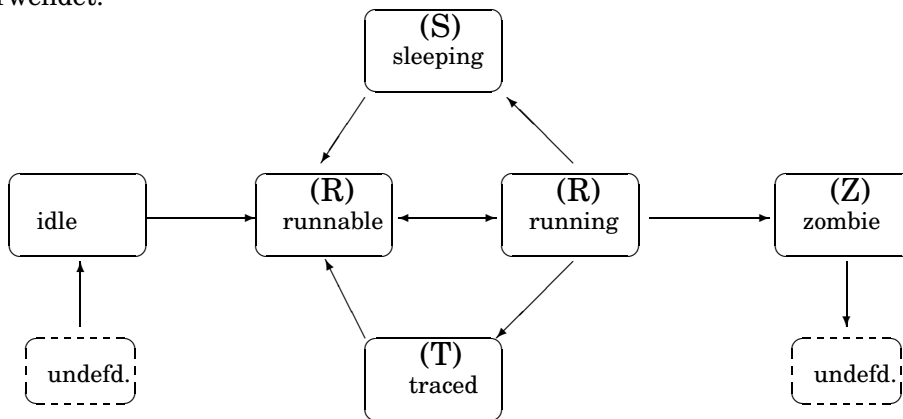


### Aufgabe 1: Prozeßzustandsmodell von UNIX

Das Bild zeigt die 6 möglichen Zustände eines Prozesses unter UNIX. Die Buchstaben in Klammern sind dabei die Codierungen, die das UNIX-Programm `ps` für die Prozesszustände verwendet.



Unmittelbar nach seiner Erzeugung durch `fork()` befindet sich ein Prozess im Zustand `idle`. Von dort gelangt er nach seiner vollständigen Initialisierung in den Zustand `runnable` und wartet - neben anderen Prozessen im gleichen Zustand - auf die Zuteilung einer/der CPU. Wenn der Scheduler des Betriebssystems einem Prozess die CPU zuteilt, dann befindet sich der Prozess im Zustand `running`. In den Zustand `sleeping` gelangt ein Prozess, indem er eine Betriebssystemressource anfordert, die nicht unmittelbar zur Verfügung steht. Sobald die angeforderte Resource zur Verfügung steht, wird der Prozess wieder in den Zustand `runnable` versetzt. UNIX-Betriebssysteme bieten zusätzlich noch die Möglichkeit, einen Prozeß in den Zustand `traced` (oder auch `stopped`) zu versetzen. Dadurch wird der Prozeß „eingefroren“ und kann z.B. von einem Debugger untersucht werden. Aus diesem Zustand kann der Prozess z.B. durch ein spezielles Signal wieder in den `runnable`-Zustand versetzt werden. Beendet sich ein Prozeß, so gelangt er zunächst in den Zustand `zombie`. Er verweilt dort so lange, bis sein Vaterprozess das Terminieren registriert hat. Wenn der Prozeß den Zustand `zombie` verläßt, verschwindet er aus dem System.

- Diskutieren Sie die Unterschiede zwischen diesem Zustandsmodell und dem vereinfachten Modell der Vorlesung.
- Auf einem Einprozessor-UNIX-Rechner gibt das Programm `top` u.a. folgendes aus: „99 processes, 96 sleeping, 3 runnable“. Würden Sie dieses System als „gesund“ oder als „krank“ einstufen und warum?
- Welche Prozeßzustände durchläuft ein Programm, welches 3 Zahlen von der Konsole einliest, aufsummiert und anschließend auf der Konsole ausgibt?

### Aufgabe 2: Verständnisfragen

- Was ist der Unterschied zwischen einem Programm und einem Prozeß?
- In einem UNIX-System gibt es 2 unterschiedliche Modi (User-Mode und Kernel-Mode) in denen Maschinencode ausgeführt wird. Der Code des Betriebssystems wird im (privilegierten) Kernel-Mode ausgeführt, Daemonen und Benutzerprogramme im User-Mode. Wieso wurde diese Trennung eingeführt? Welche Vor- und Nachteile bietet sie?

### Aufgabe 3: Scheduling-Strategien

Gegeben seien folgende *Programme (Executables)*:

$E_1$ : 4cpu, 9io, 2cpu  
 $E_2$ : 1cpu, 2io, 2cpu, 4io, 5cpu  
 $E_3$ : 4cpu, 6io, 1cpu

Die Notation soll folgendermaßen interpretiert werden (exemplarisch für  $E_1$ ): das Programm benötigt im ersten Schritt 4 Zeiteinheiten CPU-Rechenzeit, danach führt es eine E/A-Operation aus, die 14 Zeiteinheiten benötigt. Dann rechnet das Programm wieder 2 Zeiteinheiten unter Benutzung der CPU bis es schließlich terminiert.

Gehen Sie davon aus, daß der Rechner, auf dem die Programme ausgeführt werden beliebig viele E/A-Operationen parallel abarbeiten kann, jedoch lediglich über eine CPU verfügt. Die „Zeitscheibe“ für die preemptiven Scheduling-Strategien soll eine Zeiteinheit betragen.

Es werden nun folgende *Prozesse* gestartet:  $P_1$ : Programm  $E_1$  zum Zeitpunkt  $t_1 = 0$ ,  $P_2$ : Programm  $E_2$  zum Zeitpunkt  $t_2=1$ ,  $P_3$ : Programm  $E_3$  zum Zeitpunkt  $t_3=2$ .

- Bestimmen Sie die Zeitpunkte, zu denen die einzelnen Prozesse frühestens beendet sind.
- Bestimmen Sie die Zuteilung der CPU zu den Prozessen, nach der Strategie „Highest Priority First“. Ordnen Sie den Prozessen folgende Prioritäten zu:  $p_1=4$ ,  $p_2=1$ ,  $p_3=3$  (Ein kleinerer Zahlenwert bedeutet dabei größere Priorität).
- Wir verwenden „Round Robin“ als Scheduling-Strategie. Berechnen Sie, für jeden Zeitpunkt die Zuteilung der CPU. Zu welchen Zeitpunkten werden die Programme beendet? Vergleichen Sie die Zeitpunkte, zu denen die Prozesse zum ersten Mal die CPU zugeteilt bekommen mit Teil b).

### Aufgabe 4: Synchronisationsmechanismen

- Die Programmiersprache Java bietet zur Synchronisation folgende Konstrukte und Methoden:
  - mit dem zusätzlichen Attribut `synchronized` versehene Methoden, die im Methodenrumpf gegenseitigen Ausschluß realisieren.
  - die Monitor-Methoden `wait()` und `notify()`, die für jedes Java-Objekt aufgerufen werden können. Die Methode `notify()` hebt eine durch `wait()` hervorgerufene Blockierung im gleichen Objekt auf.

Wie können mit diesen Sprachkonstrukten und Methoden die Operationen P(s) und V(s) eines Semaphors nachgebildet werden?

- In UNIX kann ein Prozeß mit `signal(int sig, signal_handler)` eine Behandlungsroutine für ein Signal installieren und mit der Funktion `pause()` auf das Eintreffen eines Signals warten. Ein anderer Prozeß kann mittels `kill(int pid, int sig)` einem wartenden Prozeß `pid` eines von 32 verschiedenen Signalen zukommen lassen. Vergleichen Sie dieses Konzept mit den Synchronisations- und Kommunikationsmechanismen der Vorlesung.