

Prozessverwaltung: Prozesse

- ★ Das Prozess-Konzept
- ★ Prozess-Scheduling
- ★ Operationen mit Prozessen
- ★ Kooperierende Prozesse
- ★ Interprozess-Kommunikation
- ★ Kommunikation in Client-Server-Systemen

Das Prozess-Konzept

★ Ein Betriebssystem führt unterschiedliche Arten von Programmen aus:

- ◆ Stabelverarbeitungssysteme – jobs
- ◆ Time-Sharing-Systeme – user programs or tasks

★ Die Begriffe „Prozess“, „Job“ und „Task“ werden häufig synonym verwendet.

★ Ein Prozess ist ein Programm, das gerade ausgeführt wird. Die Ausführung eines Prozesses geschieht sequentiell.

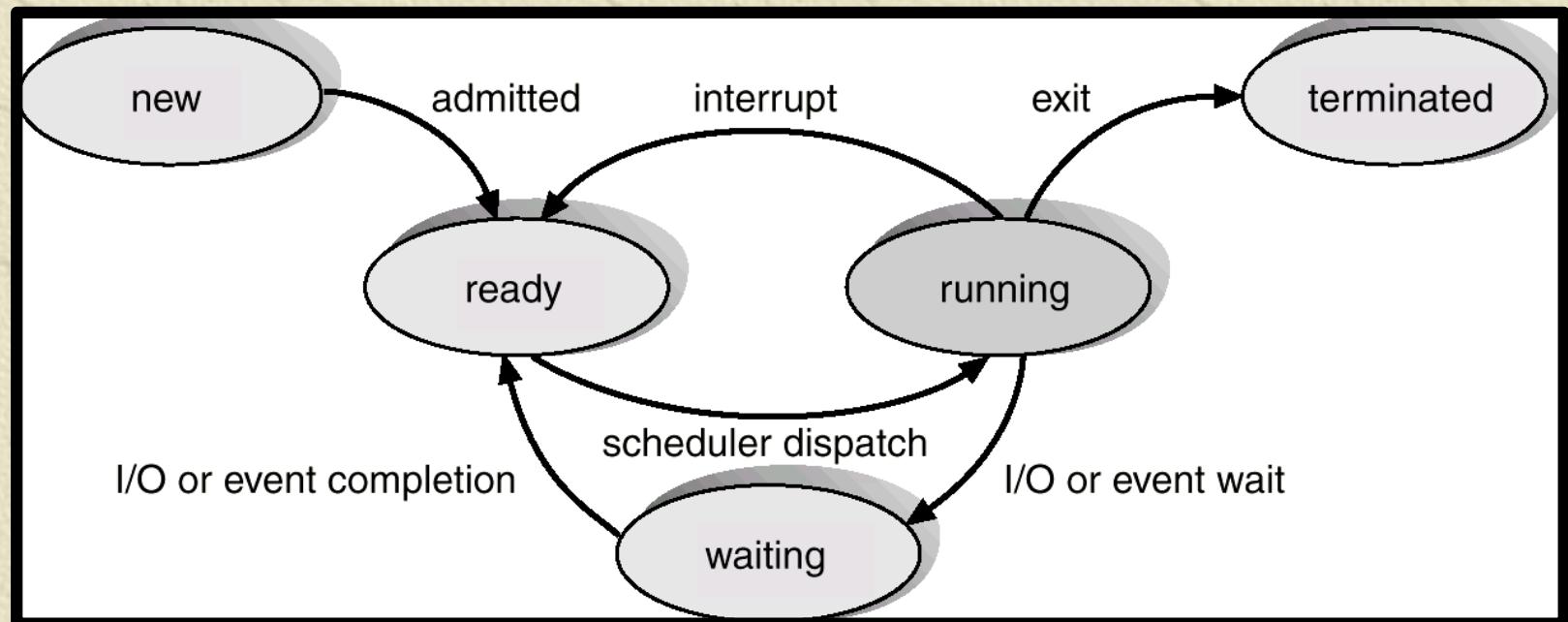
★ Ein Prozess beinhaltet:

- ◆ den Programmzähler
- ◆ den Stack (Kellerspeicher)
- ◆ den Datenbereich (data section)

Zustände eines Prozesses

- ★ Bei der Ausführung eines Prozesses durchläuft er verschiedene Zustände:
 - ◆ **neu (new)**: Der Prozess wird erzeugt.
 - ◆ **laufend (running)**: Prozessanweisungen werden ausgeführt.
 - ◆ **wartend (waiting)**: Der Prozess wartet auf das Eintreten eines Ereignisses.
 - ◆ **bereit (ready)**: Der Prozess ist bereit, ausgeführt zu werden.
 - ◆ **beendet (terminated)**: Der Prozess ist vollständig beendet.

Diagramm der Prozesszustände

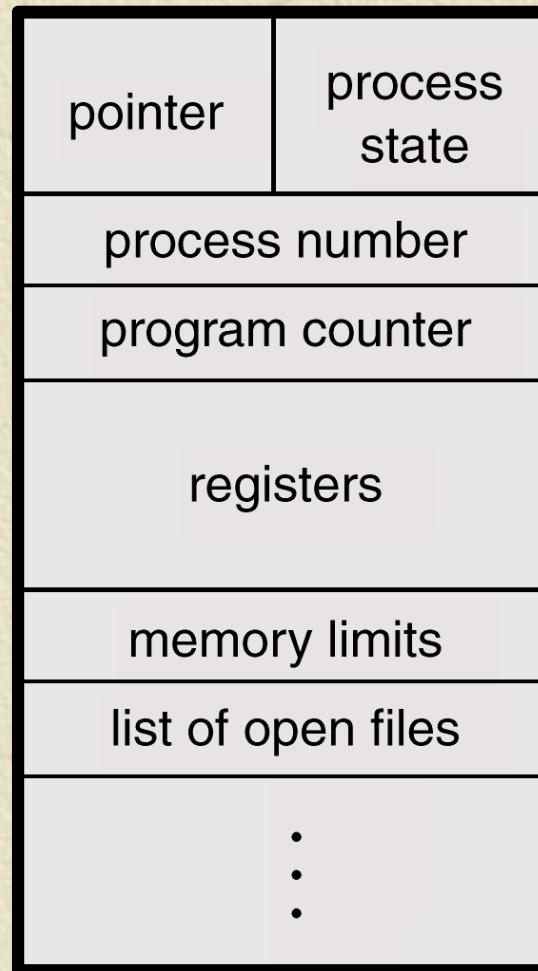


Process-Control-Block (I)

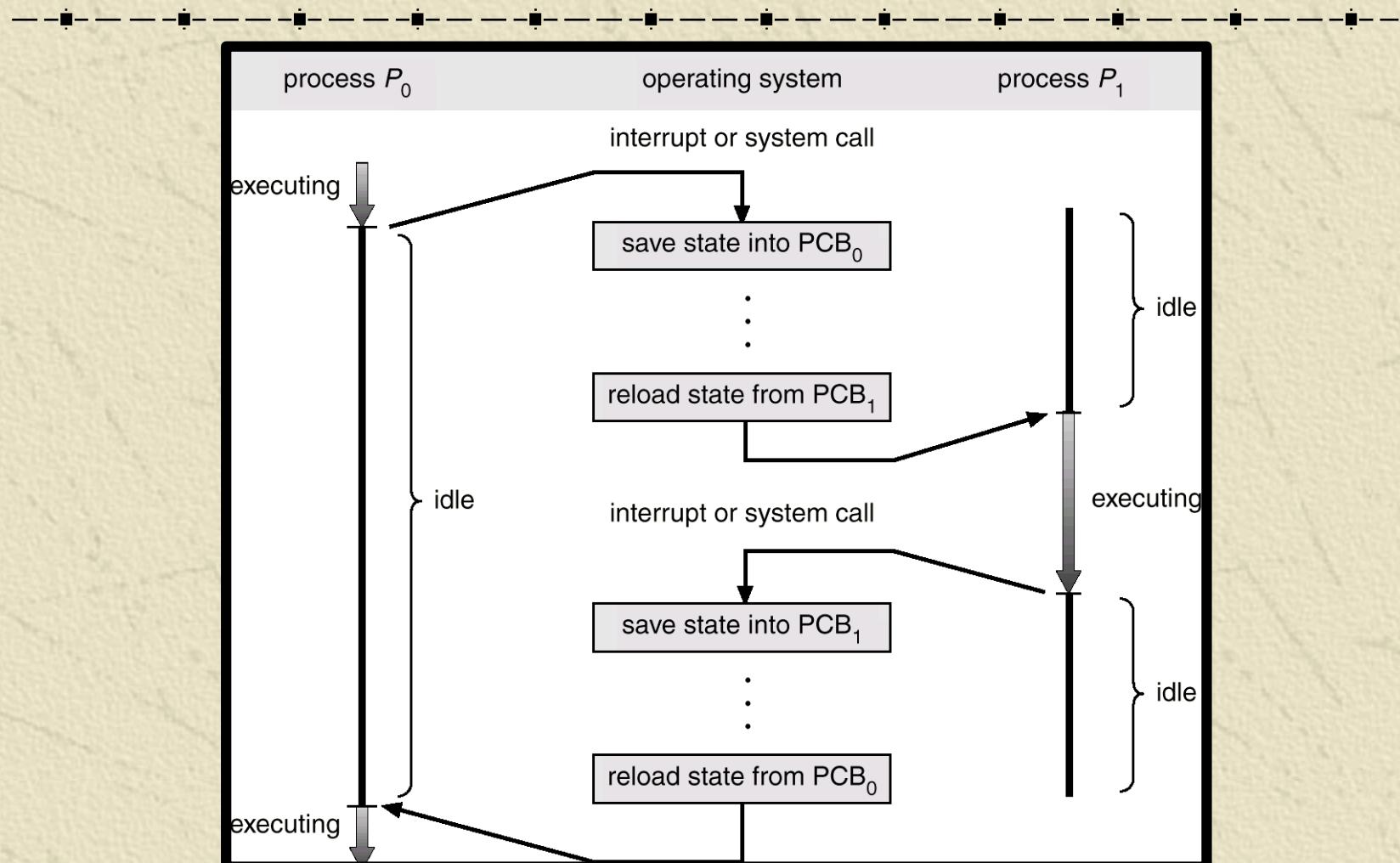
Der Process-Control-Block enthält zum Prozess gehörige Informationen:

- ◆ Prozesszustand
- ◆ Befehlszähler (program counter)
- ◆ Prozessorregister
- ◆ Scheduling-Informationen (z.B. Zeitscheibe)
- ◆ Speicherverwaltungsinformationen
- ◆ Buchhaltungs- und Verwaltungsinformationen (z.B. Prozess-ID, CPU-Nutzung)
- ◆ E/A-Statusinformationen

Process-Control-Block (II)



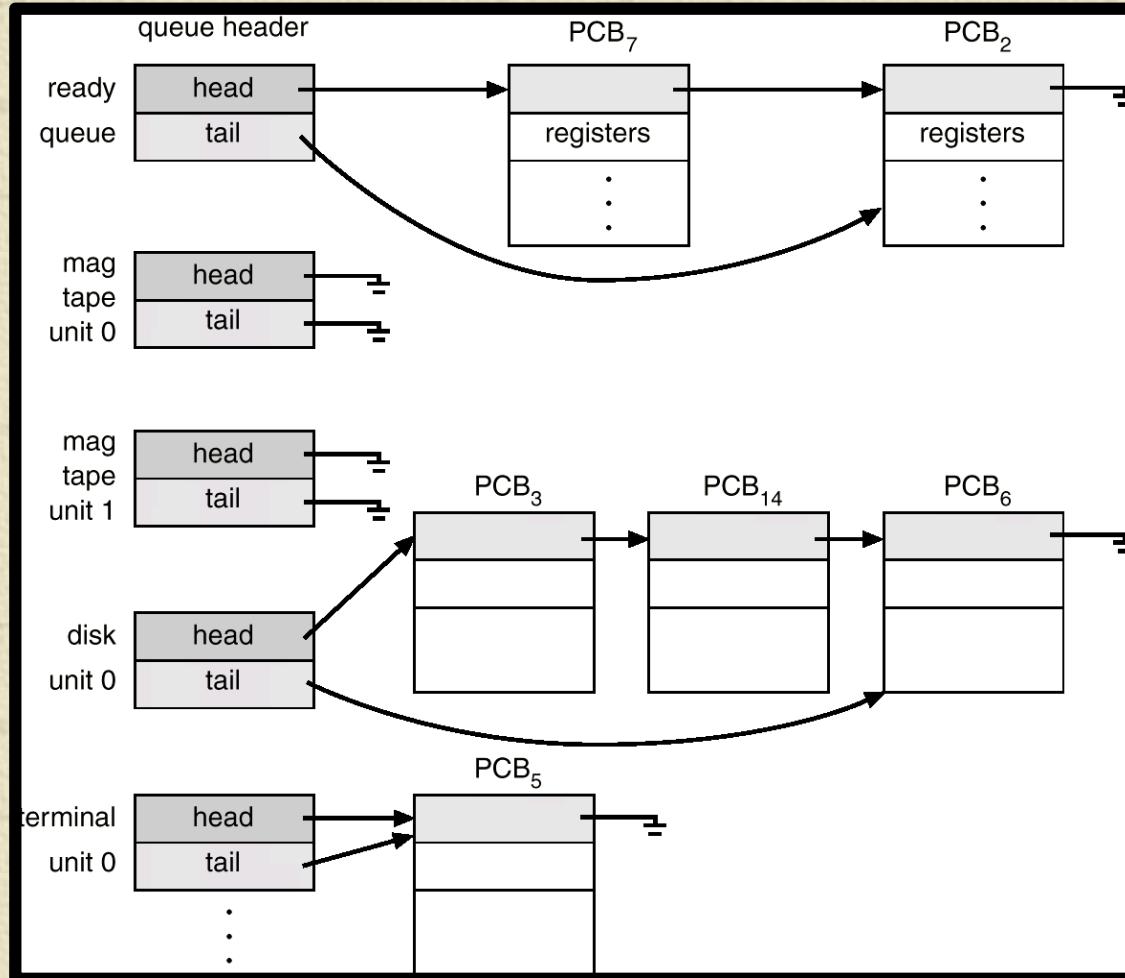
Prozesswechsel



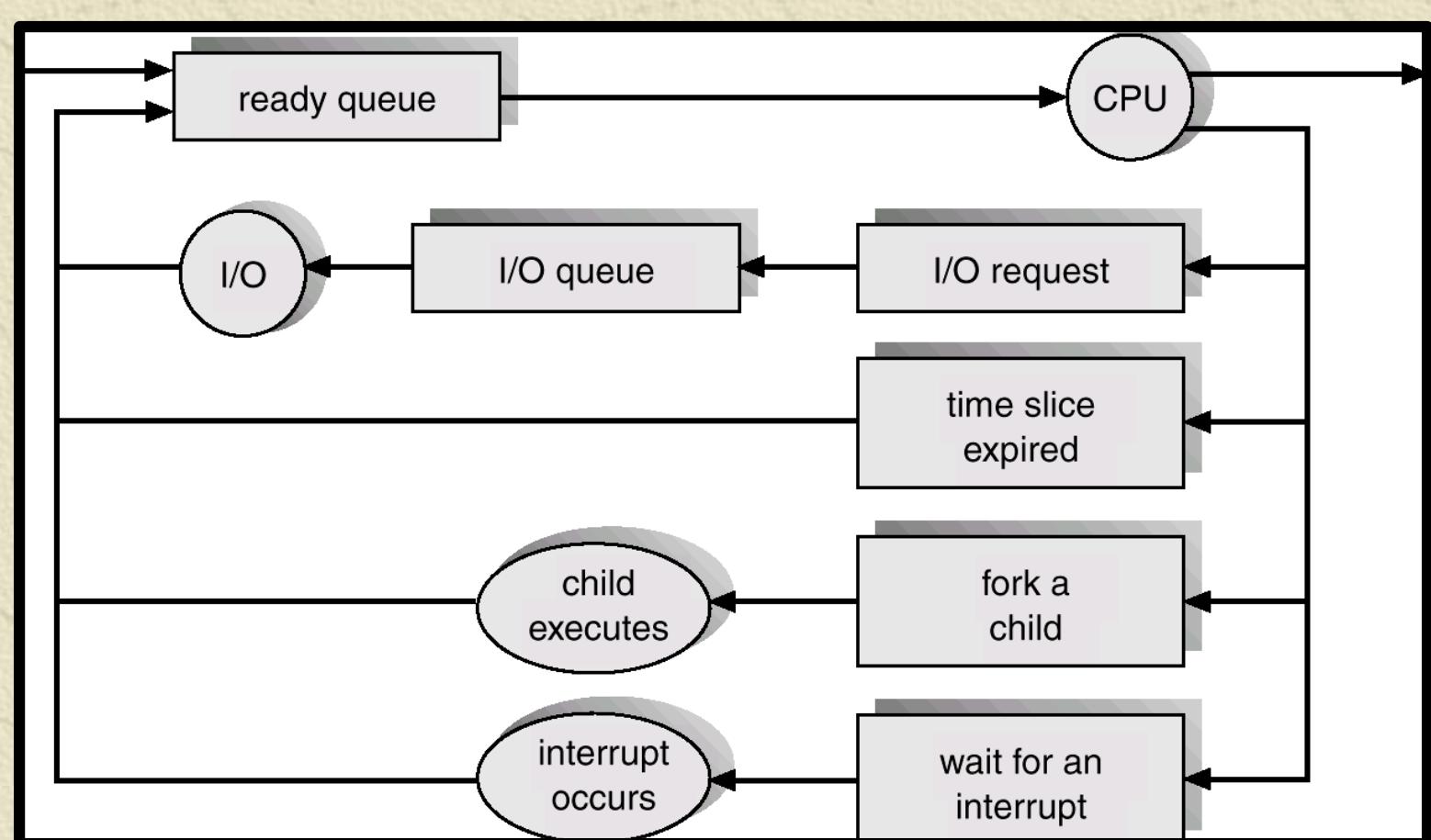
Prozesswarteschlangen

- ❖ Prozessliste – Liste aller Prozesse im System
- ❖ Bereit-Warteschlange – Liste aller Prozesse, die sich im Hauptspeicher befinden und ausführbereit sind
- ❖ Geräte-Warteschlangen – Liste der Prozesse, die auf ein E/A-Gerät warten
- ❖ Prozesse wandern zwischen den Warteschlangen hin und her.

Bereit- und E/A-Geräte-Warteschlangen



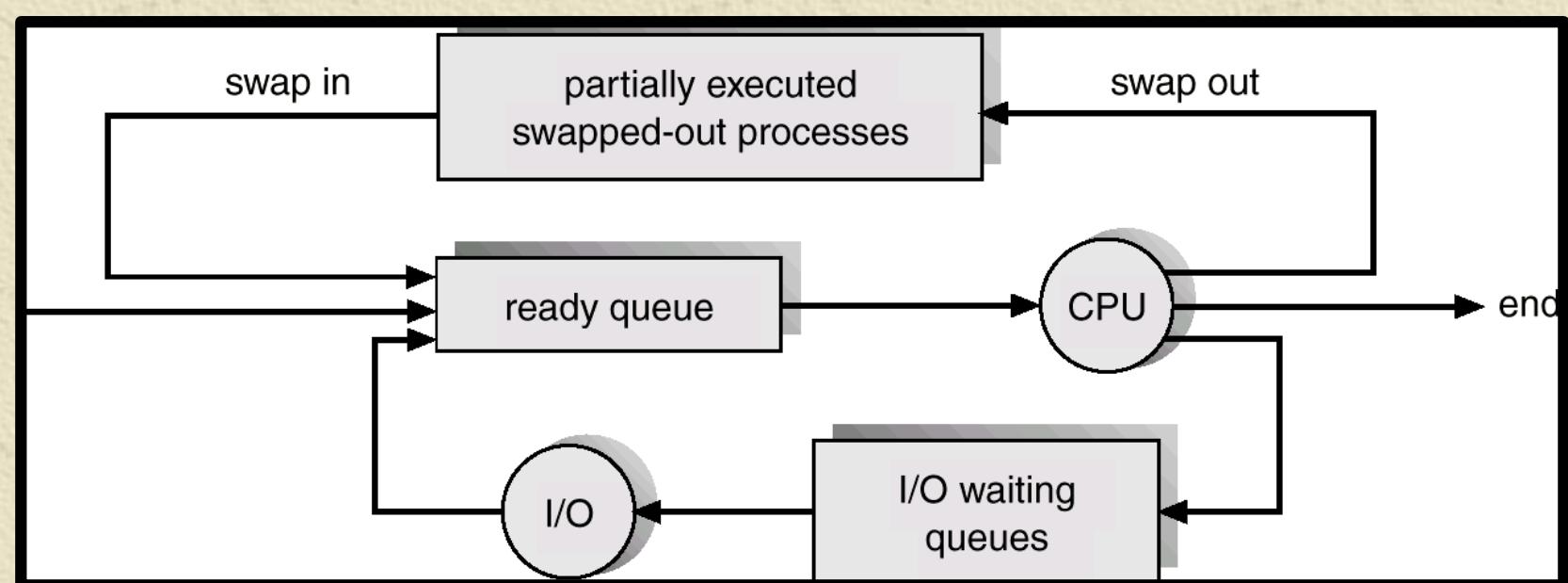
Darstellung des Prozess-Schedulings



Scheduler-Typen (I)

- ✿ Kurzzeit-Scheduler (auch Prozessor-Scheduler) – wählt aus, welcher bereite Prozess als nächstes den Prozessor erhält.
- ✿ Langzeit-Scheduler (auch Prozess- oder Job-Scheduler) – wählt aus, welche neuen Prozesse in die Bereit-Warteschlange aufgenommen werden soll.

Mittelfristiger Scheduler



Scheduler-Typen (II)

- ❖ Der Kurzzeit-Scheduler wird sehr häufig aufgerufen (im Millisekundenbereich) ⇒ muss sehr schnell sein
- ❖ Der Langzeit-Scheduler wird selten aufgerufen (im Sekunden- oder Minutenbereich) ⇒ kann langsam sein
- ❖ Der Langzeit-Scheduler kontrolliert den *Grad der Multiprogrammierung*.
- ❖ Ein Prozess kann beschrieben werden als:
 - ◆ E/A-bezogen – Der Prozess verbringt mehr Zeit mit Ein- und Ausgaben als mit Berechnungen; die CPU wird häufig, aber nur kurz benötigt.
 - ◆ CPU-bezogen – Der Prozess verbringt mehr Zeit mit Berechnungen als mit Ein- und Ausgaben; die CPU wird weniger oft, dafür aber für lange Perioden benötigt.

Kontext-Wechsel

- ❖ Wenn ein anderer Prozess den Prozessor erhält, dann muss der Zustand des alten Prozesses gespeichert und der gespeicherte Zustand des neuen Prozesses geladen werden (Wechsel des Prozess-Kontexts).
- ❖ Der Kontext-Wechsel ist Verwaltungsarbeit, während der das System keine anwendungsbezogene Arbeit leistet.
- ❖ Durch Hardware-Unterstützung kann die Zeit für den Kontext-Wechsel reduziert werden.

Prozess-Erzeugung (I)

- ❖ Elternprozesse erzeugen Kinderprozesse, die selbst ebenfalls weitere Prozesse erzeugen können. So entsteht ein Baum von Prozessen.
- ❖ Die gemeinsame Nutzung von Betriebsmitteln ist möglich:
 - ◆ Eltern- und Kindprozesse nutzen die Betriebsmittel gemeinsam.
 - ◆ Kindprozesse nutzen einen Teil der Betriebsmittel der Elternprozesse.
 - ◆ Eltern- und Kindprozesse nutzen keine gemeinsamen Betriebsmittel.
- ❖ Ausführung
 - ◆ Eltern- und Kindprozesse werden gleichzeitig ausgeführt.
 - ◆ Der Elternprozess wartet auf die Beendigung des Kindprozesses.

Prozess-Erzeugung (II)

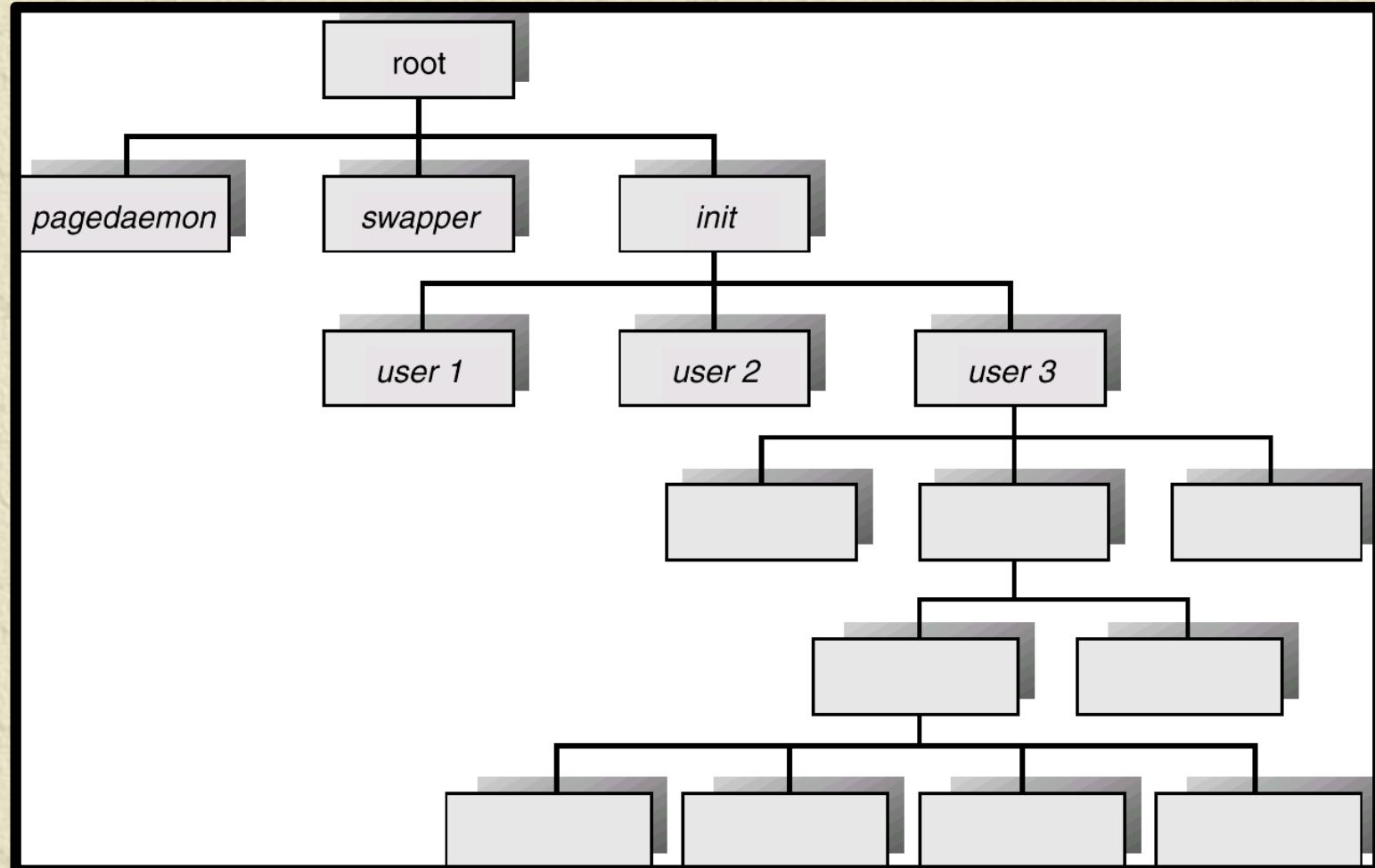
❖ Nutzung des Speicherbereichs

- ◆ Der Kindprozess ist ein Duplikat des Elternprozesses.
- ◆ Der Kindprozess lädt ein neues Programm in seinen Speicherbereich.

❖ Beispiel UNIX:

- ◆ **fork**: Systemaufruf erzeugt einen neuen Prozess (Kopie des Elternprozesses)
- ◆ **exec**: Systemaufruf, der nach **fork** verwendet wird, um in den Speicherbereich des neuen Prozesses ein neues Programm zu laden.

Prozessbaum in UNIX-Systemen



Beendigung von Prozessen



Der Prozess führt seine letzte Anweisung aus und bittet das Betriebssystem um Beendigung (**exit**)

- ◆ Der Elternprozess wird informiert, falls er auf die Beendigung des Kindprozesses wartet (via **wait**).
- ◆ Prozess-Ressourcen werden vom Betriebssystem freigegeben.



Elternprozesse können Kindprozesse beenden (durch **abort**).

- ◆ Der Kindprozess hat die verfügbaren Ressourcen überschritten.
- ◆ Die Aufgabe des Kindprozesses wird nicht länger benötigt.
- ◆ Der Elternprozess wird beendet:
 - Das Betriebssystem erlaubt dem Kindprozess nicht weiterzuarbeiten, wenn der Elternprozess beendet wird.
 - Kaskadierende Beendigung der Prozesse

Kooperierende Prozesse

❖ *Unabhängige* Prozesse können sich gegenseitig nicht beeinflussen.

❖ *Kooperierende* Prozesse können andere kooperierende Prozesse beeinflussen und von ihnen beeinflusst werden.

❖ Vorteile der Prozess-Kooperation

- ◆ Austausch von Informationen
- ◆ Schnellere Berechnungen
- ◆ Modularität
- ◆ Bequemlichkeit

Produzenten-Konsumenten- Problem

✿ Musterbeispiel für kooperierende Prozesse:
Der Produzenten-Prozess erzeugt
Informationen, die vom Konsumenten-
Prozess verbraucht wird.

- ◆ *unbegrenzter Puffer*: Puffer variabler Größe ohne vorgegebene Maximalgröße.
- ◆ *begrenzter Puffer*: Puffer mit fest vorgegebener Größe.

Begrenzter Puffer – Lösung mit gemeinsam genutzten Speicher

❖ Gemeinsam genutzte Daten

```
#define PUFFER_GROESSE 10

typedef struct {
    ...
} artikel;

item puffer[PUFFER_GROESSE];
int rein = 0;
int raus = 0;
```

Begrenzter Puffer – Produzenten-Prozess

```
artikel neuProduziert;

while (1) {
    while (((rein + 1) % PUFFER_GROESSE) == raus)
        ; /* tue nichts */
    puffer[rein] = neuProduziert;
    rein = (rein + 1) % PUFFER_GROESSE;
}
```

Begrenzter Puffer – Konsumenten-Prozess

```
artikel neuVerbraucht;

while (1) {
    while (rein == raus)
        ; /* tue nichts */
    neuVerbraucht = puffer[raus];
    raus = (raus + 1) % PUFFER_GROESSE;
}
```

Interprozess-Kommunikation (IPC: Inter-Process-Communication)

- ❖ Interprozess-Kommunikation: Prozesse kommunizieren und synchronisieren ihre Handlungen.
- ❖ IPC über Nachrichtensysteme – Prozesse kommunizieren über Nachrichten (kein gemeinsamer Speicher notwendig).
- ❖ Zwei zentrale Operationen:
 - ◆ **send(*Nachricht*)**
 - ◆ **receive(*Nachricht*)**
- ❖ Genereller Ablauf der Kommunikation:
 - ◆ Aufbau einer Kommunikationsverbindung
 - ◆ Nachrichtenaustausch mittels send und receive
 - ◆ Abbau der Verbindung

Direkte Kommunikation

❖ Kommunikationspartner müssen explizit benannt werden:

- ◆ **send** (P , *Nachricht*) – sende Nachricht an Prozess P
- ◆ **receive**(Q , *Nachricht*) – empfange Nachricht von Prozess Q

❖ Kommunikationsverbindungen

- ◆ werden automatisch aufgebaut,
- ◆ sind genau einem Paar von Prozessen zugeordnet,
- ◆ sind meist bidirektional, seltener unidirektional.

Indirekte Kommunikation (I)

❖ Nachrichten werden an ein Postfach (*mailbox*, *port*) geschickt und von dort abgeholt.

- ◆ Jedes Postfach hat eine eindeutige ID.
- ◆ Prozesse können nur kommunizieren, wenn sie eine gemeinsames Postfach verwenden.

❖ Eigenschaften

- ◆ Ein Postfach kann von mehreren Prozessen genutzt werden.
- ◆ Ein Prozess kann mehrere Postfächer verwenden.
- ◆ Die Kommunikation kann uni- oder bidirektional erfolgen.

Indirekte Kommunikation (II)

❖ Operationen

- ◆ Neues Postfach erzeugen
- ◆ Nachricht an Postfach schicken
`send(A, Nachricht)` – sende Nachricht an Postfach A
- ◆ Nachricht aus Postfach holen
`receive(A, Nachricht)` – hole Nachricht aus
- ◆ Postfach löschen

Synchronisation

- ❖ Der Austausch von Nachrichten kann synchron (Prozesse blockieren) oder asynchron (Prozesse arbeiten weiter) geschehen.
 - ◆ blocking send: warten, bis die Nachricht empfangen wurde
 - ◆ nonblocking send: Nachricht senden und weiterarbeiten
 - ◆ blocking receive: warten, bis eine Nachricht eintrifft
 - ◆ nonblocking receive: empfange Nachricht, falls vorhanden

Pufferung



Kommunikationsverbindungen können mit Nachrichtenwarteschlangen versehen werden. Es existieren drei Realisierungsformen:

1. Es werden keine Nachrichten gepuffert:
Der Sender muss auf den Empfänger warten
(Rendezvous-Technik).
2. Der Puffer ist begrenzt:
Der Sender muss warten, falls der Puffer bereits voll ist.
3. Der Puffer ist (theoretisch) unbegrenzt:
Der Sender muss nie warten.

Client-Server-Kommunikation

✿ Die Kommunikation zwischen Prozessen auf Client-Rechnern und Prozessen auf Server-Rechnern kann unterschiedlich realisiert werden. Beispiele:

- ◆ Socket-Verbindungen
- ◆ Remote-Procedure-Calls (RPC)
- ◆ Remote-Method-Invocation (Java RMI)