Einführung in die Informatik

Betriebssysteme

Meik Teßmer

Inhalt der Veranstaltung

Lernziele:

- Notwendigkeit eines Betriebssystems
- seine Aufgaben
- technische Umsetzungen

Einstieg

Frage:

Wo überall sehen wir Betriebssysteme (oder sehen sie eben nicht)?

Einstieg

- offensichtlich: Computer (PCs, Laptops)
- Smartphones, Tablets
- nicht so offensichtlich:
  - Autos
  - Industrieanlagen
  - Fernseher
  - Häuser

Interaktion

Fragen:

- Wie interagieren Sie mit einem Betriebssystem?
- Worin besteht der Unterschied zu einer *Anwendung*?

Rückblick

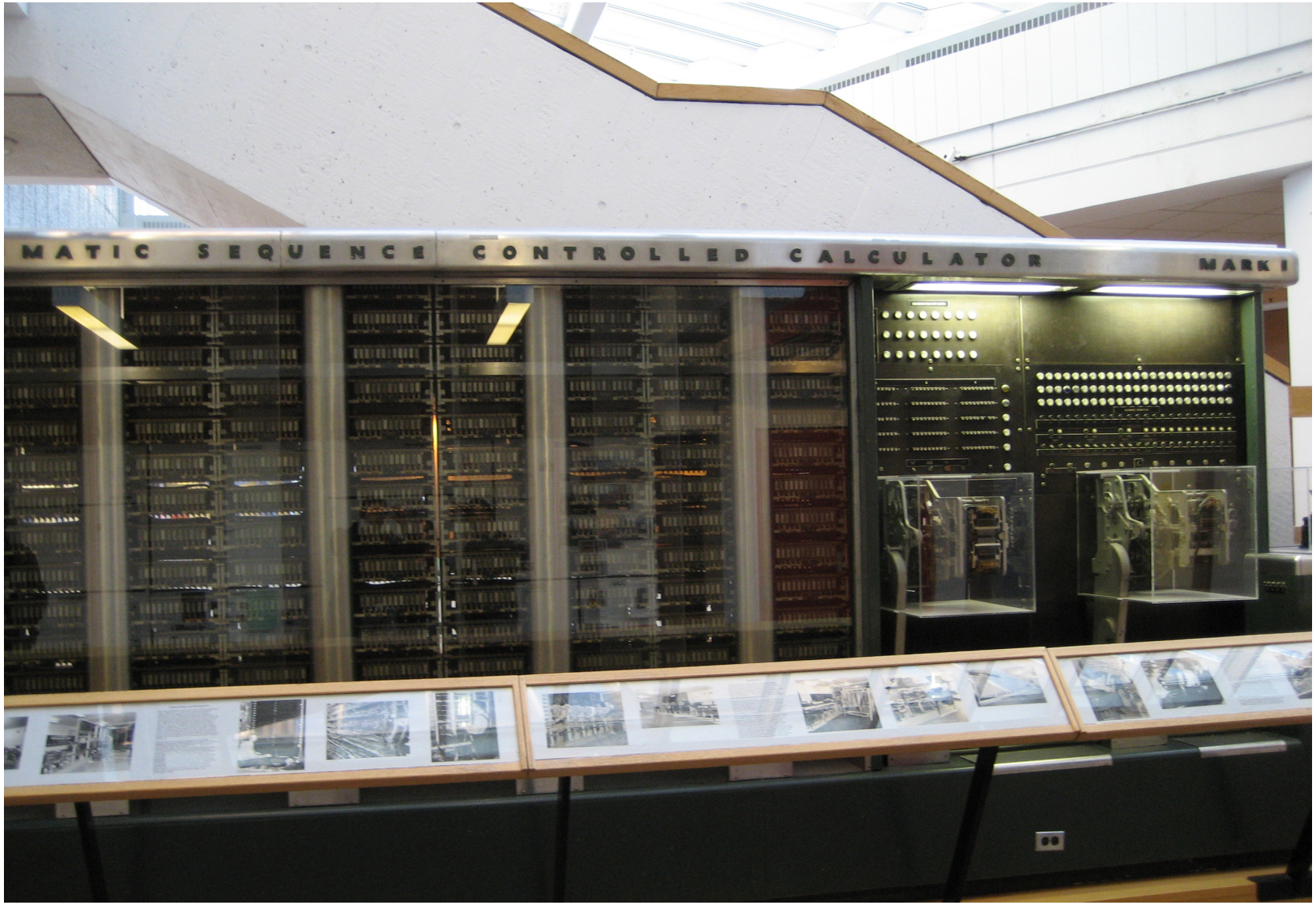
- Digitale Rechenmaschinen der 1. Generation
- Digitale Rechenmaschinen der 2. Generation
- Digitale Rechenmaschinen der 3. Generation

Digitale Rechenmaschinen der 1. Generation

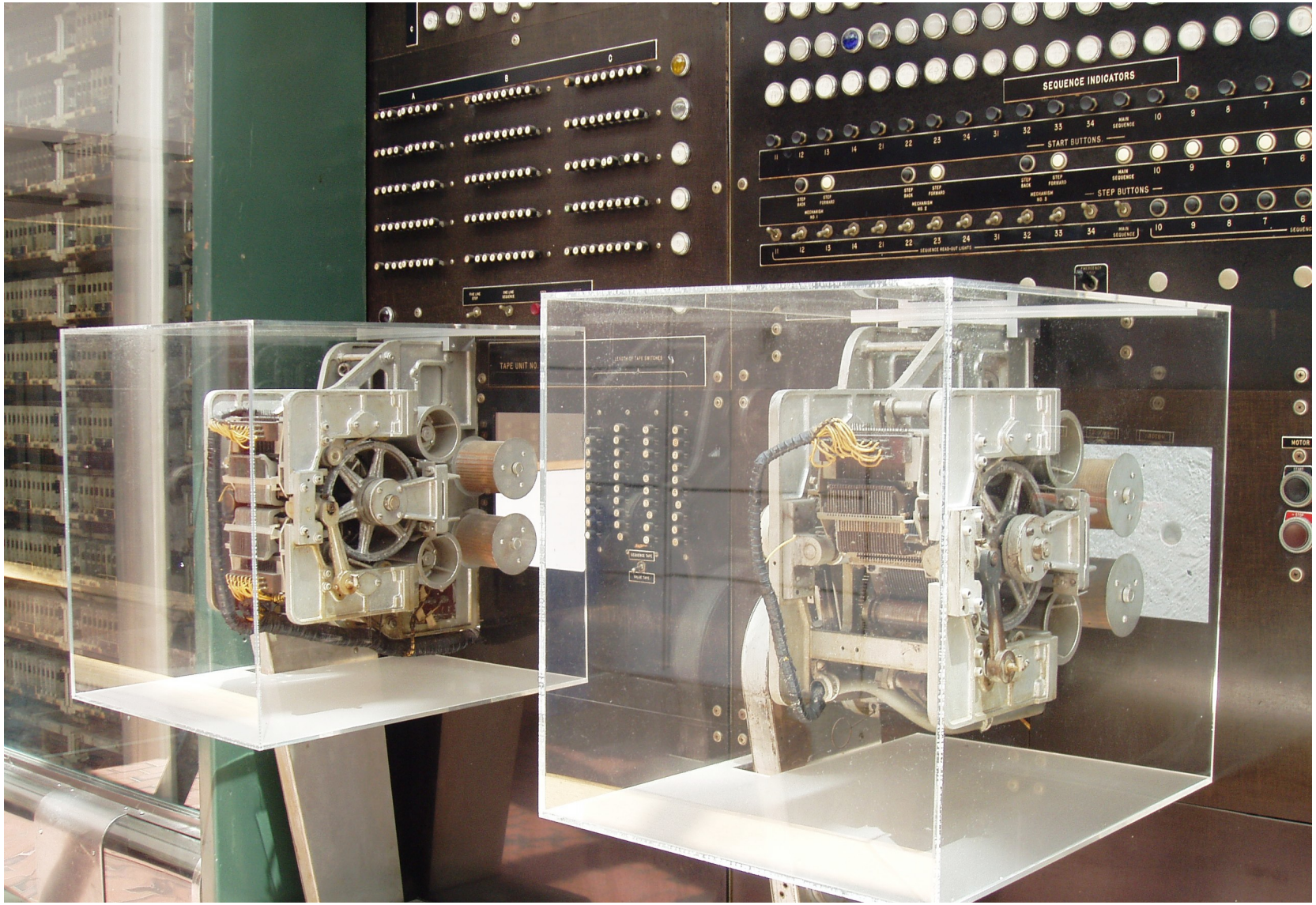
1939-1944: Howard Aiken und Grace Hopper konstruieren den ASCC

- der ASCC war der erste digitale Großrechner der Welt
- ASCC steht für *Automated Sequence Controlled Calculator*
- wurde später in *Mark I* umbenannt
- Ausmaße: ca. 16 Meter lang und 2,50 Meter hoch, Gewicht: 5 Tonnen
- bestand aus 760.000 Einzelteilen mit 3000 Kugellagern und 80 km Leitungsdraht

Digitale Rechenmaschinen der 1. Generation



Digitale Rechenmaschinen der 1. Generation



Digitale Rechenmaschinen der 1. Generation

- war frei programmierbar (Lochkarten)
- Betriebssystem: Fehlanzeige

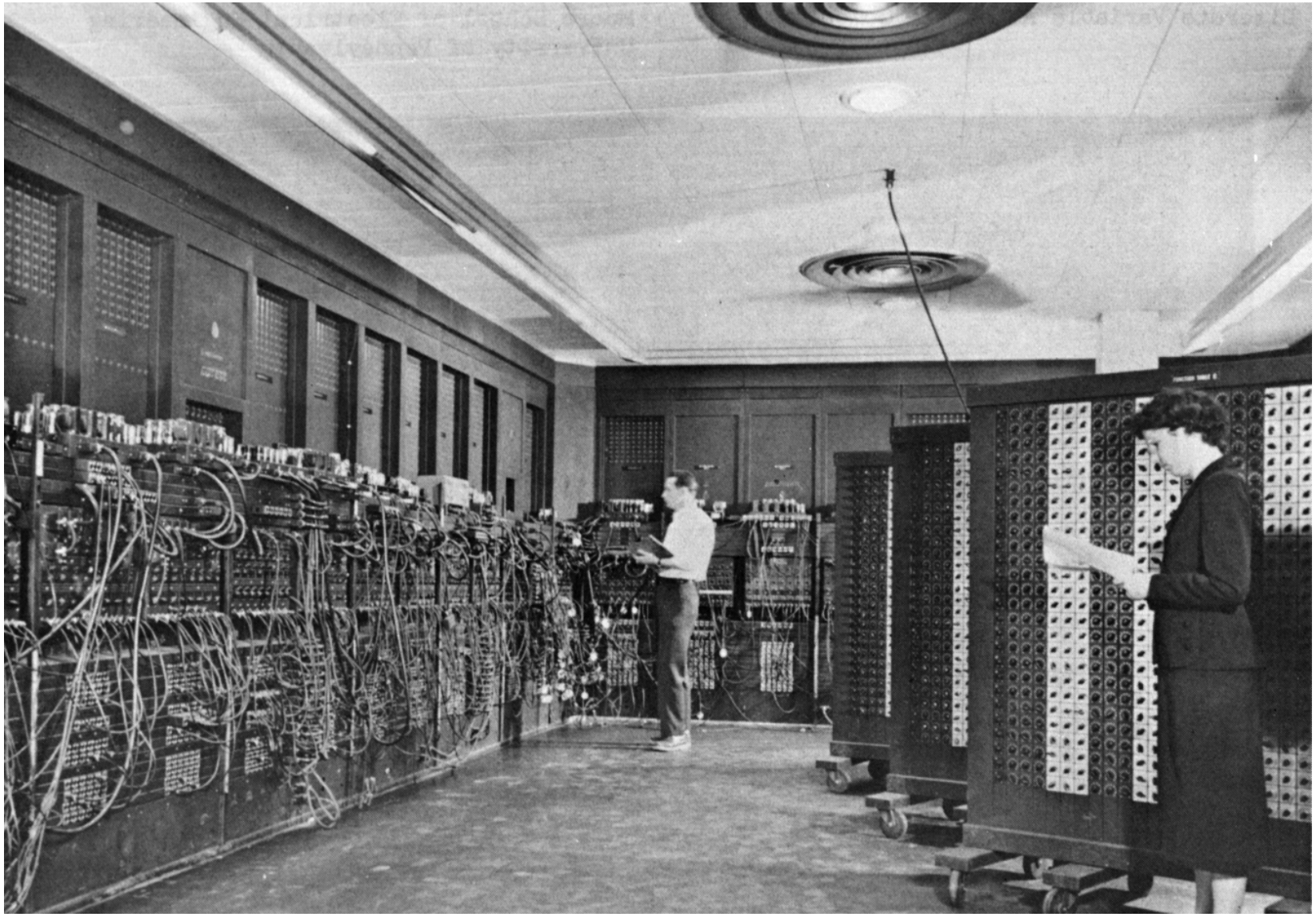
  $\rightarrow$ Betrieb „von Hand“ (Militär, wiss. Personal)

Digitale Rechenmaschinen der 1. Generation

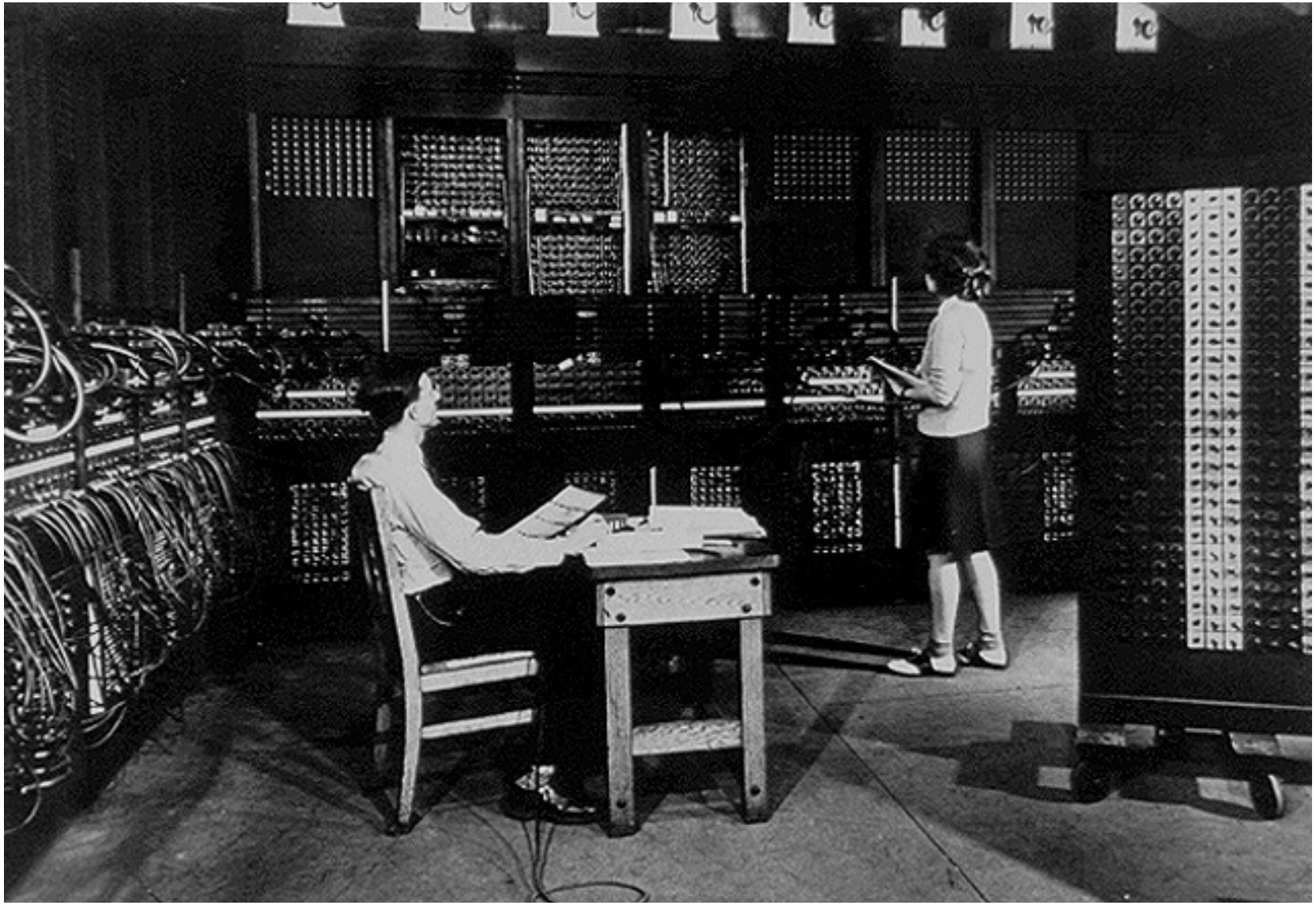
ab 1942: Eckert und Mauchly: ENIAC

- steht für *Electronic Numerical Integrator and Calculator*
- verwendete Röhren als Schaltelemente
- 17.468 Elektronenröhren, 7.200 Dioden, 1.500 Relais, 70.000 Widerstände, 10.000 Kondensatoren
- Addition, Subtraktion, Multiplikation, Division, Quadratwurzeln ziehen
- Programmierung erfolgt durch Verkabelung
- Operationen wurden auf Drehschaltern eingestellt

Digitale Rechenmaschinen der 1. Generation



Digitale Rechenmaschinen der 1. Generation



Digitale Rechenmaschinen der 1. Generation

- in gewissem Sinne frei programmierbar (Lochkarten, Verkabelung)
- Betrieb:
  - Zeitfenster reservieren (Aushang)
  - Lochkarten ins Lesegerät legen
  - warten und hoffen, dass keine der Röhren ausfällt
- Betriebssystem: auch hier Fehlanzeige

Funktionen der Rechner vorbestimmt (und eingeschränkt) durch Hardware-seitige Konstruktion

$\rightarrow$ kein Betriebssystem notwendig

Digitale Rechenmaschinen der 2. Generation

wesentlicher Unterschied: Einsatz von Transistoren

- ab 1955: Transistoren statt Relais und Röhren

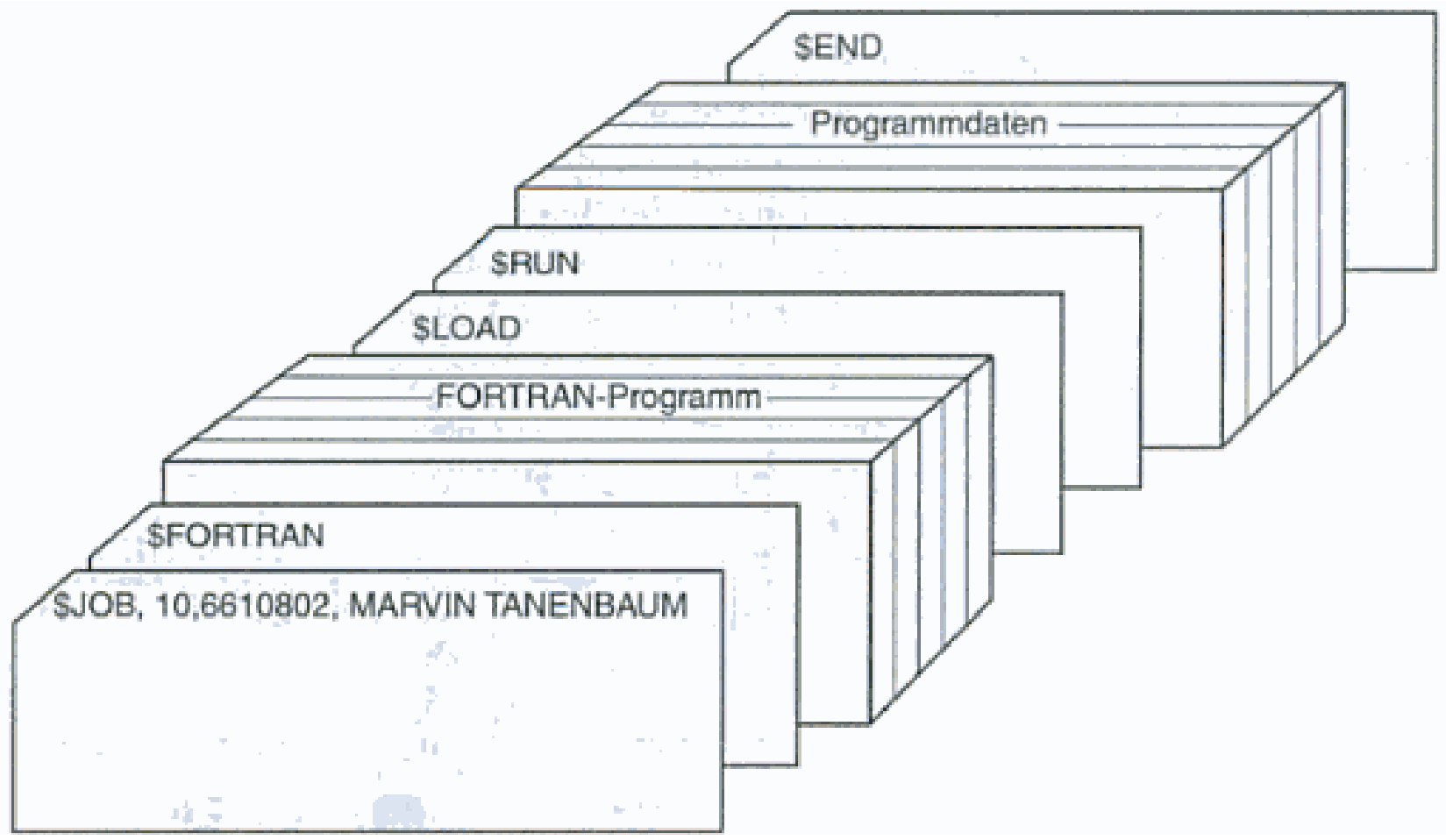
  $\rightarrow$ Rechner wurden zuverlässiger und konnten verkauft werden
- Unterscheidung: Entwickler, Hersteller, Programmierer, Wartungspersonal
- *Job*: Programm auf Lochkarten
- Operater brachte Lochkarten zum Lesegerät und nach Programmende samt Ergebnis wieder zurück

Digitale Rechenmaschinen der 2. Generation

- Problem: zeitintensive Lauferei der Operateure ist teuer
- Lösung: Stapelverarbeitung (*Batch*-Verarbeitung)
- Jobs auf Lochkarten wurden von einem kleineren Computer (IBM 1401) auf ein Magnetband gelesen
- diese Menge von Jobs wurde dann vom eigentlichen Großrechner wie der IBM 7094 verarbeitet
- Ausgabe erfolgte auf ein zweites Band, welches dann zum Ausgaberechner samt Drucker gebracht werden konnte

Digitale Rechenmaschinen der 2. Generation

- Quasi-Betriebssystem: *FORTRAN Monitoring System* (FMS)
- Struktur eines Jobs:



Digitale Rechenmaschinen der 3. Generation

- Neuerung: integrierte Schaltkreise statt Transistoren

  $\rightarrow$ Geräte wurden kleiner, sparsamer und billiger

- zwei (inkompatible) Klassen von Geräten
  - wissenschaftliche Rechner: numerische Berechnungen
  - kommerzielle Rechner: Sortieren und Drucken
- Ziel: Kosten senken

  $\rightarrow$ ein Gerät für beides: IBM System/360-Geräteserie

- Software-kompatibel, unterschiedliche Hardware

Digitale Rechenmaschinen der 3. Generation

- Problem: unterschiedliche Hardware, aber *ein* „Betriebssystem“?
- OS/360:
  - mehrere Millionen Zeilen Assembler-Code
  - mehr als 1000 Entwickler
  - besaß wahrscheinlich eine konstante Anzahl an Fehlern
- neue Konzepte: Multiprogramming, Spooling, Timesharing

Multiprogramming

- Problem: Langsame I/O-Geräte
  - CPU muss auf Abschluss von I/O-Operationen warten
  - rechenintensive Berechnungen im wissenschaftlichen Bereich waren nicht I/O-intensiv $\rightarrow$ kein Problem
  - kommerzielle Datenverarbeitung hingegen bestand zu 80 bis 90 Prozent aus I/O-Wartezeit $\rightarrow$ verursachte hohe Kosten
- Lösung: Aufteilung des Speichers für mehrere Jobs
  - mehrere Jobs liegen gleichzeitig im Speicher
  - jeder Job bekommt einen eigenen Bereich
  - wartet der aktuelle Job auf I/O-Operationen, verarbeitet die CPU einen anderen

Spooling

- Problem: Programm-/Dateneingabe dauerte lang (Lochkarten)
- Lösung: Einlesen *mehrerer* Jobs auf Magnet-Platten
- sobald ein Job beendet war, konnte der freie Speicherbereich sofort mit einem neuen belegt werden

  $\rightarrow$ SPOOL=Simultaneous Peripheral Operation On Line
- Ausgabe erfolgte ebenfalls auf Magnetplatten

aber: im Kern noch immer klassische Stapelverarbeitung

Timesharing

- Problem: Rechner waren immer noch Stapelverarbeitungssysteme

  $\rightarrow$ ein Mehrbenutzer-System war notwendig

- Lösung: Timesharing
  - Variante des Multiprogramming: jeder Job bekommt eine *Zeitscheibe*
  - Benutzer hat direkten Zugang zum System (Terminal)
  - Eingaben der Benutzer sind im Vergleich zur CPU immer langsam

    $\rightarrow$ CPU konnte mit *mehreren* Benutzern *gleichzeitig* interagieren

Die 4. Generation

- ab 1980: der Personal Computer

  $\rightarrow$ Preisverfall: neben Universitäten konnten jetzt auch Einzelpersonen Rechner besitzen

- damals marktbeherrschende Betriebssysteme: MS-DOS und UNIX

- heute: viele verschiedene Betriebssysteme

sie nutzen jedoch alle die vorgestellten Konzepte

Betriebssystemstrukturen

Schichtenmodell:

5	Anwendungsprogramme
4	Systemprogramme
3	Betriebssystem
2	Maschinensprache
1	Hardware

Hardware und Betriebssysteme

- Kernel- und User-Mode-Betrieb
- Betriebssysteme laufen im sog. *Kernel-Mode*
  - ist eine Funktionalität der Hardware
  - der *User-Mode* unterbindet einige Befehle

  $\rightarrow$ *schützt das Betriebssystem vor Fehlern, Verfälschungen etc.*
- Kernel-Mode: „darf alles“
- User-Mode (Userland): kann nur eingeschränkt Ressourcen verwenden
- Wechsel zwischen den Modi: *Kontext-Switch*

Prozesse

sind *das* Schlüsselkonzept schlechthin

- Prozess = Programm in Ausführung
- Bestandteile u.a.:
  - Programm (in Maschinensprache)
  - Daten
  - Stack
  - Program Counter
  - Stack Pointer

Unterschied Programm-Prozess

- Programm:
  - vergleichbar mit einem Kuchenrezept
  - Programm = Arbeitsanweisung
  - Daten = Zutaten
- Prozess
  - entspricht dem tatsächlichen Backvorgang
  - Program Counter = aktueller Arbeitsschritt
- Programme sind *tot*, Prozesse *leben*

Prozessverwaltung

- Timesharing: periodisch entscheidet das Betriebssystem, einen laufenden Prozess zu stoppen
- Prozesszustand muss gespeichert werden: *Prozesstabelle*

  $\rightarrow$ suspendierter Prozess besteht aus Eintrag in Prozesstabelle und Speicherabbild
- Prozess kann selbst Prozesse erzeugen

  $\rightarrow$ Kindprozesse bilden einen *Prozess-Baum*
- Kommunikation erfolgt über *Signale*

Prozess-Baum

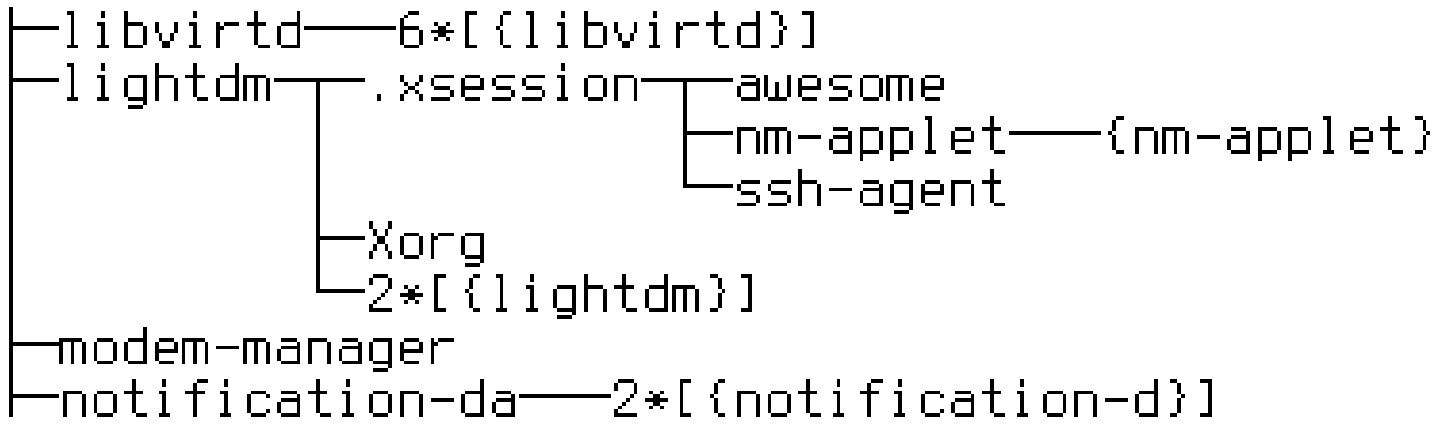


Abbildung 6: Prozessbaum

Prozesszustände

- Prozess kann also verschiedene *Zustände* haben
- running: Prozessor ist dem Prozess zugeteilt (rechnet)
- ready: Prozess ist ausführbar, aber Prozessor ist anderem Prozess zugeteilt
- waiting: Prozess wartet auf externes Ereignis (z.B. auf Daten)
- sleeping: Prozess ist suspendiert

Prozesszustände

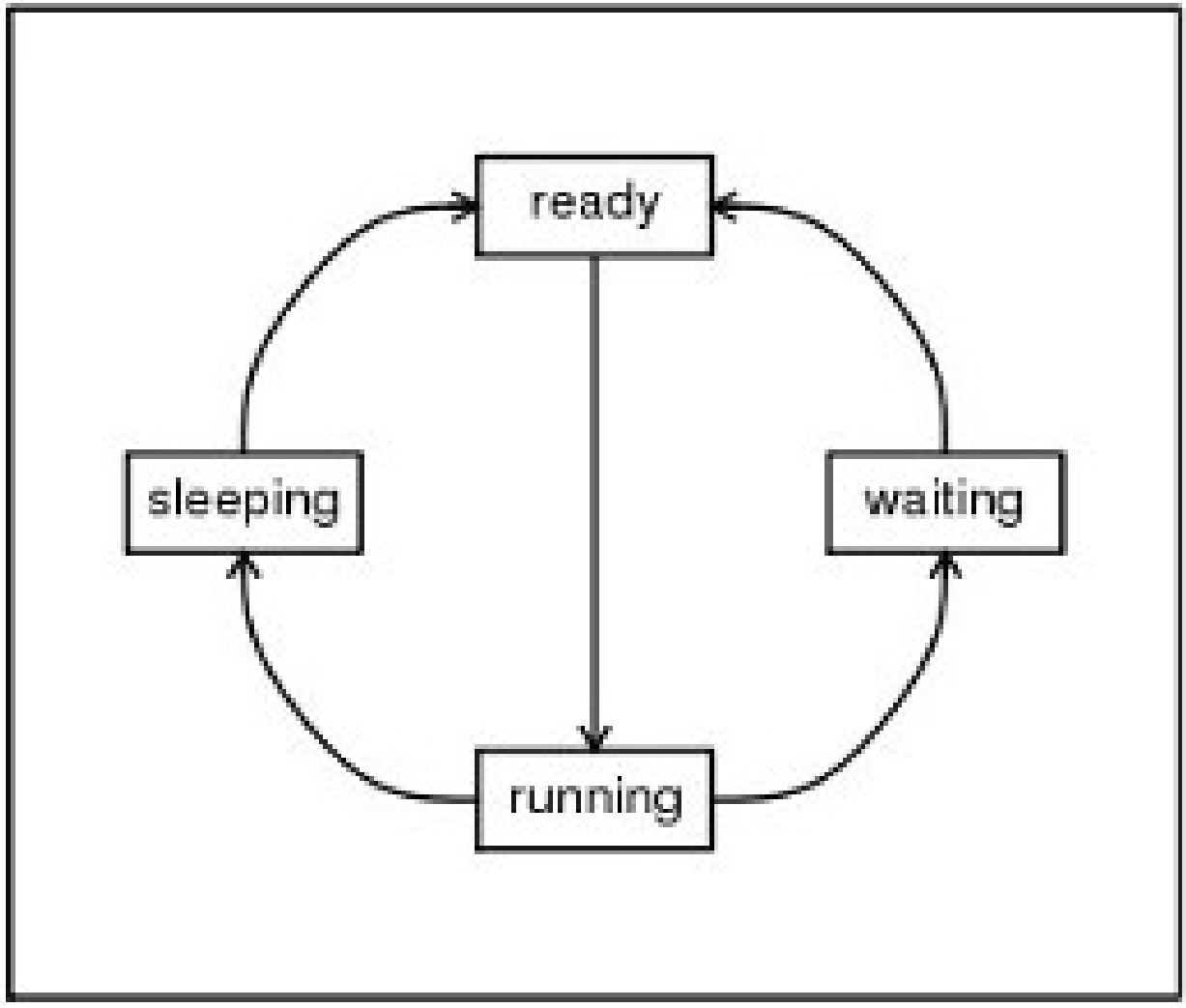


Abbildung 7: Prozesszustände

Prozess-Scheduler

- entscheidet, welcher Prozess rechnen darf
- einfachster Scheduler (Stapelverarbeitung): immer den nächsten Job vom Band
- Kriterien:
  - Fairness: gerechte Verteilung der verfügbaren Rechenzeit
  - Effizienz: vollständige Prozessorauslastung
  - Antwortzeit: Minimierung der Antwortzeit für arbeitende Benutzer
  - Verweilzeit: Wartezeit auf Ausgaben von Stapelaufträgen minimieren
  - Durchsatz: Anzahl von Aufträgen in einem Zeitintervall maximieren

Beispiel-Scheduler

Round-Robin-Scheduler

- einfach zu implementieren
- jeder Prozess bekommt ein festes Zeitintervall (Quantum) zugeordnet
- ist das Quantum verbraucht, wird der Prozess suspendiert und der nächste wird aktiviert (Kontextwechsel)
- wartet ein Prozess während seines Zeitintervalls auf I/O, wird sofort ein Kontextwechsel durchgeführt

Beispiel-Scheduler

Prioritäts-Scheduling

- Prozesse sind unterschiedlich wichtig
- jeder Prozess erhält eine Priorität
- ausgeführt wird derjenige rechenbereite Prozess, der die höchste Priorität hat
- bei jedem Kontextwechsel wird die Priorität des Prozesses dekrementiert

  $\rightarrow$ verhindert, dass ein Prozess zu lange läuft
- Prioritäten können statisch oder dynamisch zugewiesen werden

Anforderung an ein Mehrbenutzer-System

- jeder Prozess hat einen „Besitzer“
- Prozesse und Speicherbereiche müssen geschützt werden
- Prozess erhält eine UID (User ID) und eine GID (Group ID)
- dieser Ansatz ist ein Kernkonzept von Unix-Systemen

  $\rightarrow$ normale User können das System nicht gefährden

Prozesskommunikation (IPC)

- via Speicher/Datei:
  - Prozess A schreibt in einen gemeinsamen Speicher/Datei/Verzeichnis
  - Prozess B sieht periodisch dort nach, ob etwas geändert wurde

  $\rightarrow$ Problem: Was passiert, wenn A und B gleichzeitig schreiben und lesen wollen?
- Lösungen: Locking, Peterson-Algorithmus, Semaphore

Prozesskommunikation (IPC)

- über Signale:
  - Prozess kann ein Signal an einen anderen Prozess senden
  - Prozesse können beim Betriebssystem *Signal Handler* anmelden, die auf bestimmte Signale reagieren
- Beispiel-Signal: SIGHUP=Auflegen (Terminal schließen)
- über *Pipes*:
  - Pipe verbindet bspw. Ausgabekanal eines Prozesses mit dem Eingabekanal eines anderen
  - Synchronisation ist möglich, so dass bei leerer Pipe ein Prozess blockiert werden kann (verbraucht dann keine Rechenzeit)

Speicherverwaltung

- Speicherverwalter ist Teil des Betriebssystems
- Aufgaben:
  - verteilt freien Speicher
  - gibt belegten Speicher frei, wenn ein Prozess beendet ist

Was passiert, wenn das Programm größer ist als der Hauptspeicher?

Speicherverwaltung

- virtueller Speicher
  - Idee: Programmgröße darf die Größe des verfügbaren Hauptspeichers überschreiten
  - Umsetzung: Adressraum in *Seiten* einteilen
  - Seiten können ausgelagert und wieder geladen werden (*Swapping*)
  - Betriebssystem & MMU (Memory Management Unit) der CPU rechnen Speicheradressen um

Geräteverwaltung

- Betriebssystem stellt einen uniformen Zugriff auf Geräte bereit
- Unix-Ansatz: *Alles ist eine Datei*

  $\rightarrow$ Konsole ist als eines „Datei“ verfügbar
- Geräte sind spezielle Dateien im Dateibaum
- Gerätetreiber stellen Zugang her

Datenverwaltung

- Daten werden in Form von *Dateien* vorgehalten
- Dateien gruppieren: Verzeichnisse
- hierarchisch organisierter Dateibaum, beginnend beim *Wurzelverzeichnis*
- Mehrbenutzer-Systeme: Zugriffsschutz

  $\rightarrow$ Mode Bits: rwx, ugo

Interaktion

Wie treten wir nun in Kontakt mit dem Betriebssystem?

- Programm starten
- Datei erstellen
- Eingabegeräte benutzen
- Programmierung: Speicher allozieren, Geräte abfragen

Ende

Quellen:

- Tanenbaum: Moderne Betriebssysteme, 2. Auflg.
- Wikipedia