Fragenkatalog zum Kurs 1802 "Betriebssysteme"
zusammengestellt von Jan Bartelsen nach Fragen von Dr. Lihong Ma

Hinweis:
Ich habe mich bemüht, fehlerfreie Antworten zu finden, kann dies jedoch nicht garantieren. Ich übernehme daher keine Gewähr für Vollständigkeit oder Korrektheit. Über Hinweise zur Fehlerkorrektur und Ergänzungen bin ich daher sehr dankbar: fernuni (at) janbartelsen.de

##################

Eine Liste von Fragen zu KE 1:

KE 1:

1.Welche Aufgaben hat ein Betriebssystem?

- Speicherverwaltung
- Verwaltung der E/A-Geräte
- Schutz / Überwachung von Zugriffsrechten
- Entdeckung und Behandlung von Fehlern (z.B. Div durch 0, Papierstau im Drucker, etc.)
- Mehrprogrammbetrieb
- Realisierung unterschiedlicher Betriebsarten
- Bereitstellung einer Kommandosprache
- Kostenabrechnung (eher historisch; heute nicht mehr so relevant)

Als eine der wichtigsten Aufgaben kann der Mehrprogrammbetrieb angesehen werden. Er ermöglicht, dass mehrere Prozesse scheinbar parallel laufen. "Echt" parallel können sie in einem Ein-Prozessor-System nicht laufen, da ein Prozessor immer nur einen einzigen Prozess zu einem Zeitpunkt bearbeiten kann. Sie laufen scheinbar parallel, als das sie sich zeitlich überlappen. Während ein Prozess z.B. auf eine Benutzereingabe wartet, kann ein anderer ausgeführt werden.

Wie wird der Mehrprogrammbetrieb realisiert?
Man unterscheidet beim Mehrprogrammbetrieb zwischen präemptivem Multitasking und kooperativem (nicht-präemptiven) Multitasking.
Beim präemptiven Multitasking wird die CPU-Zeit wie in Zeitscheiben definierter Länge unterteilt. Um dieses Zeitscheibenverfahren zu realisieren, wird ein Zeitgeber benutzt, der rückwärts läuft. Wenn er Null erreicht, ist die Zeitscheibe abgelaufen und es gibt eine Unterbrechung. Der Prozess wird unterbrochen und ein anderer Prozess erhält eine Zeitscheibe.

Das kooperative Multitasking arbeitet ohne Zeitscheiben. Hier blockiert ein Prozess die CPU solange bis er sie freiwillig abgibt.

##################################

2.Können Sie den Ebenenmodell für Rechner im Kurseinheit 1 beschreiben?

Höhere Programmiersprachen
Assembler-Sprachen
BS-Ebene
konventionelle Maschinenebene

Mikroprogramm-Ebene
digitale Logikebene

Welche Aufgaben hat die einzelne Schicht?

Digitale Logikebene:
Man betrachtet einen Rechner als eine Menge von Gattern, die boolsche Funktionen realisieren.

Mikroprogramm-Ebene
Die Gatter werden zu elementaren Funktionen zusammengeschaltet. Ein Beispiel ist der Transport des Inhaltes eines Registers in ein anderes.

Konventionelle Maschienenebene
Jeder Maschinenbefehl eines Prozessors wird durch ein Mikroprogramm realisiert. Diese Ebene ist die tiefste Ebene, die Programmierer im Normalfall erreichen.

BS-Ebene
erweitert die konventionelle Maschinenebene mit den Systemaufrufen. Dies sind z.B.
- Operationen mit Dateien
- Operationen mit Prozessen (erzeugen, überwachen, steuern, beenden, etc.)
- Operationen zur Hauptspeicherverwaltung

Assemblerebene
Maschinenprogramme werden in einer für Menschen besser lesbaren Form dargestellt.

Höhere Programmiersprachen
sind im Gegensatz zum Assemblersprachen weitestgehend unabhängig von konkreten Prozessoren und Betriebssystemen.

Welche Vorteile hat ein Ebenenmodell?
Es erleichtert das Verständnis. (Ein Modell ist im Allgemeinen eine vereinfachte Abbildung der Wirklichkeit zur besseren Darstellung). Jede Ebene bietet nach oben eine gewisse Menge von Diensten an, unter Benutzung der Dienste der nächsttieferen Ebene.

##################################

3.Welche Unterbrechungen gibt es? (synchrone und asynchrone Unterbrechungen) Wodurch werden sie ausgelöst?

Man unterschiedet zwischen Traps und Interrupts:

Traps:
Traps werden durch den geraden laufenden Prozess verursacht, befinden sich also auch in einem zeitlichen Zusammenhang („synchron“) mit dem Prozess.

Beispiele für Traps:
- Division durch 0
- Überlauf bei einer arithmetischen Operation
- Unbekannter Befehl
- Ein Fehler bei einem Speicherzugriff

- Systemaufruf

Interrupts:
Interrupts sind externe und damit asynchrone Unterbrechungen. Sie sind Änderung in der Ablaufsteuerung, die nicht vom laufenden Prozess, sondern von etwas anderem (meist in Zusammenhang mit E/A-Geräten) verursacht werden.

Beispiele für Interrupts sind:
- Timer
- Eine Taste gedrückt
- E/A-Signale
- Seitenfehler

Der Hauptunterschied ist:
Ein Trap ist synchron, vorhersagbar und reproduzierbar und ein Interrupt ist asynchron, unvorhersagbar und nicht reproduzierbar.
##################################

4.Können Sie die Unterbrechung beschreiben, die durch ein Ein-/Ausgabe-Gerät ausgelöst ist? Die Antwort muss die Funktionen von Unterbrechungsregister, Unterbrechungsvektor, Unterbrechungsroutinen unbedingt dabei beinhalten.

In vielen Situationen muss das BS die Beendigung einer E/A-Operation abwarten. Hierbei unterscheidet man synchrone und asynchrone Verfahren. Beim synchronen Verfahren fragt die CPU immer und immer wieder ein bestimmtes Statusregister ab, um herauszufinden, ob die Operation beendet wurde ("Polling" oder "busy waiting" genannt). Dies führt jedoch zu sehr vielen Speicherzugriffen und belastet die internen Busse.
Ein alternativer asynchroner Weg wird mit Hilfe von Unterbrechungen (interrupts) wie folgt realisiert: Im Unterbrechungsregister zeigt jedes Bit dieses Registers eine Unterbrechung an. Geräte, bzw. Geräte-Controller können diese Bits setzen. Der Prozessor prüft nun zwischen der Abarbeitung von Maschinenbefehlen, ob ein Bit des Unterbrechungsregisters gesetzt ist. Falls ja, wird die normale Programmausführung unterbrochen und eine Unterbrechungsroutine (interrupt handler) ausgeführt. Dies funktioniert mit einem Array von Adressen von Unterbrechungsroutinen, dem Unterbrechungsvektor, der an einer festen Adresse im Hauptspeicher steht. Die Nummer des Bits im Unterbrechungsregister wird als Index im Unterbrechungsvektor verwendet und das Unterprogramm mit dieser Adresse wird aufgerufen.

##################################

5.Mit welchen Mechanismen wird der Betriebssystemkern geschützt? Wie wird eine Routine (Funktion) im Kern durch Benutzerprogramme erreicht? Wie funktioniert der SVC? Was ist der Unterschied zwischen dem Systemmodus und dem Benutzermodus?

Der Schutz vom BS-Kern ist notwendig, um die für den Betrieb erforderliche Systemsoftware gegenüber unerlaubten Zugriffen durch Benutzersoftware abzugrenzen. Dieser Schutz wird mit Hilfe eines Grenzregister erreicht. Es enthält die Adresse, ab der der Benutzerbereich im Hauptspeicher beginnt. Wir nehmen an, dass das BS den Bereich ab Adresse 0 belegt, das Benutzerprogramm also die höheren Adressen. Greift ein Benutzerprogramm nun auf einen Bereich unterhalb des Grenzregister zu, wird eine Unterbrechung ausgelöst, und die Kontrolle vom BS übernommen.

Mit Hilfe eines supervisor call (SVC; auch Trap genannt) kann ein Benutzerprogramm dennoch Dienste vom BS anfordern und die zu Systemaufrufen gehörenden BS-Routinen verwenden.
Im Systemmodus sind alle Maschinenbefehle erlaubt und die Speicherschutzmechanismen sind abgeschaltet. Im Benutzermodus sind nur die "normalen", nicht sicherheitskritischen Befehle erlaubt. Die nur im Systemmodus erlaubten Befehle nennt man auch "privilegierte Befehle".
Wenn ein Benutzerprogramm einen Dienst vom BS anfordert, dann wird einen Systemaufruf ausgeführt, der einen supervisor call verursacht, damit ein Systemmodus-Wechsel stattfindet. Es ist ähnlich bei Interrupt, dass jede BS-Routine eine eindeutige Nummer hat. Eine solche Nummer ist wieder die Anfangsadresse einer Routine.
Bei einem Systemaufruf wird die Nummer des Systemaufrufs als Parameter übergeben, die als Index verwendet wird, um die Routine zu lokalisieren.

##############

6.Was macht der Urlader?
Der Urlader befindet sich im ROM des Rechners. Seine Aufgabe ist es, einen Lader zu starten, der immer an einer bestimmten Speicheradresse steht. Der Lader lädt schließlich das BS.

##############

7.Welche Betriebsarten gibt es? (Eine kurze Beschreibung jeder Betriebsart ist notwendig)

- Interaktiver Betrieb
  - auch online- oder Dialogbetrieb
  - Ein Rechenauftrag besteht hier aus einer Sitzung, in deren Verlauf der Benutzer die auszuführenden Programme in einer Kommandosprache aufruft. Die Programme werden dabei immer sofort gestartet. Das BS muss dafür sorgen, dass die Antwortzeit möglichst niedrig ist, da der Benutzer so lange warten muss.

- Stapelbetrieb (auch off-line Betrieb)
  - Der Benutzer erstellt einen Stapeljob (batch job), der mehrere Programme enthält. Wenn der Job schließlich ausgeführt wird, hat der Benutzer i.A. keinen Kontakt mehr zum Rechner, es findet keine direkte Kommunikation zwischen Programm und Benutzer statt. Zur Beschreibung des Jobs gehören auch organisatorische Aufgaben, z.B. wieviel CPU-Zeit der Job maximal verbrauchen darf.

- Hintergrundausführung
  - Zwischenform zw. Dialog- und Stapelbetrieb
  - Ein Programm wird interaktiv gestartet und dann im Hintergrund ausgeführt, während die interaktiven Programme im Vordergrund weiterlaufen.
  - Viele Fenstersysteme sind so organisiert.

- Realzeitbetrieb
  - E/A müssen innerhalb von harten Zeitgrenzen verarbeitet werden
  - erfordert einen besonders konstruierten BS-Kern.

- Teilhaberbetrieb
  - Abart des Dialogbetriebs
  - mehrere Benutzer kommunizieren über je ein eigenes Terminal mit einem einzigen Prozess.
  - Beisp.: transaktionsorientierte Systeme wie Bankenterminals, Flugbuchungssysteme

Eine Liste von Fragen zur KE 2:

1.Wie kommuniziert die CPU mit Geräten? (Geräte-Treiber, Geräte-Controller)

Viele Geräte werden nicht direkt mit der CPU verbunden, sondern über einen Controller. Der Grund für die Einführung von Controllern liegt vor allem darin, dass andernfalls die CPU mit sehr viel elementaren Aufgaben bei der Steuerung der Geräte belastet würde. Ein weiterer Vorteil ist, dass man neuartige Geräte mit Hilfe eines Controllers, der einen bereits bekannten Gerätetyp simuliert, an einem Rechner betreiben kann, ohne das BS zu ändern.
Die geräteabhängigen Funktionen der E/A-Software werden in die Geräte-Treiber verlagert. Geräte-Treiber kommunizieren direkt mit deren Controllern über den Bus. Normalerweise ist ein Geräte-Treiber Teil des BS-Kerns, damit er auf die Hardware und die Register des Controllers zugreifen kann.

####################

2.Wie ist eine Festplatte aufgebaut?

Platte -> Scheibe -> Spuren (~ 100 kB) -> Sektoren (< 4 kB)

Eine Platte besteht aus mehreren (meist 2 – 10) Scheiben, die in ihrer Mitte durch eine Achse fest miteinander verbunden sind und gemeinsam um diese Achse rotieren.
Jede Scheibe ist auf beiden Seiten mit einer magnetisierbaren Beschichtung versehen. Die untere Seite der untersten Scheibe und die obere Seite der obersten Scheibe bleiben meist unbenutzt. Die Scheiben haben einen gewissen Abstand voneinander, damit ein Arm des Plattenlaufwerks zwischen je zwei Scheiben einfahren kann. An seinem Ende sind zwei Lese-/Schreibköpfe abgebracht. Alle Arme sind starr miteinander verbunden.
Die Scheibe besteht aus mehreren Spuren. Senkrecht übereinander liegende Spuren bezeichnet man als Zylinder.
Die Spur wird in Sektoren unterteilt, die aus Präambel, Nutzdaten und Fehlerkorrektur bestehen. Die Präambel beginnt mit einem bestimmten Bitmuster, damit die Hardware den Beginn eines Sektors erkennen kann.

Was kann eine Festplatte ohne Dateisystem?
Ohne Dateisystem müssten die Benutzer direkt auf dem Plattenspeicher, d.h. auf der konventionellen Maschinenebene arbeiten. Ein Arbeiten mit Dateien wäre nicht möglich.

####################

3.
Wie wird die Zugriffszeit auf eine Festplatte definiert?

Suchzeit (Positionierung)
+ Latenzzeit (Zeit bis gesuchter Sektor unter den Köpfen erscheint)
+ Übertragungszeit (Zeit zum Lesen)

Welche Maßnahmen im Controller können die Zugriffszeit verkürzen?
SSTF, SCAN und interleaving

Wie funktionieren die Strategien SSTF und SCAN?

Welche Vorteile und Nachteile haben die Strategien?

SSTF (shortest seek time first) bezeichnet die Strategie, dass derjenige Übertragungsauftrag als nächstes ausgeführt wird, bei dem die kleinste Suchzeit auftritt. Aufträge in Zylindern, die der aktuellen Position nahe liegen, werden folglich bevorzugt.
Nachteil: Die Köpfe bewegen sich u.U. aus einem engen Bereich von Zylindern nicht mehr heraus, wenn dort laufend neue Aufträge abzuarbeiten sind. Aufträge aus entfernteren Zylindern müssen übermäßig lange warten und können im schlimmsten Fall sogar "verhungern".

Abhilfe schafft die SCAN-Strategie (auch Aufzugsalgorithmus genannt), bei der die Köpfe immer abwechselnd nach innen und außen wandern, sofern in der jeweiligen Richtung Aufträge vorliegen.
Nachteil: Mittlere Suchzeit ist höher als bei SSTF
Vorteil: Mittlere Ausführungszeit eines Übertragungsauftrages inkl. der Wartezeit bis zum Beginn der Ausführung ist kürzer.

Die Strategien SSTF und SCAN finden normalerweise im Gerätetreiber statt, weil der Gerätetreiber eine Liste von Anfragen für die Festplatte verwaltet. Mit der Liste kann er die Reihenfolge des Zugriffs organisieren.

Was ist das Interleaving?
Wenn man eine Folge von Blöcken, die z.B. zu einer Datei gehören, nacheinander in die Sektoren einer noch leeren Spur schreiben möchte, bietet es sich an, beim Schreiben einen oder mehrere Sektoren zu überspringen und die Blöcke nicht in aufeinander folgende Sektoren (i-ter Block in i-ten Sektor) zu schreiben. Der Grund ist folgender: Die Zeit zum Übertragen eines Blockes in den Hauptspeicher kann länger sein, als die Zeit, die der Schreib/Lesekopf benötigt, um die Lücke zwischen zwei Sektoren zu überqueren. Wenn also der erste Block übertragen wurde, befindet sich der Kopf bereits hinter dem zweiten Sektor, wenn der zweite Sektor angefordert wird.
Wie viele Sektoren übersprungen werden, hängt vom Rechner ab. Diese Zahl nennt man "interleave factor"; sie muss beim Formatieren der Platte festgelegt werden.

##################

4.Was ist ein Dateisystem?
Ein Dateisystem ist eine Menge von Dateien und Verzeichnissen, die incl. der erforderlichen Hilfsdaten auf einem physischen oder logischen Datenträger gespeichert sind. (Def. laut Skript S. 52).

Was ist ein hierarchisches Dateisystem?
Hierarchische Dateisysteme arbeiten mit Verzeichnissen, die geschachtelt sein können. Ein Verzeichnis kann Dateien oder wieder andere Verzeichnisse enthalten. Die Struktur der Verzeichnisse ist normalerweise ein Baum mit einem Wurzelverzeichnis als Einstiegspunkt.

Wozu ist ein Dateisystem gut?
Ein Dateisystem soll die Transparenz schaffen, dass dem Besitzer der Dateien die Details verborgen bleiben. Er braucht nicht zu wissen, wo und wie die Dateien gespeichert sind.

4.Was ist eine FAT?
FAT (file allocation table) ist eine zentrale Datenstruktur, die Informationen über alle Dateien sowie die freien Blöcke enthält. Sie besteht aus einem Array, in dem jeder Sektor der Platte durch einen Eintrag repräsentiert wird.

Vorteile von FAT:
- freie und defekte Blöcke werden in der FAT als solche markiert.
- sehr gute Unterstützung von sequentieller Datei-Verarbeitung

Nachteile von FAT:
- die gesamte Tabelle muss sich zu jeder Laufzeit des Rechners im Hauptspeicher befinden -> dies benötigt unnötig viel Ressourcen
- die Tabelle wird bei großen Platten ebenfalls recht groß.
- Wenn ein Block einer Datei gesucht wird, muss man eine lineare Suche durch die Liste der Blöcke der Datei starten. Wenn eine Datei sehr groß ist, kostet das viel Zeit.

Wie wird eine Datei mit FAT verwaltet?

Die zu einer Datei gehörigen Sektoren werden in der FAT linear verkettet. Beispiel: Meine Datei text.tex steht in den Sektoren 3, 27 und 5. Im Datei-Verzeichnis gibt es einen Eintrag, dass "text.tex" im Sektor 3 startet. In der FAT gibt es im Eintrag 3 einen Verweis auf 27, welches die Nummer des Folgesektors ist. Eintrag 27 verweist wiederum auf 5 und Eintrag 5 enthält die Angabe "nil", welches anzeigt, dass der letzte Block der Datei erreicht wurde.

Die freien Sektoren werden mit Hilfe eines Bitfeldes realisiert, welches für jeden Sektor anzeigt, ob er frei ist oder nicht. Sofern in der FAT noch Platz ist, wird das Bit in den Einträgen der FAT untergebracht. Es ist außerdem möglich, die freien Sektoren als eine Datei zu verwalten.

################################

5. Was ist ein i-node?

Im Gegensatz zur zentralen Verwaltung aller Dateien in der FAT, existiert beim Konzept der i-nodes dezentral für jede Datei eine eigene Sektoradresstabelle. Zu jeder Datei existiert ein i-node (index-node); eine Tabelle, die Angaben über die Datei enthält. Darin stehen:
- 10 direkte Sektoradressen. Dies sind Adressen von D-Sektoren (Dateiseiten-Sektoren), die Seiten von Dateien enthalten und auf die direkt zugegriffen werden kann.
- eine indirekte Sektoradresse, die auf genau eine Adresse eines Blocks zeigt, in dem genau X Adressen stehen
- eine doppelt indirekte Sektoradresse, die auf X indirekte Sektoradressen zeigt.
- eine dreifach indirekte Sektoradresse, die auf X doppelt indirekte Sektoradressen zeigt. Eine Datei kann folglich $10 + X + X^2 + X^3$ Sektoren groß sein

X ist hierbei die Anzahl der Sektoradressen, die in einem Sektorblock passen. Wenn eine Sektoradresse 4 Byte lang und ein Block 1 KByte groß ist, dann passen genau 1 kByte/4 Byte= 2^{10}/ 2^2 = 2^8 = 256 Adressen in einem Block. Also gilt X=256.

-> Abb. 2.11 im Kurstext (S. 59)

Der Dateiname steht nicht im i-node, sondern im Verzeichnis. Hierfür gibt es in jedem Verzeichnis eine Liste mit dem Dateinamen und dem i-node dieser Datei. Der i-node wird identifiziert über eine eindeutige Nummer für genau die Datei, die er verwaltet (Inode-Nummer; ähnlich einem Primärschlüssel in einer Datenbank).

Welche Attribute gibt es im i-node?
- Zugriffsrechte
- Eigentümer/Gruppe
- Zeitstempel
- Größe
- Anzahl der Links
- das Datum der letzten Aktualisierung
-

Wie groß kann eine Datei sein, die mit i-node verwaltet wird?
(10+X+X^2+X^3, Was ist das X? Wie kann man das X berechnen? Wie groß kann das X sein?)

Wenn X=256 gilt, kann eine Datei max. ca. 16 GB groß sein.

6.Wie werden bei UNIX die Zugriffsrechte einer Datei realisiert?
(Sieh Schutzits in Kurseinheit 6)
Die Zugriffsrechte einer Datei werden in UNIX mit Hilfe von Schutzbits realisiert. Zu jeder Datei werden drei Gruppen á drei Bits angegeben, die für Besitzer, Gruppe und sonstige angeben, ob Leserecht (erstes Bit), Schreibrecht (zweites Bit) und Ausführungsrecht (drittes Bit) existieren.

7. Wie kann man die Zugriffszeit auf eine Festplatte mit Hilfe des Betriebssystems (oder Dateisystems) reduzieren?

Neben den bereits genannten Strategien SSTF, SCAN und interleaving bieten sich folgende Möglichkeiten an:
- Verwendung eines E/A-Puffers
- Asynchrones Schreiben (=> Die Daten werden zunächst in den Puffer geschrieben und von dort auf den Plattenspeicher übertragen. Eine Bestätigung, dass die Daten tatsächlich auf die Platte geschrieben wurden, wird nicht verlangt.)
- Optimierung der Sektorfolgen (Defragmentierung)

Eine Liste von Fragen zur KE 3:

1. Was ist ein Prozess? Was ist der Unterschied zu einem Programm?

Ein Prozess ist die konkrete Ausführung eines laufenden Programmes.

Definition lt. Skript:
Ein ablaufendes Programm bezeichnet man als Prozess, inklusive des aktuellen Wertes des Befehlszählers, der Registerinhalte und der Belegung der Variablen.

Ein Programm an sich ist etwas statisches, ein Prozess etwas dynamisches.

Vgl. die Kuchenback-Analogie von Tanenbaum:
Rezept = Programm; Mensch = Prozessor; Vorgang des Backens = Prozess

####################

2. Was gehört zu einem Prozess?
Programmsegment, Stacksegment, Datensegment und Prozesskontrollblock
=> wird auch als Prozessabbild bezeichnet.

####################

3. Was steht im Prozesskontrollblock?

Alle Daten, die das BS über einen Prozess verwalten muss, bezeichnet man als Prozesskontrollblock (PCB)

Das sind:
- Prozess-ID (eindeutig!) + Prozess-Zustand
- Speicherbereich des Prozesses
- Programmzählerstand
- Inhalte Prozesserregister
- Informationen über geöffnete Dateien
- Abrechnungs- und Statistikinformationen

####################

4. Welche Zustände hat ein Prozess?

- Erzeugt
- Bereit
- Rechnend
- Blockiert
- Beendet

(Bei der Erzeugung eines Prozesses wird ein Speicherplatz für die Programmsegment, Datensegment, Stacksegment und Prozesskontrollblock im Hauptspeicher reserviert.)

Welche Übergänge gibt es?

Malen Sie bitte das Prozesszustandübergangsdiagramm und erklären den Grund der Übergänge?

Siehe Skript S. 78, Abb. 3.2

5. Wer entscheidet, dass ein Prozess vom Zustand bereit zum Zustand rechnend gehen darf?

Der Scheduler entscheidet, welcher Prozess vom Zustand bereit zum Zustand rechnend geht, d.h. die Reihenfolge der Prozesse in der Warteschlange realisiert (präsentiert) genau die Scheduling-Strategie, die der Scheduler gewählt hat.
Die Umschaltung macht der Dispatcher.

###############

6. Was heißt non-preemptiv? Was heißt preemptiv?

Bei nicht-preemptiven BS entscheidet nur Prozess selbst, ob er den Prozessor wieder abgeben möchte.
Bei preemptiven Systemen entzieht das BS einem laufenden Prozess den Prozessor.

##############

7. Welche Qualitätsmerkmale von Scheduling-Strategien gibt es?

- Maximale Effizienz, d.h. hohe Prozessorauslastung
- Minimale Antwortzeiten
- Minimale Durchlaufzeiten
- Maximaler Durchsatz
- Fairness, also gerechte Verteilung des Prozessors.

Können sie gleichzeitig erfüllt werden?
Nein, da sie sich teilweise widersprechen. Dadurch entstehen auch Konflikte zwischen den Qualitätsmerkmalen. Ein Extrem-Beispiel: Ich habe ein rechenintensives Programm, z.B. eine Simulation, die mehrere Tage Rechenzeit benötigt. Wenn ich dieses Programm nun an einem Stück ausführen lassen, habe ich die ganze Zeit volle Prozessorauslastung und für dieses Programm eine minimale Durchlaufzeit. Gleichzeitig ist mein Prozessor jedoch mehrere Tage blockiert und ein kleines Programm müsste lange warten (Widerspruch zur Fairness und zu minimalen Antwortzeiten)

8. Welche Scheduling-Strategien sind geeignet für ein Batchbetrieb?

Beim Batchbetrieb ist das anzustrebende Ziel eine minimale Durchlaufzeit. Dies ist bei First-Come-First-Serve gegeben, sowie bei SJF, SRTF und Priority gegeben.

9. Welche Scheduling-Strategien sind geeignet für ein nicht-kooperativ (preemptiv) interaktiver Betrieb?

Beim interaktiven Betrieb besteht der Wunsch, dass Prozesse, die ständig kurze Ein/Ausgaben haben, nicht lange auf die nächste Prozessorzuteilung warten müssen. Sie würden z.B. beim Round Robin ihr Quantum oftmals nicht vollständig ausnutzen und stattdessen wegen E/A blockieren.

Mit der Strategie „Feedback Scheduling“ wird die Vergangenheit eines Prozesses bei der Auswahl des nächsten Prozesses berücksichtigt. Hierbei erhält jeder Prozess eine Priorität, wobei die Priorität bei jeder Prozessorumschaltung neu berechnet wird. Prozesse, die nicht rechnend waren, erhalten eine höhere Priorität. Vorteil hierbei ist außerdem, dass Prozesse nicht mehr verhungern können. Neue Prozesse erhalten zunächst eine hohe Priorität und eine kleine Zeitscheibe. Hat der Prozess sein Quantum komplett genutzt, erhält er eine niedrigere Priorität, aber sein Quantum verdoppelt sich.

Wie kann man diese Strategien so einstellen, dass sie interaktive Prozesse bevorzugen?

Mit der Strategie „Multiple Queues“ werden Prozesse in verschiedene Klassen eingeteilt: Es existieren die Klasse der Systemprozesse, der Dialogprozesse und der Hintergrundprozesse (batch jobs). Für jede Klasse ist ein eigener Scheduler zuständig. Nun kann das Gesamtscheduling so eingestellt werden, dass Dialogprozesse bevorzugt werden.

######################

10. Welche preemptive Scheduling-Strategien gibt es?

Round-Robin
Hierbei werden Prozesse wie bei First-Come-First-Served in einer Warteschlange verwaltet. Jeder Prozess darf den Prozessor jedoch nur für eine bestimmte Zeit behalten. Diese Zeit nennt man Zeitscheibe oder Quantum. Hat der rechnende Prozess nach dem Ablauf seiner Zeitscheibe den Prozessor noch nicht freigegeben, so wird er vom BS unterbrochen und kommt an das Ende der Warteschlange. Dort werden auch neue Prozesse eingereiht.

Vorteile RR:
- Fairness: Jeder Prozess bekommt seinen Anteil am Prozessor.
- Kurz laufende Prozesse werden nicht benachteiligt.
- Einfach zu implementieren
- Wenig Verwaltungsaufwand

Nachteil RR:
- Langlaufende Prozesse müssen bis zu ihrem Ende recht lange warten.

SRTF (Shortest Remaining Time First)
Eine Abwandlung der Shortest-Job-First-Strategie. Die Idee ist, dass jedes mal, wenn ein neuer Prozess eintrifft, der rechnende Prozess angehalten wird. Die noch verbleibende Bedienzeit dieses Prozesses wird zu seiner neuen Bedienzeit. Dann wird wieder unter allen bereiten Prozessen derjenige mit der kleinsten Bedienzeit ausgewählt.

Priority Scheduling
- Prozesse sind nicht mehr gleich wichtig, sondern werden nach Prioritäten geordnet.
- Prioritäten können intern oder extern vergeben werden.
- Prioritäten können statisch oder dynamisch sein.
- Dynamische Priorität bedeutet, dass die Priorität angepasst wird , z.B. erhalten Prozesse mit kleiner Priorität, die schon lange warten, vom System eine höhere => so wird verhungern verhindert
- Vorteil: Wichtige Aufgaben werden schnell erledigt
- Nachteil: Bei statischer Vergabe ist "verhungern" möglich => nicht fair

Weitere preemptive Strategien sind

Feedback und Multiple Queues (siehe Kurstext)

######################

11. Welche theoretische Eigenschaft hat die SJF-Strategie?

SJF (Shortest Job First) bedeutet, dass immer derjenige Prozess ausgewählt wird, der die kürzeste Bedienzeit hat. Dies ist zwei wichtige Eigenschaften: Zum einen wird eine minimale durchschnittliche Antwortzeit garantiert (Vorteil), zum anderen ist die Gefahr, dass Langläufer verhungern (starvation).

########################

12. Wie wird die Priorität festgelegt, wenn man die SJF-Strategie und die FCFS-Strategie als Priorität-Strategie betrachtet?

SJF kann man wie folgt als Prioritäts-Strategie interpretieren: Je kleiner die Bedienzeit eines Prozesses ist, desto höher ist seine Priorität.
Bei FCFS erhalten die Prozesse, die zuerst ankommen eine hohe Priorität, die späteren eine entsprechend niedrigere.

#########################

13. Wie wird die Zeitscheibe festgelegt, wenn man die FCFS-Strategie als Round Robin betrachtet? Jeder Prozess hat die Größe der Zeitscheibe unendlich.

Wie ist die Größe der Zeitscheibe sinnvollerweise zu wählen?
Die Größe der Zeitscheibe ist sorgfältig zu wählen. Wählt man das Quantum q zu klein, gibt es ein ungünstiges Verhältnis Verwaltungsaufwand <-> „echte Arbeit“. Im worst case ist der Prozessor nur mit der Prozessumschaltung beschäftigt und fast gar nicht mit „echter“ Arbeit.
Wählt man q zu groß werden die Wartezeiten für die interaktiven Benutzer unzumutbar groß.
Eine sinnvolle Größe für q ist ein Wert, der etwas größer als die für eine übliche Interaktion erforderliche Zeit ist, typischerweise in Millisekunden-Größenordnungen (s. Skript S. 91).

########################

14. Wer ist für das Umschalten des Prozessors zwischen Prozessen zuständig? Was passiert bei einem Umschalten?

Zuständig für das Umschalten des Prozessors zwischen den Prozessen ist der Dispatcher (Teil des BS). Beim Umschalten passiert folgendes:

- Anhalten des rechnenden Prozesses. Der Prozessor ist dem Prozess entzogen und dem BS zugeordnet.
- Sichern der Informationen über den bisherigen Prozess in dessen PCB.
- Aus dem PCB des nächsten Prozesses dessen alten Zustand wieder herstellen. Der nächste Prozess stammt aus der Menge der Prozesse im Zustand „bereit“.
- Den Prozessor an den neuen rechnenden Prozess übergeben. Dieser wechselt dann in den Zustand „rechnend“.

Bei heute üblichen Systemen schaltet das BS den Prozessor mehrmals pro Sekunde zwischen verschiedenen Prozessen hin und her.

######################

15. Was ist ein Thread? Was ist der Unterschied zu einem Prozess?
Was haben die Threads eines Prozesses gemeinsam?
Ein Thread ist ein leichtgewichtiger Prozess. Es hat sich als nützlich erwiesen, wenn nicht für alle Aufgaben eigene Prozesse vorhanden sind. Statt dessen werden mehrere Ressourcen von verschiedenen Prozessen gemeinsam genutzt, z.B. ein gemeinsamer globaler Datenbereich.
Mit Hilfe von Threads lassen sich auch parallele Aktivitäten innerhalb eines Prozesses beschreiben. Ein Prozess kann aus mehreren Threads bestehen.
Ein Thread ist definiert durch seinen eigenen
- Registersatz
- Programmzähler
- Stackbereich

Programm- und Datensegment teilen sich die verschiedenen Threads eines Prozesses.

Anwendungsgebiete für Threads sind bspw. Mehrprozessor-Maschinen. Ein Programm kann so mit Hilfe von Threads parallelisiert und dadurch wesentlich beschleunigt werden. Verschiedene Threads können jeweils einem eigenen Prozessor zugewiesen werden und dann parallel abgearbeitet werden. Weitere Anwendungsgebiete sind Gerätetreiber für langsame Geräte (Bsp. Diskettenlaufwerk: Der Prozessor im Controller kann schon die nächste Anfrage bearbeiten (Thread), falls der Schreib-/Lesekopf für die vorherige Anfrage gerade positioniert wird.) oder verteilte Systeme. (Client/Server)

Welche Vorteile haben Threads?
- Erreichbarkeit (Wenn ein Prozess blockiert würde, so muss nur ein Thread blockiert werden, während die anderen Threads weiterarbeiten können.)
- Teilen der gemeinsamen Ressourcen (Es ist möglich, dass verschiedene Prozesse auf die gleichen Ressourcen, z.B. Datenspeicher zugreifen)
- Mehr Effizienz (Threads arbeiten wesentlich effizienter, weil bei einem Wechsel der Threads nur die Register ausgetauscht werden. Es ist auch effizienter, was den Speicherverbrauch betrifft, wenn der Speicher nicht dupliziert wird.)

Was ist der Unterschied zwischen Kernel-Threads und Benutzer-Threads?
Bei Benutzer-Threads hat der Kern keinerlei Kenntnis davon, ob ein Prozess mehrere Threads verwendet oder nicht. Um seine Threads zu verwalten, braucht jeder Prozess seinen eigenen, privaten Thread-Kontrollblock, analog zum PCB. Er behandelt das Scheduling seiner Threads selbst. Zum Scheduling von Benutzer-Threads kommen nur die nicht-preemptiven Scheduling-Strategien in Frage. Ein Nachteil von Benutzer-Threads ist, dass der ganze Prozess blockiert, wenn ein Benutzer-Thread blockiert.
Bei Kernel-Threads verwaltet der Kern die Threads ähnlich, wie er schon Prozesse verwaltet. Es gibt eine Tabelle von Prozessen und Threads, die alle am Scheduling des BS-Kerns teilnehmen. Diese Realisierung ist aufwändiger, zusammengehörige Kernel-Threads können in Multiprozessor-Systemen jedoch auf mehreren Prozessoren parallel ablaufen.

Eine Liste von Fragen zur KE 4:

1. Was ist der Unterschied zwischen einer logischen und physischen Adresse?

Eine physische Adresse ist eine reale Adresse einer Speicherzelle in einem Hauptspeicher. Über die physische Adresse kann jede Speicherzelle einzeln adressiert werden.
Wenn Programme im Benutzermodus ausgeführt werden, werden oft logische Adressen verwendet, die mit Hilfe der MMU (Memory Management Unit) in physische Adressen umgesetzt werden. Der logische Adressraum ist dabei meistens größer (viel größer) als der physische Adressraum, da der reale Hauptspeicher deutlich teurer ist, als der Plattenspeicher, auf dem der logische Adressraum realisiert wird.
Eine logische Adresse ist die relative Position zum Anfang des Programms. Ein Prozess braucht nicht die physischen Adressen bei der Ausführung eines Prozesses zu wissen, weil die logische Adresse mit Hilfe der MMU in die physische Adresse umgesetzt wird, bevor ein Zugriff auf den Hauptspeicher stattfindet.

Vorteil des ganzen ist folgende Idee:
Zur Ausführung muss ein Prozess in den Arbeitsspeicher geladen werden. Wegen der sequentiellen Abarbeitung werden innerhalb eines bestimmten Zeitintervalls nicht alle Teile des Prozesses und analog auch nicht der gesamte Datenbereich benötigt. Es reicht also aus, nur die jeweils benötigten Programm- und Datenbereichs-Teile im Arbeitsspeicher zu halten. Die Programme und Daten werden in einzelne Teilabschnitte zerlegt, die dann nur bei Bedarf (wenn sie von der CPU benötigt werden) in den Speicher geladen werden.

################################

2. Was ist MMU?

In der MMU (Memory Management Unit) werden logische Adressen in physische Adressen umgeformt. Physisch ist die MMU entweder in der CPU integriert oder nahe angeordnet, logisch gesehen liegt sie zwischen CPU und Hauptspeicher.

################################

3. Was ist der Unterschied zwischen einer absoluten und einer relativen Adressen? Welche Vorteile haben relative Adressen?

Wir befinden uns inhaltlich beim Lader, der ein bestimmtes Programm laden möchte und dafür sorgen muss, dass die im geladenen Programm auftretenden Adressen im logischen Adressraum des Prozesses liegen. Hierzu gibt es zwei Möglichkeiten:
a) Wenn bereits der Binder die Adressen des logischen Hauptspeichers kennt, der bei der Ausführung des zu erzeugenden Lademoduls verfügbar sein wird, kann er bereits passende Adressen erzeugen. Diese Adressen nennt man auch "absolute Adressen." Nachteil der absoluten Adressen ist, dass das Lademodul nur noch an genau diesen Bereich logischer Adressen geladen werden kann.
b) Abhilfe schaffen die relativen Adressen. Hier kann das Lademodul an einen beliebigen logischen Adressbereich geladen werden. Der Binder nimmt an, dass im logischen Hauptspeicher von 0 an nummerierte Adressen zur Verfügung stehen. Diese Adressen nennt man "relative Adressen." Der Laden muss beim Laden dann nur noch die Startadresse des verfügbaren Bereichs logischer Adressen aufaddieren, um die im Programm auftretenden relativen Adressen in absolute

umzuformen.

##############################

4. Wie kann man den Speicher eines Anwenderprogramms schützen?
(Basisregister und Grenzregister)

Nehmen wir an, dass der vom BS belegte Teil des Arbeitsspeichers a Byte lang sei. Es geht nun darum, zu verhindern, dass ein Anwenderprogramm absichtlich oder unabsichtlich auf einen Bereich < a zugreift.
Eben ist der Speicherbereich eines Anwenderprogramms vor dem Zugriff durch ein anderes Anwenderprogramm zu schützen.

Es gibt zwei Möglichkeiten:
Entweder man verwendet ein internes, für das Anwenderprogramm unsichtbares Basisregister, welches zu Wert a enthält und benutzt im Anwenderprogramm nur Relativadressen zur Basis a. Das Anwenderprogramm läuft nun in einem virtuellen Adressraum ab und kann nicht auf Adressen < a zugreifen.
Wenn logische Adressen jedoch unverändert als physische Adressen verwendet werden, kann man eine Unterbrechung erzeugen, wenn ein Anwenderprogramm auf eine Adresse < a zugreift. Hierzu ist ein Grenzregister erforderlich, welches die derzeitige Grenze zwischen dem geschützten und ungeschützten Bereich, also den Wert a, enthält. Der Wert a sollte nicht fix verankert, sondern variabel sein, da die Größe des BS-Kerns variieren kann. Ein Setzen des Grenzregisters ist nur im Systemmodus möglich.

######################

5. Welche Speicherzuweisungsstrategien zur Verwaltung des Hauptspeichers gibt es?
(Einfach zusammenhängend, mehrfach zusammenhängend, MFT, MVT, Paging.
Bei MVT gibt es bei der Auswahl eines freien Segments Verfahren:
First Fit, Next Fit, Best Fit, Worst Fit und Buddy.
Eine kurze Beschreibung der Strategien ist notwendig.)

Einfach zusammenhängende Speicherzuweisung: Es befindet sich zu jedem Zeitpunkt maximal ein Benutzerprozess im Hauptspeicher.
Mehrfach zusammenhängende Speicherzuweisung: Mehrere Programme und deren Daten werden nebeneinander aufgenommen.

MFT: multiprogramming with a fixed number of tasks
Hierbei werden die Segmentgrößen, in die Speicher eingelagert wird, fix gewählt. Leider ist es fast unmöglich, günstige Segmentgrößen zu finden, da die benötigten Größen i.d.R. nicht vorher abgeschätzt werden können und Segmente i.d.R. immer zu groß sind, wodurch interne Fragmentierung entsteht. Diese kann jedoch durch geschicktes Scheduling verringert werden.
Zunächst einmal wird einem Auftrag das kleinste zur Verfügung stehende Segment zugewiesen, also kein unnötig großes (best-available-fit). Der nächste Schritt besteht darin, einem Auftrag nur ein Segment zuzuweisen, das nicht wesentlich größer als benötigt ist (best-fit-only). Ein Auftrag muss dann ggf. warten.
Eine falsche, bzw. ungünstige Segmentzuordnung ist jedoch nur dann ein Problem, wenn kein Segmentwechsel möglich ist (beim swapping (= Ein/Auslagerung von Prozessen)).

MVT: multiprogramming with a variable number of tasks
Beim MVT liegt die Segmentgröße nicht fest und wird erst dann bestimmt, wenn Speicher benötigt wird. Da die Segmentgröße dann genau an die benötigte Größe angepasst wird, kommt es nicht zu interner Fragmentierung. Dafür kann es jedoch zu erheblicher externer Fragmentierung kommen.
Beim MVT gibt es verschiedene Speicherplatzvergabestrategien, die entscheiden, wo eine Speicherplatzanforderung erfüllt wird:

First Fit:
- es wird die erste verfügbare ausreichend große Lücke genommen
- bei kleinen Adressen entstehen besonders viele kleine Lücken
- die Suche nach großen Lücken dauert lange.
- i.d.R. am schnellsten

Next Fit:
- wie First Fit mit dem Unterschied, dass immer hinter der ehem. Lücke weitergesucht wird und nicht von vorne.

Best Fit:
- es wird die kleinste ausreichend große Lücke genommen
- Nachteil: Es entstehen viele kleine Lücken, die schwer nutzbar sind.

Worst Fit:
- die größte Lücke wird gewählt
- Vorteil: Der Rest kann am Besten wieder verwertet werden.
- Nachteil: Wenig Platz für große Segmente.

Buddy:
- Hauptziel ist die Vereinfachung der Verwaltung der Segmente.
- Der zugewiesene Speicherplatz wird immer auf die nächsthöhere 2er-Potenz aufgerunden => interne Fragmentierung
- Die Gesamtgröße des Speichers sei ebenfalls eine 2er-Potenz und wird halbiert und eine obere und untere Hälfte, die man buddies (Kumpel) voneinander nennt.
- Jede Hälfte kann erneut in zwei zueinander gehörige buddies geteilt werden.
- Vorstellung am Besten als Binärbaum
- Ein angefordertes Segment der Größe 2^x kann sofort gefunden werden.
- Bei Freigabe eines Segmentes wird dieses wieder mit seinem buddy zusammengelegt, sofern dieser auch frei ist.
- Vorteil: Anforderungs- und Freigabe-Operationen sind effizient implementierbar.
- Nachteil: Erhebliche interne und ggf. externe Fragmentierung.
- Siehe auch: Abb. 4.11 und 4.12 im Kurstext (S. 127 / 128)

Eine weitere Speicherzuweisungsstrategie ist Paging (siehe Aufgabe 7)

####################

6. Was ist eine interne Fragmentierung? Was ist eine externe Fragmentierung?
Welche Fragmentierung haben die Speicherzuweisungsstrategien?

Interne Fragmentierung entsteht dadurch, dass einem Prozess ein größerer Speicherbereich zugewiesen wird, als benötigt wird.
Externe Fragmentierung bedeutet, dass nicht zugewiesene Speicherbereiche zu klein sind, um die Speicherplatzanforderungen von auf Ausführung wartenden Prozessen zu erfüllen.
Bei MFT tritt vorwiegend interne Fragmentierung auf, bei MVT externe.
Bei Paging gibt es nur interne Fragmentierung.

####################

7. Was ist paging?
Was ist eine virtuelle Adresse?
Wie wird eine virtuelle Adresse bei paging auf eine physische Adresse abgebildet? (Die Seitentabelle macht genau diese Abbildung)
Wer führt diese Abbildung durch? (MMU, siehe Abbildung 4.15 im Kurstext)
Wie kann eine Seite individuell geschützt werden? (Protection-Bit)

Der logische Hauptspeicher wird in Einheiten einer bestimmten Größe unterteilt, die page (Seite) genannt werden. Die logische Adresse wird wie folgt in eine Seitennummer und einen Offset geteilt:
Eine Seite habe p Speicherzellen, die logische (virtuelle) Adresse v liegt dann in Seite s und Offset d wobei
s = v div p und d = v mod p
Beispiel: p = 5, v = 16 => s = 3 und d = 1
Auch der physische Speicher ist unterteilt - hier nennt man die einzelnen Stücke page frame (Seitenrahmen). Seite und Seitenrahmen sind gleich groß. Ein Seitenrahmen kann folglich genau eine Seite des logischen Hauptspeichers aufnehmen. Jetzt muss man sich nur noch merken, in welchen Seitenrahmen welche Seite steht. Diese Übersicht enthält die Seitentabelle ST.
Dementsprechend existiert für jeden Prozess eine derartige Seitentabelle. Nun wird klar, warum hier keine externe Fragmentierung im Hauptspeicher mehr auftreten kann: Zwar ist der logische Speicher weiterhin zusammenhängend, im physischen Speicher können die benachbarten Seiten jedoch in weit von einander entfernt liegenden Seitenrahmen abgelegt werden.
Ein weiterer Vorteil von paging ist, dass beim swapping (Ein/Auslagerung von Prozessen) nicht mehr alle Seiten eines Prozesses ausgelagert zu werden brauchen, sondern nur so viele wie freie Seitenrahmen benötigt werden.

Paging bietet uns auch neue Möglichkeiten des Speicherschutzes: Es ist nun möglich, jede Seite individuell zu schützen. Wenn z.B. Programme und Daten auf verschiedenen Seiten stehen, kann das Programm gegen Schreiben geschützt werden und auf den Daten schreiben erlaubt werden. Hierzu enthält die Seitentabelle ein Protection-Bit (Schutzbit), welches durch die Hardware bei jedem Zugriff geprüft wird. Es können drei Bits benutzt werden, jeweils eines für Leserecht, Schreibrecht und Ausführungsrecht.

Für jeden Seitenrahmen wird ein sog. dirty bit verwendet, welches anzeigt, ob die Seite seit ihrer Einlagerung in den Hauptspeicher verändert wurde. Dieses bit wird bei Einlagerung auf 0 gesetzt und beim schreibenden Zugriff auf 1.

################################

8. Was ist Shared Memory?
Wie funktioniert sie?

Was ist der Vorteil?

Gemeinsam von mehreren Prozessen genutzter virtueller Hauptspeicher. Wenn ein Prozess auf eine Seite des Shared Memory schreibt, erscheint der modifizierte Inhalt auch im Hauptspeicher des anderen Prozesses. Dies ist eine effiziente Art, Daten zwischen Prozessen auszutauschen, weil die Daten im physischen Hauptspeicher nicht mehr kopiert werden müssen.

Welche Probleme kann entstehen, wenn mehr als ein Prozess eine Shared Memory modifiziert?
(Inkonsistenz, Inkorrekt. Siehe auch die internet-basierte Aufgabe von KE 5.)

Was muss gemacht werden, damit diese Probleme vermeiden werden können?
(Synchronisierung der Prozesse)

##########

9. Wie oft wird der Hauptspeicher bei paging zugegriffen, wenn ein Wort im Hauptspeicher liegt?

Wenn ein TLB hit vorliegt, gibt es nur einen Zugriff auf den Hauptspeicher. Wenn ein TLB miss passiert, gibt es zwei Zugriffe auf den Hauptspeicher (unter der Annahme, dass wir eine einstufige Seitentabelle haben. Bei mehrstufigen Seitentabellen entsprechend mehr).

##########

10. Wie kann der Zugriff eines Worts im Hauptspeicher bei paging mit TLB beschleunigt werden?

Die komplette Seitentabelle wird im physischen Hauptspeicher abgelegt. Ein Teil der Seitentabelle, genau gesagt die aktuell benutzten Einträge, werden in einem schnellen Assoziativspeicher (TLB; Translation Lookaside Buffer) abgelegt. Diese enthalten etwa 8 – 16 Einträge. Wenn im TLB eine gesuchte Seitennummer vorhanden ist (TLB hit), wird die zugehörige Seitenrahmennummer ausgegeben. Dieser Vorgang ist etwa 10mal schneller als ein Hauptspeicher-Zugriff. Wenn die gesuchte Seitennummer nicht vorhanden ist (TLB miss), wird die zugehörige Seitenrahmennummer aus der Seitentabelle geholt und in die TLB eingetragen. Hierbei muss i.d.R. ein schon vorhandener Eintrag verdrängt werden.

##########

11. Wie kann man die Seitentabelle verwalten, wenn sie zu groß ist?

In der Regel darf eine Seitentabelle maximal so lang sein, wie eine Seite. Sollte man eine größere Tabelle benötigten, bietet sich das Konzept der mehrstufigen Seitentabellen an. Diese Seitentabellen werden nicht nur im realen Speicher, sondern auch im virtuellen Speicher abgelegt (Ein Teil liegt im realen Speicher, ein Teil im virtuellen Speicher). Das bedeutet, dass Seitentabellen genauso wie alle anderen Seiten dem Paging unterliegen. Zweck des ganzen ist es, dass nicht mehr die gesamte Seitentabelle im physischen Speicher gehalten werden muss. Insbesondere Tabellen, die gerade nicht gebraucht werden, sollten nicht im Speicher liegen.
Bei einem zweistufigen Seitentabellen-Verfahren gibt es eine erste Stufe, bei der jeder Eintrag die Adresse oder Seitenrahmennummer einer Seitentabelle der zweiten Stufe enthält.

##########

12. Was ist die Idee (das Ziel) des virtuellen Speichers?
(Schreiben eines Programms ohne Einschränkung der Größe des Hauptspeichers, Zerlegung von Programmen, nur der Teil im Hauptspeicher bleibt, der gerade gebraucht wird. Die Lokalitätseigenschaft unterstützt die Effizienz. Das Betriebssystem realisiert das virtuelle Konzept, so dass es transparent für Benutzer und Benutzerprozesse ist.)

Unter der Lokalitätseigenschaft verstehen wir, dass überwiegend hintereinander stehende Instruktionen oder Schleifen, die nur wenige Seiten umfassen, ausgeführt werden und selten zu weiter entfernt liegenden Programmteilen gesprungen wird. Es ist daher möglich, nur einen Teil des gesamten Programms im physischen Hauptspeicher zu halten.

##############################

13. Was werden gebraucht, um das virtuelle Hauptspeicher-Konzept mit paging zu realisieren?

"Für jeden logischen Hauptspeicher brauchen wir also Platz auf einem Hintergrund-Speicher (i.A. einer Festplatte), dem sog. Swap-Bereich, der beim Start des BS reserviert wird." (Skript S. 137 ganz unten). Weiterhin brauchen wir eine Seitentabelle.

##############################

14. Was muss das Betriebssystem bei der Erzeugung eines Prozesses tun, wenn das virtuelle Hauptspeicher-Konzept mit paging realisiert wird?

Bei der Erzeugung eines Prozesses wird ganz allgemein ein Speicherplatz für Programmsegment, Datensegment, Stacksegment und Prozesskontrollblock im Hauptspeicher reserviert. Wenn wir mit Paging arbeiten, verteilt sich diese Speicherreservierung auf physischen und virtuellen Speicher.

Bei der Erzeugung eines Prozesses muss man folglich eine Seitentabelle im Hauptspeicher initialisieren und einen Swap-Bereich im Sekundärspeicher reservieren.

##############################

15. Was ist ein Seitenfehler?
Was muss das Betriebssystem bei einem Seitenfehler tun?

Unter einem Seitenfehler (page fault) versteht man, dass sich eine Seite beim versuchten Zugriff nicht im Hauptspeicher befindet. Die Abarbeitung des Befehls wird zunächst einmal abgebrochen und eine Unterbrechung wird ausgelöst. Nun wird die Seite vom Hintergrundspeicher in einen freien Seitenrahmen eingelagert. Da dies u.U. dauert, wird ein Prozesswechsel durchgeführt, falls ein anderer Prozess ausführungsbereit ist. Schließlich wird der unterbrochene Prozess fortgesetzt und der abgebrochene Befehl erneut ausgeführt, diesmal ohne Seitenfehler.

##############################

16. Was heißt demand paging? Was heißt prepaging?

Demand paging bedeutet, dass Seiten immer erst bei Bedarf eingelagert werden, nie auf Vorrat. Im Gegensatz dazu werden beim prepaging Seiten eingelagert, die aktuell noch nicht, vermutlich aber in Zukunft benötigt werden. Das prepaging erweist sich wegen des Verdrängens schon vorhandener

Seiten normalerweise als nachteilig und wird nur in Ausnahmefällen angewandt.

##########

17. Was heißt Lokalität eines Prozesses?

Unter der Lokalität des Zugriffsverhaltens von Prozessen verstehen wir, dass überwiegend hintereinander stehende Instruktionen oder Schleifen, die nur wenige Seiten umfassen, ausgeführt werden und selten zu weiter entfernt liegenden Programmteilen gesprungen wird. Es ist daher möglich, nur einen Teil des gesamten Programms im physischen Hauptspeicher zu halten.

##########

18. Was heißt Memory-mapped Files?

Memory-mapped Files bedeutet, dass Seiten einer Datei in den virtuellen Hauptspeicher eingeblendet werden. Der Prozess sieht dann im virtuellen Hauptspeicher die Daten, die in der Datei stehen. Verändert er diese Daten, verändert er implizit auch den Inhalt der Datei.
Dies ist eine Alternative zum klassischen E/A-Verfahren.

##########

19. Welche Seitenauslagerungsstrategien gibt es?

FIFO:
- First-In-First-Out
- leicht implementierbar
- deutlich höhere Seitenfehlerrate
- Nachteil: FIFO lagert auch laufend intensiv genutzte Seiten aus.

Second Chance:
- FIFO-Erweiterung
- Ausgabepuffer
- Seiten werden nicht sofort ausgelagert
- ineffizient

Clock-Algorithmus
- andere Implementierung von Second-Chance
- Schlange wird durch zirkuläre Liste erreicht
- Seite eingelagert: R-Bit = 0
- Seite benutzt: R-Bit = 1
- Auslagerung geschieht wie folgt: Zeiger findet Seite mit R-Bit=1 => R-Bit=0. Zeiger findet Seite mit R-Bit=0 => Auslagerung

LRU
- Last Recently Used
- eigentlich eine sehr gute Strategie
- Problem: effiziente Implementierung
- ohne aufwändige Hardware-Unterstützung nicht implementierbar.

Zugriffsbit

- Idee ist ähnlich wie LRU, doch man merkt sich nicht den exakten Zeitpunkt, sondern eine Beobachtungsperiode.
- Realisierung über Zugriffsbit (Referenced-Bit; kurz: R-Bit)

Wie viele Seitenrahmen sollten einem Prozess zugewiesen werden?

Es gibt keine feste Zuweisung, da die Zuweisung etwas dynamisches ist. Daher kann zu Beginn keine Zahl festgelegt werden, wie viele Seitenrahmen ein Prozess bekommen sollte. Es gibt hierfür jedoch zwei Strategien:
Die Arbeitsmengenstrategie besteht darin, einem Prozess zu einem bestimmten Zeitpunkt so viel Seitenrahmen zuzuweisen, wie dessen aktuelle Arbeitsmenge groß ist. Bei jedem Seitenfehler wird einem Prozess folglich ein weiterer Seitenrahmen zugewiesen.
Die zweite Strategie ist die Kontrolle der Seitenfehlerrate. Wenn ein Seitenfehler auftritt, registriert das BS die Zeit t, die seit dem letzten von diesem Prozess verursachten Seitenfehler vergangen ist. Man legt einen Grenzwert T fest. Wenn t <= T, wird dem Prozess noch eine Seite mehr zugeteilt.
(siehe auch Skript Kap. 4.6.3, Seite 146ff)

##########

20. Was ist Seitenflattern?

Seitenflattern (auch Trashing genannt) meint eine Systemüberlastung. Dies tritt dann auf, wenn die Summe der Arbeitsmengen aller aktiven Prozesse größer als der verfügbare physische Hauptspeicher ist. Dieselben Seiten werden nun immer wieder ersetzt und neu eingelagert.
Trashing vermeiden: jedem Prozess so viele Seiten zuordnen, wie er mind. Benötigt oder bei Überlastung die Zahl der aktiven Prozesse verringern, indem z.B. Prozesse mit geringer Priorität vorübergehend stillgelegt werden.

Eine Liste von Fragen zur KE 5:

1. Was ist ein kritischer Abschnitt eines Prozesses?

Kritische Abschnitte sind Abschnitte, die lesend und/oder schreibend gemeinsame Daten verarbeiten, die von mehreren konkurrenten Prozessen genutzt werden. Damit kritische Abschnitte determinierte Resultate zur Folge haben, müssen solche Abschnitte ohne Störung durch die anderen konkurrenten Prozesse in einer Sequenz ausgeführt werden. Innerhalb eines kritischen Abschnitts greift ein Prozess exklusiv auf gemeinsam benutze Betriebsmittel zu.
Nicht gemeinsam benutzbare Betriebsmittel (z.B. Drucker, Diskettenlaufwerk) können von konkurrenten Prozessen nur in kritischen Abschnitten belegt werden.

####################

2. Was ist die Notwendigkeit des wechselseitigen Ausschlusses?

Damit die Resultate von Abläufen konkurrenter Prozesse determiniert bleiben, müssen deren kritische Abschnitte exklusiv ausgeführt werden, d.h. EIN kritischer Abschnitt zu EINER Zeit. Das Betreten und Verlassen eines kritischen Abschnittes muss abgestimmt (synchronisiert) sein. Es muss folglich zwei Funktionen enter_critical_region und exit_critical_region geben, die das Betreten und Verlassen des kritischen Abschnittes regeln.
Die Lösung dieses Problems bezeichnet man als "wechselseitigen Ausschluss".

Welche Anforderung gibt es an einen wechselseitigen Ausschluss?
a) EIN Prozess zu EINER Zeit in seinem kritischen Abschnitt.
b) Falls mehrere Prozesse gleichzeitig versuchen, den kritischen Abschnitt zu betreten, gibt es in endlicher Zeit eine Entscheidung, wer zuerst darf.
c) Ein Prozess, der eintreten möchte, darf nicht beliebig oft von anderen überholt werden, so dass er verhungert. Die Wartezeit eines Prozesses sollte fair sein.
d) Kein Prozess darf außerhalb eines kritischen Abschnittes einen anderen Prozess blockieren.
e) Ein Algorithmus zur Realisierung des wechselseitigen Ausschlusses darf nicht auf irgendwelchen Annahmen über die relative oder absolute Ausführungsgeschwindigkeit der Prozesse beruhen.

######################

3. Welche Synchronisierungsverfahren gibt es?
Welche Vor- und Nachteile haben die Verfahren?

- Synchronisationsvariablen
Wir definieren eine globale Variable s, über die ein Prozess den kritischen Abschnitt für den jeweils anderen freigibt.
Nachteil:
- Die kritischen Abschnitte können nur abwechselnd 1,2,1,2,... ablaufen -> kein Überrunden möglich
- Ein Prozess kann nicht selbst zweimal hintereinander den gleichen kritischen Abschnitt aufrufen.

Lösungsmöglichkeit:
- 2 Synchronisationsvariablen verwenden
- -> Neuer Nachteil: Gefahr eines Deadlock

- Semaphore (siehe Frage 4)
  Vorteil: hohe Geschwindigkeit, hardwareseitige Implementierung
  Nachteil: up- und down-Operationen sind überall im Text verstreut, man verliert den Überblick

- Nachrichtenaustausch
  - Ringpuffer
  - Sender-Empfänger-Prinzip
  - Unterscheidung zwischen Einweg-Kommunikation (nur von A nach B) und Zweiweg-Kommunikation (Sender A erwartet von Empfänger B eine Bestätigung).

- Monitore (siehe Frage 5)
  - Vorteil: alle Funktionen nur im Monitor, nicht überall verteilt

##################

4. Was ist ein Semaphor?

Wenn sich ein Prozess in seinem kritischen Abschnitt befindet, sollte kein anderer Prozess ständig nachfragen, ob er in den kritischen Abschnitt eintreten darf (busy waiting -> unwirtschaftlich, da unproduktive Prozessorzeit benötigt wird). Vielmehr sollte es möglich sein, dass ein Prozess, der eintreten möchte, deaktiviert und wieder aufgeweckt wird, wenn der derzeitige aktive Prozess seinen kritischen Abschnitt verlässt.
Dijkstra hat daher das Konzept der Semaphore entwickelt. Eine Semaphore kann als eine Variable betrachtet werden, die einen ganzzahligen Wert hat. Mit Hilfe dieses Semaphores wird eine bestimmte kritische Ressource überwacht (Def. lt. Skript, S. 167).
Initialisiert man s mit k, so werden höchstens k gleichzeitige Zugriffe auf diese Ressource zugelassen => meistens ist k=1.
Zur Benutzung der Semaphore stehen die Operationen down und up zur Verfügung.

Wie funktioniert down- und up-Operationen?
down(s) bedeutet, dass der aktuelle Wert der Semaphore s um 1 vermindert wird. Wird der Wert negativ, blockiert der Prozess, der down ausführt.
up(s) erhöht den Wert von s um 1. Ist der neue Wert nicht positiv, wird ein durch eine down-Operation blockierter Prozess freigegeben, er geht folglich vom Zustand blockiert in bereit über.
Sei s der Semaphor eines kritischen Abschnittes, so wird down(s) vor und up(s) nach dem kritischen Abschnitt ausgeführt.

Wie werden die atomaren down- und up-Operationen realisiert?
Zur Implementierung von Semaphoren benutzen wir eine Warteschlange, die Prozesse aufnimmt. s kann wie folgt interpretiert werden:
s > 0: s ist die Anzahl der Prozesse, die noch down ohne Blockierung ausführen können.
s <= 0: |s| ist die Anzahl der Prozesse, die sich in der Warteschlange des Semaphores s befinden.

Interessant ist, dass down und up (die ja kritische Abschnitte schützen sollen), selbst kritische Abschnitte haben. Der Vorteil von Semaphoren ist jedoch, dass die Operationen nur sehr kurze Zeit benötigten und die Implementierung in die Hardware verlagert werden kann.

######################

5.Was ist ein Monitor?

Hauptnachteile der Semaphoren aus softwaretechnischer Sicht sind, dass die down- und up-Operationen beliebig im Quelltext verstreut sein können und es schwierig ist, den Überblick zu bewahren. Synchronisation kann auch dadurch herbeigeführt werden, dass ein spezieller Prozess damit beauftragt wird, den Zutritt zu kritischen Abschnitten zu kontrollieren. Dies ist das Konzept des Monitors.
Definition lt. Skript S. 181:
"Unter einem Monitor versteht man eine Menge von Prozeduren, Variablen und Datenstrukturen, die als Betriebsmittel betrachtet und mehreren Prozessen zugänglich gemacht sind. Die gemeinsam genutzten Betriebsmittel können geschützt werden, indem sie im Monitor platziert werden."

Ein Monitor unterstützt die Synchronisation durch Bedingungsvariablen, auf die zwei Funktionen wirken:
wait(c): Prozess wird blockiert, wenn c nicht erfüllt ist; Prozess kommt in die Warteschlange der Bedingung c
signal(c): Wenn in einer Monitorprozedur die Bedingung c wahr wird, dann wird die signal-Operation aufgerufen. Ein wartender Prozess wird ein aktiver.

Monitor-Konzept ist vergleichbar mit einem Raum, zu dem es nur einen einzigen bewachten Eingang gibt, so dass sich immer nur ein Prozess im Raum befinden kann. Andere Prozesse geraten in eine Warteschlange für blockierte Prozesse, die auf die Verfügbarkeit des Monitors (Änderung der Bedingung c) warten. Sobald ein Prozess eingetreten ist, kann er sich selbst auf Grundlage der Bedingung c blockieren, indem er wait(c) aufruft. Wenn ein Prozess, der im Monitor läuft, eine Änderung von c bemerkt, ruft er signal(c) aus.
Einer der wartenden Prozesse wird nun fortgesetzt, der signalisierende Prozess wird angehalten. Dieser wird fortgesetzt, wenn der wartende Prozess den Monitor verlässt. (Monitor Hoare-Typ)

##########################

5. Was ist das Erzeuger/Verbraucher-Problem?

Zwei zyklische Prozesse Erzeuger und Verbraucher produzieren, bzw. konsumieren Ware (analogie im BS: z.B. E/A-Vorgänge). Es ist eine Einweg-Kommunikation vorgesehen, wobei der Austausch der Ware über einen Puffer stattfindet.
Der gleichzeitige Zugriff von Erzeuger und Verbraucher auf den Puffer kann jedoch zu Störungen führen.

Wie kann man das Problem mit Semaphor und Monitor lösen?

Siehe Skript 5.2.3.1 (S. 170ff)

######################

6. Was ist ein Deadlock?

Ein Deadlock ist eine Systemverklemmung. Ganz allgemein geschrieben, ein Zustand von Prozessen, bei dem mindestens zwei Prozesse auf Betriebsmittel warten, die einem jeweils anderen beteiligten Prozess zugeteilt sind.
Beispiele: Verkehrsstau, Buchausleihe (A liest a und benötigt b; B liest b und benötigt a)

######################

7. Welche vier Deadlock-Bedingungen müssen erfüllt sein, wenn ein Deadlock vorlegt? (Die vier Bedingungen sind notwendig, d.h. wenn ein Deadlock vorlegt, dann müssen die vier Bedingungen erfüllt sein.
Wenn die vier Bedingungen erfüllt sind und keine zusätzliche externe Ressourcen angeschlossen werden dürfen, dann sind die vier Bedingungen auch hinreichend.)

a) Wechselseitiger Ausschluss (Es kann zu einem Zeitpunkt nur ein Prozess ein Betriebsmittel nutzen)
b) Nicht-Unterbrechbarkeit (no preemption): Die Betriebsmittel können temporär nicht zurückgegeben werden, sondern bleiben dem Prozess bis zum Ende der Anforderung zugeordnet.
c) Halte-und-Warte (Hold-and-wait): Die Prozesse belegen exklusiv ein Betriebsmittel während sie noch auf weitere Betriebsmittel warten.
d) Zyklische Warte-Bedingung (circular wait): Es besteht eine geschlossene Kette aus Prozessen und Betriebsmitteln in der Weise, dass jeder Prozess ein oder mehrere Betriebsmittel belegt, die vom nächsten Prozess in der Kette benötigt werden.

########################

8. Wie kann man ein Deadlock erkennen?

Das BS führt regelmäßig einen Algorithmus zur Erkennung von Deadlocks aus, der das zyklische Warten erkennt.
(siehe Skript Kap. 5.3.1, Seite 188f)
Wir benötigen dafür
Betriebsmittelvektor – Betriebsmittelrestvektor – Belegungsmatrix – Anforderungsmatrix
repeat-Schleife prüft, ob alle noch benötigten Betriebsmittel verfügbar sind, dann wird dieser Prozess als beendbar markiert.

##################

9.Wie kann man ein Deadlock vermeiden?
(Der Bankier-Algorithmus)

Vermeidung von Deadlocks bedeutet, dass Informationen über zukünftige Betriebsmittel-Anforderungen genutzt werden, um zu versuchen, nur solche Betriebsmittel-Vergaben zuzulassen, bei denen nicht die Gefahr einer Verklemmung besteht.

Dies geschieht z.B. mit dem Bankier-Algorithmus:
Wie einem Bankier nur eine begrenzte Anzahl an Geld zur Verfügung steht, um die Kreditwünsche seiner Kunden zu erfüllen, so steht einem Betriebssystem auch nur eine begrenzte Anzahl von Betriebsmitteln zur Verfügung. Der Bankier hält deswegen immer noch so viel Geld in seinem Tresor zurück, damit er noch von mindestens einem Kunden das komplette Kreditlimit erfüllen kann. Dieser eine Kunde (Prozess) kann dann sein Geschäft erfolgreich zum Abschluss bringen und das verwendete Geld wieder zurück auf die Bank bringen. Nun kann es ein anderer Kunde haben.
Übertragen auf die Informatik bedeutet dies:
Jeder Prozess muss seine maximalen Anforderungen an allen Betriebsmitteln vorab bekannt geben, diese sind in der maximalen Anforderungsmatrix M gespeichert. Ein Zustand ist sicher, wenn es

mindestens eine Ausführungsreihenfolge der Prozesse gibt, die nicht zu einer Verklemmung führt, selbst wenn alle Prozesse sofort ihre maximal Anzahl an Betriebsmitteln anfordern. Das System wird in einem sicheren Zustand gestartet. Wenn ein Prozess Betriebsmittel anfordert, wird getestet, ob der Zustand durch diese Zuteilung unsicher würde. Falls ja, bleibt das System im sicheren Zustand und blockiert den Prozess so lange, bis es sicher ist, die Anforderung zu gewähren.

Problem: Keine endgültige Lösung des Deadlock-Problems, da die maximalen Anforderungen eines Prozesses kaum im Voraus verfügbar sind.

##########

10. Wie kann man ein Deadlock verhindern?

Bei der Verhinderung von Deadlocks wird versucht, zumindest eine der vier Deadlock-Bedingungen niemals erfüllen zu lassen.

Bed. 1): scheidet aus, da die wechselseitiger Ausschluss i.d.R. erforderlich ist
Bed. 2): Möglich durch Ressourcenentzug: Prozess müsste alle Betriebsmittel zurückgeben und diese wieder zusammen mit dem neuen Betriebsmittel anfordern -> würde funktionieren, ist aber nicht immer praktikabel, da z.B. einem Prozess ein Ausgabegerät nicht entzogen werden kann, auf das er gerade etwas ausgibt.
Bed. 3): Prozess müsste alle in Zukunft benötigten Betriebsmittel von Anfang an anfordern (alle oder keines). Nachteil: Lösung ist nicht effizient und man kann nicht immer im Voraus sehen, wie viele Ressourcen benötigt werden.
Bed. 4): lineare Ordnung der Betriebsmittel ("wichtiger als") -> nur weniger wichtige Betriebsmittel können nachträglich angefordert werden; funktioniert, da die weniger wichtigen Betriebsmittel nur von Prozessen blockiert werden können, die auf noch weniger wichtige Betriebsmittel warten. Nachteil: Eine vernünftige Ordnung für alle Ressourcen kann man kaum finden.

Fragen zu KE 6

1. Was sind die Ziele der Sicherheitsmaßnahmen?
(Vertraulichkeit, Integrität, Verfügbarkeit...)

- Vertraulichkeit (Daten dürfen nur von Personen gelesen werden, die hierzu befugt sind)
- Integrität (Unversehrtheit und Korrektheit von Daten)
- Verfügbarkeit (Alle Programme und das BS müssen stets verfügbar sein => Gefahr z.B. durch denial-of-service-Angriff)
- Schutz vor finanziellem Verlust (Rechner wird von Personen benutzt, die hierfür nicht zugelassen sind und die Kosten nicht tragen)
- Schutz für Missbrauch und kriminellen Handlungen

##########################

2. Was ist das SETUID-Bit?

Def. lt. Wikipedia:
"Setuid (Set User ID, manchmal auch suid) ist ein Zugriffsbit für Dateien oder Verzeichnisse des Unix-Betriebssystems. Ausführbare Programme bei denen dieses Bit gesetzt ist, werden mit den Rechten des Benutzers ausgeführt dem die Datei gehört, anstatt mit den Rechten desjenigen Benutzers, der die Datei ausführt." [...]
"Auf Verzeichnissen bewirkt Setuid, dass Dateien, die innerhalb des Verzeichnisses angelegt werden, nicht dem Benutzer gehören, der sie anlegt, sondern dem Eigentümer des Verzeichnisses."

###################

3. Können Sie ein Beispiel der Anwendung von SETUID-Bit angeben?
(Die Änderung der passwd-Datei)
Wie funktioniert die Änderung der passwd-Datei?

Die Datei, in der unter UNIX die Passwörter aller Benutzer gespeichert sind, kann nur von root beschrieben werden. Dennoch sollte jeder Benutzer die Möglichkeit haben, sein eigenes Passwort zu ändern. Dazu gibt es das Programm usr/bin/passwd, bei welchem das s-bit gesetzt ist. Der Prozess erhält beim Aufruf die Rechte des Besitzers, also root, und kann die Passwortdatei ändern. Das Programm wird aber nur die Änderung des eigenen Passwortes erlauben.

##################

4. Wie funktioniert die Spooling-Technik?
Wie wird sie realisiert?
Ein Spooling ist eine Warteschlange oder ein Puffer, um die Dateien aufzunehmen, die z.B. ausgedruckt werden sollen. Diese Warteschlange muss geschützt werden, es kann nicht sein, dass jeder Benutzer-Prozess einfach seine Datei in die Warteschlange schreiben darf.
Also braucht man ein Druckerprogramm, bei der Ausführung des Programms wird ein spezieller Prozess erzeugt, der das Schreiben in die Warteschlange übernimmt. Dieser Prozess gehört einem speziellen Benutzer daemon, nur er darf das Programm schreiben und lesen, die anderen können das Programm nur ausführen(mit Schutzbits: rwxs--x--x) . Wenn das s-bit des Programm gesetzt wird, kann auch jeder Benutzer mit den Rechten des Besitzers das Druckerprogramms ausführen.

Das Schreiben in die Warteschlange wird durch das Programm kontrolliert.
####################

5. Was ist eine Zugriffskontrollliste?

Ganz allgemein ermöglichen Zugriffskontrollen, einzelnen Subjekten den Zugriff auf einzelne Objekte zu erlauben oder zu verbieten.

Die Zugriffskontrollliste (access control list, ACL) gehört dabei zu den granulatorientieren Implementierungen (die Bezeichnung "objektorientiert" wäre jedoch nahe liegender), da die Rechtefestlegungen beim Objekt gespeichert werden (im Gegensatz zur subjektorientierten Implementierung). Eine ACL ist ein Record, wobei jeder Eintrag für jeden zu einer Domäne gehörenden Prozess angibt, für welche Modi der Zugriff erlaubt ist, siehe Abb. 6.3., Skript S. 228.

Es gibt ferner die "benannten ACLs": Man kann ACLs als eigenständige Einheit auffassen, ihnen einen Namen geben und mehreren Objekten die gleiche benannte ACL zuweisen.

Was ist der Vorteil?
- bei ACL allgemein: Default-Wert ist "verboten", d.h. alles erlaubte muss explizit erlaubt sein
- bei benannten ACLs: Wenn mehrere Objekte die gleiche ACL haben, spart man Platz
- Der Besitzer einer Datei kann bei ACLs selbst festlegen kann, wer und welche Zugriffsrechte auf eine Datei hat.

###################

6. wie sehen die Schutzbits aus?

Bei Schutzbits werden zu jeder Datei drei Gruppen zu je drei Bits angegeben. Zu jeder Gruppe (Datei-Eigentümer; Gruppe (Eigentümer muss nicht unbedingt Mitglied in der Gruppe sein) und sonstige) legt jeweils ein Bit fest, ob Leserecht, Schreibrecht oder Ausführungsrecht erteilt ist.
Bsp. rwxrw-r-- bedeutet: Eigentümer darf lesen (r), schreiben (w) und ausführen (x), Gruppe darf lesen und schreiben, alle übrigen dürfen nur lesen.

###################

7. Wie kann man mit Schutzbits realisieren, dass eine Datei allen bis auf eine Gruppe von bestimmten Personen zugänglich ist?

Ich lege für die Personen, für die die Datei nicht zugänglich sein soll, eine Gruppe an und realisiere dann folgende Schutzbits:
rwx---rwx
Dies ist möglich, da die Schutzbits immer der Reihe nach geprüft werden. Wenn ich Mitglied in einer Gruppe bin und diese Gruppe keinen Zugriff hat, wird mir der Zugriff verweigert, auch wenn alle übrigen Zugriff bekommen.

###################

8. Was ist ein Capability?

Capabilities gehören zu den subjektorientierten Implementierungen, die Rechte sind folglich beim Subjekt gespeichert. Nun kann man leicht herausfinden, welche Rechte ein Subjekt hat, es ist jedoch schwerer herauszufinden, welche Rechte für ein Objekt vorhanden sind.

Capability ist äquivalent zu einem Profil -> Unterschied liegt in der Adressierung der Objekte.

Eine Capability besteht aus einem Objektverweis plus die Operationen auf dem Objekt. Dieser Verweis muss geschützt werden, z.B. das Betriebssystem bewahrt die Capabilities auf.

Beispiel Dateikontrollblock:
Wenn ein Prozess eine Datei mit open() öffnen will, sucht das Dateisystem zuerst die Lakalisierung der Datei auf der Festplatte. Danach erzeugt das Betriebssystem einen Dateikontrollblock (ein Dateikontrollblock heißt bei UNIX auch i-node) für die Datei, in dem der Besitzer der Datei, Gruppe, Opreationen auf der Datei (read, write, append), i-node usw. stehen. Das Dateikontrollblock wird in eine Tabelle von allen offenen Dateien (system-wide open-file table) eingetragen, die zur Verwaltung der offenen Dateien vom Betriebssystem gebraucht wird. Die Stelle des Dateikontrollblocks in der Tabelle ist eine ganze Zahl, also eine Nummer. Diese Nummer wird wieder in die Tabelle der offenen Dateien des Prozesses (per-process open-file table) eingetragen. Die Tabelle per-process open-file table heißt in Kurseinheit 7 Umsetztabelle des Prozesses, der Nummer des Dateikontrollblocks heißt Dateiidentifizierer, siehe
Abbildung 7.3 im Kurseinheit 7.
Jetzt werden die Rechte des Prozesses (Subjektes) auf die Datei im Betriebssystem aufbewahrt. Der Prozess hat auch nur die Rechte auf die Datei, die bei der Erzeugung des Dateikontrollblocks übergeben wurden. Wenn der Prozess die Datei lesen oder schreiben möchte, muss er den Dateiidentifizierer als Index zum system-wide open-file table als Parameter übergeben. Dieser Dateiidentifizierer ist der Objektverweis oder das Ticket.

Fragen zu KE 7:

1. Was ist ein Shell?

Eine Shell ist eine Kommandozeilen-Schnittstelle, bzw. ein Kommandointerpreter. Es ist der Teil des BS, der Kommandos einliest, interpretiert und alles weiter Erforderliche zur Ausführung der Aktivität veranlasst. Sollte nicht Teil des BS-Kerns sein.

##########

2. Wo kann man Shell am besten einsetzen?

In jedem BS. Eine Shell ist auch eine Art Benutzungsoberfläche um den BS-Kern (s. Skript S. 261) Shell ist sehr mächtig für die z.B. administrativen Aufgaben.

##########

3. Was ist ein Kommandoprozedur?

Ein Programm, welches in einer Kommandosprache geschrieben ist. Eine Kommandoprozedur enthält im einfachsten Fall eine Folge von Kommandos, die bei Ausführung der Kommandoprozedur der Reihe nach ausgeführt werden.

##########

4. Warum ist manchmal vorteilhafter mit einer Kommandoprozedur als mit einem z.B. C-Programm?

Eine kleine Kommandoprozedur kann "mal eben schnell" geschrieben werden, während ein C-Programm erst kompiliert und gelinkt werden muss.

##########

5. Was ist das Kommando make?
Was ist eine Makefile? Wie funktioniert eine Makefile?

Das Kommando make sorgt dafür, bei einer Programm-Aktualisierung nur die von der Veränderung betroffenen Programmteile zu aktualisieren, um das gesamte Programm auf den neuesten Stand zu bringen.
Dafür benötigt make eine Beschreibung der Abhängigkeiten zwischen den Dateien und die Definition der auszuführenden Aktion. Diese Angaben werden in einer Datei "makefile" abgelegt.
"make" prüft das Modifikationsdatum jeder Datei, ermittelt damit, welche Dateien geändert wurden, leitet aus den im "makefile" beschriebenen Abhängigkeiten die notwendigen Aktionen ab und führt sie aus. Das heißt, dass das letzte Veränderungsdatum in den Attributen einer Datei eine wichtige Rolle bei make spielt.
Siehe auch Abb. 7.4 auf S. 258.