1) **Betriebsarten**

Stapelverarbeitung (Lochkarten)

- Rechnerfamilien für wissenschaftliche und kommerzielle Berechnungen
- günstig (ICs statt Röhren)
- Software sollte auf diversen Rechnern laufen

Mehrprogrammbetrieb (Mutliprogramming, Multitasking)

- Gleichzeitiges Bereithalten mehrerer Jobs im Hauptspeicher (Partitionierung)
- Auf anderen Job umschalten statt auf E/A zu warten
- Timesharing
    - Jeder Benutzer hat Zugang zum System über sein Terminal

2) **Betriebssystemstrukturen**

Kernaufruf

- Anwendungsprogramm springt über TRAP in den Kern und führt den Code selbst aus.
- BS Code bestimmt die Nummer des angeforderten Dienstes.
- BS Code lokalisiert Prozedur-Code für Systemaufruf und ruft sie auf.
- Kontrolle wird an das Anwendungsprogramm zurückgegeben.
- Wichtig: Kern selbst ist passiv (Menge von Datenstrukturen und Prozeduren)

Betriebsmodi

- meisten 2 Modi (privilegiert, nicht-privilegiert), bei x86 4 Modi.
- Hardwaresicht (Privilegierter Modus)
    - Sperren von Unterbrechungen, Zugriff auf Speicherverwaltung-Hardware
    - Exceptions (Interrupts,TRAPs und Faults (z.B. division by 0)) schalten in den privilegierten Modus
- Betriebssystemsicht (Benutzungsmodus = nicht-privilegiert)
    - Beschränkter Zugriff auf Betriebsmittel
    - Unberechtigter Zugriff auf Betriebsmittel lösen Faults aus
    - Unerlaubte Operationen lösen Faults aus
    - Systemcall = expliziter TRAP Befehl
- Betriebssystemsicht (privilegiert)
    - Uneingeschränkter Zugriff auf alle Betriebsmittel
    - Faults und Exceptions führen zum Absturz

Monolithische Systeme (Windows, Unix, ...)

- prozedurorientiert
    - Kern ist passiv und der Code besteht aus einer Menge von Prozeduren
    - Struktur: Hauptprogramme → Dienstprozeduren → Hilfsprozeduren
- Nachteil: Viel Code → viele Fehler, nicht alle Anwedungen benötigen alle Dienste , Art und Anzahl der Dienste vom Kern vorgegeben

Client/Server-Strukturen (Mikrokerne)

- Ansatz: Nur Dienste die im Kernmodus laufen müssen, dürfen in diesem laufen
- Dateisystem, Netzwerkprotokolle, Speicherverwaltung müssen nicht im Kern sein , „Server“-Prozesse (ohne besondere Privilegien) bieten diese Dienste an
- Kern bietet nur Dienste zur Kommunikation zwischen Klienten (Anwendungen) und Servern untereinander an
- Dienste werden durch Nachtrichten per IPC: Interprozesskommunikation von Servern angefordert (send & receive)
- Server liefern Dienste auch mit IPC-Nachrichten (reply & wait)

-Vorteile: Isolation der Systemteile, Erweiterbarkeit, Nachrichtenbasiert
- Policy & Mechanism
    - Beispiel Speicherverwaltung
        - Strategie (policy): Zuteilung von Speicher an Prozesse
        - Mechanismus (mechanism): Konfiguration der Hardware
- µKern SOLLTE klein und wenig komplex sein

-Single Server: Monolithisches BS in Server umwandeln
- Mehrere BS in einem Rechner, große Trusted Code Base, schlechte Performance

Virtualisierung
- Virtuelle Maschinen (Beispiel VM/370)
    - Trennen der Funktionen „Mehrprogrammbetrieb“ und „erweiterte Maschine“
    - Virtualisierung durch Hypervisor
    - virtuelle Maschinene als identische Kopien der Hardware
    -in jeder virtuellen Maschine: übliches Betriebssystem
- Virtualisierbarkeit (Anforderung: Identisches Verhalten der VM)
    - Emulation: Nachbild der HW ins SW (ineffizient!) [Bochs, JWVM]
    - Virtualisierung: die meisten Befehle werden von der realen Hardware ausgeführt,der Rest emuliert(schnell, Architekturabhängig)[QEMU,VMWare]
    - Paravirtualisierung (Falls nicht virtualisierbar): Priviligierte Befehle des Gast-BS durch „Hypercalls“ (= Aufrufe in den Hypervisor) ersetzen. Schnell oder schneller als Virtualisierung, aber Gast-BS muss angepasst werden. [Xen, KVM, Hyper-V]

3) **Prozesse und Threads**

Prozessmodell
- Prozess: ein in sich in Ausführung befindliches Programm inkl.Stack, Register,Pc
    - Menge von (virtuellen) Adressen, von Prozess zugreifbar
    - Programm und Daten in Addressraum sichtbar
- Verhältnis Prozessor - Prozessor
    - Prozess besitzt konzeptionel eigenen virtuellen Prozessor
    - Reale(r) Prozessor(en) werden zwischen virtuellen Prozessoren umgeschaltet (Mehrprogrammbetrieb)
    - Umschaltungseinheit heißt Scheduler oder Dispatcher
    - Umschaltvorgang heißt Prozesswechsel oder Kontextwechsel
- Prozesserzeugung
    1) feste Menge von Prozessen werden beim Systemstart erzeugt
    einfache, meist eingebettete System [Motorsteuerung, Videorekorder]
    einfache Verwaltung, deterministisches Zeitverhalten, unflexibel

    2) dynamisch (es können im Laufe der Zeit neue Prozesse erzeugt werden)
    impliziert die Bereitstellung geigneter Systemaufrufen durch BS
- Prozessende
    1) freiwillig: Prozess ist fertig (egal ob erfolgreich oder nicht)
    2) unfreiwillig: Prozess WIRD beendet (Bsp: Division 0, Segmentation Fault)
- Prozesshierarchie (Unix ja, Windows nein [Prozesse gleichwertig])

-Prozesszustände (aktiv, bereit, schlafend/blockiert) selten auch initiiert, terminiert

Implementierung
- PCB (Process Control Block)
  - Prozessverwaltung: Register, Id, Pc, StackPtr, Flags, Signal, Parent, Zustand
  - Speicherverwaltung: zeiger auf .text .data .bss, real und effektiv UID & GID
  - Dateisystem: effektive UID & GID, Flags, Wurzel- & aktuelles Verzeichnis
  - Zeiger zur Verkettung des PCB in (verschiedenen) Warteschlangen
- Scheduler-Aktivierung
  - kooperatives Multitasking: Problem MUSS Kontrolle an BS abgeben
  - preemptiv: Code wird unterbrochen bei z.B. Ablauf eines Timers, Scheduler wird aufgerufen
- Unterbrechungsbehandlung
  - Interrupt-Handler: Interrupt-Vektor-Tabelle (IVT) enthält Interrupts mit IDs
  - Ablauf: Pc (u.a.) wird durch HW auf dem Stack abgelegt
    HW lädt Pc-Inhalt aus Unterbrechungsvektor
    Assembly-Routine rettet Registerinhalte
    Assembly-Routine bereitet den neuen Stack vor
    C-Prozedur markiert den unterbrochenen Prozess als bereit
    Scheduler bestimmt den nächsten auszuführenden Prozess
    C-Prozedur gibt Kontrolle an die Assembly-Routine zurück
    Assembly-Routine startet den ausgewählten Prozess
- Interrupts aus Sicht des Prozesses
  - IRT: Interrupt Response Time
  - PDLT Process Dispatch Latency Time
  - SWT Process Switch Time

Threads (Leichtgewichtsprozesse für billige Nebenläufigkeit im Prozessadressraum)
- Idee einer „parallel ausgeführten Programmfunktion“
- eigener Prozessor-Context (Registerinhalte usw.)
- eigener Stack (i.d.R. 2, getrennt für unser und kernel mode)
- eigener kleiner privater Datenbereich (Thread Local Storage)
- Threads nutzen alles Betriebsmittel, Programm- & Adressraum des Prozesses
- WICHTIG: Bein 1-Prozessorsystemen kein Performancegewinn
- Kooperationsformen: Verteiler-/Arbeitermodell, Teammodell, Fließbandmodell
- Implementierung
  - Thread-Bibliothek (User level threads)
    - Threadfunktionen/Kontextwechsel auf Applikationsebene
    - einfache Implementierung, keine Nutzung von MehrprozessorArch
  - Im BS-Kern (Kernel level threads)
    - Threads als Einheiten denen Prozessoren zugeordnet sind
    - Nutzung von Mehrprozessor Architekturen,Kernelunterstüzung nötig

4) **Scheduling (Priorität- oder Zeitscheiben-basiert)**

Begriffe
-Bedienzeit: Zeitdauer für reine Bearbeitung des Auftrags
-Antwortzeit: Zeitdauer vom Eintreffen bis zur Fertigstellung des Auftrags
Bei Dialogaufträgen Zeitdauer von Benutzereingabe bis Ausgabe
Bei Stapelaufträgen auch Verweilzeit genannt
-Wartezeit: Antwortzeit - Bedienzeit
-Durchsatz: Anzahl erledigter Aufträge pro Zeiteinheit
-Auslastung: Anteil der Zeit im Zustand „belegt“

-Fairness: „Gerechte“ Behandlung aller Aufträge

Moderne Anforderungen

-Scheduling wird von Applikationsebene gesteuert

Non-Preemptive Scheduling (Annahme: Bekannte Bedienzeiten)

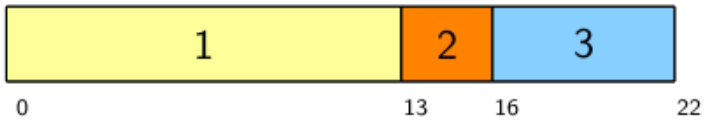
-FCFS (first come first served): Ready-Queue als FIFO Liste

Gegeben: Prozessmenge mit 3 Prozessen

Prozess	Bedienzeit
1	13
2	3
3	6

Alle Aufträge seien zur Zeit Null bekannt

Resultierender Schedule:



Prozess	Wartezeit	Antwortzeit
1	0	13
2	13	16
3	13+3=16	22

Durchschnittliche Wartezeit:
$(13 + 16)/3 = 29/3$

Im Falle der Ausführungsfolge 3, 2, 1 hätte sich ergeben:
Durchschnittliche Wartezeit: $(6 + 9)/3 = 5$

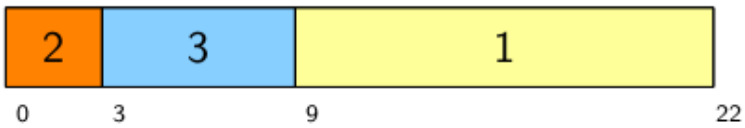
-SJF (Shortest Job First)

Gegeben: Prozessmenge mit 3 Prozessen

Prozess	Bedienzeit
1	13
2	3
3	6

Alle Aufträge seien zur Zeit Null bekannt

Resultierender Schedule:



Prozess	Wartezeit	Antwortzeit
1	3+6=9	22
2	0	3
3	3	9

Durchschnittliche Wartezeit:
$(9 + 3)/3 = 4$

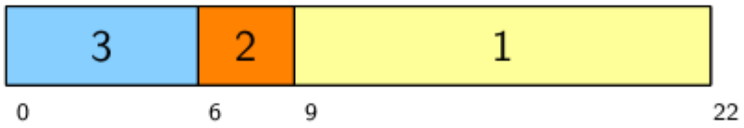
-Prioritäts-Scheduling: Jeder Auftrag hat statische Priorität
höchste Priorität hat Vorrang
Bei gleicher Priorität FCFS

Gegeben: Prozessmenge mit 3 Prozessen

Prozess	Bedienzeit	Priorität
1	13	2
2	3	3
3	6	4

Alle Aufträge seien zur Zeit Null bekannt

Resultierender Schedule:



Prozess	Wartezeit	Antwortzeit
1	6+3=9	22
2	6	9
3	0	6

Durchschnittliche Wartezeit[4]:
$(9+6)/3 = 5$

[4] In diesem Beispiel – abhängig von Prioritätsvergabe sind auch alle anderen Ergebnisse möglich

Preemptive Scheduling