Betriebssysteme

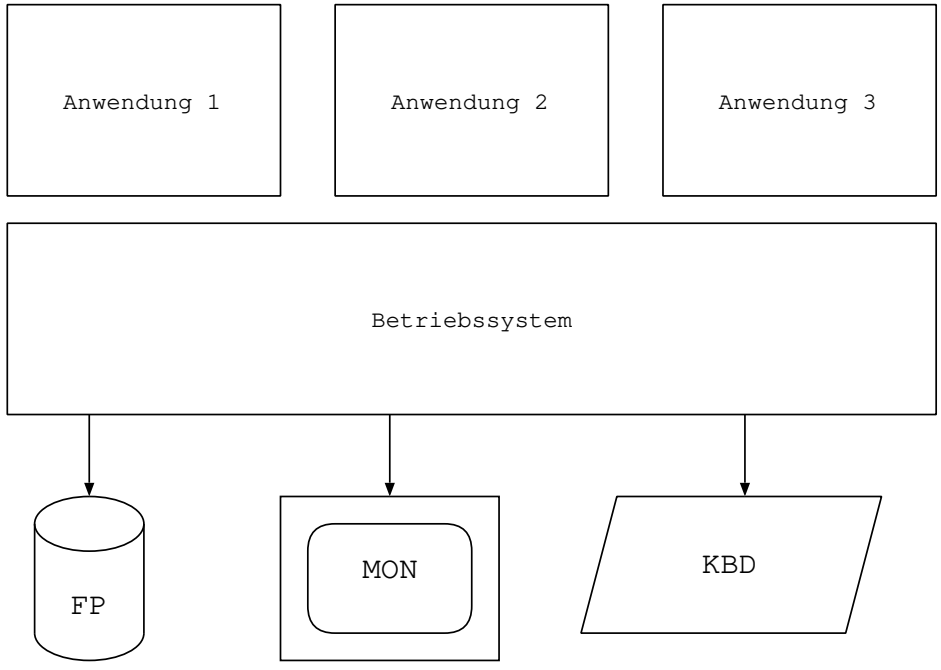
Speicherverwaltung, Prozessverwaltung

G. Richter

5. April 2017

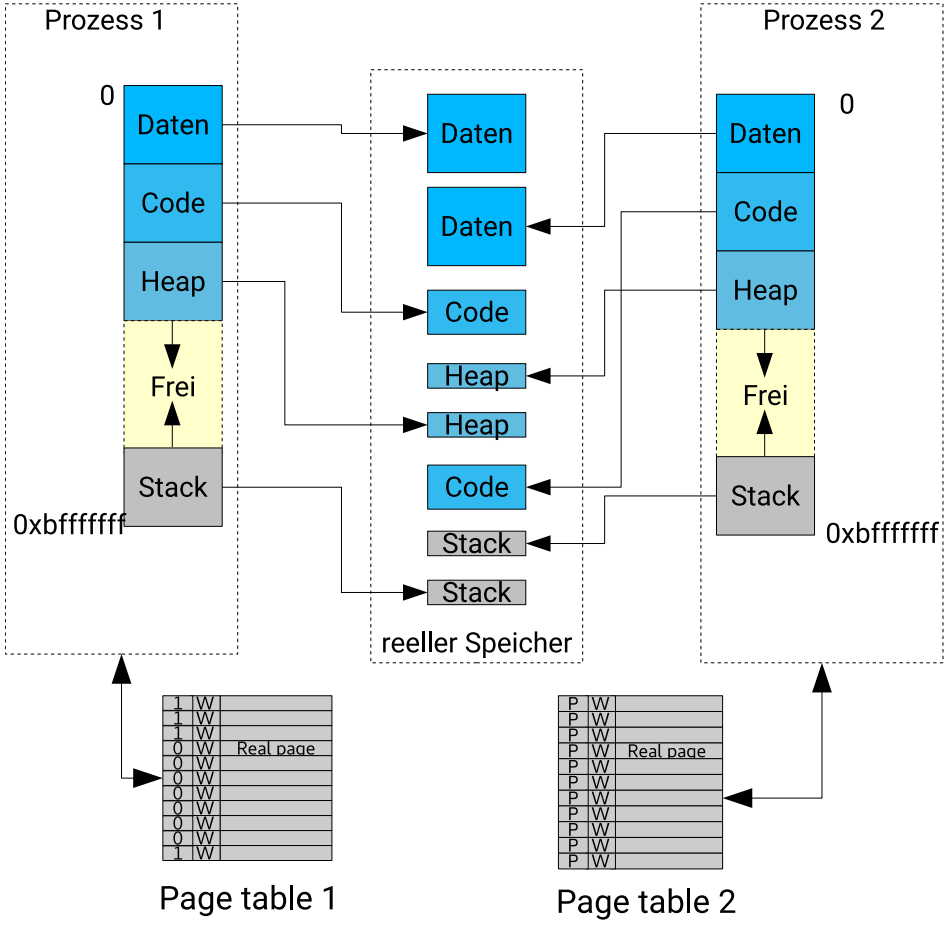
Softwarestruktur

- Anwendungen haben ihren eigenen Adressraum
- Anwendungen können auf den Adressraum anderer Anwendungen nicht zugreifen
- Zugriff von Anwendungen auf Hardware *nur* über das Betriebssystem
- Anwendung, die in Ausführung sind, nennt man Prozesse oder Task



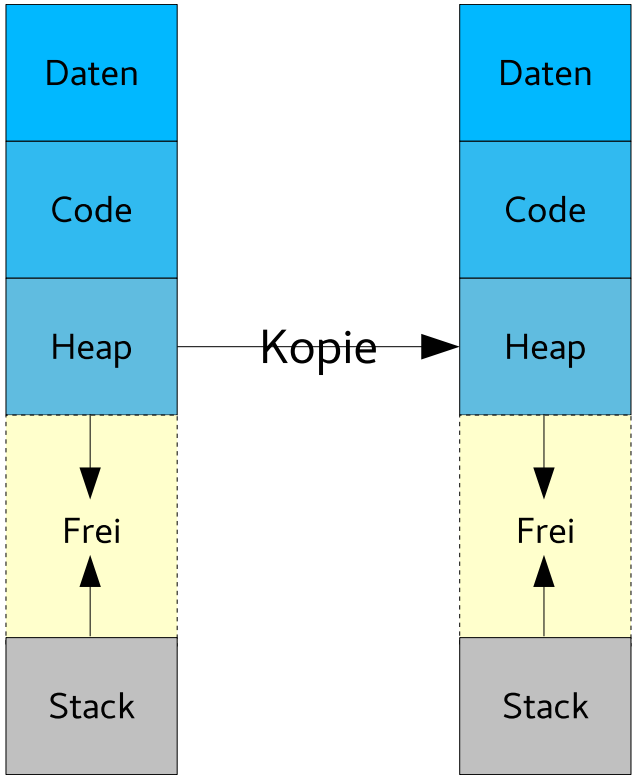
Adressräume der Anwendungen

- Jede Anwendung hat ihre eigene Pagetable
- So wird verhindert, dass auf fremde Adressräume zugegriffen wird
- Der ungenutzte virtuelle Adressraum (Frei) wird über P = 0 gesperrt



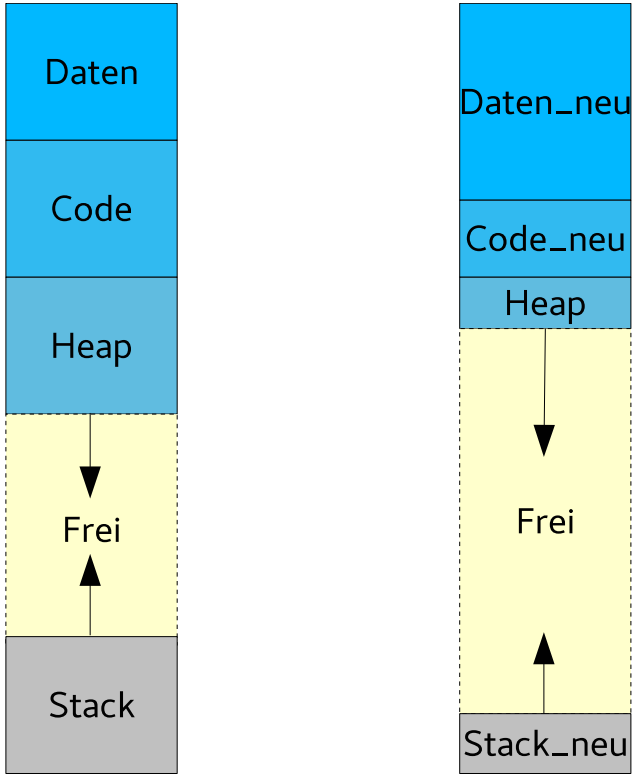
Entstehung von Prozessen

- Prozesse entstehen durch Kopie aus Elternprozessen (Systemaufruf „fork()“)



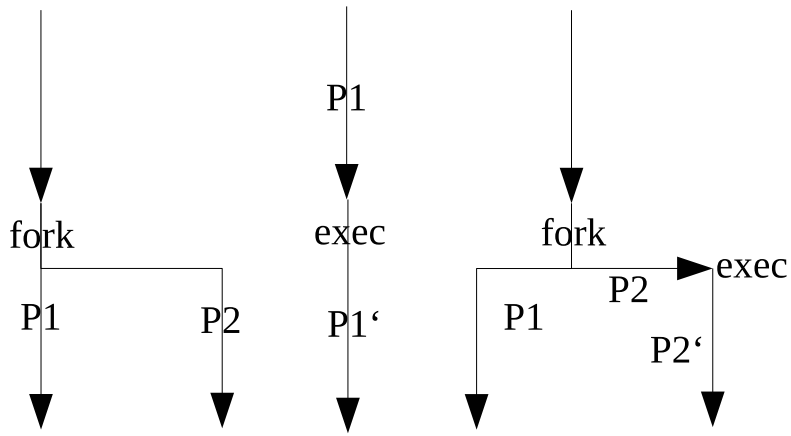
Entstehung von Prozessen

- Prozesse entstehen durch Kopie aus Elternprozessen (Systemaufruf „fork()“)
- Die Kopien können durch Überladen mit anderem Code und Daten andere Funktionen ausführen (Systemaufruf „exec()“)
- Der erzeugte Prozess ist das KInd des Prozesses, der „fork()“ aufgerufen hat



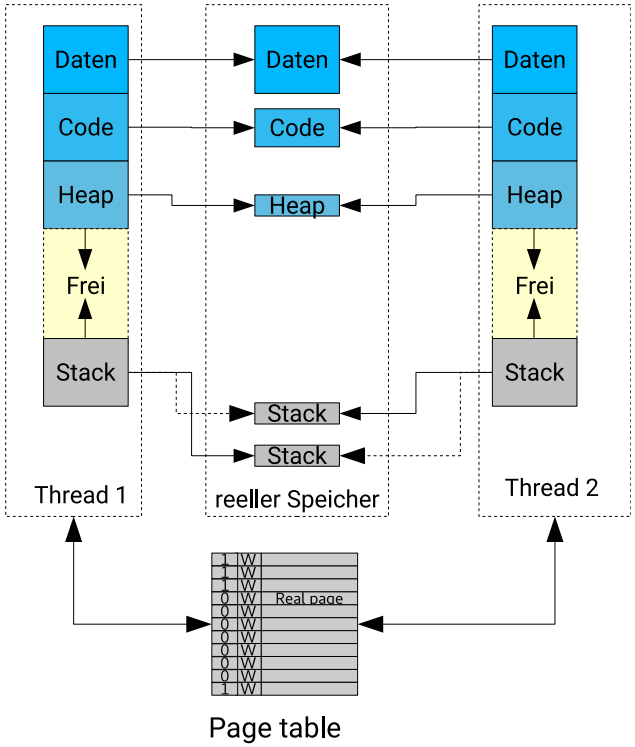
Einsatz von „fork()“ und „exec()“

- Die Aufrufe „fork()“ und „exec()“ sind voneinander unabhängig
- „fork()“ erzeugt ein Kind des aufrufenden Prozesses
- „exec()“ überlädt den Adressraum mit einer anderen Anwendung
- Damit sind 3 Variationen möglich:
- „fork()“ alleine
- „exec()“ alleine
- „fork()“ mit anschließendem „exec()“



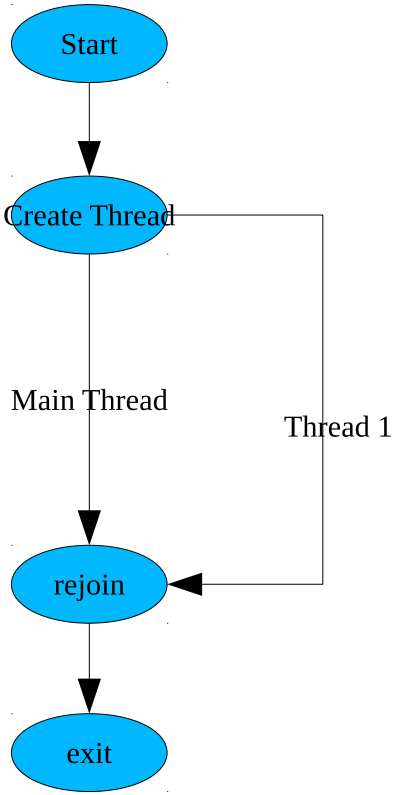
Threads

- „Leichtgewichtige“ Prozesse
- Der Adressraum wird nicht dupliziert
- Alle Threads greifen auf denselben Code, Daten, Heap zu
- Jeder Thread bekommt einen eigenen Stack, kann aber auf andere Stacks zugreifen, da nur eine Page-Table existiert



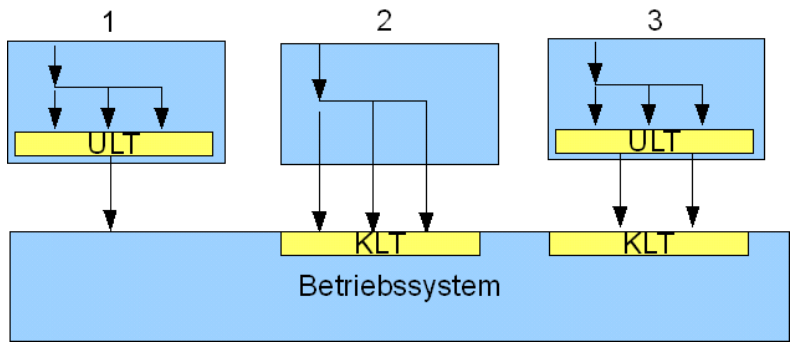
Threaderzeugung

- Jeder Prozess startet mit einem Hauptthread
- Dieser kann weitere Threads erzeugen (Systemaufruf z.B. „pthread_create()“)
- Der Erzeuger kann auf Beendigung des erzeugten Threads warten („pthread_join()“)
- Threads nutzen alle Ressourcen des Prozesses gemeinsam



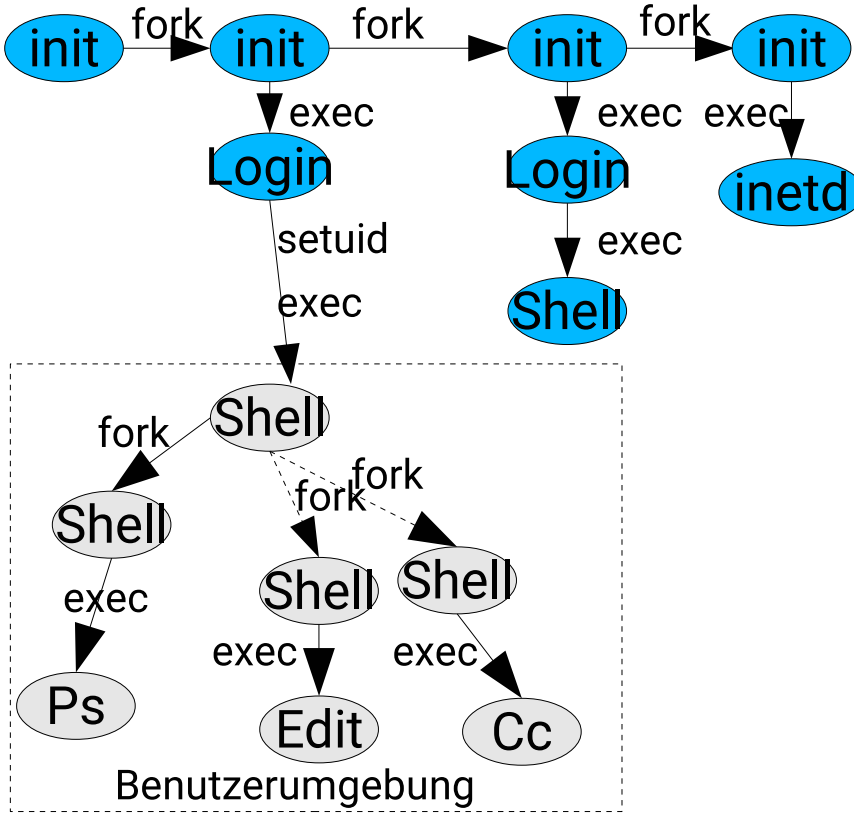
Arten von Threads

- Kernel-Level-Threads (KLT) werden durch Systemaufruf erzeugt und vom Betriebssystem verwaltet
- KLT können ein Mehrprozessorsystem ausnutzen
- User-Level-Threads (ULT 1) werden durch Bibliotheksaufrufe erzeugt und verwaltet, Betriebssystem „sieht“ nur einen Thread
- ULT dürfen keine blockierenden Systemaufrufe verwenden, da sonst alle ULT blockieren
- Mischformen (3) existieren. Die ULT-Bibliothek bildet ihre ULT auf KLT ab



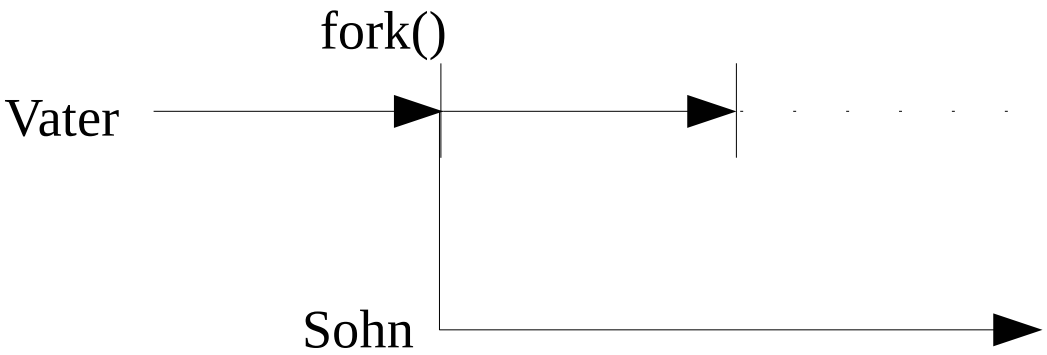
Systemstart und Aufbau der Prozesshierarchie

- Zum Start des Systems wird lediglich ein Urprozess (heute systemd, früher init) erzeugt
- Aus diesem Prozess gehen durch „fork()“ alle Prozesse des Systems hervor
- Systemprozesse laufen mit „Root“-Rechten
- Anwenderprozesse laufen mit den Rechten des Anwenders („setuid()“)



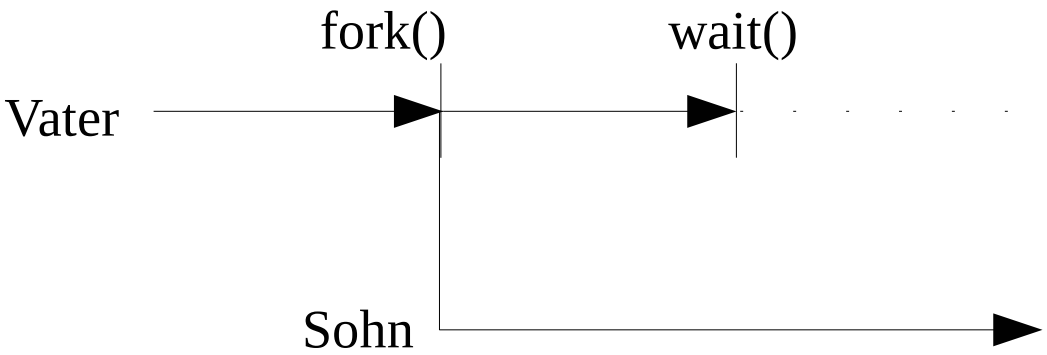
Warten auf Prozessende des Kindes

Erzeugerprozess (Vater) ruft „fork()“ auf, Kindprozesse läuft parallel weiter



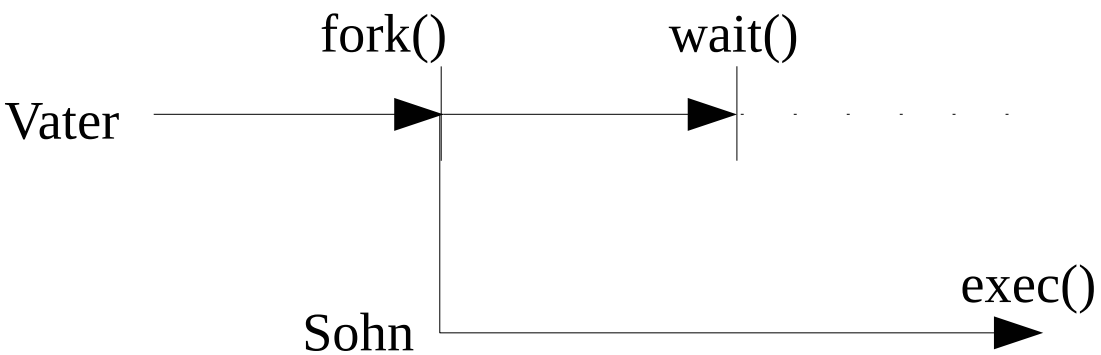
Warten auf Prozessende des Kindes

Vaterprozess ruft „wait()“ auf und wird angehalten, solange Kind läuft



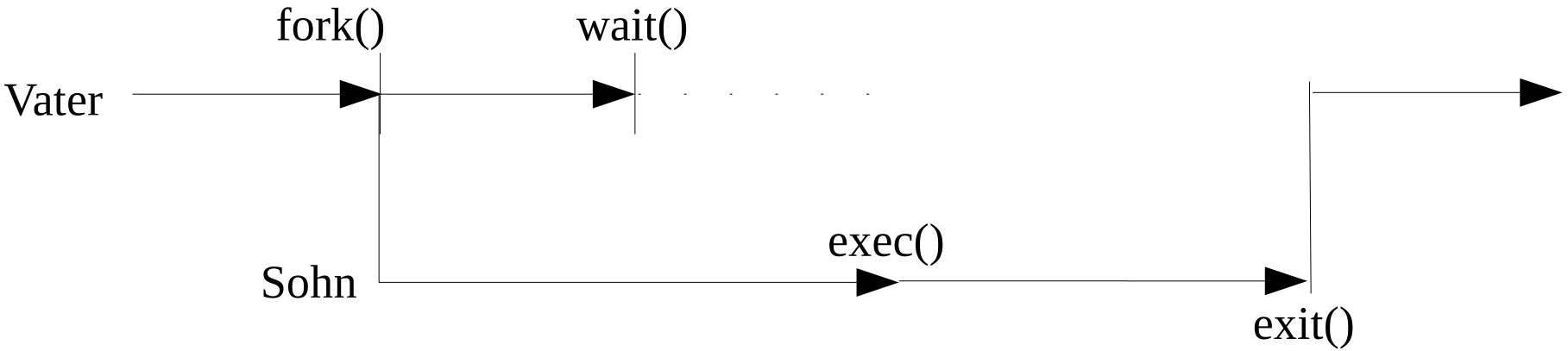
Warten auf Prozessende des Kindes

Kindprozess überlädt sich mit „exec()“ (kann auch fehlen)



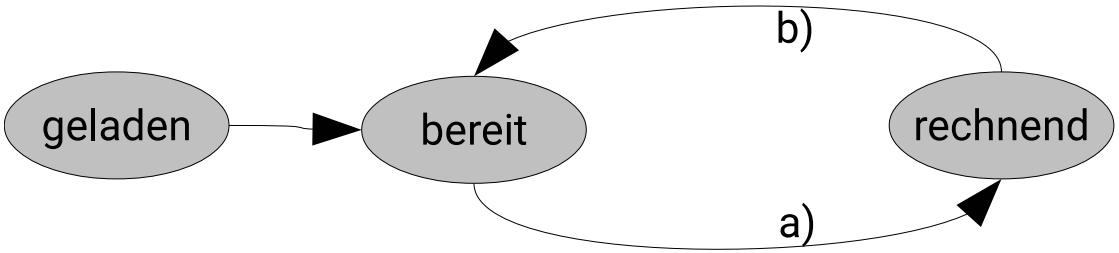
Warten auf Prozessende des Kindes

Kindprozess beendet sich, Vaterprozess wird fortgesetzt und kann Status des Kindes (Rückgabewert von „exit(val)“ auswerten



Prozesszustände

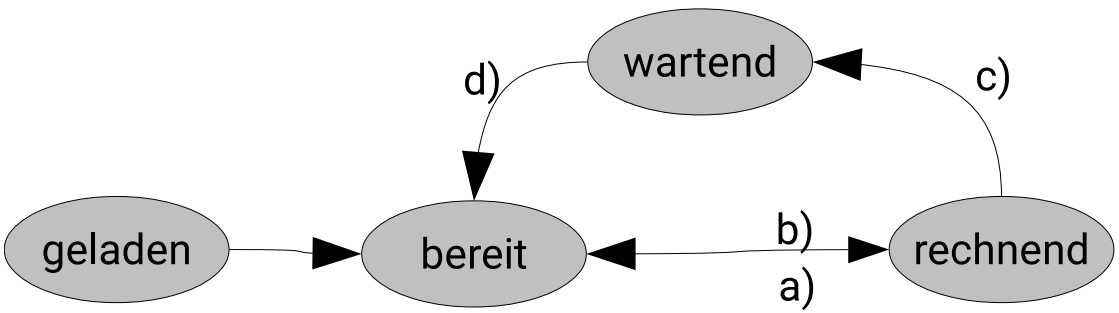
Wenn in einem System (hier mit einem Prozessor) mehrere Prozesse ausgeführt werden sollen, kann zu einer Zeit auch nur ein Prozess ausgeführt werden. Den anderen sind zur Ausführung zwar bereit, werden aber nicht ausgeführt



a) Prozessorzuteilung
b) Prozessorentzug

erweiterte Prozesszustände

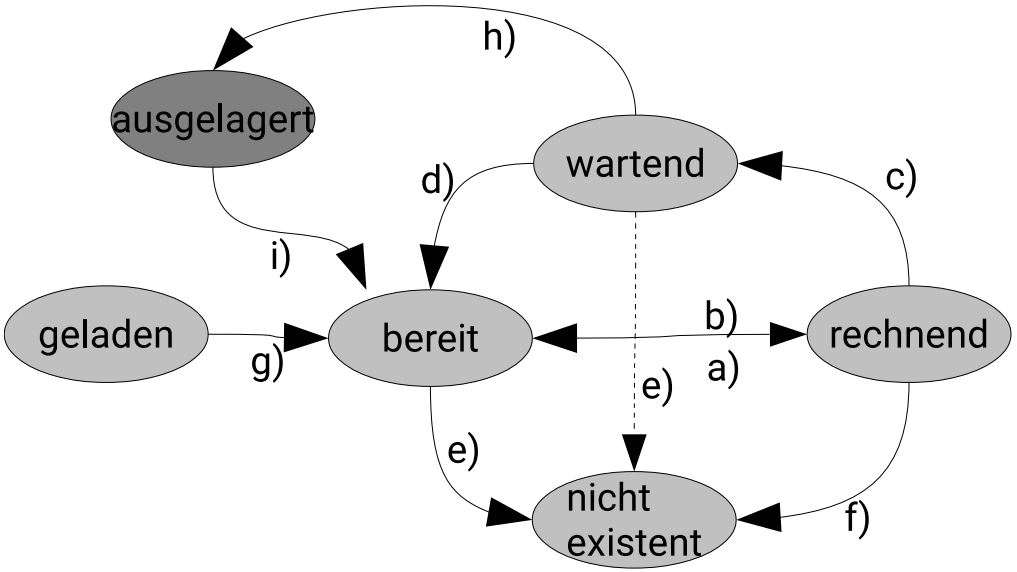
Wenn eine Task einen Systemaufruf absetzt, der in der Regel (z. B. hole Zeichen von der Tastatur) länger dauert, hat es keinen Sinn, ihr in der Zwischenzeit den Prozessor zu übergeben.
Daher wird ein weiterer Zustand eingeführt, der durch den Abschluss des Systemaufrufs verlassen wird.



a) Prozessorzuteilung
b) Prozessorentzug
c) Systemaufruf,
Warten auf Ergebnis
d) Ende Systemaufruf

vollständige Taskzustände

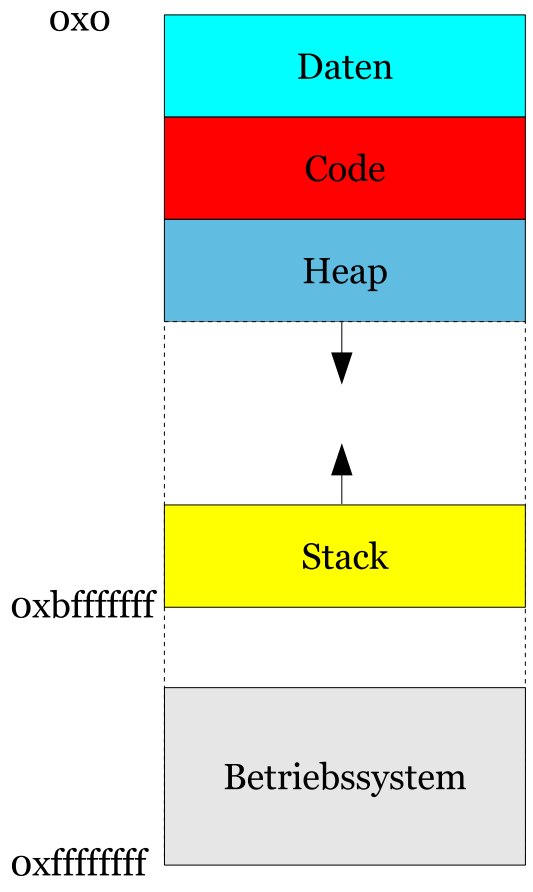
Wenn eine Anwendung aufgrund von Platzbedarf (System benötigt mehr Speicher als verfügbar ist), kann eine wartende Task aus dem Hauptspeicher auf den Sekundärspeicher (Swap-Area auf Festplatte) ausgelagert werden.



a) Prozessorzuteilung
b) Prozessorentzug
c) Systemaufruf,
Warten auf Ergebnis
d) Ende Systemaufruf
e) Taskabbruch
f) reguläres Taskende
g) Start
h) Auslagerung
i) Wiedereinlagerung

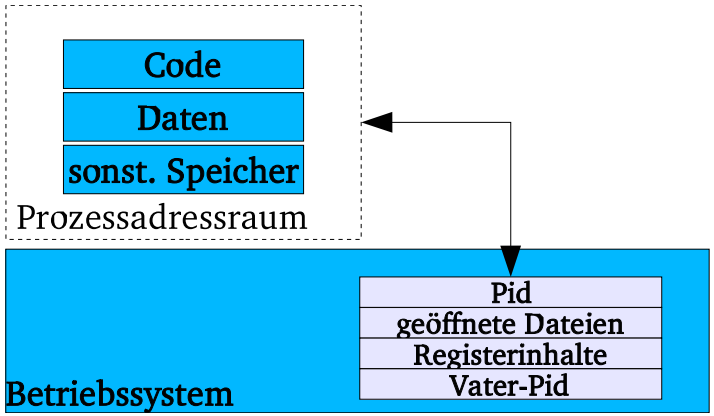
Aufteilung des virtuellen Adressraums

- Der verfügbare virtuelle Adressraum der Anwendungen wird in der Praxis eingeschränkt.
- Für das Betriebssystem wird ein Teil davon ausgespart, in den das Betriebssystem eingeblendet wird.
- Damit wird ermöglicht, dass das BS mit den virtuellen Adressen der Anwendungen auf deren Daten zugreifen kann, ohne seinen Adressraum umzuschalten.
- Bei Windows sind es 2GB, bei LInux 1 GB



Prozessverwaltung

Zum Prozess gehören nicht nur sein Adressraum mit den darin befindlichen Daten, Code und Stack, sondern auch Daten, die das Betriebssystem zur Verwaltung des Prozesses benötigt. Diese sind im Adressraum des Betriebssystems gespeichert, da diese Informationen dem Betriebssystem ständig zur Verfügung stehen müssen, der Anwenderadressraum aber jederzeit auf die Festplatte ausgelagert werden kann.



Inhalt des Prozesskontrollblocks

- Prozesspriorität

Inhalt des Prozesskontrollblocks

- Prozesspriorität
- bisherige Rechenzeit

Inhalt des Prozesskontrollblocks

- Prozesspriorität
- bisherige Rechenzeit
- Erzeugungszeitpunkt

Inhalt des Prozesskontrollblocks

- Prozesspriorität
- bisherige Rechenzeit
- Erzeugungszeitpunkt
- Kontext (Register etc., insbes. Befehlszeiger)

Inhalt des Prozesskontrollblocks

- Prozesspriorität
- bisherige Rechenzeit
- Erzeugungszeitpunkt
- Kontext (Register etc., insbes. Befehlszeiger)
- Prozessnummer (process identifier, PID)

Inhalt des Prozesskontrollblocks

- Prozesspriorität
- bisherige Rechenzeit
- Erzeugungszeitpunkt
- Kontext (Register etc., insbes. Befehlszeiger)
- Prozessnummer (process identifier, PID)
- Eigentümer (user ID (UID), group ID (GID))

Inhalt des Prozesskontrollblocks

- Prozesspriorität
- bisherige Rechenzeit
- Erzeugungszeitpunkt
- Kontext (Register etc., insbes. Befehlszeiger)
- Prozessnummer (process identifier, PID)
- Eigentümer (user ID (UID), group ID (GID))
- Eltern-PID (PPID)

Inhalt des Prozesskontrollblocks

- Prozesspriorität
- bisherige Rechenzeit
- Erzeugungszeitpunkt
- Kontext (Register etc., insbes. Befehlszeiger)
- Prozessnummer (process identifier, PID)
- Eigentümer (user ID (UID), group ID (GID))
- Eltern-PID (PPID)
- Dateideskriptoren

Inhalt des Prozesskontrollblocks

- Prozesspriorität
- bisherige Rechenzeit
- Erzeugungszeitpunkt
- Kontext (Register etc., insbes. Befehlszeiger)
- Prozessnummer (process identifier, PID)
- Eigentümer (user ID (UID), group ID (GID))
- Eltern-PID (PPID)
- Dateideskriptoren
- Signalmaske

Aufgaben des Scheduling

- Optimierung der Ressourcennutzung

Aufgaben des Scheduling

- Optimierung der Ressourcennutzung
- Maximierung des Durchsatzes

Aufgaben des Scheduling

- Optimierung der Ressourcennutzung
- Maximierung des Durchsatzes
- Minimierung der Antwortzeiten

Aufgaben des Scheduling

- Optimierung der Ressourcennutzung
- Maximierung des Durchsatzes
- Minimierung der Antwortzeiten
- Auslastung der CPU

Aufgaben des Scheduling

- Optimierung der Ressourcennutzung
- Maximierung des Durchsatzes
- Minimierung der Antwortzeiten
- Auslastung der CPU
- Fairness (es sollen keine Prozesse oder Anwender benachteiligt werden)

Aufgaben des Scheduling

- Optimierung der Ressourcennutzung
- Maximierung des Durchsatzes
- Minimierung der Antwortzeiten
- Auslastung der CPU
- Fairness (es sollen keine Prozesse oder Anwender benachteiligt werden)
- Garantie von Antwortzeiten (Echtzeit)

Arten des Scheduling

Nicht-präemptiv

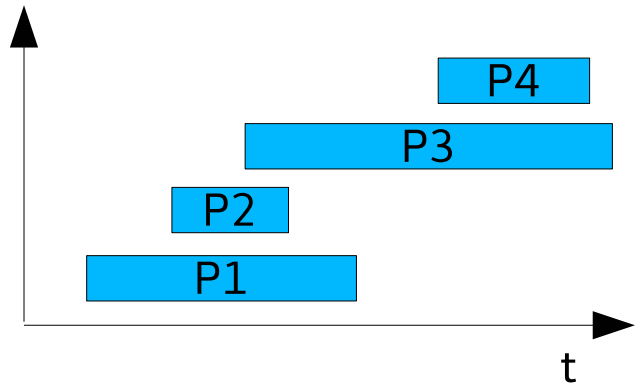
Die Prozesse werden gestartet und während der Laufzeit nicht unterbrochen. Erst wenn sie beendet sind, kann die CPU einem anderen Prozess zur Verfügung gestellt werden. Lediglich für eine I/O-Operation wird der Prozess blockiert und gibt die CPU an den folgenden Prozess ab. Er erhält sie dann aber erst zurück, wenn dieser sie wieder abgibt.

Präemptiv

Prozessen kann die CPU entzogen werden und einem anderen Prozess gegeben werden. Zu einem gegeben Zeitpunkt, der von der Scheduling-Strategie abhängt, erhält der Prozess die CPU zurück. Auch für I/O-Operationen, die das Betriebssystem für den Prozess ausführt, wird der Prozess gestoppt und erst nach der Operation wieder fortgesetzt.

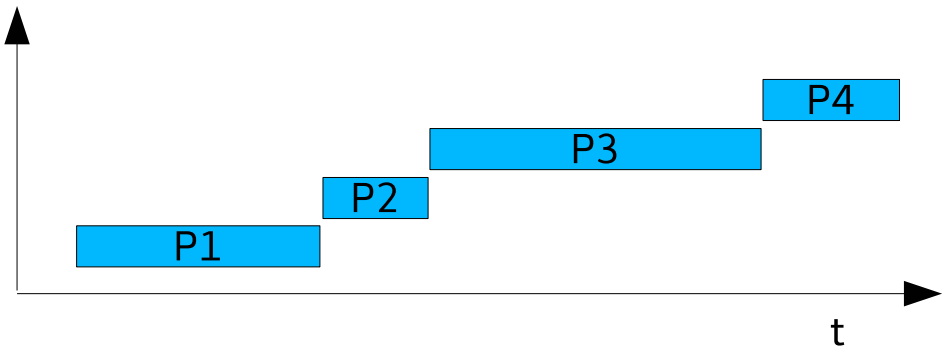
First in first out

Beispiel: 4 Prozesse mit verschiedenen Laufzeiten treffen in der Reihenfolge P1,P2,P3,P4 ein.



First in first out

Prozesse werden in der Reihenfolge abgearbeitet, in der sie eintreffen.

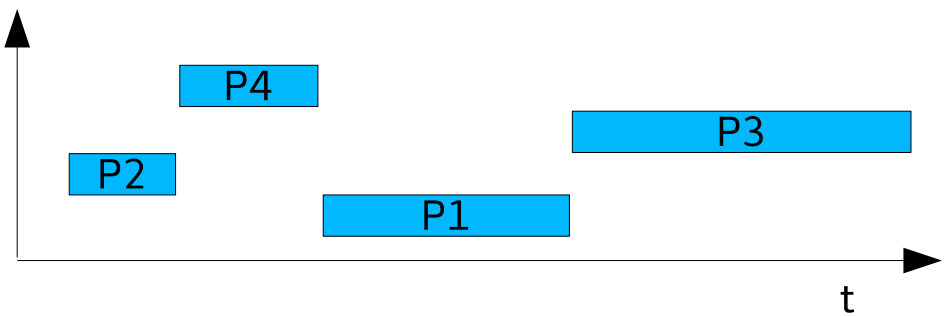


Shortest Job first

Prozesse werden so sortiert, dass der Prozess mit der kleinsten Laufzeit als erster bearbeitet wird. Dies minimiert die mittlere Wartezeit auf das Ende der Prozesse. Im Beispiel:
FCFS: MWZ = (10 + 13 + 28 + 32) / 4 = 20,75 ms
SJF: MWZ = (3 + 7 + 17 + 32) / 4 = 14,75 ms

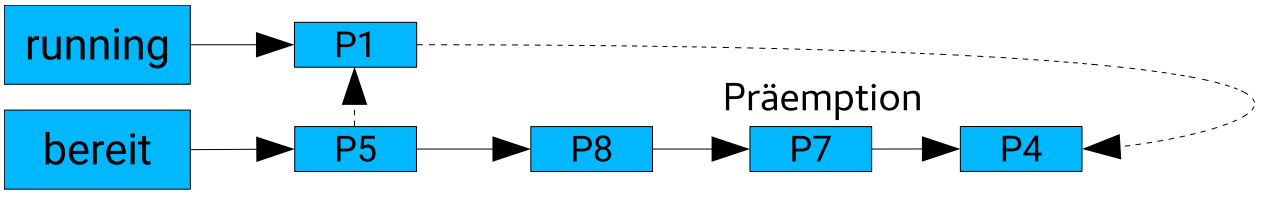
FCFS	LZ	Su.	WZ
P1	10ms	10ms	10ms
P2	3ms	13ms	13ms
P3	15ms	28ms	28ms
P4	4ms	32ms	32ms

SJF	LZ	Su.	WZ
P2	3ms	3ms	3ms
P4	4ms	7ms	7ms
P1	10ms	17ms	17ms
P3	15ms	32ms	32ms



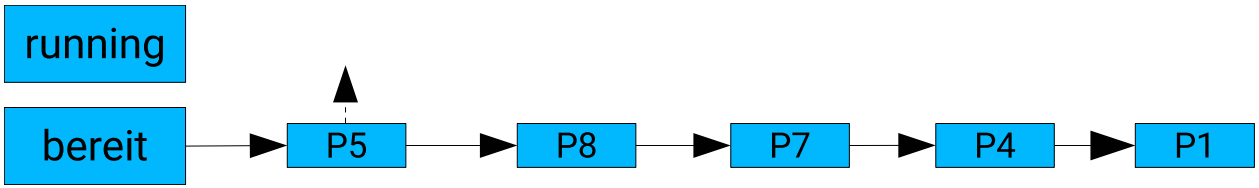
Round Robin

Ein Hardwaretimer unterbricht den laufenden Prozess zu äquidistanten Zeitpunkten.
Dem Betriebssystem wird dadurch die Kontrolle über den Prozessor gegeben und es kann entscheiden, ob der Prozess den Prozessor wieder bekommt oder ein anderer Prozess.
Bei Round Robin bekommen alle Prozesse den Prozessor max. für die Länge dieser sog. Zeitscheibe. Ist sie abgelaufen oder der laufende Prozess blockiert wegen einer I/O-Operation, kommt der nächste Prozess an die Reihe.



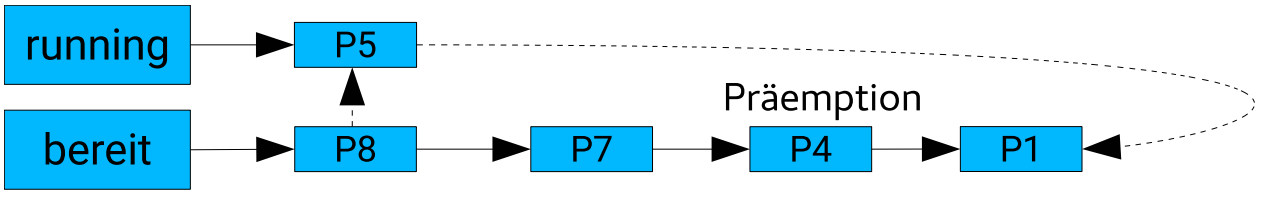
Round Robin

Ein Hardwaretimer unterbricht den laufenden Prozess zu äquidistanten Zeitpunkten.
Dem Betriebssystem wird dadurch die Kontrolle über den Prozessor gegeben und es kann entscheiden, ob der Prozess den Prozessor wieder bekommt oder ein anderer Prozess.
Bei Round Robin bekommen alle Prozesse den Prozessor max. für die Länge dieser sog. Zeitscheibe. Ist sie abgelaufen oder der laufende Prozess blockiert wegen einer I/O-Operation, kommt der nächste Prozess an die Reihe.



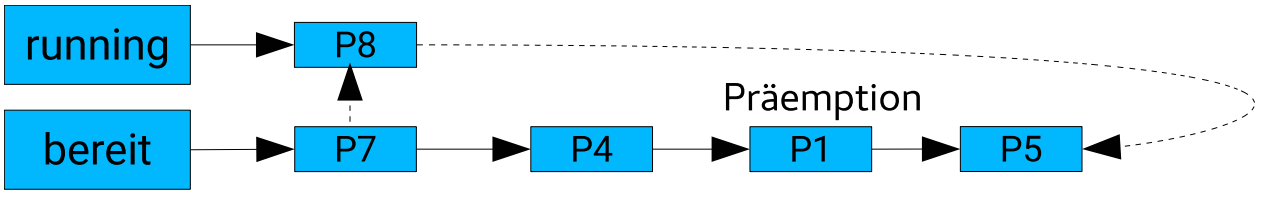
Round Robin

Ein Hardwaretimer unterbricht den laufenden Prozess zu äquidistanten Zeitpunkten.
Dem Betriebssystem wird dadurch die Kontrolle über den Prozessor gegeben und es kann entscheiden, ob der Prozess den Prozessor wieder bekommt oder ein anderer Prozess.
Bei Round Robin bekommen alle Prozesse den Prozessor max. für die Länge dieser sog. Zeitscheibe. Ist sie abgelaufen oder der laufende Prozess blockiert wegen einer I/O-Operation, kommt der nächste Prozess an die Reihe.



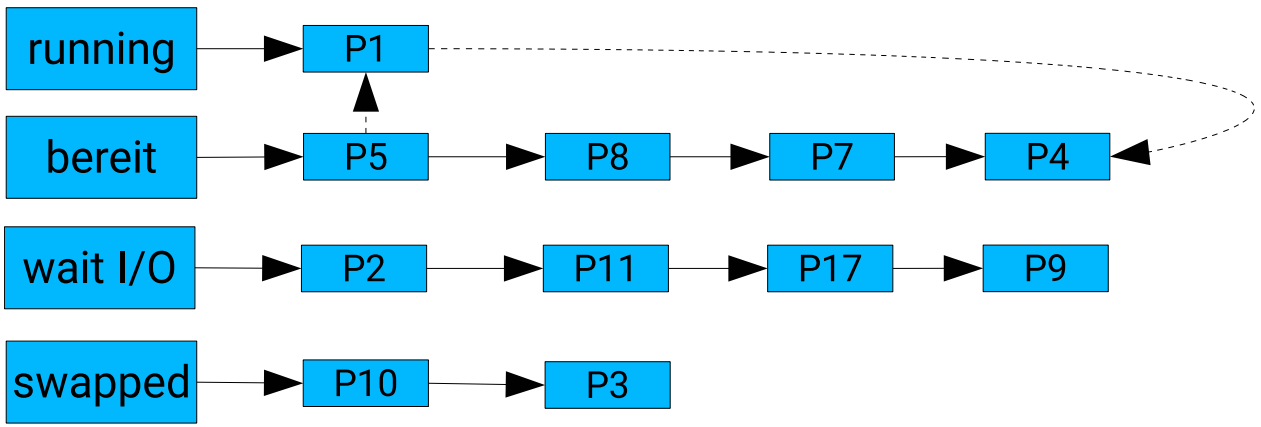
Round Robin

Ein Hardwaretimer unterbricht den laufenden Prozess zu äquidistanten Zeitpunkten.
Dem Betriebssystem wird dadurch die Kontrolle über den Prozessor gegeben und es kann entscheiden, ob der Prozess den Prozessor wieder bekommt oder ein anderer Prozess.
Bei Round Robin bekommen alle Prozesse den Prozessor max. für die Länge dieser sog. Zeitscheibe. Ist sie abgelaufen oder der laufende Prozess blockiert wegen einer I/O-Operation, kommt der nächste Prozess an die Reihe.



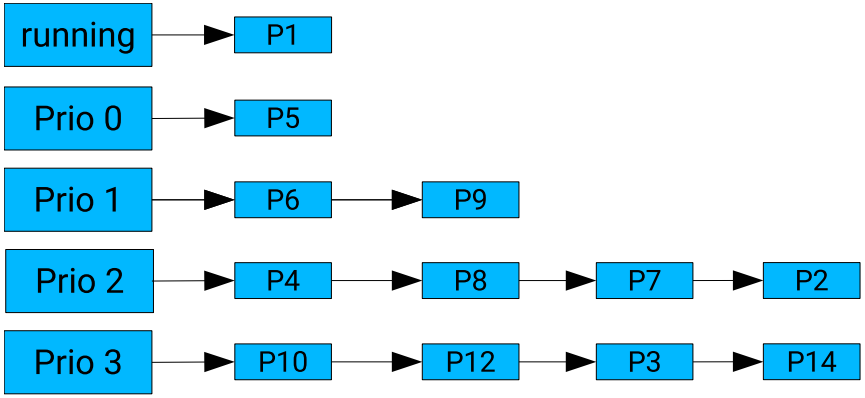
Round Robin

Damit Prozesse, die auf eine I/O-Operation warten nicht den Prozessor bekommen, werden diese gesondert verwaltet, ebenso die Prozesse, die wegen Speichermangels auf die Festplatte ausgelagert wurden.



Multilevel Scheduling

- Für Systeme mit Anwendungen mit unterschiedlichen Zeitanforderungen (Echtzeitsysteme), z. B. zeitkritische Prozesse und Prozesse ohne zeitliche Anforderungen.
- Warteschlangen mit Prioritäten (P0 - P3). Prozesse z.B. in Prio 1 werden erst ausgeführt, wenn keine Prozesse mehr in Prio 0 sind.
- In jeder Priorität unterschiedliche Scheduling-Verfahren.



Multilevel-Feedback Scheduling

- Ähnlich wie Multi-Level-Scheduling
- Prozesse starten in der höchsten Priorität
- Gibt ein Prozess den Prozessor vorzeitig ab, verbleibt er in der gleichen Priorität (hier Prio 1)
- Erreicht der Prozess das Ende der Zeitscheibe, wird er in die nächst niedere Priorität (Prio 2) abgestuft