Prüfungsfragen

1) Prozesse und Threads

1) Prozesszustände
New - Prozess wurde erzeugt und enthält eine pid, ist jedoch noch nicht ausführbar.
Ready - Prozess bereit zur Ausführung und wartet in der Queue auf die CPU.
Running - Prozess läuft momentan auf der CPU.
Blocked - Prozess wartet auf z.B. I/O Interrupts, ist nicht lauffähig bis dies eintritt.
Exit - Prozess terminiert und ist nicht mehr ausführbar.

Durch Swapping (Auslagerung der Prozesse auf Sekundärspeicher) zwei neue Zustände:
Ready, suspend - Blocked, suspend
Ausführung dieser Prozesse sehr teuer, da sie wieder zurück geswappt werden müssen.

2) Process Switch und Mode Switch
Führt zu Process Switch:
- Supervisor Call (explizierter Aufruf durch Programm)
- Trap (bedingt durch aktuelle Instruktionen z.B. Auftreten eines Fehlers)
- Interrupt (Ursache liegt außerhalb des Prozesses -> Interrupt Handler)

Process Switch:
- Kontext des Prozessors speichern (Register, PC)
- PCB des momentanten Prozesses updaten
- PCB in die entsprechende Queue verschieben
- Anderen Prozess zur Ausführung auswählen
- PCB dieses Prozesses updaten (und auf Running setzen)
- Memory managment updaten
- Prozessorkontext updaten

Um Datenstruktur des BS zu schützen gibt es zusätzlich noch den Mode Switch (user mode und privileged mode)
Mode Switch ist Basis von Process Switch, aber nicht jeder Mode Switch bewirkt einen Process Switch.

3) Swapping
Ist das Auslagern von Prozessen in den Sekundärspeicher, bei zuvielen Prozessen im Hauptspeicher -> Schlechtere Performance

4) Process Image
Wird vom BS zur Prozessverwaltung benötigt, Primary Process Table verweist auf Process Image (im Virtual Memory)
Enthält: User Program
User Data (modifizierbarer Bereich des User Spaces)
System Stack (Parameter und Calling Adresse von System Calls)
Process Control Block (Execution Context)

5) Process Control Block (PCB)
Ist Teil des Process Images, besteht aus:
- Process Identification: eindeutige Prozessnummer, User-Identifier (Benutzer, dem Prozess gehört), Parent-Process-Identifier (Nummer des Elternprozesses)
- Processor State Information: Registerinhalte, Stack Pointers, Kontroll- und Statusregister (Befehlszähler)

- Process Control Information: Verweise auf andere Prozesse (Parent, Child, Realisierung von Process-Queues), Scheduling und Zustandsinformationen (Prozesszustand, Priorität, Ereignis auf das der Prozess wartet, etc.)

6) OS Implementations
- Nonprocess Kernel:

Prozess nur für Benutzerprogramme, BS arbeitet getrennt von Prozessen im privileged Mode (eigener Speicherbereich)
- Prozessbasiertes BS:

BS ist eine Sammlung von Systemprozessen, nur Basisservices (Process Switching, Interrupts, I/O, etc.) sind nicht als Prozess implementiert

7) Kernel Architekturen
Monolithic OS - BS = Menge von Prozeduren, jede Prozedur kann jede andere aufrufen
Layered OS - Hierarchische Organisation der BS-Funktionen. Mehrheit der Schichten exekutieren im Kernel Mode.

Microkernel:
Basisservices befinden sich im Kernel (Interrupts, I/O Access, Process-Switching, etc.)
Nicht-zentrale Services als Server Prozess simuliert (Device Driver, File Server, Virtual Memory, etc.)
Vorteil: einheitliches Interface, Flexibilität, Portabilität,
Nachteile: mehr Kontextwechsel, komplexe Synchronisation -> langsamer als monolithisch

8) Process vs Thread
Prozess: Einheit für Ressourcenverwaltung (virtueller Adressraum mit Process Image)
Thread: Einheit für Dispatching (teilen sich Adressraum und PCB)
Mehrere Threads pro Prozess möglich

Vorteil: Thread Erzeugung, Terminierung und Umschaltung zwsichen Threads schneller als Process-Switch und können ohne Kernel miteinander kommunizieren
Nachteil: Synchronisation notwendig

9) User-Level Threads vs Kernel-Level Threads
ULT: Threads sind für Kernel unsichtbar, Managment mittels Thread Library
Vorteil: Für Thread-Switching kein Mode-Switch notwendig (geschieht im User-Mode)
Nachteil: Keine Verteilung auf Prozessoren möglich, Blockierung durch System-Call blockiert alle Threads

KLT: Thread Managment durch Kernel, Thread-Switching durch Kernel
Vorteil: Thread-weises blockieren, multi-threading
Nachteil: Thread-Switch benötigt 2 Mode-Switches -> langsamer als ULT

---

2) Mutual Exclusion und Synchronisation

1) Interaktion von Prozessen
Wettstreit um Ressourcen
Mutual Exclusion (Verhindern des gleichzeitigen Ressourcenzugriffs -> Konsistenz)
Bedingungssynchronisation (definierte Abfolge von Operationen)

2) Anforderungen für kritschen Abschnitt Lösung
Mutual Exclusion
Progress (kein ewiges warten bevor Prozess in k.A. eintritt -> Lifelock)
Bounded Waiting (Starvation)

3) Semaphoren Bsp 1)
A und B greifen auf Shared Memory zu, A hat Priorität

init: init(S, 1); init(X, 0)

A:
P(S);
read();
V(S);

B:
P(X);
P(S);
read();
V(S)
V(X);

Bsp 2) Producer-Consumer mit Ringbuffer

Initialisierung:

```
init (S, 1);   init (N, 0);   init (E, K);
in := out := 0;
```

```
append (v):
  b[in] := v;
  in := (in + 1)
        mod K;

take ( ):
  w := b[out];
  out := (out + 1)
         mod K;
  return w;
```

Producer:

```
loop
    produce (v);
    P (E);
    P (S);
    append (v);
    V (S);
    V (N);
end loop
```

Consumer:

```
loop
    P (N);
    P (S);
    w := take ( );
    V (S);
    V (E);
    consume (w)
end loop
```

Bsp 3) Reader Writer

```
init (x, 1); init (y, 1); init (z, 1); init (wsem, 1); init (rsem, 1);
rc := 0; wc := 0;
```

Reader:

```
loop
    P (z);
    P (rsem);
    P (x);
    rc := rc + 1;
    if rc = 1 then P (wsem);
    V (x);
    V (rsem);
    V (z);
      read;
    P (x);
    rc := rc - 1;
    if rc = 0 then V (wsem);
    V (x);
end loop
```

Writer:

```
loop
    P (y);
    wc := wc + 1;
    if wc = 1 then P (rsem);
    V (y);
    P (wsem);
      write;
    V (wsem);
    P (y);
    wc := wc - 1;
    if wc = 0 then V (rsem);
    V (y);
end loop
```

Writer haben Priorität

4) Monitor bei der Prozesssynchronisation
Monitor = Softwaremodul, sorgt für Mutual Exclusion, kein explizites Programmieren notwendig

Komponenten: Prozeduren/Zugriffsfunktionen, lokale Daten (gemeinsamer Zugriff), Monitorvariable (regelt Monitoreintritt)

Eigenschaften: gemeinsame Objekte sind gekapselt, max 1 Prozess im Monitor, Monitoreintritt wenn Bedingungsvariable erfüllt ist, Gemeinsamer Speicher im Monitorbereich

Funktionen: cwait(c): blockiert aufrufenden Prozess bis c (Bedingungsvariable) true annimmt
csignal(c): Setze einen Prozess, der auf c wartet fort (keine speicherende Wirkung wie Semaphoren)

5) Message Passing
Methode zur Interprozesskommunikation (auch für verteile Systeme). Eine Nachricht ist dabei eine atomare Datenstruktur. Es gibt außerdem unterschiedliche Synchronisationssemantiken (blocking, non-blocking, test for arrival)

Funktionen: send(destination, message)
receive(source, message)

3) Deadlock

1) Deadlock Bedingungen
- Mutual Exclusion - exklusiver Zugriff auf Ressourcen
- Hold and Wait - Prozess kann Ressourcen halten, während er auf andere Ressourcen wartet
- No Preemption - zugewiesene Ressourcen werden dem Prozess nicht weggenommen
- Circular Wait - geschlossene Kette von Prozessen, Prozess hält eine Ressource die von einem anderen Prozess benötigt wird und umgekehrt.

2) Deadlock Avoidance
Strategie um Deadlocks zu vermeiden, selektives Vergeben von Ressourcen
- Process Initiation Denial - Prozess wird nicht gestartet wenn seine Anforderung zu einem Deadlock führen könnte
- Ressource Allocation Denial - Ressourcenanforderungen eines Prozesses werden verwehrt wenn sie zu einem Deadlock führen könnten (Bankers Algorithm)

3) Safe und Unsafe state
Safe State: Ressourcenbelegung ist „Safe“, wenn jeder Prozess fertiggestellt werden kann ohne einen Deadlock zu verursachen -> Abarbeitungsreihenfolge erlaubt Fertigstellung aller Prozesse; bei einem Safe State ist kein Deadlock möglich

Unsafe State: keine solche Reihenfolge; Deadlock ist möglich, muss aber nicht eintreten

4) Banker Algorithmus

Vektor R - Anzahl der verfügbaren Ressourcen

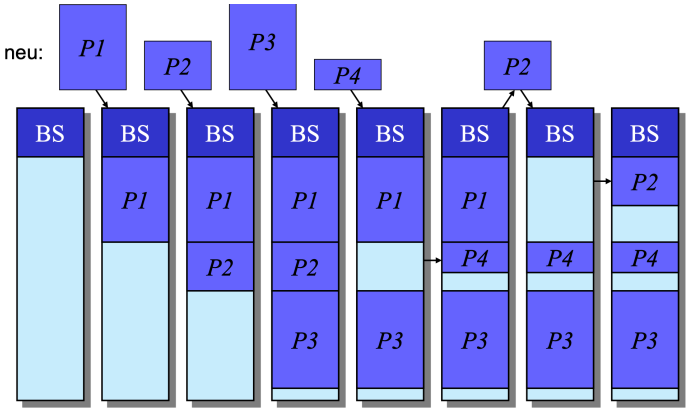
Matrix A - Momentane Allocation der Prozesse (get und free durchgehen)
Matrix C - Gesamtressourcen der Prozesse (maximale Auslastung der gesamten Folge)
Matrix N - Wieviele Ressourcen noch benötigt werden (C - A)

- A, C und N bestimmen
- Q Vektor zu entsprechenden Prozess in A addieren
- W berechnen (R - jede Spalte von A)
- Schauen ob ein Prozess in N <= W
- Falls Nein -> Unsafe State
- Falls Ja -> Prozess auf 0 setzen und A + W rechnen

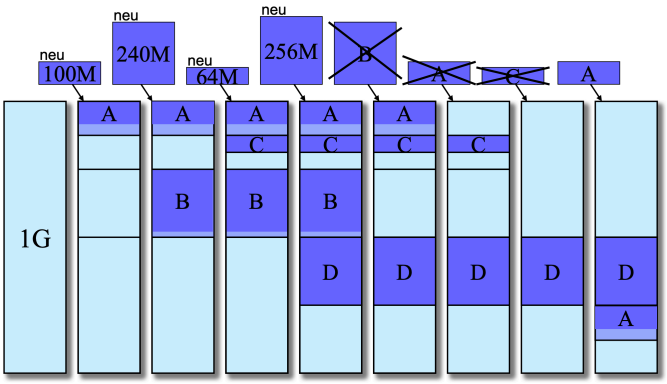
4) Memory managment

1) Anforderungen des Memory managments
- Partitioning (Speicheraufteilung auf Prozesse)
- Relocation (Positionierung von Coder und Daten im Speicher)
- Protection (Speicherschutz)
- Sharing (gemeinsamer Zugriff auf den Speicher)
- Performance (effektive log./phys. Organisation)

2) Partitioning
Fixed Partitioning: Unterteilung des Hauptspeichers in nicht überlappende Teile
Dynamic Partitioning: Partitionen variabler Länge und Anzahl



Buddy System: Speicher wird immer zur Hälfte zerkleinert bis 2^U-1 < S <= 2^U herscht.

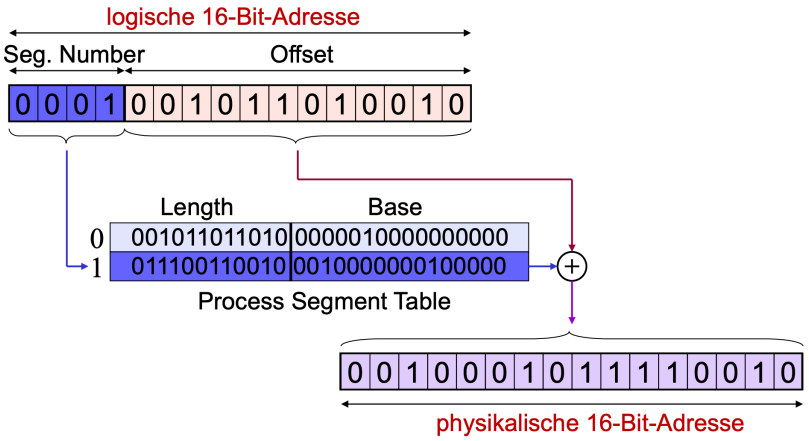


3) Fragmentierung
Interne Fragmentierung: Daten füllen eine Partition nicht aus -> Speicherplatzverschwendung (z.B. bei fixer Partitionierung)
Externe Fragmentierung: Speicherplatzverschwendung durch Zerstückelung des Speicherbereichs der Partiotionen (z.B. Löcher bei dynamische Partitionierung = Partitionen variabler Länge)

4) Relocation
Positionierung von Code oder Daten im Speicher
Speicherbereich eines Prozess ist nicht statisch fixiert (Speicherbelegung bei Prozessaufruf, Swapping, etc.)
Daher müssen Daten an verschiedenen Orten positioniert werden können -> Referenz auf physikalischen Speicher veränderbar
Unterschied zwsichen logischer und physikalischer Adresse

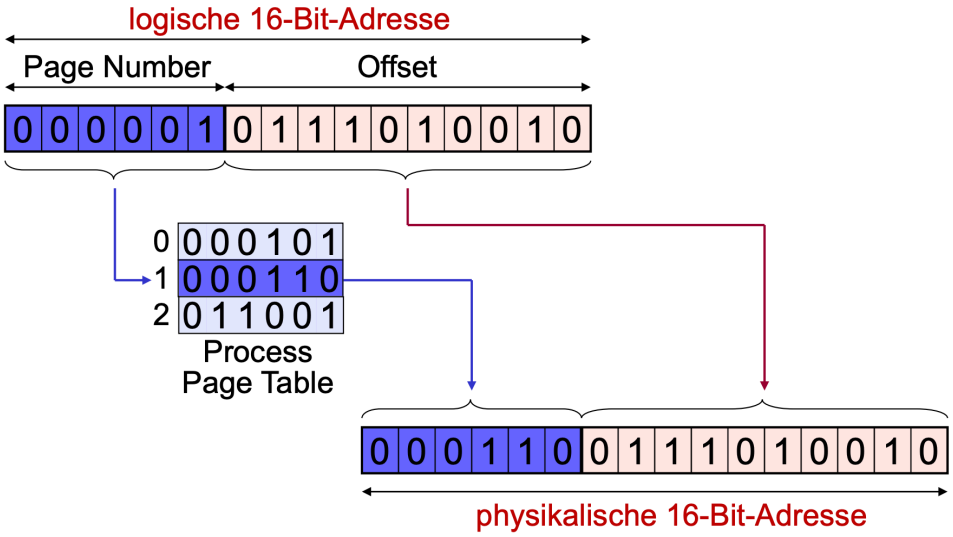
5) Segmentierung
Unterteilung von Programmen in Blöcke unterschiedlicher Länge. Diese Segmente enthalten Code oder Daten. Segmenttabelle pro Prozess (jedes Segment hat einen Eintrag: Start, Länge)

Adressübersetzung:

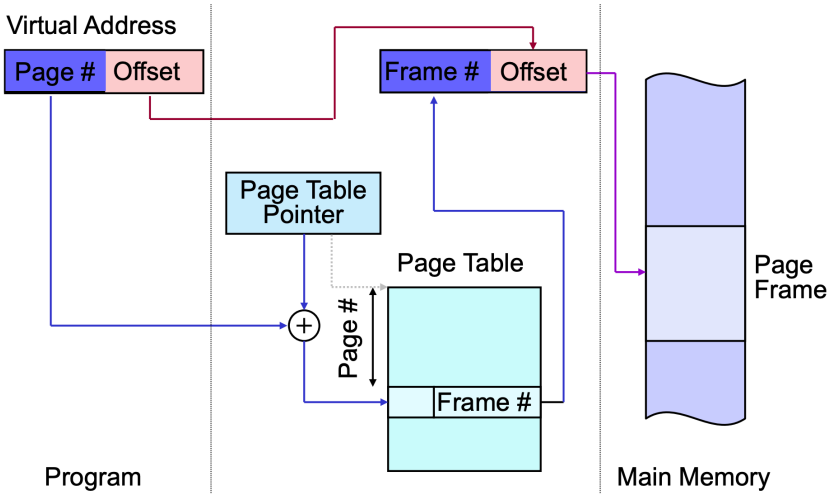


6) Paging/Page Table

Paging dient dazu den Hauptspeicher in Frames (gleicher Größe) zu unterteilen. Prozesse werden dann in Pages unterteilt. Pages werden dann von den Frames geladen.



Page Table Dient dazu um eine logische Adresse (Page Number, Offset) in eine physische Adresse (Frame Number, Offset) umzuwandeln.
Paging:



Pro Prozess gibt es ein Page Table:
Present Bit - Befindet sich Page im Hauptspeicher
Frame Number - Wo die Seite zu finden ist
Modified Bit - Wurde Seite seit dem Laden verändert
Control Bits - r/w-Bits, Locking, Kernel vs User Page

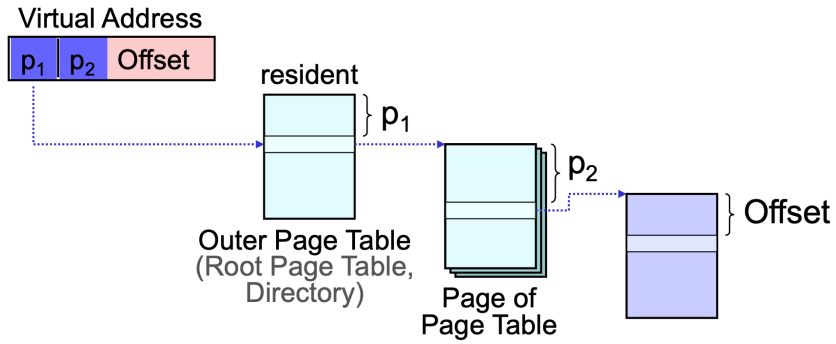
7) Virtual Memory
Verwendung von Speicher ohne Berücksichtigung der Größe des physikalischen Speichers. Nur aktuell benötigte Seiten laufender Prozesse werden im Hauptspeicher gehalten (Resident Set), Rest im Sekundärspeicher (Arbeitsspeicher).
Logische Adressen referenzieren Virtual Memory.

Vorteil: Mehr Prozesse passen in den Hauptspeicher (mehr als der eigentliche Speicher)

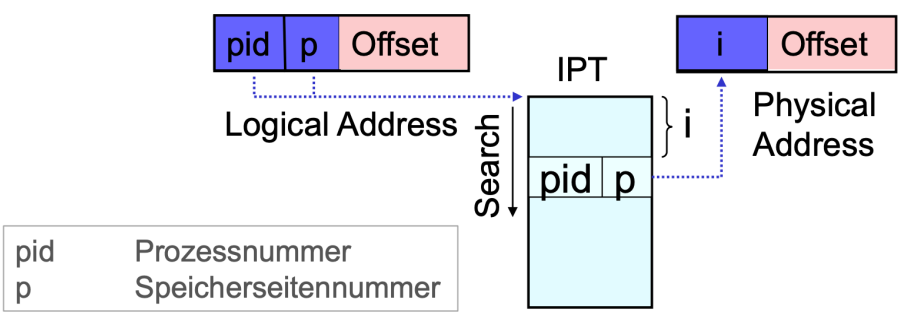
8) Page Faults
Ausnahmebehandlung falls eine Adresse im VM referenziert wird, die sich nicht im Hauptspeicher befindet. Zwei Ursachen:
- Seite vorübergehend nicht vorhanden, Betriebssystem lädt die Seite aus dem Hauptspeicher, aktualisiert Page Table und setzt Programmausführung fort
- Seite dauerhaft nicht vorhanden, Prozess wird mit Fehlermeldung beendet (Seg fault)

9) Thrashing
Durch häufige Page Faults (z.B. zu kleine Resident Sets) verbringt der Prozessor zuviel Zeit mit Laden von Sekundärspeicher -> Sehr langsam
Kann durch Working Set behoben werden.

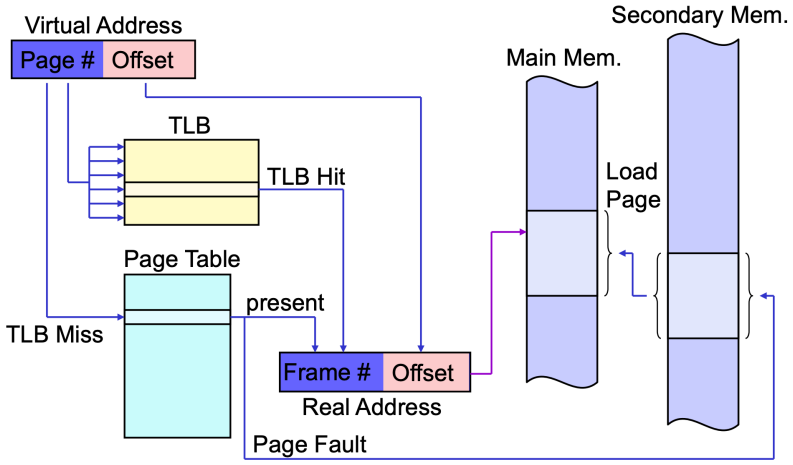
10) Multilevel Page Table
Bei großen Prozessen besteht Page Table selbst oft aus mehreren Seiten. Als Page Nummer speichert die logische Adresse die Adresse zur Outer/Root Table. Diese enthält dann die Adresse der gesuchten Seite, deren Nummer im Offset der logischen Adresse bereits angegeben ist.



11) Inverted Page Table (IPT)
Ein IPT für das gesamte System, ein IPT-Eintrag pro physikalischen Frame. Page fault falls IPT EIntrag nicht vorhanden (assoziative Suche oder Hashing um Suche zu beschleunigen)



12) Translation Lookaside Buffer (TLB)
TLB ist ein Cache für die Einträge einer Page Tables. Enthält die Nummern der zuletzt verwendeten Seiten -> erhöht Effizienz. Page Table im Hauptspeicher -> pro Zugriff auf VM sind zwei Zugriffe auf den physikalischen Speicher notwendig.
Bei jedem Process Switch werden Einträge des TLB gelöscht!



13) Fetch Policy
Bestimmt wann eine Seite geladen wird.
- Demand Paging - lädt eine Seite sobald diese referenziert wird -> viele Page Faults bei Prozessstart
- Prepaging - Localityprinzip -> lädt Seiten in der Nachbarumgebung im vorraus

14) Replacement Policies
- LRU:

Ersetzt die Seite die am längsten nicht benutzt wurde. Anzahl der Pagefaults kaum höher als OPT-Policy. Implementierung jedoch aufwendig (Speichern der Zugriffszeit, Suche nach Seite)

- FIFO:

Älteste Seite wird ersetzt. Schlecht, da eine häufig benutzte Seite die letzte sein könnte und ersetzt wird, aber einfache Implementierung (zyklischer Pointer)

- Clock Policy:

Ist eine Strategie die bestimmt welche Seite beim Laden einer neuen Seite in den Hauptspeicher aufgenommen wird.
Ist ein Ringpuffer mit Frames, die für den Austausch in Frage kommen und einem Positionszeiger. Jedes Frame hat ein Bit, das auf 1 gesetzt wird, wenn die Seite geladen oder referenziert wird. Bei Page-Fault -> Zeiger wird auf den nächsten Frame gesetzt, ab dort wird nach dem ersten Frame mit Use Bit 0 gesucht und ausgetauscht. Use Bits mit 1 werden auf 0 gesetzt.

15) Working Set
Ist eine Strategie die bestimmt wieviele Frames einem Prozess zugeordnet werden. Jeder Prozess hat ein Working Set = Menge an Seiten, die in einem letzten Zeitintervall referenziert wurden. Wenn das Resident Set (Teile im Hauptspeicher) weniger Einträge hat als das Working Set, werden Frames für den Prozess allokiert. Dies optimiert das Paging, da zu wenige Frames viele Page Faults verursachen und zuviele Frames die Parallelität beeinträchtigen.

---

5) Scheduling

1) Schedulingebenen
Long-term Scheduling - Erstellen von Prozessen, bestimmt Grad der Parallelität, Mix von CPU und I/O-intensiven Prozessen
Mid-term Scheduling - Ein- und Auslagern von Prozessen (Memory Managment)
Short-term Scheduling - bestimmt nächsten Prozess zur Ausführung (CPU Scheduler/Dispatcher)

2) CPU Scheduling Strategien

- FCFS (First Come, First Served) - Prozess der am längsten in der Ready Queue ist wird ausgewählt, Non-preemptive, Begünstigt lange und CPU-intensive Prozesse

- Round Robin - Wie FCFS mit Zeitscheibe fixer Länge, die nacheinander an Prozesse vergeben werden. Ist Zeit abgelaufen gelangt Prozess wieder ans Ende der Ready-Queue (preemptive). Benachteiligt I/O intensive Prozesse da diese die Zeitscheibe nicht voll ausnutzen können.

- Virtual Round Robin - Wie Round Robin nur mit zusätzlicher Auxiliary Queue. I/O Prozess kommt in eine eigene I/O queue, tritt das Event ein gelangt der Prozess in die Auxiliary Queue und bekommt CPU vor Prozessen in der normalen Ready Queue. Keine Benachteiligung wie bei normalen Round Robin mehr.
- SPN (Shortest Process Next) - Wählt Prozess mit kürzester zu erwarteten CPU-Zeit (wird geschätzt). Kann zu Starvation von langen Prozessen führen.

- SRT (Shortest Remaining Time) - Wie SPN aber preemptive. Kürzere Prozesse werden fair behandelt und weniger Interrupts als bei Round Robin.

- Highest Response Ratio Next - Selection function RR = (w+s) / s (w = bisherige Wartezeit, s = geschätzte Servicezeit) -> Auswahl des Prozesses mit größtem RR -> kürzere Prozesse fair behandelt, keine Starvation, Schätzung der Servicezeit jedoch notwendig

---

6) Input/Output - Disk Scheduling

1) I/O Strategien
- Programmed I/O - Programm selbst führt die I/O Operation aus (einfach)
- Interrupt-driven I/O - Programm setzt I/O command ab und geht in den wait Zustand, BS übernimmt I/O und benachrichtigt Task mit einem Interrupt
- DMA - Direct Memory Access Controller erledigt I/O Operationen. Prozess schickt I/O Kommando an Controller, dieser kopiert autonom Daten zwischen Speicher und I/O Modul und unterbricht den Prozess nach Fertigstellung. Kein Context Switch notwendig und Daten werden quasi parallel mit CPU-Aktivität transferiert.

2) I/O Schichten des BS

- Logical I/O - Oberste Schicht, stellt grundlegende Funktionen zur Verfügung (open, close, read)

Logical I/O hat nochmal drei Schichten:
Directory Managment, File System, Physical Organisation

- Device I/O - Übersetzung von Operationen in I/O- und Kontroll Kommandos + Buffering
- Scheduling and control - Queuing, Verwaltung von I/O Operationen, Interrupt handling, etc.

3) Synchrone vs Asynchrone I/O / Blocking vs non-blocking

Blocking - Prozess vom Zustand Running in Blocked Queue
Non-blocking - I/O Operation wird sofort fertiggestellt

Synchron: Prozess schickt I/O Request und wartet bis I/O Response da ist (von Running auf Blocked gestellt)

Asynchron: Prozess schickt I/O Request aber arbeitet dazwischen weiter, I/O wird zu späterem Zeitpunkt ausgeführt, kein Blockieren.

4) Buffering

Buffering ist das Zwischenspeichern von Daten
Bsp: Prozess A will Daten an Drucker Treiber schicken, Drucker jedoch beschäftigt mit anderen Tätigkeiten (Drucken von Seiten) -> BS speichert die Daten in einem Buffer

Vorteil: Erhöhte Performance und Flexibilität, weniger Overhead durch Zusammenfassen der Daten
Nachteil: Buffermanagment (Zuordnung Buffer - Prozess) und Speicher (Buffer-Overflow)

3) Berechnen der Zugriffszeit

$T(A) = T(S) + T(RD) + T(TF)$
$T(RD) = 1 / (2 * r)$
$T(TF) = b / (r * N)$

T(A) = Average Access Time
T(S) = Seek Time (Zeit um Disk Arm auf gewünschte Spur zu setzen)
T(RD) = Rotational Delay (Verzögerung bis gewünschter Sektor gefunden ist)
T(TF) = Transfer Time

b = Anzahl der zu übetragenden Bytes
N = Anzahl der Bytes pro Spur
r = Umdrehungsgeschwindigkeit

4) Disk Scheduling

Ziel: Minimierung der Seek-Time (Erhöhung der Zugriffsgeschwindigkeit) durch ordnen der I/O-Requests

FIFO - erster eingetragener I/O Request als erstes ausgeführt
LIFO - letzter eingetragener I/O Request als erstes ausgeführt
Priority - I/O-Requests erhalten eine bestimmte Priorität
Shortest Seek Time First (SSFT) - I/O Request mit kürzester Seek-Time als erstes

7) File Managment

1) File Allocation Table (FAT)
Wird bei Indexed Allocation verwendet
FAT ist eine Tabelle im Speicher, die Pointer auf Folgeblöcke einer Datei enthält. Pointer werden daher in einer eigenen Tabelle (FAT) gespeichert und nicht in den Blöcken der Datei selbst (wie bei Chained Allocation) -> mehr Platz für Nutzdaten, aber FAT benötigt mehr Platz im Arbeitsspeicher

2) Layout einer Disk/Filesystem
Disk in Partitionen unterteilt. MBR (Master Boot Record befindet sich im Sektor 0 der Disk: enthält Boot Code und Partition Table. Beim Hochfahren des BS wird Code des MBR ausgeführt. Partitionen werden lokalisiert und geladen durch Ausführung des Boot Blocks.

3) Block Allokierungsmöglichkeiten
Contigues Allocation: Datei belegt aneinander grenzende Blöcke.
Vorteil: gute Performance
Nachteil: externe Fragmentierung, Platzprobleme bei Vergrößern einer Datei

Chained Allocation: Blöcke sind über Pointer miteinander verbunden
Vorteil: keine externe Fragmentierung
Nachteil: langsamer Zugriff, Pointer verbrauchen Speicher

Indexed Allocation: Wie Chained Allocation aber Pointer befinden sich im FAT.
Vorteil: Blöcke nur für Nutzdaten verwendet.
Nachteil: Großer Platzverbrauch im Arbeitsspeicher

i-Nodes: Jedes File hat eine eigene Datenstruktur (i-Nodes) die alle wichtigen Informationen enthält (Attribute der Files und Referenzen auf Fileblöcke. Erster i-Node verweist auf root directory.
Vorteil: i-Node wird nur im Speicher gebraucht wenn File verwendet wird.
Nachteil: Anzahl der Referenzen ist begrenzt -> viele indirekte Blöcke