Scheduling

Linux 2.4 vs. 2.6

Johannes Segitz

Was ist Scheduling/ein Scheduler?

Scheduling Erstellen einer zeitlichen Ordnung fuer Ereignisse

Scheduler Die Einheit, die diese Entscheidung trifft

Warum brauchen wir Scheduling?

- Bestmoegliche Ausnutzung existierender Ressourcen

Warum brauchen wir Scheduling?

- Bestmoegliche Ausnutzung existierender Ressourcen
- Reaktionszeiten des Systems verbessern

Warum brauchen wir Scheduling?

- Bestmoegliche Ausnutzung existierender Ressourcen
- Reaktionszeiten des Systems verbessern
- Ermoeglichung bestimmter Anwendungen

Geschichte

Scheduler gab es nicht immer in Betriebssystemen. Sie wurden begleitend zu dem technischen Fortschritt der Hardware eingefuehrt

- Batch operating systems

Geschichte

Scheduler gab es nicht immer in Betriebssystemen. Sie wurden begleitend zu dem technischen Fortschritt der Hardware eingefuehrt

- Batch operating systems
- Multiprogrammed systems

Geschichte

Scheduler gab es nicht immer in Betriebssystemen. Sie wurden begleitend zu dem technischen Fortschritt der Hardware eingefuehrt

- Batch operating systems
- Multiprogrammed systems
- Time-sharing system

Arten des Schedulings

Es gibt verschiedene computerrelevante Bereiche, in denen Scheduling eingesetzt wird, z.B.:

- Befehlscheduling

Arten des Schedulings

Es gibt verschiedene computerrelevante Bereiche, in denen Scheduling eingesetzt wird, z.B.:

- Befehlscheduling
- I/O Scheduling

Arten des Schedulings

Es gibt verschiedene computerrelevante Bereiche, in denen Scheduling eingesetzt wird, z.B.:

- Befehlscheduling
- I/O Scheduling
- CPU Scheduling

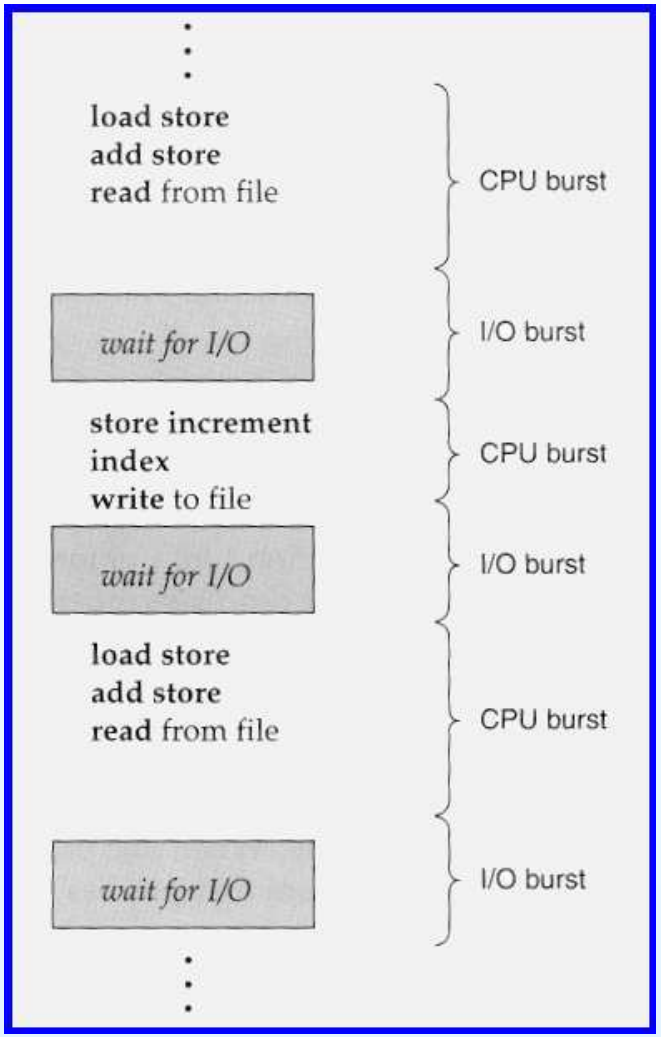
Arten des Schedulings

Es gibt verschiedene computerrelevante Bereiche, in denen Scheduling eingesetzt wird, z.B.:

- Befehlscheduling
- I/O Scheduling
- CPU Scheduling

Kombination verschiedener Scheduler um optimale Leistung zu erreichen (z.B. CPU Scheduling mit I/O Scheduling)

Kombination von Schedulern



CPU Scheduling

Bestimmt welcher **ablauffaehige** Prozess die CPU benutzen darf.

Abhaengig vom Zeitraum, ueber den man die Entscheidung trifft, unterscheiden wir:

- Kurzzeitscheduler
- Langzeitscheduler

CPU Scheduler sind Kurzzeitscheduler

Wichtig: Scheduler != Dispatcher

Ziele

Wie vergleicht man Schedulingalgorithmen gegeneinander?

Uebliche Ziele sind:

- Maximiere Ressourcenauslastung

Ziele

Wie vergleicht man Schedulingalgorithmen gegeneinander?

Uebliche Ziele sind:

- Maximiere Ressourcenauslastung
- Maximiere Durchsatz

Ziele

Wie vergleicht man Schedulingalgorithmen gegeneinander?

Uebliche Ziele sind:

- Maximiere Ressourcenauslastung
- Maximiere Durchsatz
- Minimiere Ausfuehrungszeit

Ziele

Wie vergleicht man Schedulingalgorithmen gegeneinander?

Uebliche Ziele sind:

- Maximiere Ressourcenauslastung
- Maximiere Durchsatz
- Minimiere Ausfuehrungszeit
- Minimiere Antwortzeit

Ziele

Wie vergleicht man Schedulingalgorithmen gegeneinander?

Uebliche Ziele sind:

- Maximiere Ressourcenauslastung
- Maximiere Durchsatz
- Minimiere Ausfuehrungszeit
- Minimiere Antwortzeit
- Minimiere Wartezeit

Bewertung von Schedulingalorithmen

Man muss eine sinnvolle Auswahl an Anforderungen treffen und die Algorithmen basierend auf dieser Auswahl vergleichen

- Analytische Auswertung
- Warteschlangenmodelle
- Simulation
- Implementierung

Analytische Auswertung

Wichtigste Methode ist die deterministische Auswertung. Diese berechnet die Leistung eines Algorithmus anhand einer bestimmten Arbeitslast

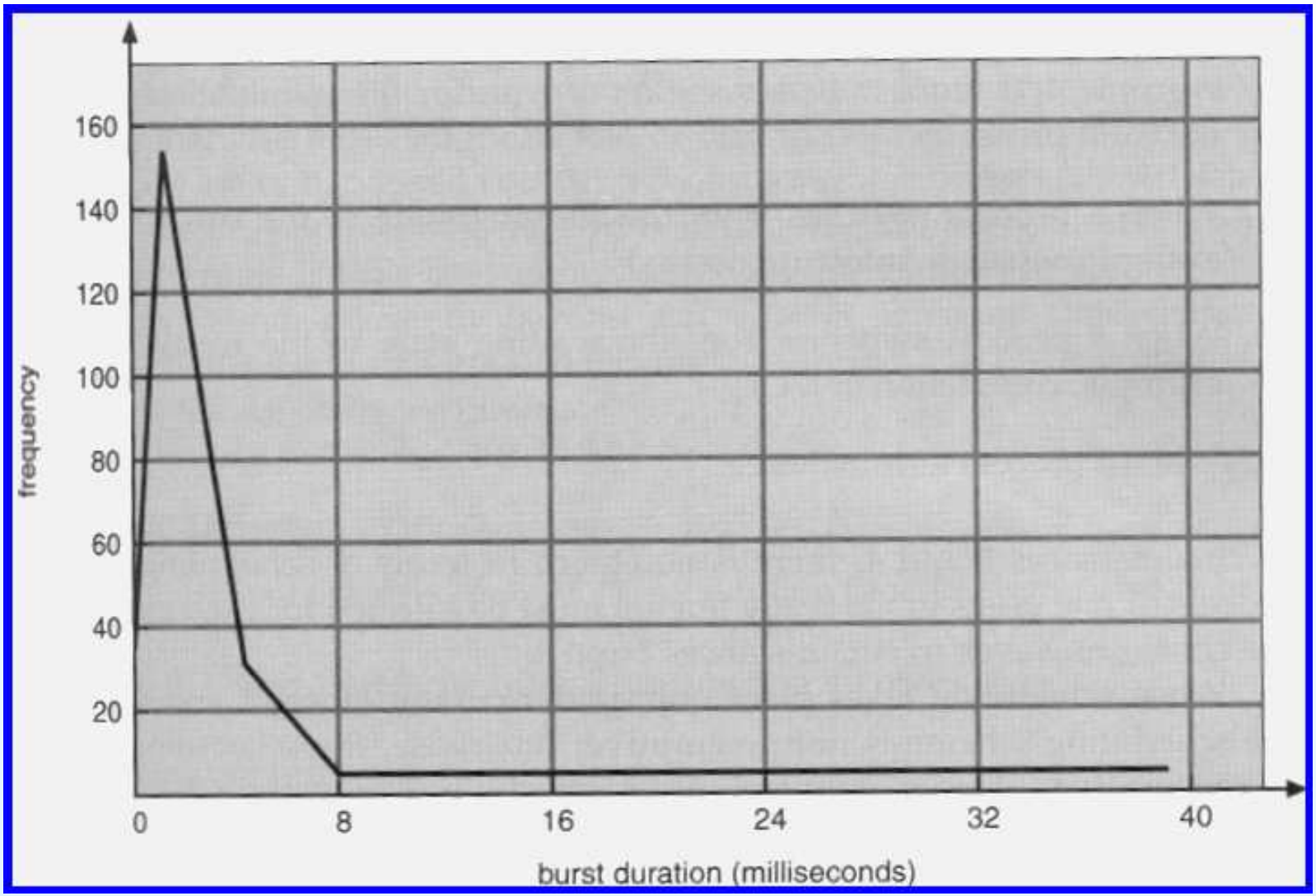
Vorteile:

- einfach und schnell
- liefert exakte Zahlen

Nachteil:

- Aussage nur gueltig fuer untersuchte Arbeitslast

Verteilung der CPU Nutzungszeit-Laengen



Warteschlangemodelle

Das System wird als ein Netzwerk von Warteschlangen dargestellt.

Vorteil:

- Nicht spezifisch zur untersuchten Arbeitslast

Nachteile:

- Nur eine Abschaetzung
- Mathematisch anspruchsvoll

Simulation

Wir konstruieren ein Modell, welches das System nachbildet. Dieses Modell wird verwendet um Daten ueber das zu erwartende Verhalten des Systems zu sammeln.

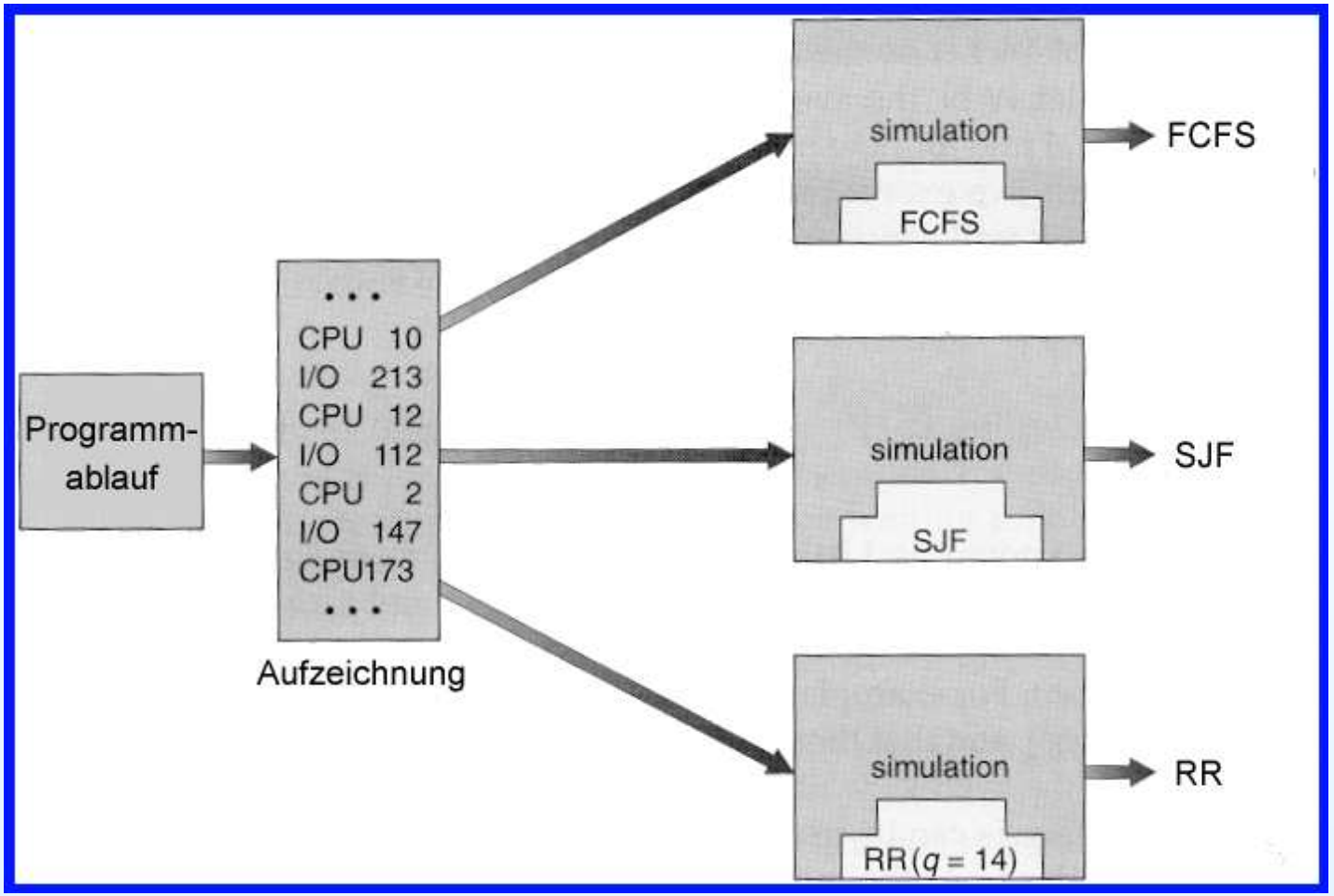
Vorteil:

- Wenn das Modell korrekt ist kann man gute Resultate erzielen

Nachteile:

- Hoher Aufwand das Modell zu erzeugen
- Eventuell hoher Rechenzeitbedarf

Beispiel einer Simulation



Implementierung

Man programmiert den Algorithmus und baut ihn in ein Betriebssystem ein, um dieses zu untersuchen

Vorteil:

- Genaue Resultate

Nachteile:

- Sehr hoher Aufwand
- Nicht immer moeglich (z.B. bei wichtigen Systemen)
- Veraenderung der Umgebung

Scheduling Algorithmen

- First come, first serve (FCFS)

Scheduling Algorithmen

- First come, first serve (FCFS)
- Shortest job first
  - unterbrechend (PSJF)
  - nicht unterbrechend (SJF)

Scheduling Algorithmen

- First come, first serve (FCFS)
- Shortest job first
  - unterbrechend (PSJF)
  - nicht unterbrechend (SJF)
- Priority scheduling
  - unterbrechend
  - nicht unterbrechend

Scheduling Algorithmen

- First come, first serve (FCFS)
- Shortest job first
  - unterbrechend (PSJF)
  - nicht unterbrechend (SJF)
- Priority scheduling
  - unterbrechend
  - nicht unterbrechend
- Round-robin scheduling (RR)

Scheduling Algorithmen

- First come, first serve (FCFS)
- Shortest job first
  - unterbrechend (PSJF)
  - nicht unterbrechend (SJF)
- Priority scheduling
  - unterbrechend
  - nicht unterbrechend
- Round-robin scheduling (RR)
- Multilevel queue scheduling

Scheduling Algorithmen

- First come, first serve (FCFS)
- Shortest job first
  - unterbrechend (PSJF)
  - nicht unterbrechend (SJF)
- Priority scheduling
  - unterbrechend
  - nicht unterbrechend
- Round-robin scheduling (RR)
- Multilevel queue scheduling
- Multilevel feedback queue scheduling

Mehr-Prozessor Scheduling

Zusaetzlich zu der Entscheidung, welcher Prozess laufen soll, kommt die Entscheidung auf welcher CPU.

Probleme:

- Gleichmaessige Belastung der Prozessoren
- Verwaltungsaufwand (z.B. Relokalisieren von Prozessen)

Scheduling in Linux

Der Linux Scheduler muss vielen Anforderungen genuegen, da Linux auf den verschiedensten Plattformen laeuft, vom Handheld bis zum Mainframe

- Mainframes muessen primaer ihre Resourcen effizient nutzen

Scheduling in Linux

Der Linux Scheduler muss vielen Anforderungen genuegen, da Linux auf den verschiedensten Plattformen laeuft, vom Handheld bis zum Mainframe

- Mainframes muessen primaer ihre Resourcen effizient nutzen
- PCs sollen allzeit kurze Reaktionszeiten bieten

Scheduling in Linux

Der Linux Scheduler muss vielen Anforderungen genuegen, da Linux auf den verschiedensten Plattformen laeuft, vom Handheld bis zum Mainframe

- Mainframes muessen primaer ihre Resourcen effizient nutzen
- PCs sollen allzeit kurze Reaktionszeiten bieten

Von nun an wird nur die Versionsnummer verwendet um auf die korrespondierenden Kernel zu verweisen

- 2.4 == Linux Kernel 2.4.*
- 2.6 == Linux Kernel 2.6.0-test*

Scheduling Klassen in Linux

Sowohl 2.4 als auch 2.6 bieten drei Scheduling-Klassen

SCHED_OTHER Traditionelle UNIX Prozesse

SCHED_FIFO POSIX.1b FIFO realtime Prozesse

SCHED_RR POSIX round-robin realtime Prozesse

Der alte 2.4 Scheduler

- Globale, ungeordnete Liste haelt alle lauffaehigen Prozesse

Der alte 2.4 Scheduler

- Globale, ungeordnete Liste haelt alle lauffaehigen Prozesse
- Vergabe von Punkten in goodness() basierend auf verbleibender Zeit, **nice**-Wert und CPU Affinitaet

Berechnung des goodness()-Wertes

- Realtime-Prozesse: goodness=1000+rt_priority

Berechnung des goodness()-Wertes

- Realtime-Prozesse: goodness=1000+rt_priority
- Time-Sharing:
  - quantum>0: goodness=quantum+priority
  - quantum=0: goodness=0

Probleme des 2.4 Schedulers

- Neuberechnung der Zeitscheiben erst nachdem alle aufgebraucht wurden

Probleme des 2.4 Schedulers

- Neuberechnung der Zeitscheiben erst nachdem alle aufgebraucht wurden
- Rescheduling erfordert vollstaendiges Durchlaufen der Prozessliste => O(n)

Probleme des 2.4 Schedulers

- Neuberechnung der Zeitscheiben erst nachdem alle aufgebraucht wurden
- Rescheduling erfordert vollstaendiges Durchlaufen der Prozessliste => O(n)
- Schlecht skalierbar wegen globaler Prozessliste

Probleme des 2.4 Schedulers

- Neuberechnung der Zeitscheiben erst nachdem alle aufgebraucht wurden
- Rescheduling erfordert vollstaendiges Durchlaufen der Prozessliste => O(n)
- Schlecht skalierbar wegen globaler Prozessliste

Der neue O(1) Scheduler fuer 2.6

Vorteile des neuen Schedulers:

- O(1) Algorithmen

Der neue O(1) Scheduler fuer 2.6

Vorteile des neuen Schedulers:

- O(1) Algorithmen
- CPU Affinitaet

Der neue O(1) Scheduler fuer 2.6

Vorteile des neuen Schedulers:

- O(1) Algorithmen
- CPU Affinitaet
- Verbesserte Interaktivitaet

Der neue O(1) Scheduler fuer 2.6

Vorteile des neuen Schedulers:

- O(1) Algorithmen
- CPU Affinitaet
- Verbesserte Interaktivitaet
- Verbesserte Skalierbarkeit

Der neue O(1) Scheduler fuer 2.6

Vorteile des neuen Schedulers:

- O(1) Algorithmen
- CPU Affinitaet
- Verbesserte Interaktivitaet
- Verbesserte Skalierbarkeit
- Unterbrechbarer Kernel

Der neue O(1) Scheduler fuer 2.6

Vorteile des neuen Schedulers:

- O(1) Algorithmen
- CPU Affinitaet
- Verbesserte Interaktivitaet
- Verbesserte Skalierbarkeit
- Unterbrechbarer Kernel
- Fuer weiche Echtzeit geeignet

Prozesslisten

Der Scheduler benutzt zwei nach Prioritaet geordnete Felder.

- Eines fuer Prozesse, die noch Prozessorzeit uebrig haben (active array)
- Eines fuer Prozesse, die ihre Prozessorzeit verbraucht haben (expired array)

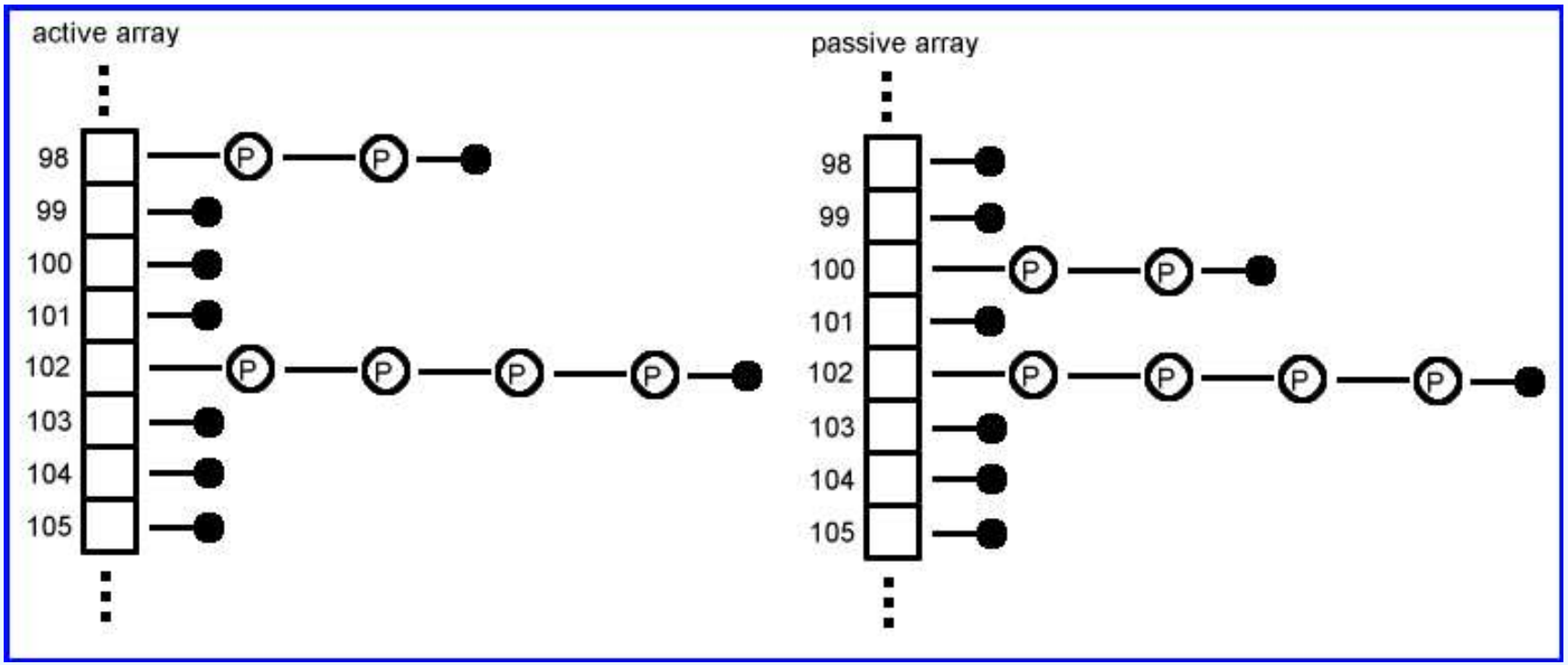
Prozesslistenfelder

Jedes dieser Felder enthaelt 140 Eintraege, die auf Listen zeigen.

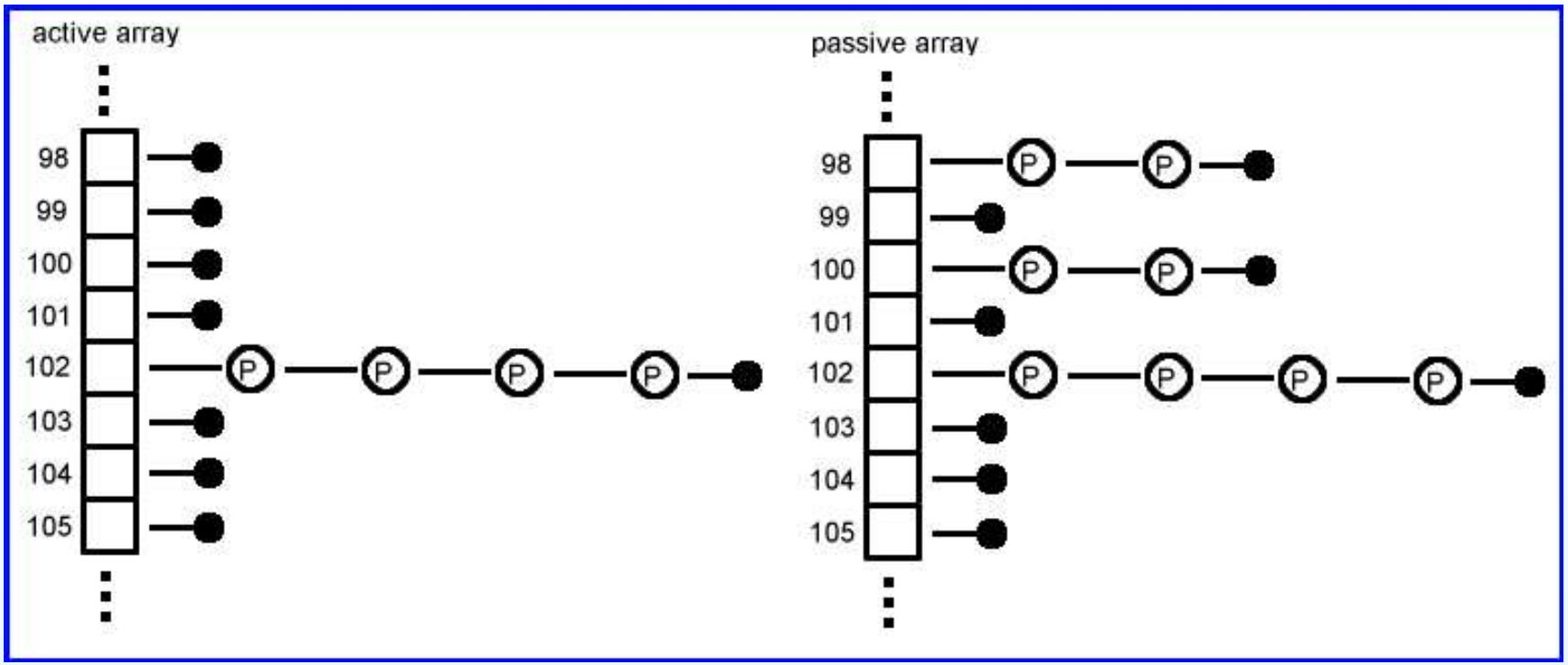
- Die ersten 100 Eintraege sind fuer Echtzeitprozesse
- Die letzten 40 representieren die 40 **nice** Werte von -19 bis 20

Prozesse, die ihre Prozessorenzeit verbraucht haben, werden vom "active array" ins "expired array" verschoben.

Beispiel


active array
98
99
100
101
102
103
104
105
P
passive array
98
99
100
101
102
103
104
105
P

Beispiel


active array
98
99
100
101
102
103
104
105
P
P
P
P
passive array
98
99
100
101
102
103
104
105
P
P
P
P
P
P
P
P
P
P

Vorteile dieses Ansatzes

- Der Wechsel der Felder erfordert nur das Wechseln zweier Zeiger

Vorteile dieses Ansatzes

- Der Wechsel der Felder erfordert nur das Wechseln zweier Zeiger
- Der Prozess mit der hoechsten Prioritaet laesst sich schnell ueber einen bitmap cache finden

Vorteile dieses Ansatzes

- Der Wechsel der Felder erfordert nur das Wechseln zweier Zeiger
- Der Prozess mit der hoechsten Prioritaet laesst sich schnell ueber einen bitmap cache finden
- Hochpriorisierte Prozesse koennen niedrig priorisierte nicht am Laufen hindern

Veraendern der Prioritaeten und Zeitscheiben

Pro Prozess wird eine Last-Abschaetzung durchgefuehrt

- Ringbuffer mit vier Eintraegen
- Wird verwendet um die Last-Historie des Prozesses aufzuzeichnen
- Prozesse, die viel Last zum System beitragen, werden depriorisiert
- Gegebener Rahmen fuer Prioritaetsaenderung durch statische Prioritaet

Prozesse mit hoeherer Prioritaet bekommen groessere Zeitscheiben

Fazit

Scheduling:

- Scheduling ist einer der wichtigsten Teile des Betriebssystems
- Theoretisch einfach, praktisch schwierig

Fazit

Scheduling:

- Scheduling ist einer der wichtigsten Teile des Betriebssystems
- Theoretisch einfach, praktisch schwierig

Linux:

- Alter Scheduler bereits gut
- Neuer Scheduler wesentlich besser fuer interaktive Nutzung und Extremsituationen