Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Kap. 4: Scheduling

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Grundlegende Begriffe:

- Wiederholung aus Kap. 3.1:
  - Scheduler oder Dispatcher: Umschalteinheit
  - Scheduling-Algorithmus: zugehöriger Algorithmus
  - Prozesswechsel oder Kontextwechsel: Umschaltungsvorgang, verursacht Kosten.
- Preemptives Scheduling: rechnende Prozesse können suspendiert werden (Prozessorentzug)
  - preemptive-resume: Fortsetzung ohne Verlust
  - preemptive-repeat: Beginn von vorne
- Non-preemptives Scheduling-Verfahren (Run-to-Completion, d.h. Prozess ist solange aktiv, bis er endet oder sich selbst blockiert).
- Non-preemptive-Scheduling-Verfahren sind für General Purpose Systeme mit interaktiven Benutzern nicht geeignet.

Grundlegende Begriffe (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Prioritäten-basierte Scheduling-Verfahren: ordnen Prozessen Prioritäten zu (relative Wichtigkeit).
- Prioritäten können extern vorgegeben sein, oder intern durch das Betriebssystem selbst bestimmt werden.
- Prioritäten können statisch sein, d.h. ändern sich während der Bearbeitung nicht, andernfalls dynamischen Prioritäten.
- Scheduling-Verfahren, die Prozessorzuteilung auf der Basis gewählter Zeitspannen mittels Uhrunterbrechungen steuern, heißen Zeitscheiben-basierte Scheduling-Verfahren.
- Mischformen sind möglich.

Grundlegende Begriffe (3)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

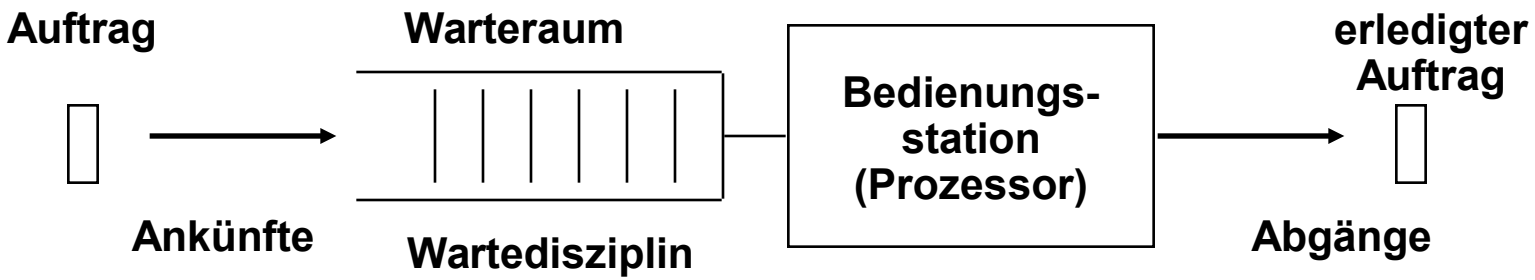
- Mehrstufiges Scheduling: Scheduler auf mehreren Ebenen

  Nur die auf einer höheren Ebene durch den entsprechenden Scheduler ausgewählten Prozesse sind dem Scheduler der nächst niederen Ebene für sein Scheduling sichtbar. Scheduling der niederen Ebene wird dann häufig als Dispatching bezeichnet.

  Typisches Anwendungsbeispiel: 2-stufiges Scheduling;
  Der Scheduler der höheren Ebene transportiert Aufträge vom Hintergrundspeicher in den Arbeitsspeicher und zurück (Swapping); der Scheduler der niederen Ebene berücksichtigt ausschließlich Prozesse, die sich im Arbeitsspeicher befinden.

Begriffe der Bedientheorie

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

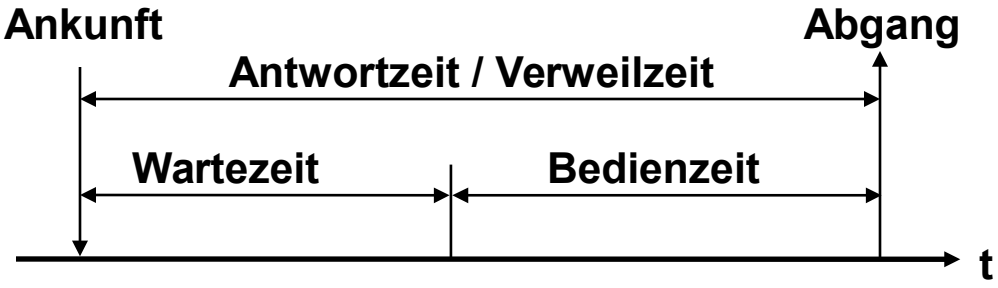


- Auftrag: Einheit zur Bearbeitung (z.B. Stapeljob, Dialogschritt).
- Bedienzeit: Zeitdauer für die reine Bearbeitung eines Auftrags durch die Bedienstation (den Prozessor).
- Wartedisziplin: einfache:
  - FCFS (First-Come-First-Served) oder FIFO (First-In-First-Out),
  - LIFO (Last-In-First-Out),
  - Random (zufällig ausgewählt).
- Ankünfte und Bedienungszeiten werden bei Bedienungsmodellen häufig durch stochastische Prozesse modelliert.
- komplexere Bedienmodelle können mehrere Bedienstationen (Multiprozessorsystem), mehrere Warteräume, mehrphasige Bedienung sowie die Rückführung teilweise bearbeiteter Aufträge enthalten.

Begriffe der Bedientheorie (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Bedienung eines Auftrags:



- Antwortzeit: Zeitdauer vom Eintreffen eines Auftrags bis zur Fertigstellung.

  bei Dialogaufträgen: Zeitdauer von einer Eingabe eines Benutzers (z.B. Drücken der return-Taste) bis zur Erzeugung einer zugehörigen Ausgabe (z.B. auf dem Bildschirm).

  bei Stapelaufträgen auch Verweilzeit genannt.
- Wartezeit: Antwortzeit - Bedienzeit.
- Durchsatz: Anzahl der erledigten Aufträge pro Zeiteinheit.
- Auslastung: Anteil der Zeit im Zustand belegt.
- Fairness: "Gerechte" Behandlung aller Aufträge, z.B. alle rechenwilligen Prozesse haben gleichen Anteil an der zur Verfügung stehenden Prozessorzeit.

"Guter" Scheduling-Algorithmus

Ziele:

Nutzer-bezogen:

- Kurze Antwortzeiten bei interaktiven Aufträgen.
- Kurze Verweilzeiten für Stapelverarbeitungsaufträge.

Betreiber-bezogen:

- Hoher Durchsatz.
- Hohe Auslastung.
- Fairness in der Behandlung aller Aufträge.
- Geringer Aufwand für die Bearbeitung des Scheduling-Algorithmus selbst (Overhead!).

Vorabwissen über die Bedienzeit:

- Einige Verfahren verlangen Kenntnis der Bedienzeit eines Auftrags bei dessen Ankunft (vgl. Verfahren zur Maschinenbelegung, Operations Research).
- Nur realistisch für wiederkehrende Stapeljobs, nicht realistisch bei interaktiven Aufträgen.

Trennung von Strategie und Mechanismus:

- Ziel: höhere Flexibilität.
- Parametrisierbarer Scheduling-Mechanismus auf niederer Ebene (im Betriebssystemkern).
- Parameter können auf höherer Ebene (z.B. über Systemaufrufe in Benutzerprozessen) gesetzt werden, um applikationsbezogene Strategie zu implementieren.

⇒ Das Scheduling wird damit weiter auf niederer Ebene durchgeführt, aber von der Applikationsebene aus gesteuert.

- Beispiel: Ein Datenbank-Management-Prozess kann für seine Kindprozesse, die z.B. bestimmte ihm bekannte Anfragen bearbeiten oder interne Dienste durchführen, deren optimale Einplanung und Abfolge bewirken.

Moderne Forderungen (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

User-Level-Scheduler:

- Über die obige Forderung hinaus kann ein Applikationsprogramm dem Betriebssystem z.B. über einen Systemaufruf einen Scheduler übergeben, der für eine Teilmenge der Prozesse deren Scheduling durchführt (Höchstmaß an Flexibilität).
- Nur in wenigen Betriebssystemen vorhanden

4.2 Non-Preemptive Scheduling

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

(für Stapeljobs mit bekannten Bedienzeiten)

1. First-Come-First-Served (FCFS)
2. Shortest-Job-First (SJF)
3. Prioritäts-Scheduling (Prio)

4.2.1 First-Come-First-Served (FCFS)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

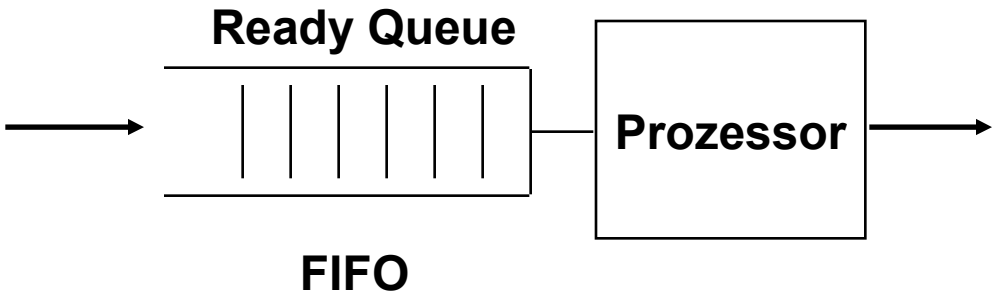
Algorithmus:

- Einfachst-Algorithmus: "Wer zuerst kommt, mahlt zuerst".
- Vollständige Bearbeitung jedes Auftrags, bevor ein neuer begonnen wird.

Implementierung:

- Die Ready-Queue wird als FIFO- Liste verwaltet.

Bedienungsmodell:



Rechenbeispiel

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Prozess	Bedienzeit
1	13
2	3
3	6

Alle Aufträge seien zur Zeit Null bekannt.

Resultierender Schedule:

t=0 13 16 22

Prozess	Wartezeit	Antwortzeit
1	0	13
2	13	16
3	13+3=16	22

Durchschnittliche Wartezeit:
(13+16)/3 = 29/3

Im Falle der Ausführungsfolge 3, 2, 1 hätte sich ergeben:
Durchschnittliche Wartezeit: (6+9)/3 = 5

4.2.2 Shortest-Job-First (SJF)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Algorithmus:

- Von allen rechenwilligen Prozessen wird derjenige mit der kleinsten Bedienzeitanforderung ausgewählt.
- Bei gleicher Bedienzeitanforderung wird nach FCFS gewählt.

- Der Algorithmus SJF ist in dem Sinne optimal in der Menge aller möglichen Algorithmen, dass er die kürzeste mittlere Wartezeit für alle Aufträge sichert.

- Notwendigkeit der Kenntnis der Bedienzeit!

Rechenbeispiel

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Prozess	Bedienzeit
1	13
2	3
3	6

Alle Aufträge seien zur Zeit Null bekannt.

Resultierender Schedule:

t=0 3 9 22

Prozess	Wartezeit	Antwortzeit
1	3+6=9	22
2	0	3
3	3	9

Durchschnittliche Wartezeit: (9+3)/3 = 4

4.2.3. Prioritäts-Scheduling (Prio)

Algorithmus:

- Jeder Auftrag besitze eine statische Priorität.
- Von allen rechenwilligen Prozessen wird derjenige mit der höchsten Priorität ausgewählt.
- Bei gleicher Priorität wird nach FCFS ausgewählt.

Rechenbeispiel:

Prozess	Bedienzeit	Priorität
1	13	2
2	3	3
3	6	4

Alle Aufträge seien zur Zeit Null bekannt.

Resultierender Schedule:

t=0 6 9 22

4.3 Preemptive Scheduling

(realististisch für heutige Rechensysteme)

1. Round-Robin-Scheduling (RR)
2. Dynamisches Prioritäts-Scheduling
3. Mehrschlangen-Scheduling
4. Mehrschlangen-Feedback-Scheduling
5. Earliest-Deadline-First-Scheduling

4.3.1 Round-Robin-Scheduling (RR)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Algorithmus:

- Menge der rechenwilligen Prozesse linear geordnet.
- Jeder rechenwillige Prozess erhält den Prozessor für eine feste Zeitdauer q, die Zeitscheibe (time slice) oder Quantum genannt wird.
- Nach Ablauf des Quantums wird der Prozessor entzogen und dem nächsten zugeordnet (preemptive-resume).
- Tritt vor Ende des Quantums Blockierung oder Prozessende ein, erfolgt der Prozesswechsel sofort.
- Dynamisch eintreffende Aufträge werden z.B. am Ende der Warteschlange eingefügt.

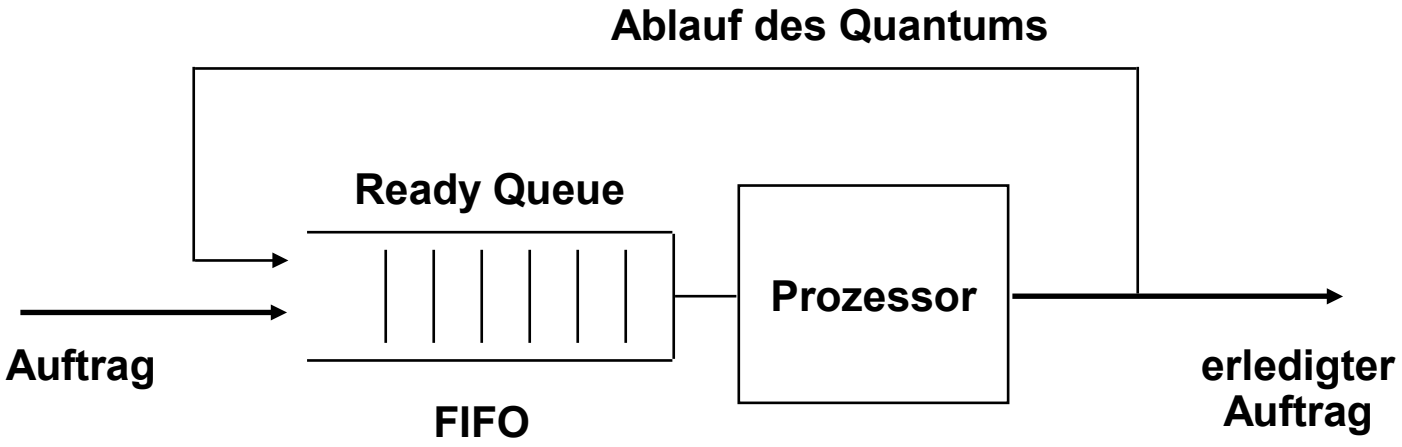
Implementierung:

- Die Zeitscheibe wird durch einen Uhr-Interrupt realisiert.
- Die Ready Queue wird als lineare Liste verwaltet, bei Ende eines Quantums wird der Prozess am Ende der Ready Queue eingefügt.

Round-Robin (RR) (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Bedienungsmodell:



Bewertung:

- Round-Robin ist einfach und weit verbreitet.
- Alle Prozesse werden als gleichwichtig angenommen und fair bedient.
- Einziger kritischer Punkt: Wahl der Dauer des Quantums.
  - Quantum zu klein ⇒ häufige Prozesswechsel, sinnvolle Prozessornutzung sinkt
  - Quantum zu groß ⇒ schlechte Antwortzeiten bei kurzen interaktiven Aufträgen.

Rechenbeispiel

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Prozess	Bedienzeit
1	13
2	3
3	6

Alle Aufträge seien zur Zeit Null bekannt.

Quantum sei q = 4.

Resultierender Schedule:

t=0 4 7 11 15 17 21 22

Prozess	Wartezeit	Antwortzeit
1	3+4+2= 9	22
2	4	7
3	4+3+4=11	17

Durchschnittliche Wartezeit: (9+4+11)/3 = 8

Grenzwertbetrachtung für Quantum q:

- q → unendlich:

  Round-Robin verhält sich wie FCFS.

- q → 0:

  Round-Robin führt zu sogenanntem processor-sharing:
  jeder der n rechenwilligen Prozesse erfährt 1/n der Prozessorleistung. (Kontextwechselzeiten als Null angenommen).

4.3.2 Unterbrechendes Prio-Sched.

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Algorithmus:

- Jedem Prozess ist ein statische Priorität zugeordnet.
- Prozesse werden gemäß ihrer Priorität in eine Warteschlange eingereiht.
- Es wird jeweils der rechenwillige Prozess mit der höchsten Priorität ausgewählt und bedient.
- Wird ein Prozess höherer Priorität rechenwillig (z.B. nach Beendigung einer Blockierung), so wird der laufende Prozess unterbrochen (preemption) und in die Ready Queue eingefügt.

4.3.3 Mehrschlangen-Scheduling

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

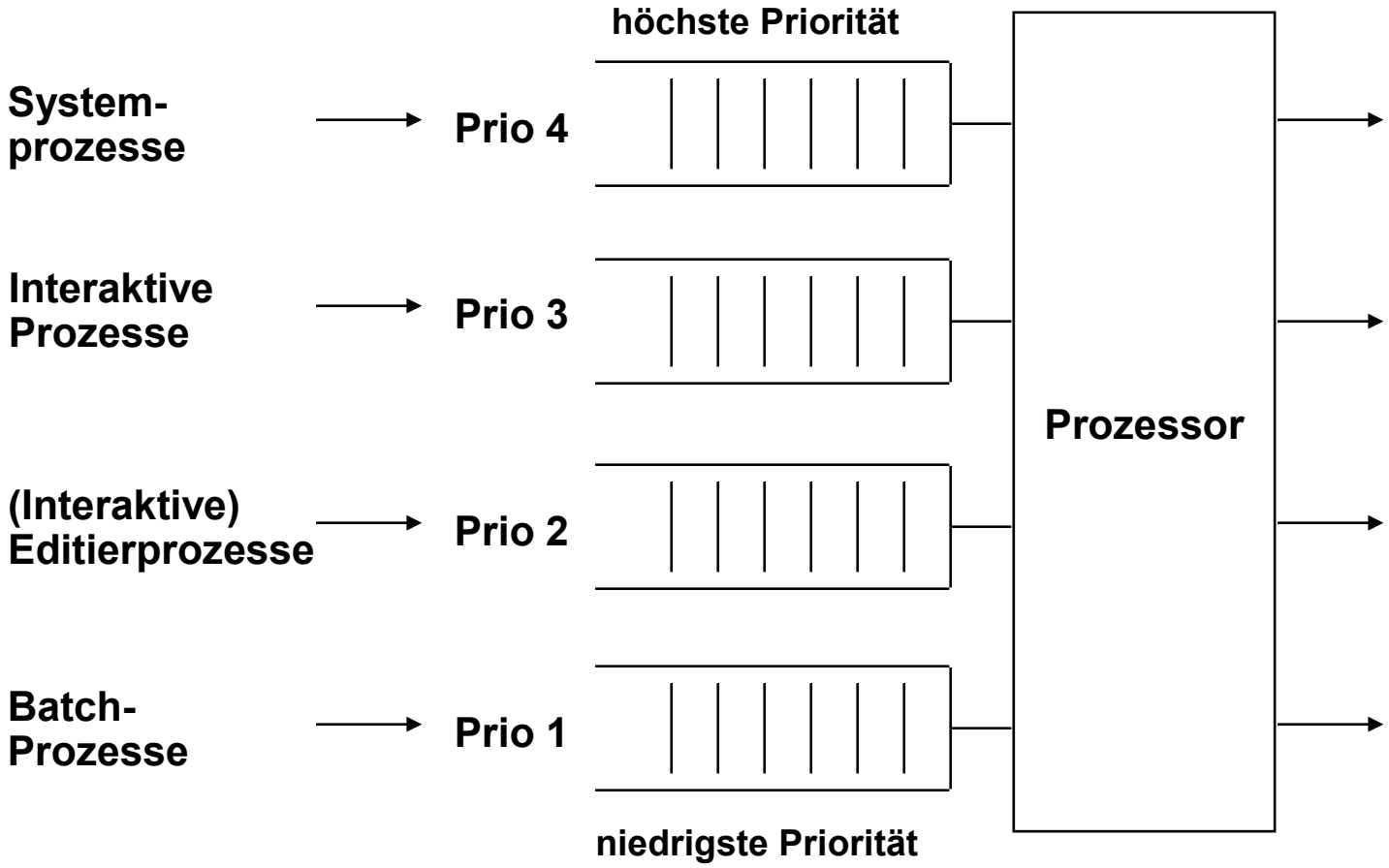
Algorithmus:

- Prozesse werden statisch klassifiziert als einer bestimmten Gruppe zugehörig (z.B. interaktiv, batch).
- Alle rechenwilligen Prozesse einer bestimmten Klasse werden in einer eigenen Ready Queue verwaltet.
- Jede Ready Queue kann ihr eigenes Scheduling-Verfahren haben (z.B. Round-Robin für interaktive Prozesse, FCFS für batch-Prozesse).
- Zwischen den Ready Queues wird i.d.R. unterbrechendes Prioritäts-Scheduling angewendet, d.h.: jede Ready Queue besitzt eine feste Priorität im Verhältnis zu den anderen; wird ein Prozess höherer Priorität rechenwillig, wird der laufende Prozess unterbrochen (preemption).

Mehrschlangen-Scheduling (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Bedienungsmodell (Beispiel):



4.3.4 Mehrschlangen-Feedback-Scheduler

Prinzip:

- Erweiterung des Mehrschlangen-Scheduling.
- Rechenwillige Prozesse können im Verlauf in verschiedene Warteschlangen eingeordnet werden (dynamische Prioritäten).
- Algorithmen zur Neubestimmung der Priorität wesentlich

  Beispiel 1: Wenn ein Prozess blockiert, wird die Priorität nach Ende der Blockierung um so größer, je weniger er von seinem Quantum verbraucht hat (Bevorzugung von I/O-intensiven Prozessen).

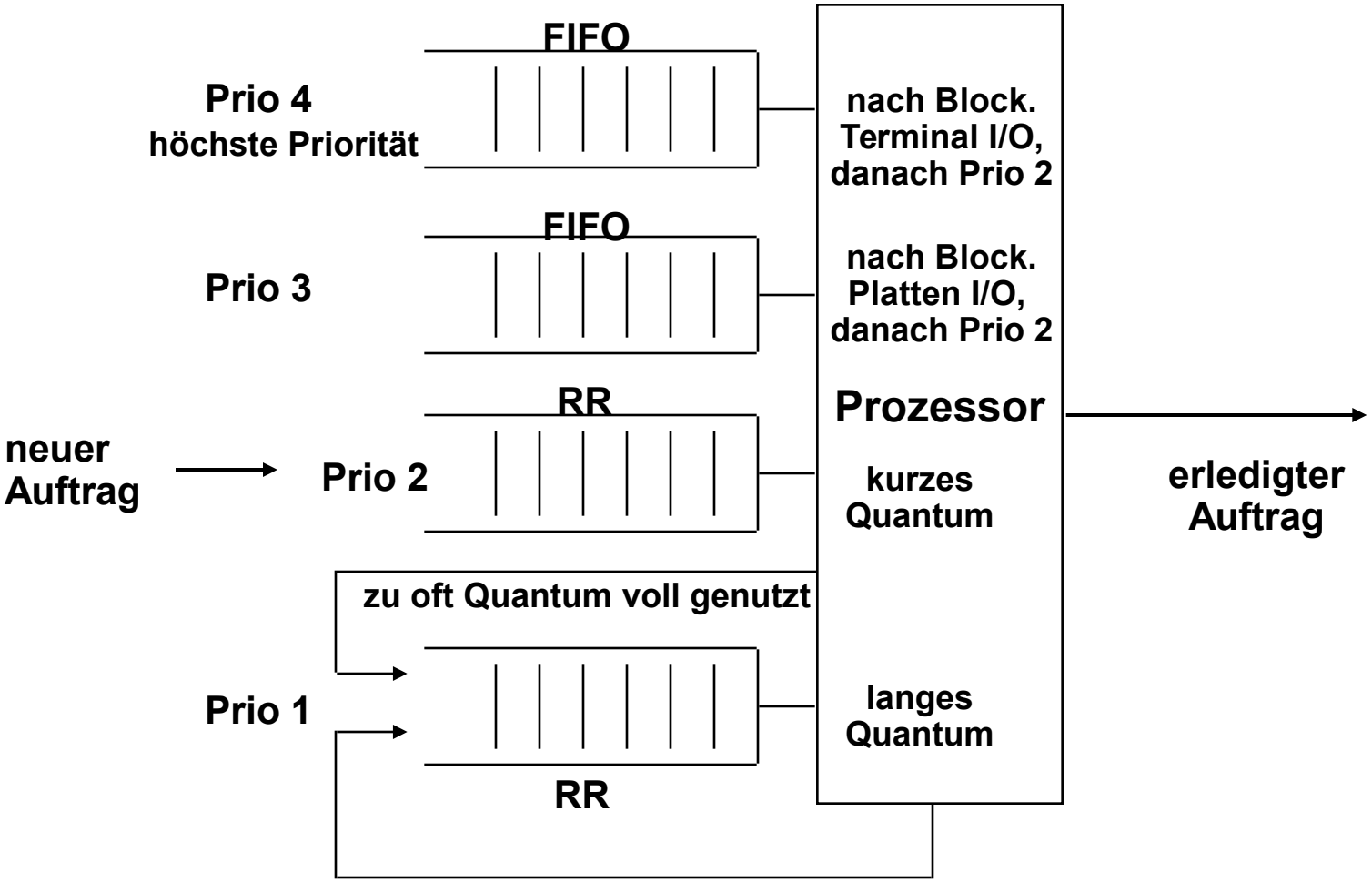
  Beispiel 2: Wenn ein Prozess in einer bestimmten Priorität viel Rechenzeit zugeordnet bekommen hat, wird seine Priorität verschlechtert (Bestrafung von Langläufern).

  Beispiel 3: Wenn ein Prozess lange nicht bedient worden ist, wird seine Priorität verbessert (Altern, Vermeidung einer "ewigen" Bestrafung).

Beispiel

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Bedienungsmodell:



Bewertung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

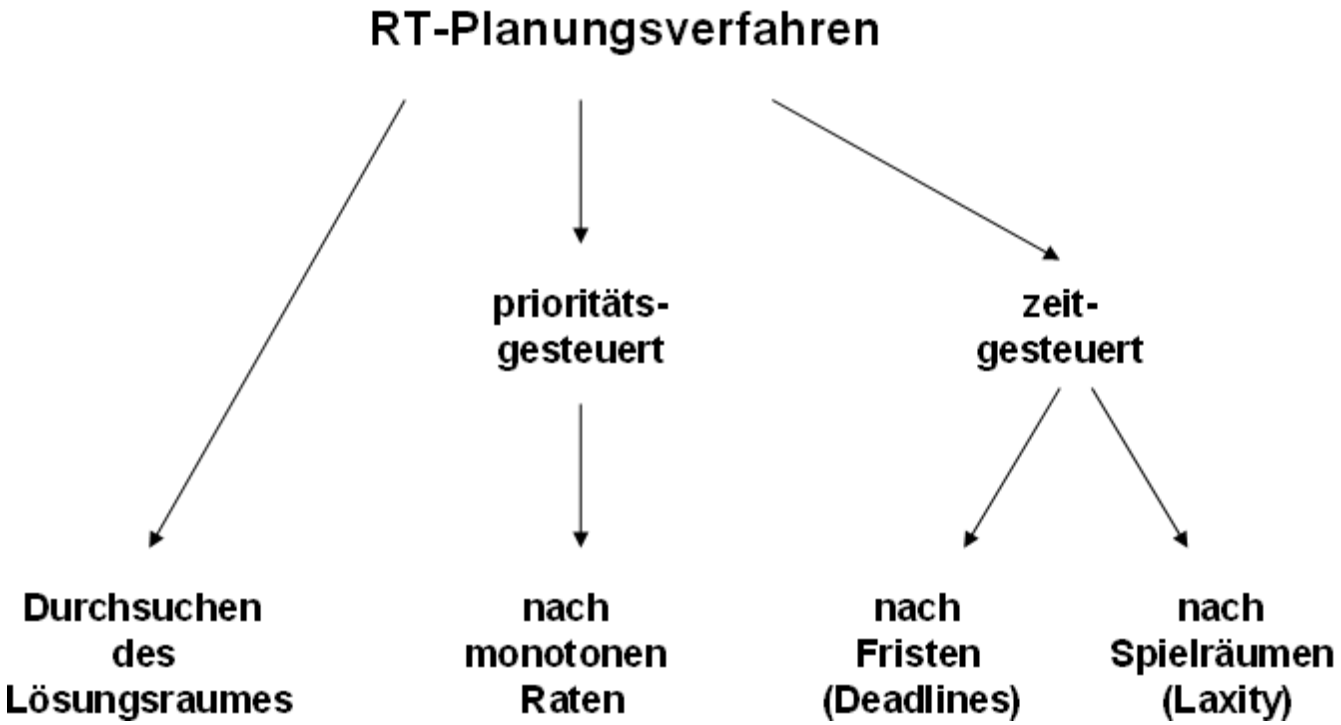
- Mit wachsender Bedienzeit sinkt die Priorität, d.h. Kurzläufer werden bevorzugt, Langläufer werden zurückgesetzt.
- Wachsende Länge des Quantums mit fallender Priorität verringert die Anzahl der notwendigen Prozesswechsel (Einsparen von Overhead).
- Verbesserung der Priorität nach Beendigung einer Blockierung berücksichtigt I/O-Verhalten (Bevorzugung von I/O-intensiven Prozessen). Durch Unterscheidung von Terminal I/O und sonstigem I/O können interaktive Prozesse weiter bevorzugt werden.
- sehr flexibel.

4.3.5 Echtzeit-Scheduling

- Scheduling in Realzeit-Systemen beinhaltet zahlreiche neue Aspekte. Hier nur erster kleiner Einblick.
- Varianten in der Vorgehensweise
  - *Statisches Scheduling:*
    Alle Daten für die Planung sind vorab bekannt, die Planung erfolgt durch eine Offline-Analyse.
  - *Dynamisches Scheduling:*
    Daten für die Planung fallen zur Laufzeit an und müssen zur Laufzeit verarbeitet werden.
  - *Explizite Planung:*
    Dem Rechensystem wird ein vollständiger Ausführungsplan (Schedule) übergeben und zur Laufzeit befolgt (Umfang kann extrem groß werden).
  - *Implizite Planung:*
    Dem Rechensystem werden nur die Planungsregeln übergeben.

Klassifizierung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



Beispiel: Earliest-Deadline-First (EDF)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Gewisse Prozesse müssen häufig zyklisch ausgeführt werden.
- Hard-Realtime-Prozesse müssen unter allen Umständen ausgeführt werden (ansonsten sind z.B. Menschenleben bedroht).
- Zeitliche Fristen (Deadlines) vorgegeben, zu denen der Auftrag erledigt sein muss.
- Scheduler muss die Erledigung aller Hard-Realtime-Prozesse innerhalb der Fristen garantieren.
- Scheduling geschieht in manchen Anwendungssystemen statisch vor Beginn der Laufzeit (z.B. Automotive). Dazu muss die Bedienzeit-Anforderung (z.B. worst case) bekannt sein.
- Im Falle von dynamischem Scheduling ist das Earliest-Deadline-First-Scheduling-Verfahren verbreitet, das den Prozess mit der frühesten fälligen Frist zur Ausführung auswählt.

4.4 Scheduling in UNIX

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Ziel:

- Gute Antwortzeiten für interaktive Prozesse.

Vorgehensweise:

- Zweistufiges Scheduling:
  1. Hintergrundspeicher-Hauptspeicher-Swapping.
  2. Scheduling von im Hauptspeicher befindlichen Prozessen.

Anmerkungen:

- Im folgenden nur die untere (2.) Scheduling-Stufe betrachtet.
- UNIX-Varianten unterscheiden sich bzgl. des Scheduling. Im folgenden wird UNIX System V Rel. 2 betrachtet
  (relativ einfach, vgl. [Bach]).

Mehrschlangen-Feedback-Scheduling:

- Rechenwillige Prozesse werden in Ready Queues gehalten. Blockierte Prozesse werden außerhalb geführt.
- Jede Ready Queue besitzt eine Priorität. Prioritäten sind ganze Zahlen. Je kleiner (negativer) der Wert, je vorrangiger der Prozess.
- Prozesse im Kernmodus haben negative Prioritäten, Prozesse im Benutzermodus haben nicht-negative Prioritäten.
- Jeder Benutzerprozess hat Basispriorität (bestenfalls 0). Die Basispriorität kann mittels `nice` verschlechtert werden ("be nice to *others*", nur super-user kann verbessern).
- Ein ausgewählter Prozess wird für die Dauer eines Quantums ausgeführt, typ. 100 msec (4-6 Ticks der Systemuhr), wenn er nicht vorher blockiert.
- Am Ende des Quantums wird ein Benutzerprozess an das Ende derselben Ready Queue zurückgestellt, d.h. Round-Robin auf jeder Benutzer-Prioritätsebene.

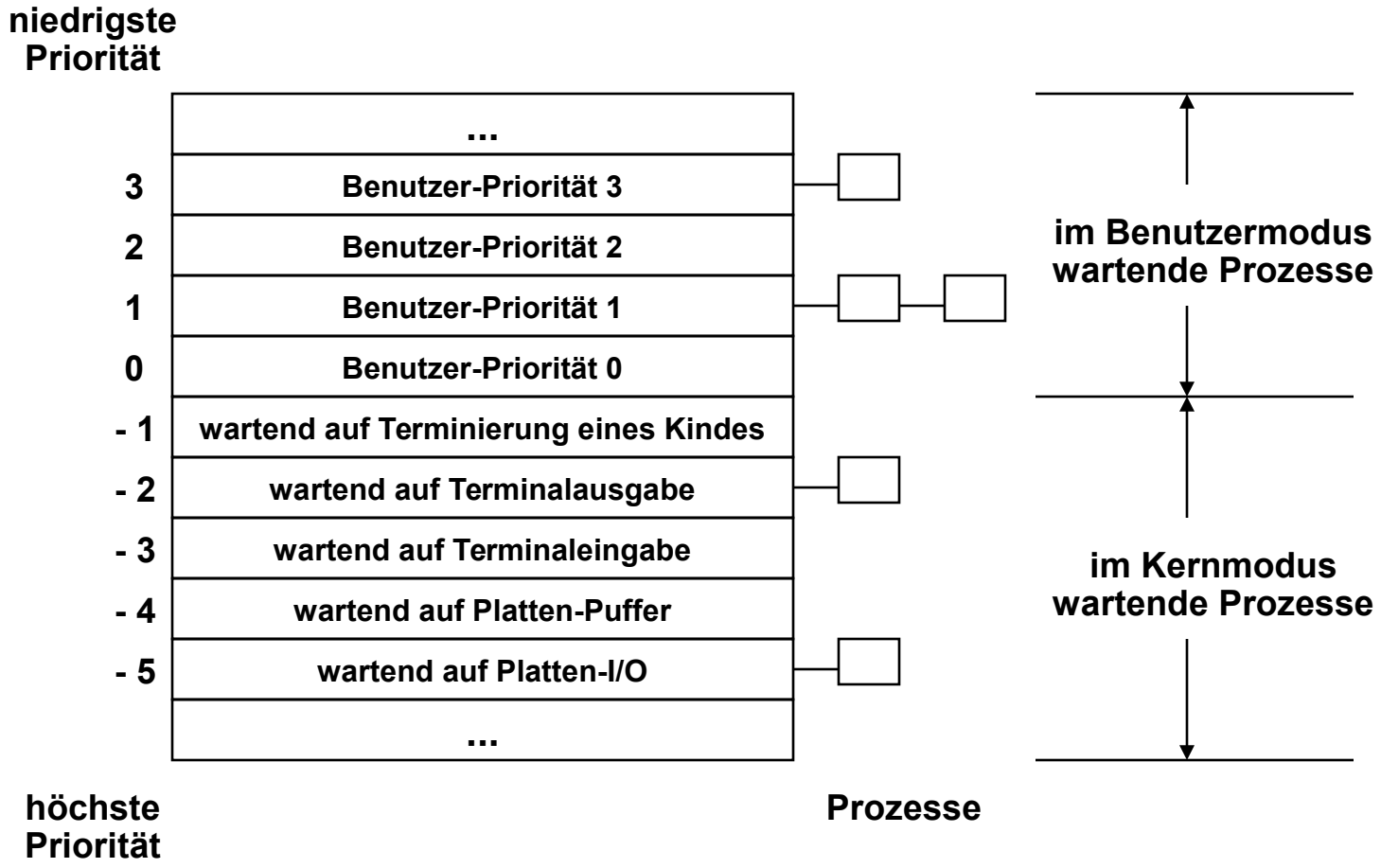
Feedback-Verfahren (vereinfacht):

- Bei jedem Uhr-Tick wird CPU-Nutzungszähler des aktuellen Prozesses inkrementiert.
- CPU-Nutzungszähler wird auf die Basispriorität das Prozesses addiert (d.h. die Priorität wird schlechter, und der Prozess wird in eine nieder priore Ready Queue verdrängt).
- Einmal je Sekunde wird aktuelle Priorität jedes Prozesses neu berechnet.
  - CPU-Nutzungszähler durch 2 dividieren (Alterungsmech. zum langsamen Vergessen der vergangenen CPU-Nutzung).
  - neue Priorität = Basispriorität + CPU-Nutzungszähler.
- Benutzerprozess, der im Verlauf eines Kernaufrufs im Kern blockiert, erhält, abhängig von der Art der Ereignisses, auf das er wartet, eine Kernpriorität zugeordnet, gemäß der er eingeordnet wird, wenn er wieder rechenwillig wird. Ziel: Schnelles Verlassen des Kerns, Ordnen der Ereignisse nach Wichtigkeit).

Scheduling in UNIX (4)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Resultierende Warteschlangenstruktur:



Scheduling in Linux

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Linux 1.2
  - Zyklische Liste, Round-Robin
- Linux 2.2
  - Scheduling-Klassen (Echtzeit, Non-Preemptive, Nicht-Echtzeit)
  - Unterstützung für Multiprozessoren
- Linux 2.4
  - O(n)-Komplexität (jeder Task-Kontrollblock muss angefasst werden)
  - Round-Robin
  - Teilweiser Ausgleich bei nicht verbrauchter Zeitscheibe
  - Insgesamt relativ schwacher Algorithmus

Scheduling in Linux (2)

- Linux 2.6
  - O(1)-Komplexität (konstanter Aufwand für Auswahl unabhängig von Anzahl Tasks)
  - Run Queue je Priorität
  - Zahlreiche Heuristiken für Entscheidung I/O-intensiv oder rechenintensiv
  - Sehr viel Code
- Ab Linux Kernel 2.6.23: Completely Fair Scheduler (CFS)
  - Sehr gute Approximation von Processor Sharing
  - Task mit geringster *Virtual Runtime* (größter Rückstand) bekommt Prozessor
  - Zeit-geordnete spezielle Baumstruktur für Taskverwaltung ($\rightarrow$ O(log n)-Komplexität)
  - Kein periodischer Timer-Interrupt sondern One-Shot-Timer

4.5 Zusammenfassung

Was haben wir in Kap. 4 gemacht?

- Scheduling-Verfahren
  - Kenngrößen wie Antwortzeit, Auslastung und Fairness relevant
  - einfache Verfahren: FCFS, SJF, PRIO
  - unterbrechende Verfahren Round-Robin, statische Prioritäten, Mehrschlangen-Verfahren
  - flexible Feedback-Algorithmen, nach denen Prioritäten dynamisch neu berechnet werden
  - Beispiel UNIX/Linux