Operating Systems

Betriebssystem: Abstraktion von Hardware, Ressourcenverwaltung, Sicherheit/Schutz

User Mode	unprivilegierte Befehle teilweiser Zugriff auf Speicher	schützt Betriebssystem und Nutzerprozesse vor anderen Nutzerprozessen
Kernel Mode	alle Befehle kompletter Zugriff auf Speicher	bsp. privilegierte Instruktionen: └ Kontrollregister lesen/schreiben └ Interrupts de-/maskieren └ Gerätezugriff

Position-Independent-Code / dynamische Linking:
indirekten mit GOT / PLT

CPU-bound: Prozess arbeitet viel ohne I/O
I/O-bound: kurze Berechnung + viel I/O

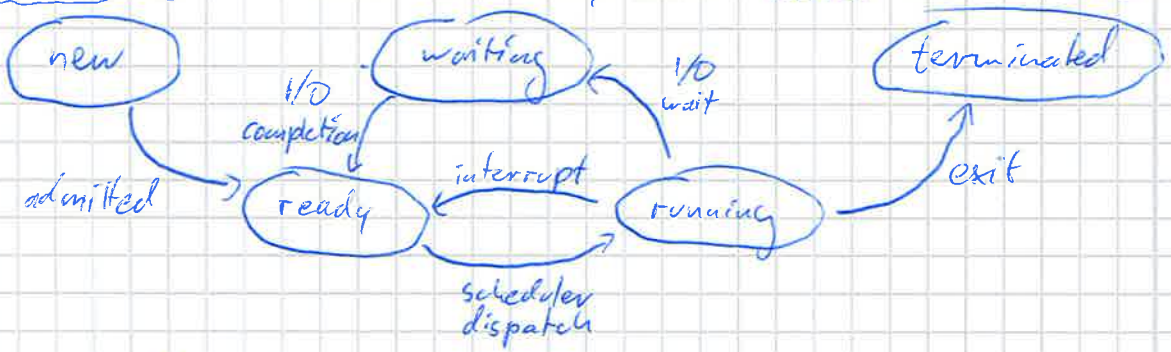
Betriebssystem schaltet sich ein:	Interrupts	Exceptions	System Call
freiwillig	nein	nein	ja
synchron	nein	ja	ja

Trap-Instruktion Prozessor wechselt in Kernel Mode
Syscall-Nr wird übergeben
Syscall-Parameter: Register / Stack / Pointer im Hauptspeicher

Scheduling : Prozesse ↔ Ressourcen
CPU scheduling
I/O scheduling

long-term : Prozess zur ready-Queue hinzufügen (admit)

short-term : Prozess aus ready-Queue auf CPU einplanen (dispatch)



Metriken für scheduling policy:
- Prozessor-Auslastung (processor utilization)
- Durchsatz (throughput)
- Zeit von Einreihen bis Fertigstellung (turn around time)
- Wartezeit (waiting time)
- Zeit von Einreihen bis erstem Laufen (response time)

nicht-kooperatives Scheduling braucht Hardware-Unterstützung (Interrupts)
weil Prozesse direkt auf CPU laufen (direct execution)

kurze Zeitscheiben	lange Zeitscheiben
- viele Kontext-Switches	- wenige Kontext-Switches
- Interaktivität	- wenig Overhead
- kurze Ladezeiten	- höheren Durchsatz (throughput)

Scheduling Policies Time-Slices werden wenden Widerholt an Prozesse in der Reihenfolge gegeben, wie sie gekommen sind.
Round Robin : ~~Prozesse werden~~
Shortest Job First (präemptiv : periodisch; bei neuen Job)

Priority Scheduling Prozess → Prio, Prozess mit max Prio läuft
Starving, wenn mehrere Prozesse selbe Prio.

Multi Level Feedback Queue

mehrere Warteschlangen mit verschiedenen ~~Prios~~
Prios (hohe Warteschlange ⇒ hohe Prio)
Zeitscheiben (hohe Warteschlange → kurze Zeitscheide)

Wähle Prozess aus höchster Warteschlange (round robin)
Prozess steigt in niedrige Warteschlange (Prio), wenn Zeitscheibe aufgebraucht, selbst er wurde Blockiert.

kooperatives Scheduling : Prozesse geben durch yield() CPU freiwillig ab.

Präemptiv : erzwungenes Scheduling
regelmäßiges Interrupts
Nutzer muss nicht von Scheduling können (yield)
verhindert nicht automatisch Starvation!

Process Switching

Process Controll Block: instruction pointer, stack pointer, general purpose Register, Adressraum, Scheduler-Info (Prio), offene Dateien

Threads

Kernel-level	User-Level	Kernel-Mode
= One-to-One	= Many-to-One	
Kernel hat Threads + Process-/Thread-tabelle	Userspace hat Threads + Thread-tabelle Kernel hat Prozess-Tabelle kein Parallelismus Blockieren alle Threads bei Syscall	

Many-to-Many (= Hybrid Threads)
N User-Threads ↔ M Kernel-Threads ($N \geq M$)
Parallelismus, kein kompletter Blockieren

Segmentation

Virtuelle Adressen: genutzt von Anwender / Betriebssystem
Physische Adressen: Hauptspeicher, MMU

Segmenttabelle: Segment (Base, Limit)
Adresse (Segmentnummer, Offset)

virt. adr.
0x4020
0100 0000 0010 0000
offset
segment nr

Segment table		
Nr.	Base	Limit
1	0xf154	0x013a

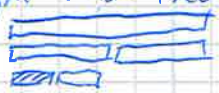
phys. adr.
0xf154 + 0x0020
= 0xf174 bis 0xf174 + 0x13a

Fragmentierung

intern: übrigbleibender Platz bei Allokation fester Größe
extern: ~~zerstückelter~~ zerstückelter Platz bei Allokation variabler Größe (wenn Speicher wieder freigegeben wird)

Memory Allocation Policies: First-fit, Best-fit, Worst-fit

Buddy-Allocator: Speicherbereiche der Größe 2^n; Große Blöcke werden rekursiv geteilt
Kombinieren von freien Buddies; Schneller Zugriff durch Free-Lists



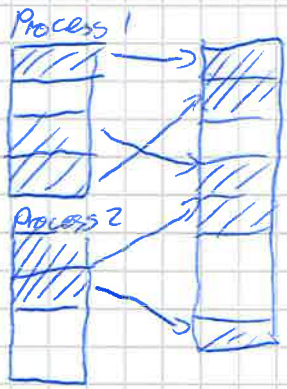
Slab-Cache: Verwaltet Objekte selber Größe
keine interne Fragmentierung
keine externe Fragmentierung

Paging virtueller → physischer Speicher
Pages → Frames
jeder Prozess hat zusammenhängenden virt. Adressraum
Zuordnung durch Page Table

virtual Address
0x30123
↓ vpn
Page nr | Offset
0x03 | 0x0123
vpn | n

Page Table
nr	pfn	Flags
3	0x15	...
:	pfn	
2^{vpn}		

→ Frame nr | Offset
0x15 | 0x0123
pfn | n
↓
physadr
0x150123
pfn



Multilevel Page Table: virt adr. hat mehrere Page nr
Tabelle muss nicht voll im Speicher sein
Adressübersetzung komplizierter

Inverted Page Table: eine Page Table im gesamten System
Zuordnung physisch → virtuell
PTE Eintrag: VPN + PID
platzsparend aber lineare Suche

Hashed Inverted Page Table

Translation-Lookaside-Buffer (TLB): Cache für Page-Table
schaut erst in TLB nach.
erst dann in PT und updatet TLB

Copy-on-write: bei fork greifen beide Prozesse auf Speicher zu, bis ein Prozess auf Speicher schreibt, erst dann wird kopiert

Page-Replacement-Policy: First-In; First-Out; Optimal (Theory)
Least-Recently-Used;
Clock
Random

Caches

Zeitliche Kohärenz
Räumliche Kohärenz

Interprocess Communication (IPC)
- geteilter Speicher
- über OS: MessageQueues, Pipes, Sockets

Synchron: Sender wartet bis Nachricht bei Empfänger
Blockierend: Sender wartet bis Nachricht im Nachrichtensystem

Race Conditions

Critical Section: Code-Abschnitt, in dem auf geteilte Ressourcen zugegriffen wird. Nur je ein Thread darf zugreifen.
Entry Section: Eintritts-Anfrage
Critical Section: geteilte Daten modifizieren
Exit Section: Abschnitt freigeben.

Anforderung an Synchronisations-Lösung
Mutual Exclusion (gegenseitiger Ausschluss)
Progress (Fortschritt)
Bounded Waiting (begrenztes Warten)
Performance (Leistung)

Spinlock: lock() { while (locked) {wait();}; locked=true; }
~~Problem: nicht selbst race cond.~~
benötigt atomare xchg operation!
User-space, kurze Wartezeiten

Mutex: lege thread schlafen und wecke ihn wenn unlocked wurde.
Kernel-Funktionen,
für lange Wartezeiten.

Futex: Spinlock am Anfang, Dann Mutex

Semaphore: wait(): counter--; thread schläft, wenn counter <0
signal(): counter++; sonst thread wecken

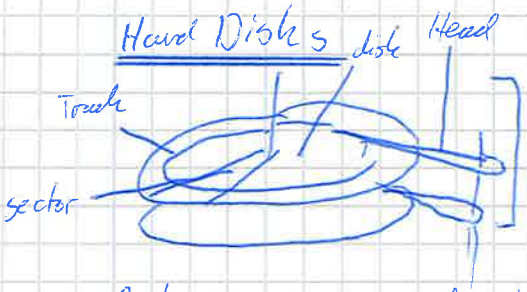
Condition Variable: Mutex mit Bedingung
~~wait, sign~~
wait(cond, mutex), signal(cond), broadcast(cond)

Deadlock: kein Fortschritt, weil mehrere Threads auf gegenseitig zugewiesene Ressourcen warten

Voraussetzungen: Mutual Exclusion: Ressourcen können nicht geteilt werden
Hold and wait: Gleichzeitiges Halten und Warten auf Ressourcen
No preemption: Ressourcen können Prozess nicht genommen werden
Circular Wait: keine zirkulären Abhängigkeiten

Deadlock-Detection: zyklische Abhängigkeit im Resource Allocation Graph

Hard Disks



Cylinder Head Sector (physisch)

Logical Block Addressing (abstrahiert)

Sector Sparing: ersetze kaputte Sektoren durch neue (Lokalität zerstört)

Sector Slipping: kaputte Sektoren überspringen

Slingbild: mehr Daten, weil verzettelt/überlagert geschrieben, langsames schreiben

Solid State Drive

- beim schreiben muss ein kompletter Block gelöscht und neu geschrieben werden
- durch remapping kann auf einen leeren Block geschrieben werden und später gelöscht werden

Trim: SSD mitteilen, welche Pages nicht mehr genutzt werden.

Raid

SLED (Single Large expensive Disk): hohe Kapazität + Geschwindigkeit
Robust + Fehlersicher

Raid 0: Block-Striping: Blöcke werden einfach auf Disks verteilt

Raid 1: Mirroring

Raid 2: Für Disks wird werden die Hamming-Codes auf weiteren Disk gespeichert

Raid 3: word-interleaved-parity: parity-disk, high throughput

Raid 4: block-interleaved-parity: parity-disk für mehrere Blöcke
z.B. „0“: update braucht 2 reads + 2 writes
parity = Bottle neck

Raid 5: block-level distributed parity: distribute parity on all disks

Raid 01: mirror of stripes
fail if 2 drives in 2 stripes fail

Raid 10: stripe of mirror
fail if 2 drives in same group fail
Lower probability of failure

Disk Space Allocation

Contiguous Allocation: Datei hat Start-Adr und Länge
hat interne/externe Fragmentierung

Chained / Linked List Allocation:
wachsende Dateien variabler größe
keine externe Fragmentierung
schlechte random R/W Performance

File Allocation Table: Tabelle mit Referenzen zu anderen Einträgen
variable Dateigröße
kann sehr groß werden → hohe RAM kosten

Indexed Allocation: ein Block hat Liste von Pointern auf Datenblöcke
variable Dateigröße
schnelles zufälliges Lesen/schreiben
geringe RAM-kosten
Platzverschwendung bei kleine Dateien

Inode
direct blocks
single indirekt
double indirekt
triple indirekt

Inhalt: Größe
Typ: regulär, dir, link
Zugriffsrechte
Timestamps
Anzahl Hardlinks
Nummern der Datenblöcke