

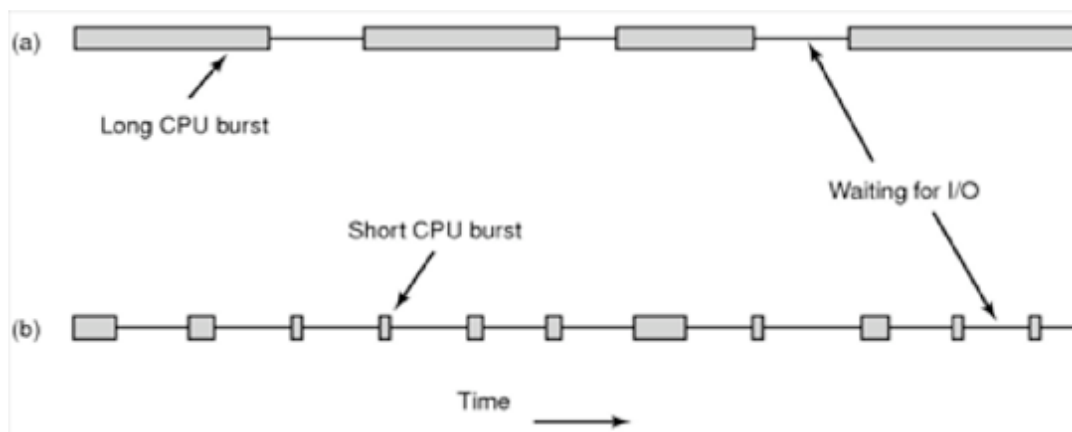
4 Scheduling

Scheduler: Der Teil des BS, welcher die Wahl trifft, welcher von mehreren konkurrierenden Prozessen CPU-Zeit bekommt. Benutzt Scheduling-Algorithmus.

Abwägen zwischen Ausführungsgeschwindigkeit und Komfort für den Benutzer

Am PC meist nur ein Prozess „vorne“, weitere „hinten“, inklusive der Dämonen. Der Scheduler kennt diese Unterscheidung meist gar nicht.

- Beispiel1: Nach Schließen eines Fensters zuerst E-Mail verschicken oder Hintergrundfenster neu zeichnen?
- Beispiel2: Bekommen eher CPU-lastige oder eher E/A-lastige Prozesse Rechenzeit (siehe Abb.)



Wann erfolgen Scheduling-Entscheidungen?

- nach Erzeugen eines Prozesses
- nach Beenden eines Prozesses
- Wenn ein Prozess blockiert (E/A, Semaphore)
- Nach einer zyklischen Unterbrechung durch Taktgeber

Scheduling verfolgt unterschiedliche Ziele (die sich teilweise ausschließen):

- Fairness – alle P. müssen CPU-Zeit bekommen
- Balance – alle Teile des Systems sind ausgelastet
- Durchsatz – Jobs pro Zeiteinheit
- Turnaround – Minimierung der Durchlaufzeit eines P.
- CPU-Belegung
- Antwortzeit
- Garantierte Reaktionszeit bei Echtzeit-Systemen
- Scheduler muss schnell sein (!)

Einfache Scheduling-Algorithmen (P. nicht unterbrechbar)

First-Come First-Served

- Einfachster Algorithmus, der P. nicht unterbricht

Shortest Job First

- Nicht unterbrechbar, aus der Warteschlange wird der kürzeste bearbeitet

Shortest Remaining Time Next

- S. wählt den Prozess, der die geringste Restlaufzeit hat
- Problem: Die Restlaufzeit muss bekannt sein.

Round-Robin-Scheduling

- Jedem Prozess wird ein Zeitabschnitt zugewiesen („Quantum“)
- Spätestens nach Ablauf des Quantums bekommt der P. die CPU entzogen (bei Blockierung oder Beendigung des P. natürlich früher)
- Verwalten der P. in einer Liste,

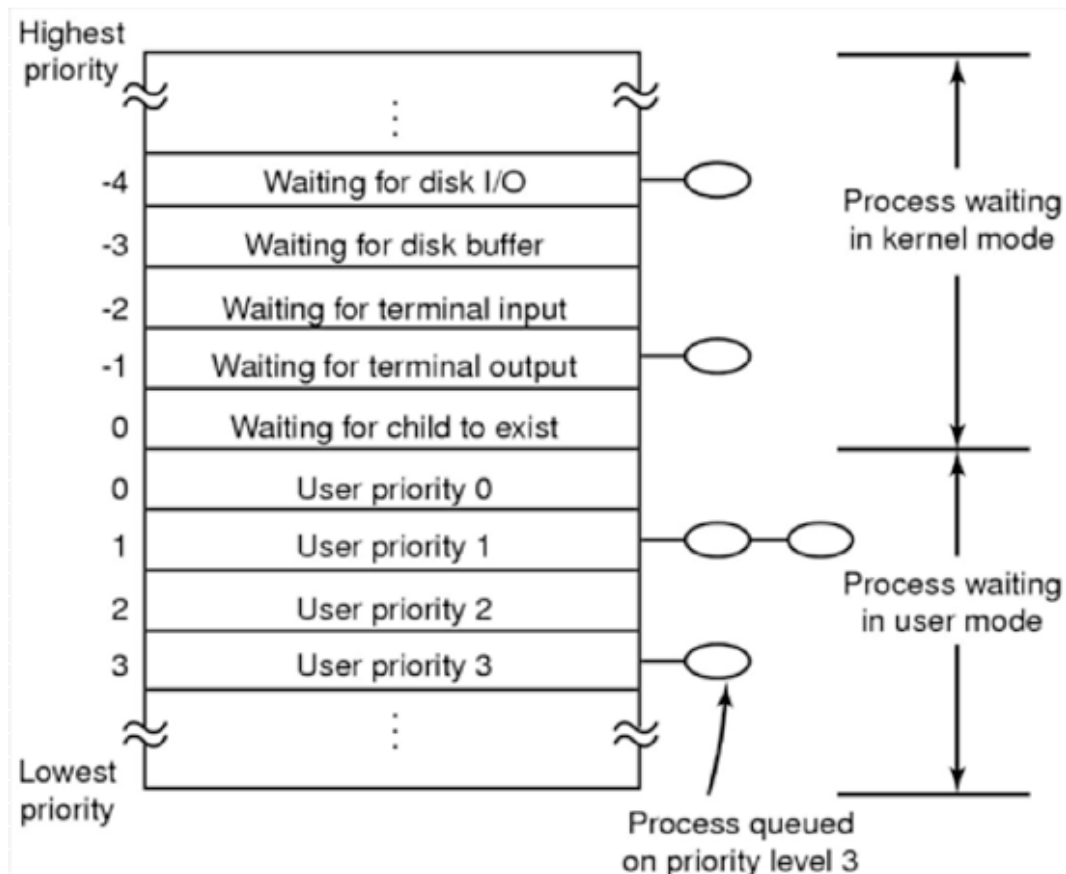


Abb.: Scheduling nachdem B sein Quantum verbraucht hat.

- Frage: Wie groß ist Quantum? (typischerweise 100ms oder kleiner)
- Populärster S.-Algorithmus
- Trade-off: Taskwechsel benötigt Zeit. Je kleiner das Quantum, desto mehr Rechenzeit wird für Taskwechsel verschwendet, aber desto geringer die Antwortzeit.

Prioritätenbasiertes Scheduling

- Jeder Prozess bekommt Priorität
- Der P. mit höchster Priorität bekommt CPU-Zeit
- Priorität verringert sich mit Laufzeit („Alterung“)
- P. können vorgegeben sein oder vom BS dynamisch ermittelt. Beispiel: E/A-lastige P. bekommen hohe Prio. Gerade gestartete P. sehr hohe Prio (da sie sich evtl. sehr schnell wieder beenden).



Typischer Scheduler in Unix: Kombination von Prioritätenbasiert und Round-Robin.

Basisidee: Prozesse möglichst schnell aus dem Kernel-Modus befördern: P., welche E/A durchgeführt haben, werden mit hoher Wahrscheinlichkeit wieder E/A anfordern

- und dann in den Zustand Blocked gehen und die CPU wieder abgeben
- E/A kann dann parallel zur CPU-Nutzung anderer P. durchgeführt werden
- E/A ist häufig Benutzerinteraktion (Tastatur, Bildschirm, Maus)

Weitere Problem bei Mehrprozessorsystemen (und Multi-Core-Systemen):

- Cacheinvalidierung bei Wechsel eines P. von einer CPU zur anderen
- Läuft ein zentraler Scheduler oder mehrere (pro CPU einer?)