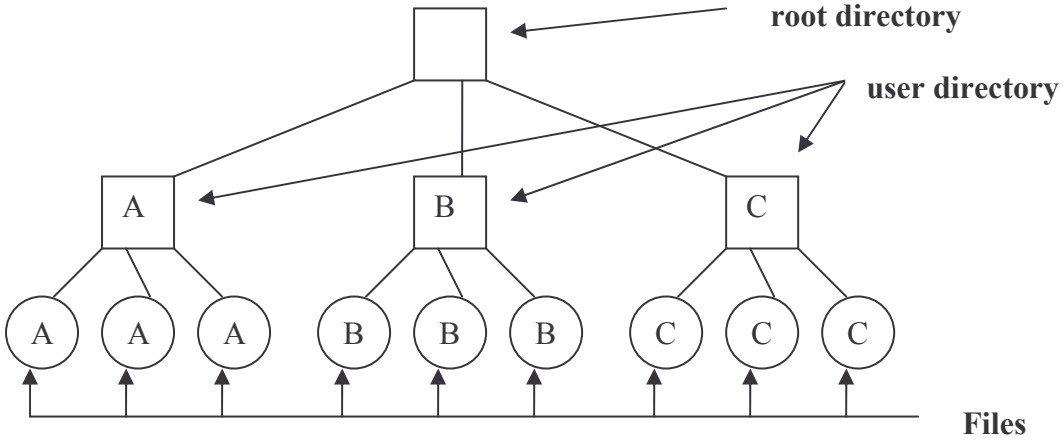
Vorteile des einstufigen Verzeichnissystems:

- Einfachheit
- Dateien können sehr schnell gefunden werden

Nachteile:
Benennung der Dateien durch verschieden Benutzer → es kann zu ungewollten Effekten kommen (keine gleichen Dateinamen möglich) → daher wird dieses Schema bei Mehrbenutzersystemen nicht mehr verwendet (man findet es aber nach wie vor bei einfachen, kleinen Systemen im Embedded Bereich).

2. Beschreiben Sie das Schema eines zweistufigen Verzeichnissystems mit Skizze! Welche Vor- und Nachteile hat dieses und warum verwendet man ein eigenes Verzeichnis für Systemprogramme?

Skizze: zweistufiges Verzeichnissystem



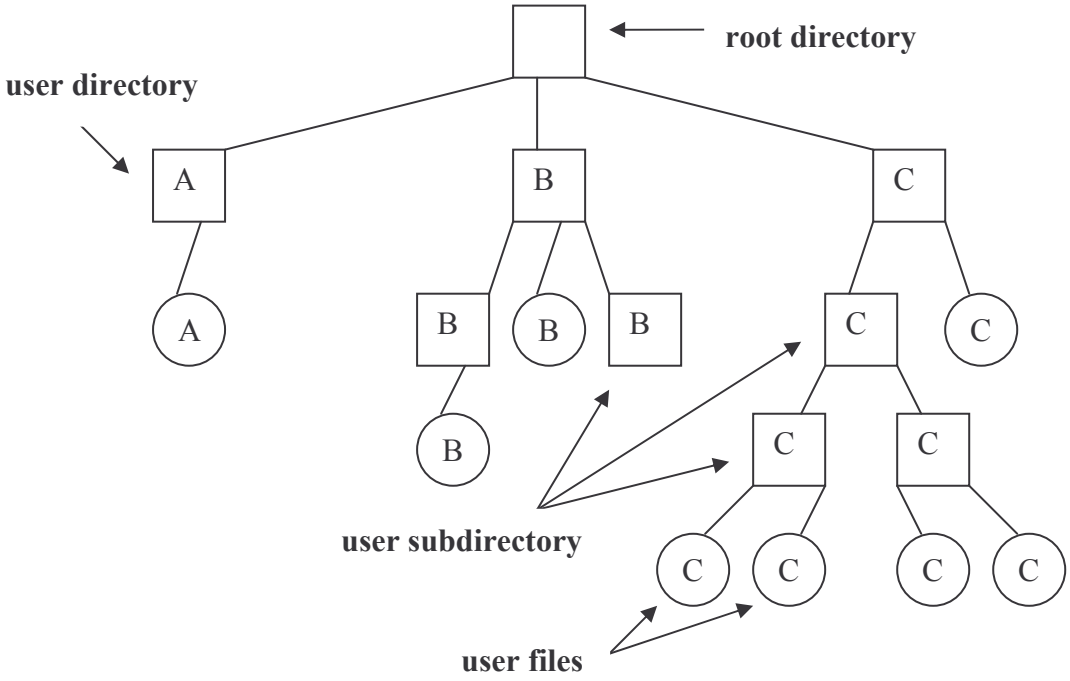
Vorteile:
Dateien eines Benutzers kommen mit Dateien eines zweiten Benutzers nicht in Konflikt → Dateien von verschiedenen Benutzern können gleich benannt sein

Nachteile:
Login (mit Benutzername und Passwort) ist notwendig, da das System sonst nicht weiß, in welchem Verzeichnis es suchen muß, wenn ein bestimmter Benutzer eine Datei öffnen will.

Es wird ein **eigenes Verzeichnis für Systemprogramme** angelegt, da dies von allen Benutzern benötigt wird. Wenn es bei einem bestimmten Benutzer im Verzeichnis liegen würde, könnte es ein anderer Benutzer nicht verwenden (kein Zugriff). Kopien des Verzeichnis in jeden Benutzer wäre ineffizient.

3. Beschreiben Sie das Schema eines hierarchischen Dateisystems mit Skizze! Warum benötigt man solche Systeme (Motivation)?

Skizze: hierarchisches Dateisystem



Bei Benutzern mit vielen Dateien ist es notwendig diese zu gruppieren. Das hierarchische Dateisystem (=Baum von Verzeichnissen) ermöglicht dem Benutzer, dass er eigene Verzeichnisse anlegen kann um seine Dateien auf eine natürliche Weise zu gruppieren.

4. Nennen Sie die zwei gebräuchlichsten Methoden zur Pfadangabe und erläutern Sie diese! Was ist das Konzept des Arbetisverzeichnisses?

Erste Methode:
Jeder Datei wird ein **absoluter Pfadname** zugewiesen (besteht aus Pfad vom Wurzelverzeichnis ausgehend bis zur Datei), Trennzeichen sind systemabhängig (zb.: /, \, >)

Zweite Methode:
- Jeder Datei wird ein relativer Pfadname zugewiesen.
- Er wird in Verbindung mit dem **Konzept des Arbeitsverzeichnisses** benutzt.
- Benutzer zeichnet ein Verzeichnis als sein Arbeitsverzeichnis → alle Pfadnamen, die nicht im Wurzelverzeichnis beginnen werden relativ zum Arbeitsverzeichnis benutzt.

5. Warum ist der Wechsel des Arbeitsverzeichnisses durch eine Bibliotheksprozedur kritisch?

In den meisten Systemen besitzt jeder Prozess sein eigenes Arbeitsverzeichnis, sodass wenn ein Prozess sein Arbeitsverzeichnis wechselt und später terminiert, keine weiteren Prozesse betroffen werden und keine Spuren des Wechsels hinterher im System verbleiben. Bei dieser Art ist es immer absolut sicher für einen Prozess, sein Arbeitsverzeichnis zu wechseln, wann immer er es will.

Auf der anderen Seite, wenn eine Bibliotheksfunktion das Arbeitsverzeichnis wechselt und nach ihrer Abarbeitung den Wechsel nicht rückgängig macht, wird das restliche Programm nicht vernünftig Arbeiten können, da die Annahme darüber, wo das Arbeitsverzeichnis ist, ungültig geworden ist. Aus diesem Grund wechseln Bibliotheksfunktionen äußerst selten das Arbeitsverzeichnis. Sollte es jedoch einmal notwendig sein, so kehren sie vor Abschluss wieder in das ‚Ausgangs'-Arbeitsverzeichnis zurück.

Was bedeuten die Punkt und Punkt-Punkt Einträge in einem Verzeichnis?
Die meisten Betriebssysteme die ein hierarchisches Dateisystem unterstützen, haben zwei besondere Einträge in jedem Verzeichnis, nämlich . und .. (Punkt und Punkt-Punkt). Punkt referenziert auf das aktuelle Verzeichnis und Punkt-Punkt referenziert auf das darüberliegende Verzeichnis (Elternverzeichnis). Die beiden Verzeichniseinträge vereinfachen das Navigieren im Verzeichnisbaum bzw. die Pfadspezifikation erheblich.

Kapitel 1

1) Was versteht man unter einem Prozess? Dürfen bei der Programmierung a priori Annahmen über den zeitlichen Verlauf des Programms gemacht werden (Begründung)?

Ein Prozess ist im Prinzip ein sich in Ausführung befindliches Programm. Ein Prozess besteht aus den Daten des Programms, dem Stack (lokaler Datenspeicher), dem Programmzähler, dem Stackpointer, anderen Registern und vielen anderen Informationen die zur Ausführung des Programms benötigt werden.

A priori Annahmen über den zeitlichen Verlauf dürfen während der Programmierung nicht gemacht werden, da beim Wechsel zwischen mehreren Prozessen auf dem Prozessor die Ausführgeschwindigkeit der Berechnung eines Prozesses nicht gleichmäßig ist, und möglicherweise selbst dann nicht mehr reproduziert werden kann, wenn die selben Prozesse nochmals gestartet werden.

2) Welche vier grundsätzlichen Ereignisse gibt es, die das Entstehen von Prozessen verursachen? Erläutern Sie diese kurz!

1. Systeminitialisierung

Beim booten des Betriebssystems werden typischerweise mehrere Prozesse erzeugt.

- Vordergrundprozesse
  Prozesse für die Interaktion mit dem Benutzer
- Hintergrundprozesse (Dämonenprozesse)
  Prozesse die nicht mit dem Benutzer assoziiert werden.

2. Ausführung eines Systemaufrufes zur Prozesserzeugung aus einem Laufenden Prozess heraus

Neue Prozesse werden von laufenden Prozessen durch einen Systemaufruf erzeugt, um z.B: Aufgaben zu verteilen
(besonders hilfreich bei zwar zusammenhängend, zugleich aber auch relativ unabhängig interagierenden Prozessen)

3. eine Benutzeranforderung für einen neuen Prozess

In interaktiven Systemen kann der Benutzer Programme durch Eingabe auf der Kommandozeile oder durch Klick auf ein Icon starten.

Bei kommandobasierten UNIX Systemen (die X-Windows benutzen)übernimmt der Prozess das Fenster, in dem er gestartet wurde.
Bei Win32 Systemen besitzt der Prozess kein eigenes Fenster, kann sich aber eines oder mehrere erzeugen.

4. die Initialisierung eines neuen Prozesses aus einem Batchjob

Nur auf Batch Systemen großer Mainfraims.
Benutzer können Batchjobs an das System aufgeben. Das Betriebssystem entscheidet schließlich ob genügend Ressourcen vorhanden sind um einen anderen Job zu starten, erzeugt einen neuen Prozess und exekutiert in diesem den nächsten Job aus dem Eingabe-Queue.

3) Durch welche Ursachen kann ein Prozess beendet werden? Erläutern Sie diese kurz!

1. Normale Beendigung (freiwillig)

Prozesse werden im Allgemeinen beendet, wenn sie ihre Aufgabe erledigt haben. Hierfür stehen spezielle Systemaufrufe zur Verfügung.

2. Beendigung durch Fehler (Fehlerbehandlung, freiwillig)

Wenn schwere, aber vorhersehbare Fehler auftreten, die durch das Programm abgefangen werden können, beendet sich der Prozess mit einem *Exit* in Folge der Fehlerbehandlungsroutine kontrolliert selbst.

z.B: Compiler mit Datei aufgerufen die nicht existiert, etc.

3. Schwerer Fehler (nicht handhabbar, unfreiwillig)

Auf Grund eines Bugs im Programm.

z.B: Ausführung einer illegalen Instruktion, Referenzieren eines ungültigen Speicherbereichs, Division durch 0, etc.

In manchen Systemen können Prozesse auch solche Fehler selbst behandeln, indem vorher dem System dieses bekannt gegeben wird, und der Prozess im Fehlerfall durch entsprechende Signale (*events*) informiert wird.

4. Terminierung durch anderen Prozess (unfreiwillig)

Durch einen Systemaufruf zur Terminierung des Prozesses durch einen anderen Prozess mit entsprechender Berechtigung.
Systemaufruf in UNIX Systemen => *kill*
Systemaufruf in Win32 Systemen => *Terminate Process*

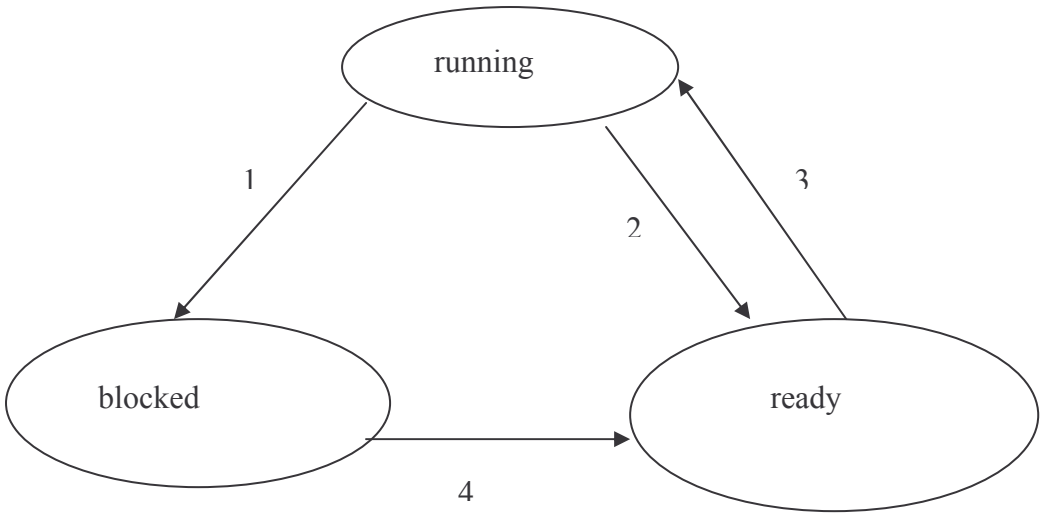
In manchen Systemen (nicht UNIX oder Win32) werden durch Terminierung eines Prozesses, freiwillig oder unfreiwillig, alle verwandten Prozesse ebenfalls beendet.

4. In welchen Zuständen kann sich ein Prozess befinden? Welche Übergänge zwischen den Prozesszuständen gibt es? Geben Sie ein Zustandsdiagramm an und erläutern Sie dieses kurz!

rechnend (der Prozessor ist dem Prozess zugeteilt)

Rechenbereit (der Prozess ist ausführbar, aber der Prozessor ist einem anderen Prozess zugeteilt) und

Blockiert (der Prozess kann nicht ausgeführt werden, bis ein externes Ereignis eintritt)



Ad 1) Prozess kann nicht weiter ausgeführt werden, da keine Eingabe vorhanden ist. (Process blocks for input)
Ad 2) Scheduler entscheidet, daß ein Prozess lange genug ausgeführt worden ist und nun ein anderer an die Reihe kommen soll.(Scheduler picks another process)
Ad 3) wenn alle anderen Prozesse ihren Anteil an Prozessorzeit zugeteilt bekommen haben und der erste Prozess wieder an die Reihe kommt. (Scheduler picks this process)
Ad 4) Wenn das externe Ereignis, auf das der Prozess wartet, eintritt. (Input becomes available)

5. Erläutern Sie das Prozessmodell in Bezug auf Unterbrechungsbehandlung und Scheduling (mit Skizze)! Was versteht man unter einer Prozesstabelle?

Falls ein Gerät eine Unterbrechung auslöst, kann das System entscheiden, die Ausführung des aktuellen Prozesses zu unterbrechen, und dann den Geräteprozess auszuführen, der blockiert war, weil er genau auf diese Unterbrechung gewartet hat. Die Aufgabe des Schedulings ist es zu entscheiden, welcher Prozess für wie lange ausgeführt wird.

0	1	…	n-2	n-1
Scheduler				

Prozesstabelle: enthält einen Eintrag für jeden Prozess. Jeder dieser Einträge enthält Informationen über den Zustand, den Programmzähler, den Stack - Zeiger, den belegten Speicher, den Zustand der geöffneten Dateien, Abrechnungs- und Scheduling-Information und was noch alles benötigt wird, um einen Prozess, der unterbrochen worden ist, weiter ausführen zu können, *so als wäre er nie unterbrochen worden*.

Kapitel 2

1. Erklären Sie das Thread-Modell! Was fügen Threads dem Prozess-Modell hinzu?

Ein Thread besitzt einen Programmzähler, der verfolgt, welche Instruktion als nächstes zur Ausführung gelangt, und einen Zustand. Weiters Register, welche die momentanen Variablen beinhalten, und einen Stack, welcher die Ausführungshistorie mit einem Frame für jede aufgerufene und noch nicht returnierte Prozedur abbildet. Ein Thread muss in einem Prozess ausgeführt werden. Prozesse werden zum Gruppieren von Ressourcen verwendet und die Threads sind die Entitäten, die für die Ausführung auf dem Prozessor verplant werden.
Threads fügen dem Prozessmodell folgendes hinzu: Die Möglichkeit einer mehrfachen Ausführung – mehrere Kontrollflüsse – in derselben Prozessumbebung unter weitgehender Unabhängigkeit (der Threads) voneinander. (Hinweis: Mehrere parallel laufende Threads in einem Prozess zu haben ist analog zu mehreren parallel laufenden Prozessen auf einem Computer.)

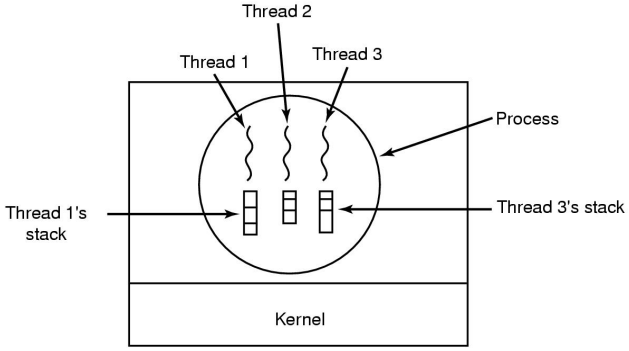
2 Nennen Sie die „per process items" und die „per thread items", die vom System verwaltet werden! Was können Sie über das Ressourcenmanagement aussagen?

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

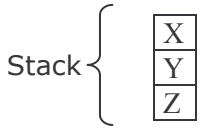
Ressourcenmanagement:
Der Prozess und nicht der Thread implementiert das Ressourcenmanagement. Deswegen ist eine Datei, die von einem Thread geöffnet wurde auch für alle anderen Threads (innerhalb eines Prozesses) sichtbar und kann gelesen oder beschrieben werden.
Hätte ein Thread die Ressourcen, wäre er ein Prozess und es wäre viel aufwendiger die Ressourcen zwischen Prozessen zu teilen(unterschiedliche Adressräume…)

3. Erläutern Sie unter Einbeziehung eines Beispiels und einer Skizze, warum jeder Thread seinen eigenen Stack benötigt!

Jeder Thread benötigt deshalb seinen eigenen Stack, weil jeder Thread meistens andere Prozeduren aufruft und sich dadurch eine unterschiedliche Ausführungshistorie ergibt.



Das ist so zu verstehen:
Jeder Stack besitzt ein Frame für jede aufgerufene & noch nicht zurückgegebene Prozedur und jedes Frame enthält z.B. die (lokalen) Variablen einer Prozedur. Ruft beispielsweise eine Prozedur X die Prozedur Y und diese wiederum eine Prozedur Z auf, befinden sich während der Ausführung von Z, die Frames von X,Y und Z auf dem Stack & so entsteht eine Ausführungshierachie.



4. Nennen Sie mindestens drei Gründe, die für die Verwendung von Threads sprechen! Geben Sie ein einfaches Anwendungsbeispiel für Threads an! Was ist das Konzept der Pop-Up Threads?

1. Hauptgrund: Anwendungen werden in mehrere sequentielle Threads zerlegt. Laufen quasi-parallel → das Programmiermodell wird einfacher & die „parallelen" Threads nutzen einen Gemeinsamen Adressraum um Daten untereinander zu teilen

2. Threads sind keine Ressourcen zugeordnet, weswegen sie einfacher zu erzeugen & zu vernichten sind als Prozesse

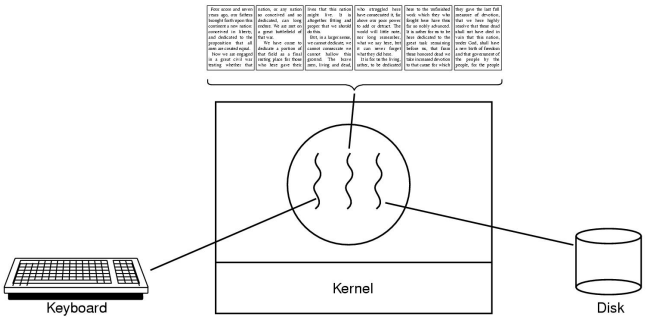
→ Erzeugung von Threads ist 100x schneller als die von Prozessen

3. Threads auf Rechnersystemen mit mehreren Prozessoren sind effektiv, da echte Parallelität der Threads stattfindet → Gewinn für Performance

einfaches Beispiel für Anwendung von Threads:

z.B. Textverarbeitungsprogramm:

Der 1. Thread nimmt Eingaben der Tastatur entgegen(interagiert mit dem Benutzer), der 2. sichert den Inhalt des Dokuments auf der Festplatte (periodisches Speichern) und der 3. verwaltet bzw. reformatiert das Dokument auf dem Speicher

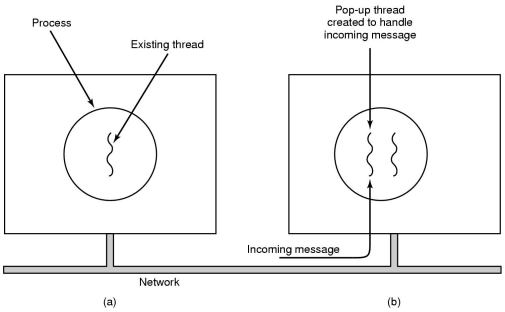


Pop-Up Threads:

z.B. wichtig bei eingehenden Nachrichten auf einem System:
die Ankunft einer Nachricht führt zum Erzeugen eines Threads für die Verarbeitung der Nachricht, dieser Thread wird Pop-Up Thread genannt;

Vorteil: Thread wird neu erstellt & besitzt keine Historie (Stack, Register...), die wiederhergestellt werden muss → nur kurze „Pause" zwischen Ankommen & Verarbeitung der Nachricht

Systemzustand (a) vor und (b) nach ankommender Nachricht



5. Erläutern Sie die drei grundsätzlichen Möglichkeiten für die Implementierung eines Servers!
Geben Sie dabei zu jedem Modell die entsprechenden Charakteristiken an!

Multi-Threaded Server

Ein weiteres Beispiel für die Anwendung von Threads ist der Einsatz dieser in Servern, z.B. Web-Servern. Einen Weg einen Web-Server aufzubauen bzw. zu organisieren findet sich in nachfolgender Abbildung.

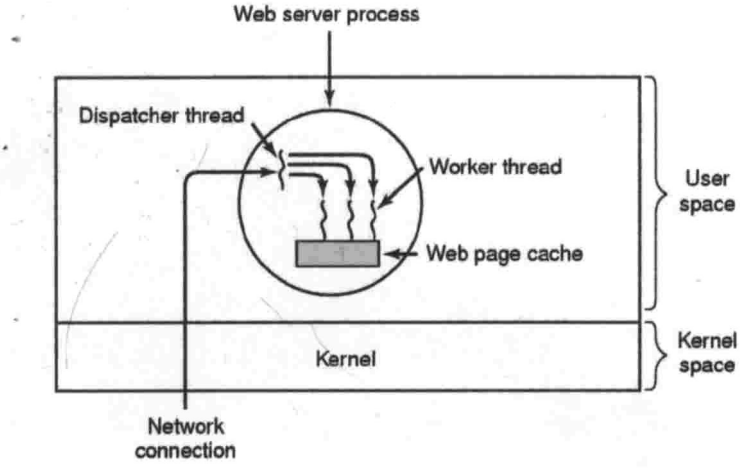


Abbildung 9: Ein *multi-threaded* Web-Server (vgl. [1])

Ein Thread, der Dispatcher-Thread, liest eingehende Arbeitsanforderungen seitens des Netzes. Nachdem dieser eine Anforderung überprüft hat, wählt er einen untätigen (oder geblockten) Worker-Thread aus und assoziiert die Anforderung mit diesem. Anschließend aktiviert der Dispatcher-Thread den schlafenden Worker-Thread durch einen Zustandswechsel vom Zustand blocked auf ready. Wenn der Worker-Thread seine Arbeit aufnimmt, überprüft er als erstes, ob die Anforderung aus dem Cache, auf den alle Threads zugreifen können, erfüllt werden kann. Ist dies nicht der Fall, so startet der Worker-Thread eine Leseoperation, um die entsprechende Seite von der Festplatte zu holen und blockiert nachdem die Operation auf der Festplatte abgeschlossen ist. Wenn der Thread während der Festplattenoperation blockiert wird, so wird sofort ein anderer Thread, z.B. der Dispatcher-Thread oder ein anderer Worker-Thread, ausgewählt. Das dargestellte Modell erlaubt es, den Server als eine Sammlung von sequentiellen Threads zu codieren.

Single-Threaded Server

Betrachten wir nun den Fall der Implementierung eines Web-Servers ohne Threads. Eine mögliche Lösung ist die Ausführung als single-threaded Server. Die Hauptschleife des Web-Servers nimmt dabei eine Anforderung entgegen, prüft diese, und führt sie komplett aus, bevor die nächste Anforderung entgegen genommen wird. Während der Festplattenzugriffe ist der Prozessor im Leerlauf und verarbeitet keine anderen eingehenden Anforderungen. Läuft der Web-Server auf einer dezidierten Maschine, was normalerweise der Fall ist, so ist der Prozessor untätig, während der Web-Server auf die Festplattenzugriffe wartet.
Das Ergebnis dieser Art von Implementierung eines Web-Servers ist, dass viel weniger Anforderungen pro Sekunde verarbeitet werden können. Folglich führt der Einsatz von Threads zu einer beachtlichen Steigerung der Performance, wobei jeder Thread aber auf gewöhnliche (sequentielle) Art programmiert werden kann.

Finite-State Machine
Man nehme nun an, dass ein System keine Threads unterstützt, aber die Systemdesigner den Performanceverlust durch das single-threading nicht hinnehmen wollen. Wenn eine nicht blockierende Version des read-Systemaufrufes vorhanden wäre, wird eine dritte Variante möglich. Wird eine Anforderung an den Web-Server gestellt, so wird diese von dem einen und einzigen Thread bearbeitet. Kann die Anforderung aus dem Cache bedient werden ist alles soweit in Ordnung, wenn nicht, wird eine nichtblockierende Leseoperation für den Zugriff auf die Festplatte gestartet. Der Server sichert dabei den Zustand der aktuellen Anforderung in einer Tabelle und bedient dann das nächste Ereignis. Das nächste Ereignis kann entweder eine Anforderung für eine neue Aufgabe

oder eine Antwort von der Festplatte (vom nichtblockierenden Systemaufruf) über eine vorhergehende Operation sein. Handelt es sich um eine neue Anforderung, so wird die Arbeit aufgenommen. Ist es aber die Antwort von der Festplatte, so wird zuerst die relevante Information aus der Tabelle geholt und dann die Verarbeitung fortgesetzt. Mit einer nichtblockierenden Festplatten-Ein-/Ausgabe hat die Antwort einen Unterbrechungs- oder Signal-Charakter.
In diesem Design geht das sequentielle Prozessmodell, welches in den beiden ersten Fällen vorhanden war, verloren. Der Zustand der gerade bearbeiteten Anforderung bzw. der Prozesszustand muss explizit in einer Tabelle gesichert und wiederhergestellt werden, wenn der Server von einer gerade in Bearbeitung befindlichen Anforderung zu einer anderen Anforderung wechselt. Im Prinzip werden durch dieses Design Threads und ihre Stacks auf die "harte Tour" simuliert. Ein Design wie dieses, in welchem jede Berechnung einen sicheren Zustand besitzt und es genau definierte Ereignisse gibt, welche die Übergänge zwischen den Zuständen veranlassen, wird finite-state machine genannt. Dieses Konzept wird häufig in allen Gebieten der Computerwissenschaft eingesetzt.

Es sollte nun eigentlich klar sein, was Threads anzubieten haben. Threads ermöglichen die Erhaltung der Idee der sequentiellen Programmierung (mit blockierenden Systemaufrufen) und erreichen bzw. realisieren dabei Parallelität. Blockierende Systemaufrufe vereinfachen die Programmierung und Parallelität verbessert die Performance. Single-threaded Server bewahren zwar die Einfachheit der Programmierung durch blockierende Systemaufrufe, weisen aber eine schwache Performance auf. Der dritte dargestellte Ansatz erreicht eine hohe Performance durch Parallelität, verwendet aber nicht blockierende Systemaufrufe und Unterbrechungen, und ist folglich schwer zu Programmieren.

Drei Wege um einen Server zu implementieren

Model	**Characteristics**
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls, interrupts

Kapitel 1

1. Was versteht man unter zeitkritischen Abläufen (race conditions) und was können Sie über deren Fehlerverhalten aussagen? Was versteht man unter einem kritischen Bereich (critical section)?

Zeitkritische Abläufe + Fehlerverhalten:
In manchen Betriebssystemen benutzen kooperierende Prozesse einen gemeinsamen Speicher, den jeder lesen und schreiben kann. Der gemeinsam benutzte Speicher kann ein Teil des Hauptspeichers oder eine gemeinsam benutzte Datei sein. Z.B. Drucker-Spooler.
Nehmen wir nun an, dass unser Spooler-Verzeichnis aus einer großen Anzahl von Einträgen besteht, die mit 0, 1, 2, … nummeriert sind und die jeweils einen Dateinamen aufnehmen können. Nehmen wir weiters an, dass es zwei gemeinsam benutzte Variablen *out* und *in* gibt, wobei *out* auf die Datei, die als nächstes gedruckt werden soll, und *in* auf den nächsten freien Eintrag verweist.
Prozess A liest die Variable in und speichert ihren Wert, 7, in einer lokalen Variablen mit dem Bezeichner next_free_slot. Der Scheduler entscheidet nun, dass Prozess A lange genug ausgeführt worden ist, und wechselt zu Prozess B (dieser erhält also nun den Prozessor). Der Prozess B liest ebenfalls die Variable *in* und erhält auch den Wert 7. Dann speichert er den Namen der auszudruckenden Datei im Eintrag 7 ab und erhöht den Wert der Variablen *in* auf 8. Danach wendet sich der Prozess B anderen Dingen zu. Die beschriebene Situation ist in nachfolgende Abbildung dargestellt.

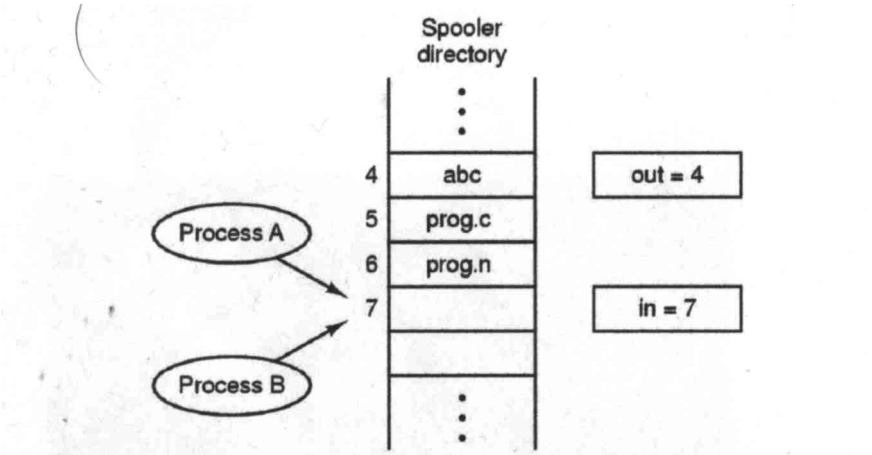


Abbildung 1: Gleichzeitiger Speicherzugriff zweier Prozesse auf *Shared-Memory* (vgl. [1])

Irgendwann wird der Prozess A an der Stelle weiter ausgeführt, an der er unterbrochen worden ist. Er findet in der Variablen next_free_slot den Wert 7, trägt den Namen seiner auszudruckenden Datei in den Eintrag 7 ein und überschreibt damit den Dateinamen, den Prozess B dort eingetragen hat. Als nächstes erhöht er den Wert der Variablen next_free_slot auf 8 und schreibt diesen Wert in die Variable in. Das Spooler-Verzeichnis ist nun in einem konsistenten Zustand, und der Drucker-Dämon wird keinen Fehler bemerken. Der Druckauftrag von Prozess B wird aber niemals ausgeführt werden.
Solche Situationen, in denen zwei oder mehr Prozesse gemeinsame benutzte Daten lesen und schreiben und die Endergebnisse von der zeitlichen Reihenfolge der Lese- und Schreiboperationen abhängig sind, nennt man zeitkritische Abläufe (race conditions). Die Fehlersuche in Programmen, die zeitkritische Abläufe enthalten, ist sehr schwierig. Die Ergebnisse der meisten Testläufe sind nämlich korrekt, aber in manchen seltenen Situationen treten dann unerklärliche Fehler auf.
Kritischen Bereich:
Es gilt es also zeitkritische Abläufe zu vermeiden. Es muss ein Verfahren gefunden werden, mit dem verhindert werden kann, dass zu einem Zeitpunkt mehr als ein Prozess gemeinsam benutzte Dateien liest oder schreibt. Also benötigt man so etwas wie einen

wechselseitigen Ausschluss das, falls ein Prozess eine gemeinsam benutzte Variable verwendet, alle anderen Prozesse davon ausschließt, ebenfalls diese Variable zu benutzen.
Der Teil eines Programms, in dem auf gemeinsam benutzten Speicher zugegriffen wird, wird als kritischer Bereich (critical section) bezeichnet. Falls man es so arrangieren kann, dass sich zu keinem Zeitpunkt zwei Prozesse in ihrem kritischen Bereich befinden können, so hat man zeitkritische Abläufe vermieden.

2. Nennen Sie die vier Bedingungen für eine gute Lösung zur Vermeidung von zeitkritischen Abläufen!
Skizzieren Sie den wechselseitigen Ausschluss durch kritische Bereiche anhand zweier Prozesse!
Was ist das Prinzip des aktiven Wartens und welcher Unterschied besteht dabei zum Blockieren?

4 Bedingungen:
Eine gute Lösung muss folgenden vier Bedingungen genügen:

- Nur ein Prozess darf sich zu jedem Zeitpunkt in seinem kritischen Bereich befinden.
- Es dürfen keine Annahmen über die Ausführungsgeschwindigkeit oder die Anzahl der Prozessoren gemacht werden.
- Kein Prozess, der sich nicht in seinem kritischen Bereich befindet, darf andere Prozesse blockieren.
- Kein Prozess soll unendlich lange warten müssen, bis er in seinen kritischen Bereich eintreten kann.

Skizze:

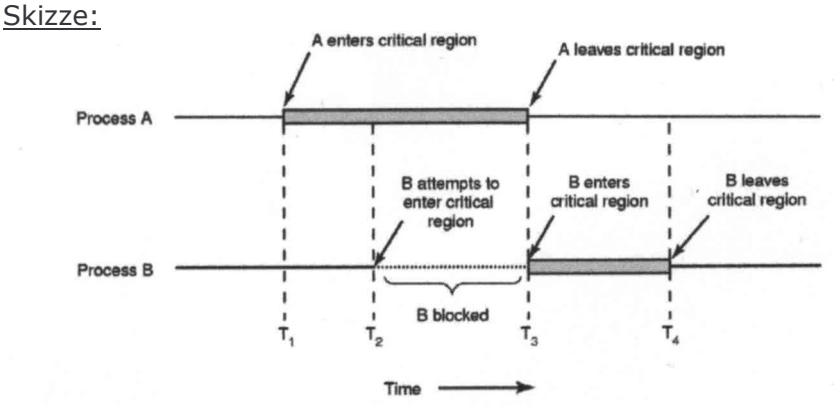


Abbildung 2: Wechselseitiger Ausschluss durch kritische Bereiche (vgl. [1])

Prinzip des aktiven Wartens:
Das fortlaufende Überprüfen einer Variablen, bis sie einen bestimmten Wert annimmt, wird aktives Warten (busy waiting) genannt. Weil jedoch Rechenzeit verschwendet wird, sollte aktives Warten soweit wie möglich vermieden werden. Nur wenn die begründete Erwartung besteht, dass die Wartedauer kurz sein wird, sollte aktives Warten eingesetzt werden.
Unterschied zum Blockieren:
Beim aktiven Warten wird der Prozess solange blockiert bis eine Variable einen bestimmten Wert annimmt. Der Prozess bekommt aber den Prozessor zugewiesen und wird ausgeführt, obwohl er nichts machen kann da er ja auf den bestimmten Wert der Variable warten muss.
Beim Blockieren bekommt er den Prozessor erst gar nicht zugewiesen und wird daher auch nicht ausgeführt, da er sich im Zustand *Blocked* befindet. Er bekommt den Prozessor erst wenn er wieder an der Reihe ist und sich im Zustand *Ready* befindet.

3. Erklären Sie die Prozesskommunikation durch Semaphore! Was sind dabei die wesentlichen Operationen und Mechanismen bzw. Prinzipien? Erläutern Sie den wichtigen Unterschied zwischen wechselseitigem Ausschluss und Synchronisation bei der Anwendung von Semaphoren! Was ist eine Mutex-Variable?

Semaphor = Integer Variable, die dazu dient Wecksignale zu zählen.
2 Operationen:

Operatoren
down überprüft ob Wert größer Null, wenn ja wird Semaphor um eins erniedrigt
up erhöht den Wert der Semaphore um eins

Prinzip
Voraussetzung für den gegenseitigen Ausschluss der Prozesse ist, dass jeder Prozess vor dem Eintritt in den kritischen Bereich die *Down* Operation aufruft und nach dem Verlassen die *Up* Operation ausführt.

Mechanismus
Bei der Kommunikation zwischen zwei Prozessen mittels Semaphoren werden üblicherweise 3 Semaphoren Variablen verwendet:
full zum Zählen der belegten Einträge im Buffer
empty zum Zählen der freien Einträge im Buffer
mutex um sicherzustellen dass Erzeuger und Verbraucher nicht zugleich auf einen kritischen Bereich zugreifen.

Wechselseitiger Ausschluss
Beim wechselseitigen Ausschluss (in diesem Fall durch die Semaphore *mutex* realisiert), wird gewährleistet dass nie mehrere Prozesse zugleich auf einen kritischen Speicherbereich zugreifen können.

Synchronisation
In diesem Beispiel werden die Semaphoren full und empty dazu benutzt dass der Erzeuger wenn der Puffer voll ist und der Verbraucher wenn der Puffer voll ist, ihre Ausführung unterbrechen.

Mutex-Variable
= vereinfachte Form einer Semaphore; wird nicht zum Zählen verwendet; auch ‚binäre Semaphore' genannt; wird zum wechselseitigen Ausschluss verwendet.
hat zwei Zustände: *locked* und *unlocked*

4. Erläutern Sie die Prozesskommunikation durch Nachrichtenaustausch! Nennen Sie drei wichtige Problembereiche von Nachrichtenaustauschsystemen!

Es wird eine Nachricht an eine bestimmte Adresse gesendet, falls keine Nachricht verfügbar ist, kann der Empfänger blockieren, bis eine Nachricht eintrifft.
Diese Methode ist vor allem sinnvoll in Systemen wo Prozesse nicht am gleichen Rechner sondern über ein Netzwerk verteilt sind.
(mögliche Anwendungen: *mailboxes*, *pipes*)

Operatoren
send (destination, &message)
recieve (source, &message)

Probleme
Nachrichten gehen im Netzwerk verloren

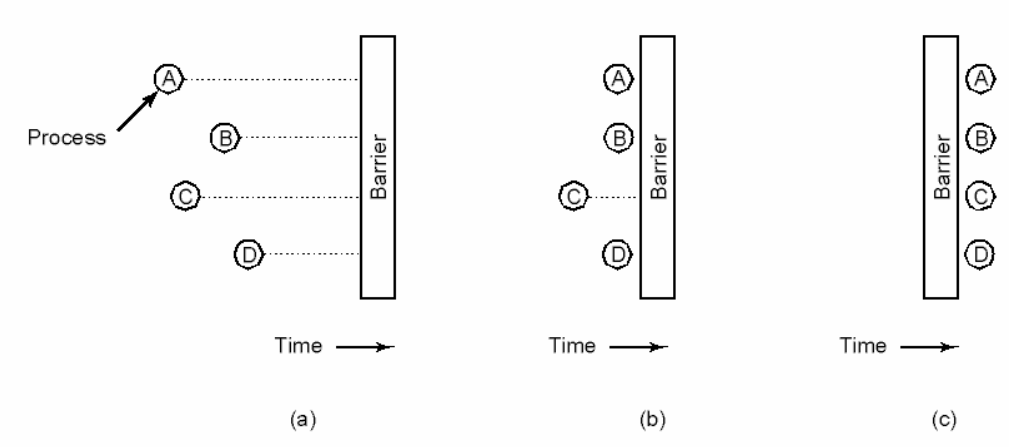
Wenn eine Nachricht gesendet wird aber im Netzwerk verloren geht, kann der Empfönger nicht weiterarbeiten. Um das zu Umgehen wird eine Empfangsbestätigung verwendet, wenn diese wiederum im Netzwerk verloren geht muss der Sender die ursprüngliche Message erneut abschicken. Um diese wiederholten Messages von neuen zu unterscheiden bekommen sie die gleiche Nummer wie die vorherige Message, neue Messages hingegen werden fortlaufend durchnumeriert.

Benennung von Prozessen
Um einen Prozess eindeutig adressieren zu können, benötigt er einen eindeutigen Namen.
Oft verwendet: *process@machine* oder *machine:process*
Wenn der Rechnerverbund sehr gross ist, kann es vorkommen dass Rechner gelich benannt werden und daher keine eindeutigen Namen mehr bestehen, dem Problem kann entgegengewirkt werden, wenn die Rechner in Domänen zusammengefasst werden (*process@machine.domain*).

Authentifizierung von Nachrichten
wird meist über Verschlüsselung bewerkstelligt. Der Schlüssel ist nur authorisierten Nutzern bekannt.

5. Beschreiben Sie die Prozesskommunikation durch Barrieren anhand einer Skizze in drei Stufen und erläutern Sie dabei jede Stufe! Für welchen Anwendungstyp ist diese Methode interessant? Wie lautet die zugehörige Regel?

= mehr für Gruppen von prozessen als für Zwei-Prozess Systeme interessant.
Viele Anwendungen leben davon dass bestimmte Abkäufe nacheinander geschehen müssen und kein Prozess in die nächste Phase übergehen darf bevor nicht alle Prozesse bereit für den Übertritt sind.



Phase (a)
Prozesse sind kurz vor Beendigung der Ausführung, bei erreichen der Barriere wird Operation ausgeführt, die den Prozess blockiert.

Phase (b)
Alle bis auf einen Pozess sind blockiert, kommt der letzte Prozess zu der Barriere, wird diese geöffnet.

Phase (c)
Alle Prozesse haben die nächste Phase eingeleitet.

Kapitel 2

1. Erklären Sie die Aufgabe eines Schedulers! Welche fünf offensichtlichen Kriterien für einen guten Scheduling-Algorithmus gibt es? Erklären Sie diese kurz! Was kann man über sich widersprechende Kriterien bez. des Schedulings aussagen?

In vielen Anwendungsbeispielen tritt des Öfteren die Situation auf, wie zum Beispiel beim Erzeuger-Verbraucher-Problem, dass mehr als ein Prozess (z.B. Erzeuger- und Verbraucher-
Prozess) ausführbar sind. In solchen Fällen muss das Betriebssystem entscheiden, welcher
Prozess als erster ausgeführt wird. Der Teil des Betriebssystems, der mit dieser Entscheidung betraut ist, wird Scheduler genannt und der von ihm verwendete Algorithmus Scheduling. Algorithmus Die Aufgabe eines Schedulers ist es, auf der Basis einer vorgegebenen Strategie Entscheidungen zu treffen und nicht einen Mechanismus zur Verfügung zu stellen.

Einige der offensichtlichen Kriterien sind:

- Fairness: Jeder Prozess erhält einen gerechten Anteil der Prozessorzeit.
- Effizienz: Der Prozessor ist immer vollständig ausgelastet.
- Antwortzeit: Die Antwortzeit für die interaktiv arbeitenden Benutzer wird minimiert.
- Verweilzeit: Die Wartezeit auf die Ausgabe von Stapelaufträgen wird minimiert.
- Durchsatz: Die Anzahl der Aufträge, die in einem bestimmten Zeitintervall ausgeführt

werden, wird maximiert.

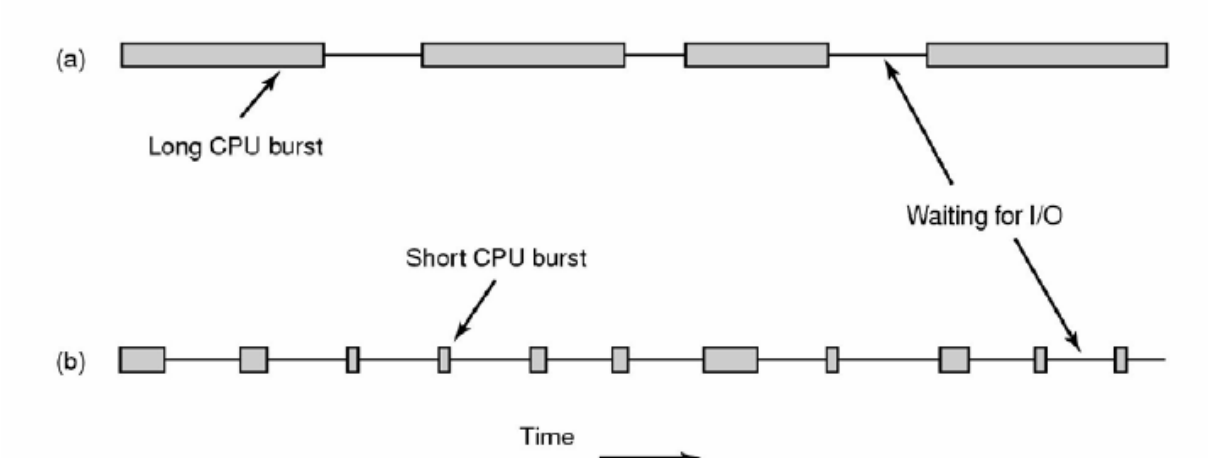
Es ist leicht zu erkennen, dass sich einige dieser Ziele widersprechen. Damit z.B. die Antwortzeit der interaktiv arbeitenden Benutzer minimiert wird, sollte der Scheduler keinerlei Stapelaufträge ausführen. Für die Auftraggeber von Stapelaufträgen ist das aber ganz und gar
nicht gut. Abgesehen davon verletzt es das oben aufgeführte vierte Kriterium. Es kann gezeigt werden, dass jeder Scheduling-Algorithmus, der eine Klasse von Aufträgen bevorzugt, eine andere benachteiligt. Die zur Verfügung stehende Prozessorzeit ist endlich, d.h. die Zeit, die einem Benutzer gegeben wird, muss bei den restlichen Benutzern abgezogen werden.

Eine große Schwierigkeit, die von einem Scheduler behandelt werden muss, ist, dass jeder Prozess anders und der Verlauf seiner Ausführung nicht vorhersagbar ist. Einer verbringt viel Zeit damit, auf E/A zu warten, und ein anderer würde den Prozessor stundenlang belegen, wenn er die Möglichkeit dazu bekäme. Wenn der Scheduler einen Prozess zur Ausführung bringt, weiß er nicht, wie lange es dauern wird, bis der Prozess blockiert, weil er auf E/A, in einem Semaphor oder aus einem anderen Grund warten muss.

2. Welche zwei grundsätzlichen Arten von Scheduling-Verfahren gibt es? Erläutern Sie die Begriffe CPU-gebundener und E/A-gebundener Prozess mit einer Skizze! Für Scheduler-Optimierungen ist die Art der Systemumgebung von besonderer Bedeutung. Nennen Sie drei Systemumgebungsklassen mit ihren Zielen sowie Ziele, die für alle Klassen gelten!

Es gibt preemptive-Scheduling und nonpreemptive-Scheduling (run-to-completion).

(a) CPU-gebundener, (b) E/A gebundener Prozess



Drei Systemumgebungen sind eine nähere Betrachtung wert.

- Batch
- Interactive
- Real-Time

Ziele

Aller Klassen:

- Fairness: Giving each process a fair share of the CPU
- Policy enforcement: seeing that stated policy is carried out
- Balance: keeping all parts of the system busy

Batch systems

- Throughput: maximize jobs per hour
- Turnaround time: minimize time between submission and termination
- CPU utilization: keep the CPU busy all the time

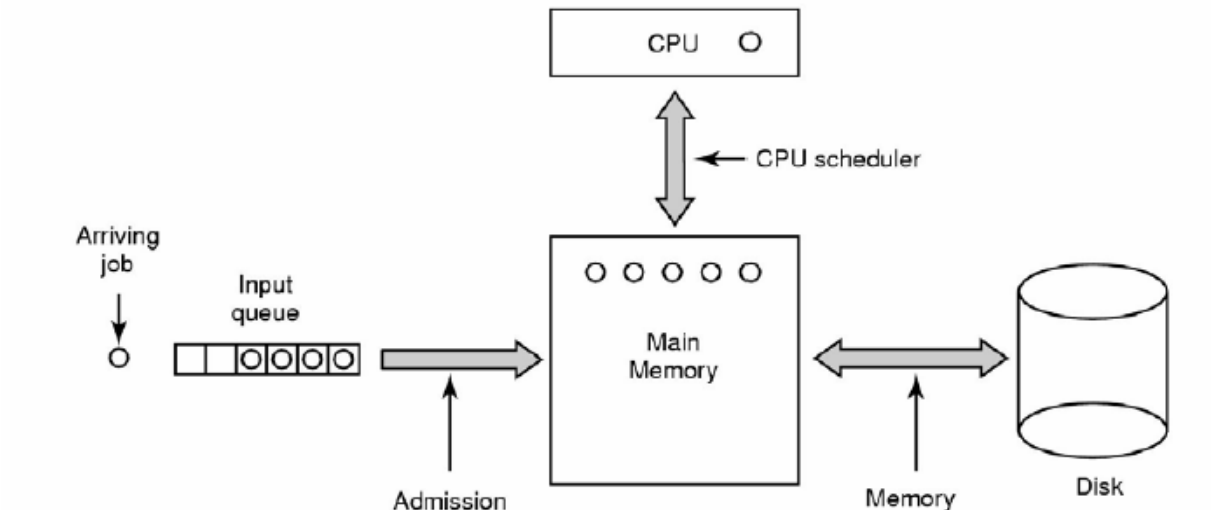
Interactive systems

- Response time: respond to requests quickly
- Proportionality: meet users' expectations

Real-time systems

- Meeting deadlines: Avoid losing data
- Predictability: Avoid quality degradation in multimedia Systems

3. Erläutern Sie das Three-Level-Scheduling in Batch-Systemen anhand einer Skizze! Erklären Sie dabei die jeweils speziellen Aufgaben der diversen Scheduler! Welche vier Entscheidungskriterien werden in der Regel von einem Memory- Scheduler berücksichtigt?



Der **Admission-Scheduler** entscheidet, welche Jobs Zugang zum System erhalten. Die anderen werden, bis sie ausgewählt werden, in der Eingabe-Queue behalten. Alternativ könnten sehr kurze Jobs schnell den Zugang erhalten, während längere Jobs warten müssten. Dem Admission-Scheduler steht es dabei frei einige Jobs in der Eingabe- Queue zu halten und Jobs die eigentlich später in das System gelangen vorzuziehen. Wenn ein Job nun den Zugang zum System erhält, so kann für diesen ein Prozess erzeugt werden, welcher dann sofort Prozessorzeit beansprucht. Jedoch kann die Anzahl der Prozesse im System so groß sein, dass nicht genügend Platz im Hauptspeicher vorhanden ist. In diesem Fall werden einige Prozesse auf den Festplattenspeicher ausgelagert.

Der **Memory-Scheduler** entscheidet welche Prozesse im Hauptspeicher bleiben. Die Entscheidungen des Memory-Schedulers müssen sehr häufig überprüft werden, um den ausgelagerten Prozessen die Möglichkeit für ihre Diensterfüllung zu geben. Zu beachten ist hierbei, dass die Prozessauslagerung mit hohen Kosten verbunden und daher sorgfältig einzusetzen ist, um nicht unnötig (Transfer-) Bandbreite zu verschwenden. Der Memory- Scheduler überprüft für eine Entscheidungsfindung periodisch jeden ausgelagerten Prozess.

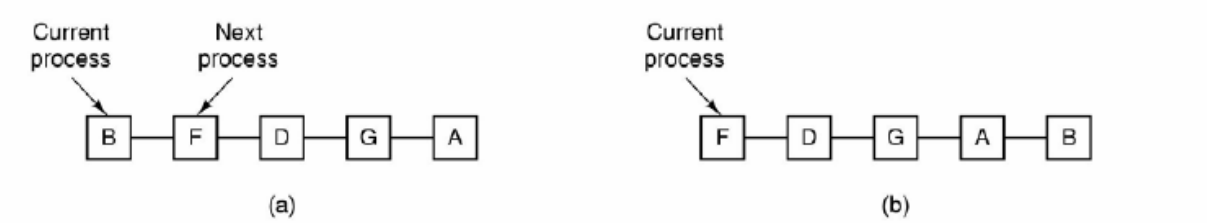
Der CPU-Scheduler wählt einen bereiten Prozess im Hauptspeicher aus, welcher als nächster den Prozessor erhalten soll. Wenn man allgemein von Scheduling spricht, meint man oft das CPU-Scheduling

Unter den Kriterien, die er für eine Entscheidungsfindung anwendet, befinden sich folgende.

- Wie viel Zeit ist vergangen, seit der Prozess ein- oder ausgelagert wurde?
- Wie viel Prozessorzeit hatte der Prozess zuletzt?
- Wie groß ist der Prozess?
- Wie wichtig ist der Prozess?

4. Wie funktioniert das Round-Robin- und das Prioritäts-Verfahren in interaktiven Systemen? Verwenden Sie für ihre Erklärungen jeweils eine Skizze mit einem Beispiel! Nennen Sie für beide Verfahren je einen Vor- und Nachteil!

Round –Robin



Jeder Prozess bekommt ein gewisses Zeitintervall, welches Quantum genannt wird, für seine Ausführung zugeteilt. Falls das Quantum eines Prozesses abgelaufen ist, wird ihm der Prozessor entzogen und einem anderen Prozess gegeben. Wenn ein Prozess blockiert oder seine Ausführung beendet, bevor das Quantum abgelaufen ist, wird natürlich zu diesem Zeitpunkt ein Prozesswechsel durchgeführt.

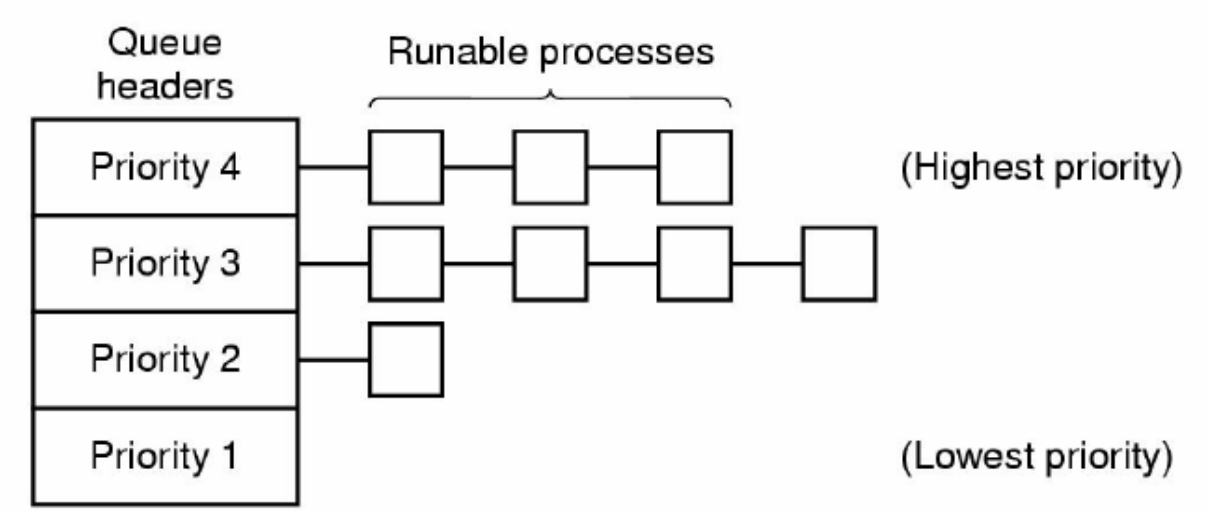
Round-Robin ist einfach zu implementieren. Der Scheduler muss nur, wie in (a) dargestellt,

eine Liste der ausführbaren Prozesse verwalten. Wenn das Quantum eines Prozesses abläuft,
so wird er, wie in (b) gezeigt, an das Ende der Liste angefügt.

Vorteil:
Es ist sehr fair, da jeder Prozess das gleiche Quantum bekommt.

Nachteil:
Je größer das Quantum, desto länger die Wartezeit des Benutzers. Je geringer das Quantum, desto höher der Verwaltungsaufwand.

Prioritätsverfahren



Die Prozesse werden häufig in Prioritätsklassen eingeteilt, wobei dann ein Prioritäts-Scheduling zwischen den Klassen und ein Round-Robin-Scheduling innerhalb jeder Klasse verwendet wird. Der Algorithmus arbeitet folgendermaßen: Solange es ausführbare Prozesse in der P-Klasse 4 gibt, wird jeder dieser Prozesse gemäß dem Round-Robin-Scheduling für ein Quantum ausgeführt. Wenn die P-Klasse 4 keine ausführbaren Prozesse mehr enthält, kommen die Prozesse der P-Klasse 3 an die Reihe usw. Falls aber die Prioritäten nicht von Zeit zu Zeit angepasst werden, kann es geschehen, dass die Prozesse aus niedrigen P-Klassen nie ausgeführt werden.

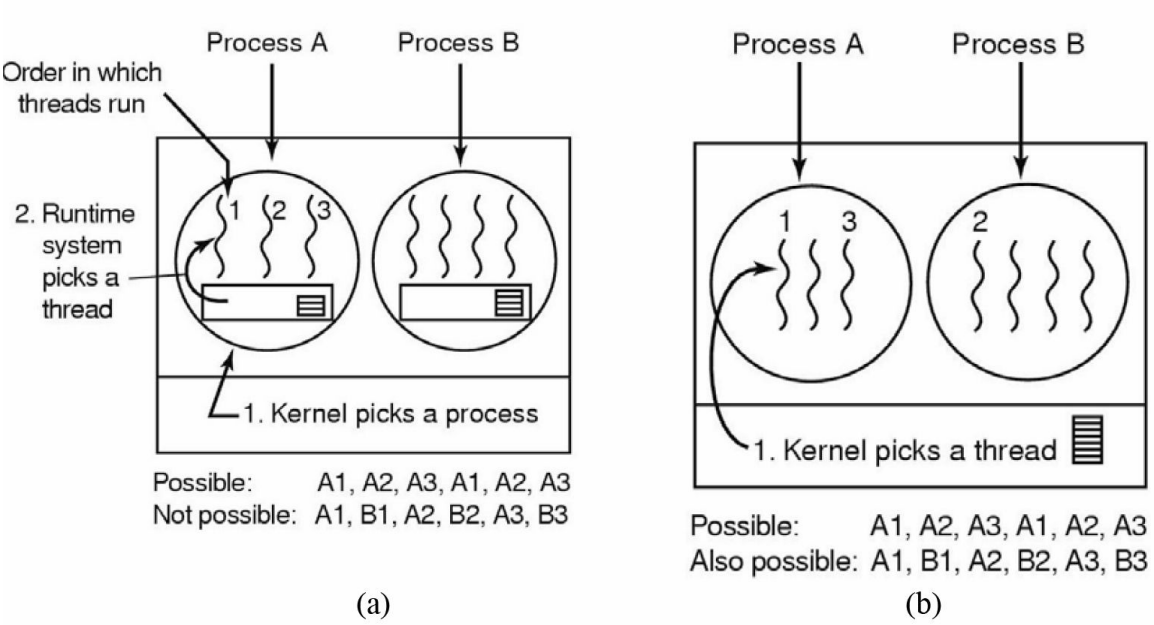
Vorteil:
Wichtige Prozesse werden sofort durchgeführt

Nachteil:
Es kann sein, dass Prozesse aus niedrigen Klassen nie ausgeführt werden.

5. Warum ist es günstig den Scheduling-Mechanismus von der Scheduling-Strategie zu trennen?

Weil der Scheduling Mechanismuss nicht die beste Entscheidung treffen kann. Die Scheduling Strategie wird hierbei durch einen Benutzerprozess vorgegeben wird.

Erläutern Sie die beiden Verfahren des Thread-Schedulings jeweils anhand einer Skizze mit einem Beispiel:



Im Allgemeinen kann ein für Anwendung spezifischer Thread-Scheduler die Application besser einstellen bzw. optimieren, als es der Kernel kann.
a=Benutzermodus b=Kernelmodus

<u>Nennen Sie für beide Verfahren je einen Vor- und Nachteil!</u>

Vorteil Benutzermodus: Performance Threadwechsel gut, weil
bei Threadwechsel im Benutzermodus lediglich einige Maschineninstruktionen nötig sind
Nachteil Benutzermodus: Beim blockieren eines Threads, wird der ganze Prozess blockiert

Vorteil Kernelmodus: Beim blockieren eines Threads, wird nicht der ganze Prozess blockiert

Nachteil Kernelmodus: Performance Threadwechsel schlecht, weil
der Threadwechsel im Kernelmodus einen vollen Kontextwechsel erfordert

Kapitel 1

1. Was versteht man unter einem Deadlock? Bei welchen Arten von Betriebsmitteln können Deadlocks entstehen? Nennen Sie jeweils ein Beispiel! Welche Schritte für die Benutzung eines Betriebsmittels sind notwendig? Wie definiert man einen Deadlock-Zustand?

Ein **Deadlock** (auch *Verklemmung* genannt) ist in der Informatik ein Zustand von **Prozessen**, bei dem mindestens zwei Prozesse untereinander auf **Betriebsmittel** warten, die dem jeweils anderen Prozess zugeteilt sind. Z.B. Kann einem Prozess *P1* der Bildschirm zugeteilt worden sein. Gleichzeitig benötigt *P1* allerdings den Drucker. Auf der Gegenseite ist der Drucker dem Prozess *P2* zugeteilt, der wiederum den Bildschirm fordert.
Ein Betriebsmittel kann sowohl ein Hardwaregerät sein (z.B Drucker) als auch eine Information (z.B. Datensatz in einer Datenbank). Man unterscheidet zwischen *unterbrechbaren* (z.B. Speicher) und *ununterbrechbaren* (z.B. Drucker) Betriebsmitteln, wobei Deadlocks nur bei letzteren auftreten.
Folgende Schritte sind für die Benutzung eines Betriebsmittels erforderlich:

- **Anforderung** des Betriebsmittels
- **Benutzung** des Betriebsmittels
- **Freigeben** des Betriebsmittels

Definition eines Deadlocks: *Eine Menge von Prozessen befindet sich in einem Deadlock-Zustand, falls jeder Prozess der Menge auf ein Ereignis wartet, das nur ein anderer Prozess der Menge auslösen kann.*

2. Nennen Sie die vier Bedingungen die für eine Deadlock-Situation erfüllt sein müssen und erklären Sie diese kurz! Was sind Betriebsmittelgraphen und welche Elemente kennzeichnen diese?

Ein solcher Zustand kann nur unter bestimmten Voraussetzungen zu Stande kommen:

1. Die Bedingung des wechselseitigen Ausschlusses:

   Jedes Betriebsmittel wird entweder von genau einem Prozess belegt oder ist verfügbar.

2. Die Belegungs- und Wartebedingung:

   Ein Prozess, der bereits Betriebsmittel belegt kann weitere Betriebsmittel anfordern.

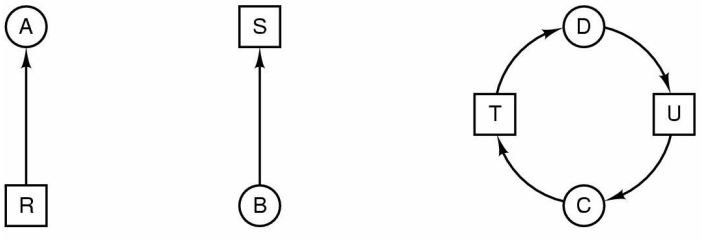
3. Die Unterbrechbarkeitsbedingung:

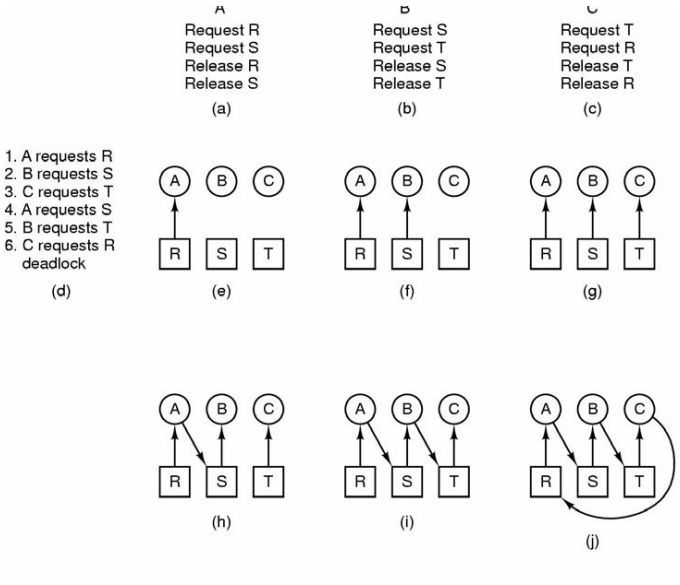
   Die Betriebsmittel, die von einem Prozess belegt werden, können nicht entzogen werden, sondern müssen explizit vom belegten Prozess freigegeben werden.

4. Die zyklische Wartebedingung:

   Es muss eine zyklische Kette aus zwei oder mehr Prozessen existieren, sodass jeder Prozess ein Betriebsmittel anfordert, das von dem nächsten Prozess in der Kette belegt wird.

Betriebsmittelgraphen dienen dazu, um die Situation von Prozessen und Betriebsmitteln visuell darzustellen. Prozesse werden als Kreise dargestellt, Betriebsmittel als Quadrate. Ein Pfeil von einem Betriebsmittelknoten auf einen Prozessknoten bedeutet, dass das BM diesem Prozess zugeteilt ist.





3. Welche vier Strategien gibt es, mit denen Deadlock-Situationen behandelt werden können?

- Ignorieren des gesamten Problems
- Erkennung und Behebung
- Dynamische Verhinderung durch vorsichtige Betriebsmittelzuteilung
- Vermeidung durch konzeptionelles Verbieten einer der vier notwendigen Bedingungen