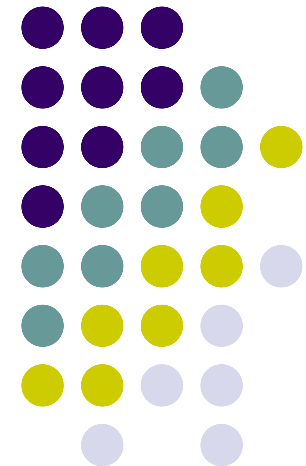
Systemsoftware 7

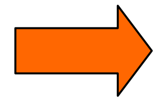
Scheduling II



hochschule mannheim

Prof. Dr. M. Föller
Fakultät Informatik

Echtzeit-Scheduling



1. Echtzeit-Scheduling
2. Scheduling in Linux
3. Scheduling in Windows 2000

Beispiele für Echtzeitsysteme

- Prozesssteuerungen für Produktionsanlagen
- Kontrolle von Laborexperimenten
- Roboter
- Überwachung des Flugverkehrs
- Telekommunikation
- Medizintechnik
- Kfz-Technik
- ....

Echtzeitsysteme

- **Nicht-Echtzeitsystem**:
  - logische Korrektheit

- **Echtzeitsystem**:
  - logische Korrektheit +
  - zeitliche Korrektheit

- Ein Ergebnis ist nur korrekt, wenn es logisch korrekt ist und zur rechten Zeit zur Verfügung steht!

Echtzeittasks

- Harter Echzeit-Task: das Nichterfüllen einer zeitlichen Beschränkung hat fatale Folgen (ABS, Airbag, Werkzeugmaschinen)
- Weicher Echtzeit-Task: Einhalten zeitlicher Bedingung, deren Verletzung allenfalls eine verminderte Nutzung bedeutet (Voice over IP, Audio/Video-Streaming)
- Aperiodischer Task: Startzeitpunkt oder Terminierungszeitpunkt ist definiert
- Periodischer Task: Anforderung: "einmal pro Zeitraum T"

Features von Echtzeitbetriebssystemen

- Schneller Prozess- bzw. Thread-Wechsel
- Geringe Größe
- Schnelle Reaktion auf Unterbrechungen
- Interprozesskommunikation (Semaphore, Signale, Ereignisse)
- Scheduling mit Vorrangunterbrechung (Prioritäten)
- Minimierung von Zeitintervallen in denen Interrupts gesperrt sind
- Funktionen zur Verzögerung von Tasks für eine feste Zeitdauer und zum Anhalten/Wiederaufnehmen von Tasks
- Spezielle Warn- und Time-Out-Meldungen
- Zentrales Element: *Kurzfristiger Task-Scheduler*

Kurzfristiges Scheduling in Echtzeit-BTS

- Wichtig:
  - Alle harten Echtzeit-Tasks in vorgegebenen Fristen abarbeiten
  - So viele weiche Echtzeit-Tasks wie möglich in vorgegebener Frist abarbeiten
- Untergeordnet:
  - Fairness
  - Minimierung der mittleren Antwortzeit
  - Optimierung der Prozessorauslastung
- Problem: Viele RT-OS können mit Fristen nicht direkt umgehen

  => OS muss so konzipiert sein, dass es auf Echtzeit-Tasks *so schnell wie möglich* reagiert

Rückblick: Scheduling-Strategien

FCFS (First Come First Served)	$\max(w)$	nicht unterbrechend
Round Robin	C	unterbrechend
SPN (Shortest Process Next)	$\min(s)$	nicht unterbrechend
SRT (Shortest Remaining Time)	$\min(s-e)$	unterbrechend
HRRN (Highest Response Ratio Next)	$\max \frac{w+s}{s}$	nicht unterbrechend
Feedback	s.u.	unterbrechend

w = bis dahin im System verbrachte Zeit (wartend und in Auführung

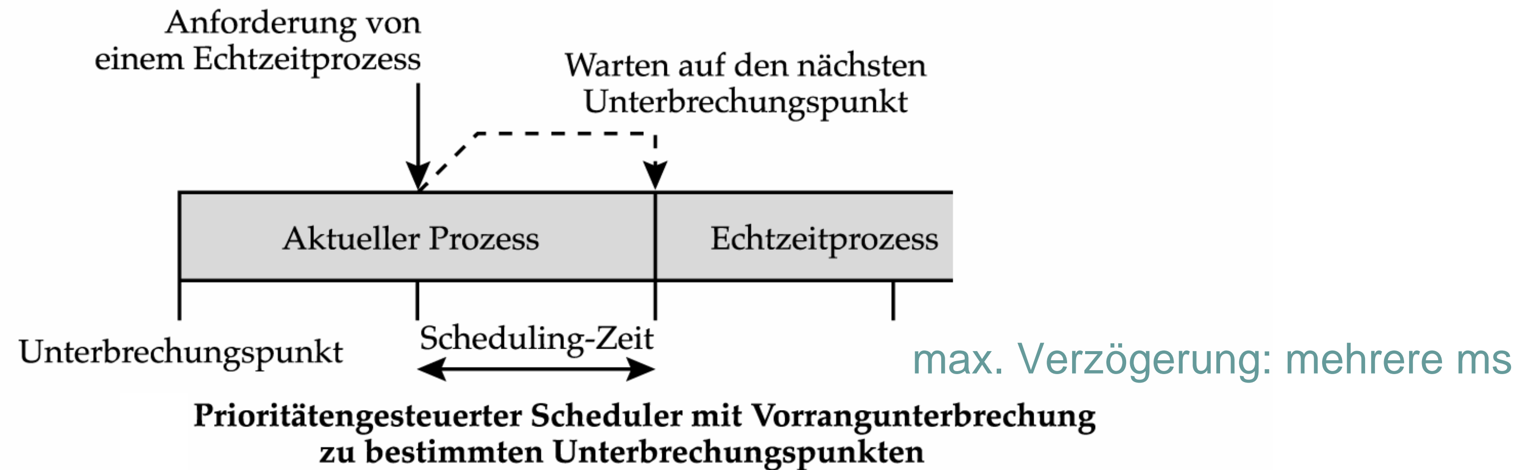
e = in Ausführung verbrachte Zeit

s = gesamte vom Prozess benötigte Ausführungszeit

C = Konstante

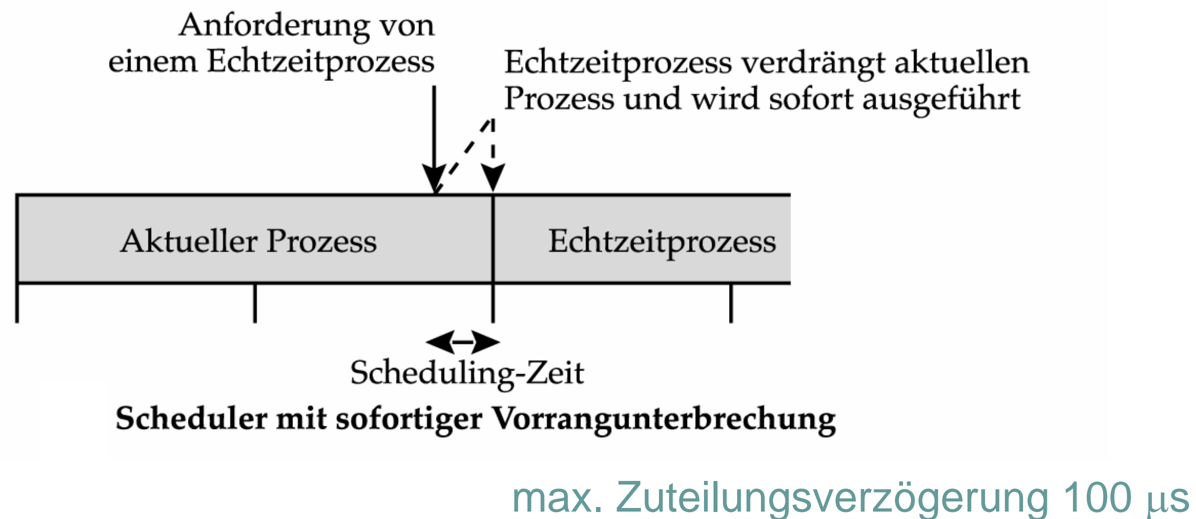
Ansätze fürs Echtzeit-Scheduling

(1)



max. Verzögerung: mehrere ms

(2)



max. Zuteilungsverzögerung 100 μs

Klassen von Echtzeit-Scheduling

Statische Verfahren

- Tabellengesteuerte Strategien
  - Bei periodischen Tasks
  - Vorhersagbar, unflexibel
  - statische Analyse der Durchführbarkeit von Scheduling-Plänen
  - ⇒ Plan der zur Laufzeit festlegt, wann welcher Task beginnt

- Prioritätengesteuerte Strategien
  - mit Vorrangunterbrechung
  - statische Analyse ohne Erstellung eines Plans
  - Analyse wird für Zuweisung von Prioritäten verwendet
  - ⇒ *RMS(Rate Monotonic Scheduling)*

Klassen von Echtzeit-Scheduling

Dynamische Verfahren

- Planungsgesteuerte Strategien
  - Durchführbarkeit wird zur Laufzeit festgestellt
  - Ankommender Task wird nur dann zugelassen, wenn Zeitbeschränkungen aller Tasks weiterhin eingehalten werden können
- Best-Effort-Strategien
  - Wird von den meisten heutigen RT-OS angewendet
  - Aperiodische Tasks
  - Bei Ankunft eines Tasks weißt ihm OS anhand seiner Eigenschaften eine Priorität zu
  - Ob Frist eingehalten wurde ist erst nach Ablauf der Frist bekannt

  ⇒ *EDF-Scheduling (Earliest Deadline First)*

Deadline Scheduling

- Benötigte Zusatzinformationen:
  - Bereitzeit
  - Startzeit
  - Abschlusszeit
  - Verarbeitungszeit
  - Ressourcenanforderungen
  - Priorität (absolute/relative)
  - Subtaskstruktur
- Fragen:
  - welcher Task wird als nächstes eingeplant
  - welche Art der Vorrangunterbrechung wird ausgeführt

Earliest Deadline First (EDF)

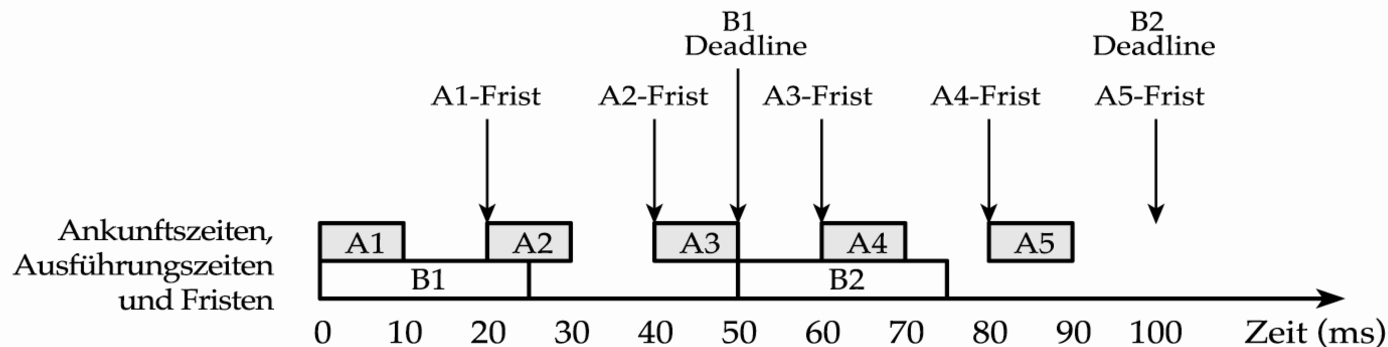
- Strategie mit:
  - Verwendung von Start- oder Abschlusszeiten
- Beste Methode für diese Strategien, da
- Anzahl der Tasks für die Fristen nicht eingehalten werden minimiert ist
- Bei Verwendungn von Abschlusszeiten:
  - Mit Vorrangunterbrechung
  - Aktiver Task X kann zugunsten Task Y unterbrochen werden, um Fristen einzuhalten
- Bei Verwendung von Startzeiten:
  - Scheduler ohne Vorrangunterbrechung
  - Task kann sich nach Ausführung des obligatorischen Teils selbst blockieren

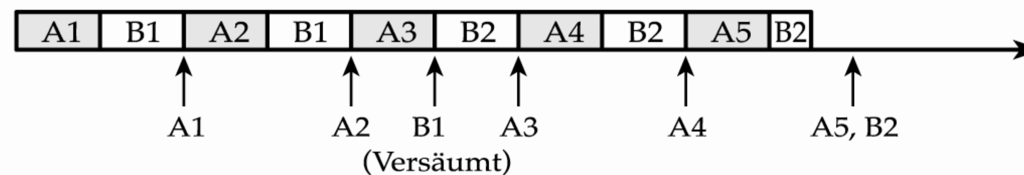
Periodische Echtzeit-Tasks mit Abschlusszeiten

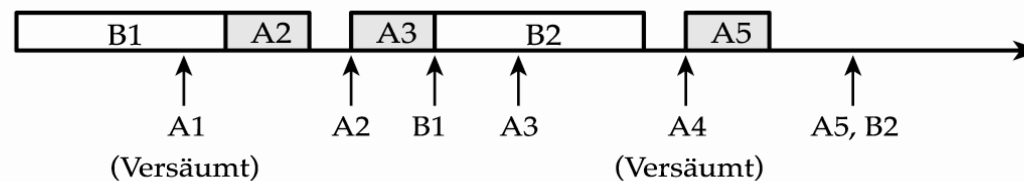
- System verarbeitet Daten von zwei Sensoren (A und B)
- Datenerfassung von A muss alle 20 ms erfolgt sein, Dauer 10 ms
- Datenerfassung von B muss alle 50 ms erfolgt sein, Dauer 25 ms

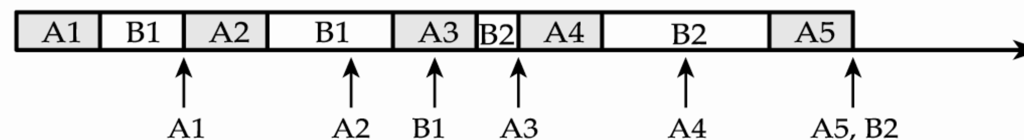
Prozess	Ankunftzeit	Ausführungszeit	Abschlusszeit
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
...	...	...	...
B(1)	0	25	50
B(2)	50	25	100
B(3)	100	25	150
...	...	...	...

Periodische Echtzeit-Tasks mit Abschlusszeiten







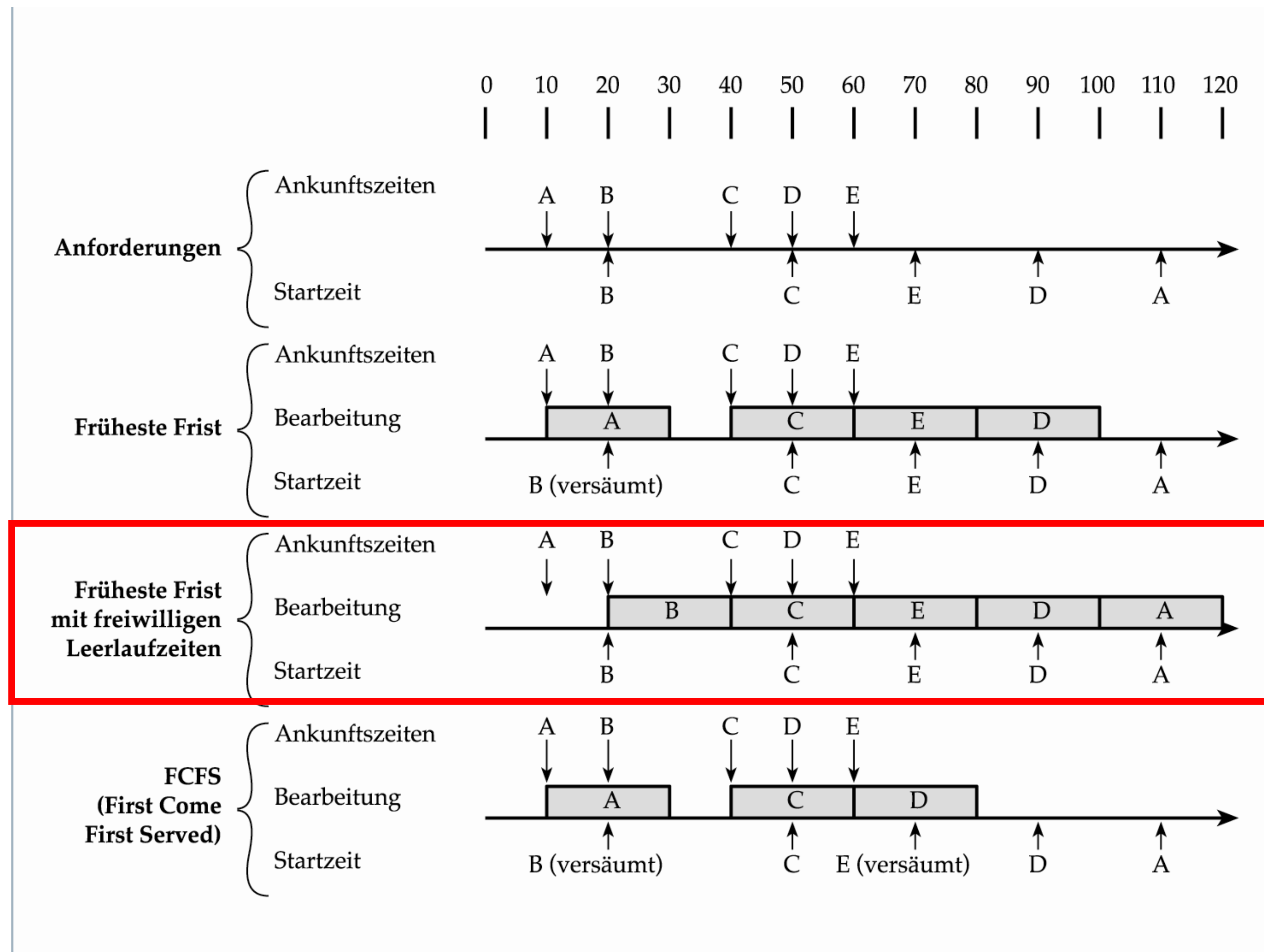


Aperiodische Echtzeit-Tasks mit Startzeiten

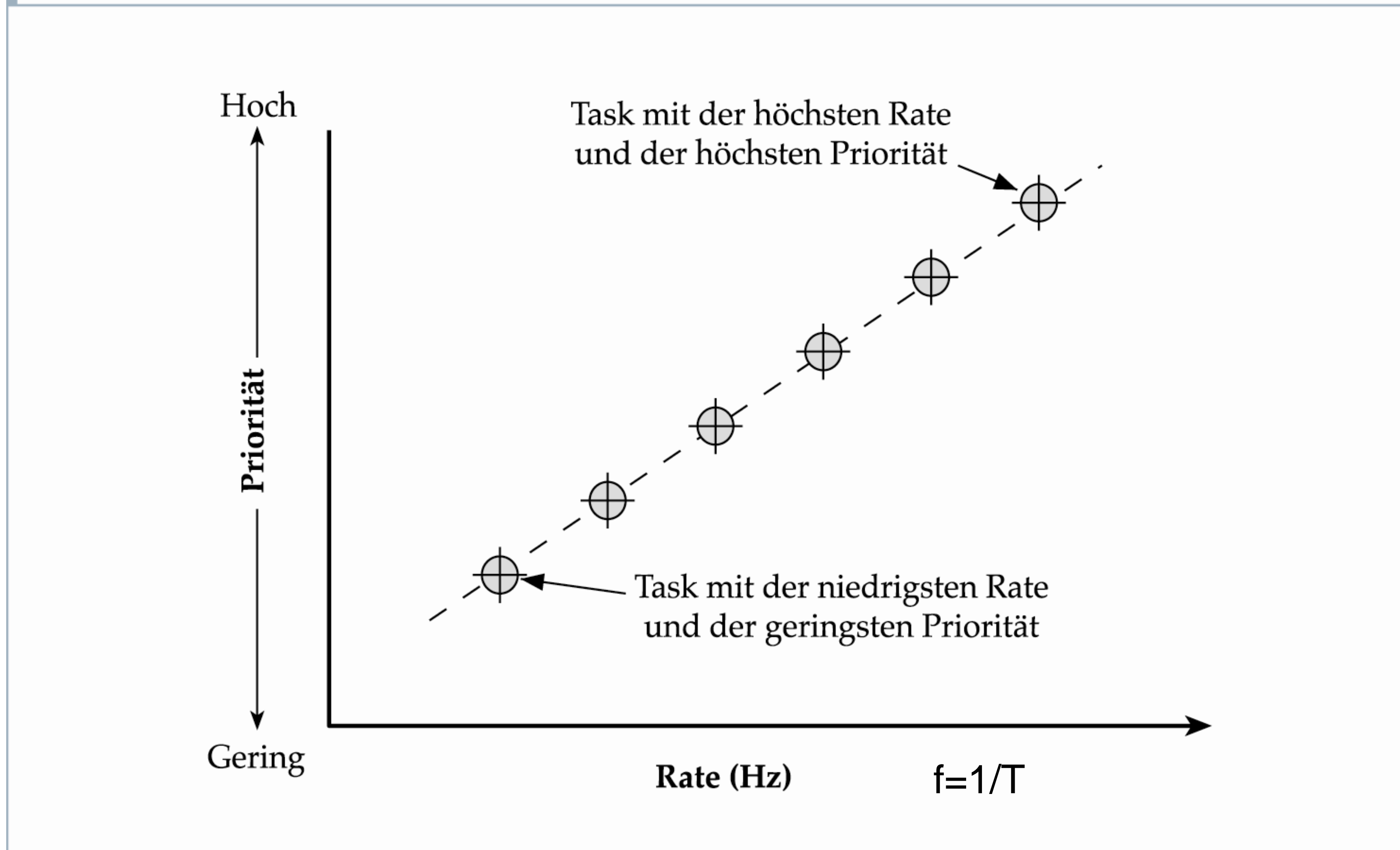
- 5 Tasks, jeweils 20 ms Ausführungszeit

Prozess	Ankunftzeit	Ausführungszeit	Startzeit
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

Aperiodische Echtzeit-Tasks mit Startzeiten



Task-Menge mit RMS

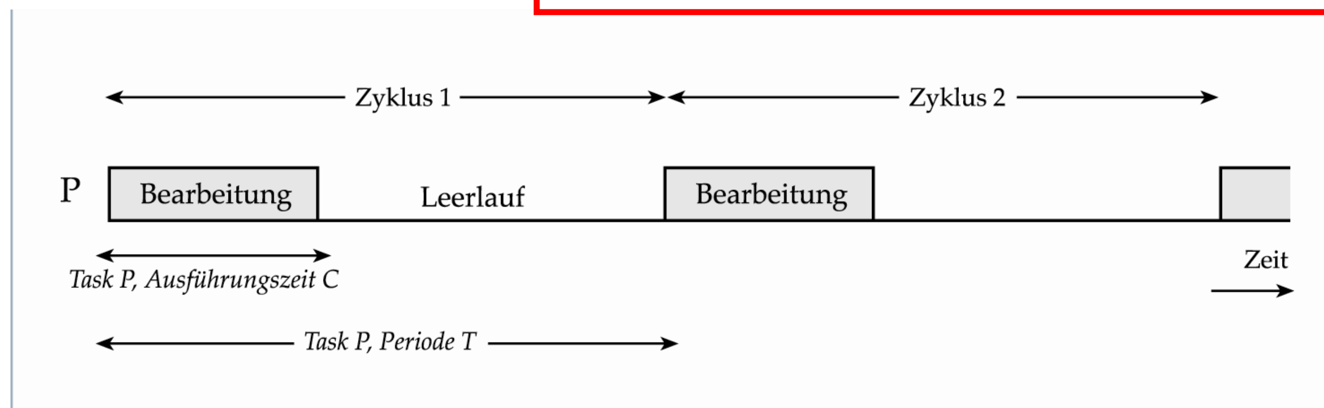


Rate Monotonic Scheduling (RMS)

- Lösen von Zuteilungskonflikten in Multitasking-Systemen mit periodischen Tasks
- Zuweisung von Prioritäten nach den Perioden der Tasks
- Kürzeste Periode ⇔ höchste Priorität
- Es gilt allgemein:

$$\text{Auslastung}: U = \frac{C}{T} \quad \text{mit} \quad C \leq T$$

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \leq 1 \; \text{(idealer Wert)}$$



Rate Monotonic Scheduling (RMS)

- Für RMS gilt:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \leq n\left(2^{1/n} - 1\right)$$

- Beispiel: 3 Tasks

$$\text{Task } P_1 : C_1 = 20; T_1 = 100 \Rightarrow U_1 = 0{,}2$$

$$\text{Task } P_2 : C_2 = 40; T_2 = 150 \Rightarrow U_2 = 0{,}267$$

$$\text{Task } P_3 : C_3 = 100; T_3 = 350 \Rightarrow U_3 = 0{,}286$$

$$\Rightarrow \text{Gesamtnutzung}: 0{,}2 + 0{,}267 + 0{,}286 = 0{,}753$$

- Zuteilung erfolgreich, da:

$$0{,}753 \leq 3\left(2^{1/3} - 1\right) = 0{,}779$$

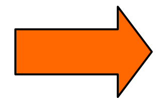
Vergleich RMS ⇔ EDF

- RMS: $$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \leq \mathrm{n}\left(2^{1/\mathrm{n}} - 1\right)$$
- EDF: $$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \leq 1$$

- Bessere Gesamtauslastung bei EDF-Zuteilung
- Dennoch hat sich RMS für industrielle Anw. durchgesetzt
  - Leistungsunterschied in Praxis gering
  - Harte Echzeitsysteme haben auch weiche Echtzeittasks, die Porzessorzeit ohne RMS nutzen können
  - Stabilität mit RMS besser
  - Bei EDF ändert sich Task-Priorität teilw. von einer Periode zur nächsten => Vorhersagbarkeit eingeschränkt

Echtzeit-Scheduling

1. Echtzeit-Scheduling



2. Scheduling in Linux

3. Scheduling in Windows 2000

Traditionelles UNIX-Scheduling

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Basis_j + \frac{CPU_j(i-1)}{4} + nice_j$$

mit

$CPU_j(i)$: Maß der Prozessorauslastung durch Prozess j im Intervall i

$P_j(i)$: Priorität von Prozess j zu Beginn von Intervall i

$Basis_j$: Basispriorität von Prozess j

$nice_j$: durch den Benutzer kontrollierbarer Regulierungsfaktor

Neuberechnung der Prozesspriorität einmal pro Sekunde

Beispiel UNIX Scheduler

Zeit	Prozess A		Prozess B		Prozess C	
	Priorität	CPU-Zähler	Priorität	CPU-Zähler	Priorität	CPU-Zähler
0	60	0 1 2 • • 60	60	0	60	0
1	75	30	60	0 1 2 • • 60	60	0
2	67	15	75	30	60	0 1 2 • • 60
3	63	7 8 9 • • 67	67	15	75	30
4	76	33	63	7 8 9 • • 67	67	15
5	68	16	76	33	63	7

- Prozessorinterrupt 60x pro Sekunde
- Nach jeder Sekunde Neuberechnung der Priorität

Scheduling in Linux

- Traditionelles UNIX-Scheduling
- Drei Klassen Zuteilungsklassen
  - SCHED_FIFO:FIFO-Echtzeit-Threads
  - SCHED_RR: RR-Echtzeit-Threads
  - SCHED_OTHER: Andere Threads ohne Echtzeitanforderung
- Innerhalb jeder Klasse können Prioritäten vergeben werden
- Prioritäten der Echtzeitklassen sind höher als der Klasse SCHED_OTHER

FIFO und RR-Echtzeitklassen

- FIFO-Klasse
  - Das System unterbricht aktiven FIFO-Thread nur wenn:
    - FIFO-Thread höherer Priorität in Zustand bereit versetzt
    - Moment aktive FIFO-Thread in Zustand blockiert versetzt wird
    - Moment aktive FIFO-Thread Prozessor "freiwillig" frei gibt durch Aufruf von *sched-yield*.
  - Unterbrochener Prozess wird Priorität entsprechend in Warteschlange abgelegt
  - Bei Threads gleicher Priorität wird Thread mit längerer Wartezeit ausgewählt.
- RR-Klasse
  - Mit jedem Thread ist ein Zeitquantum verknüpft
  - Nach Ablauf des Zeitquantums: Suspendierung des Threads, anderer Echtzeitthread wird zur Ausführung gebracht

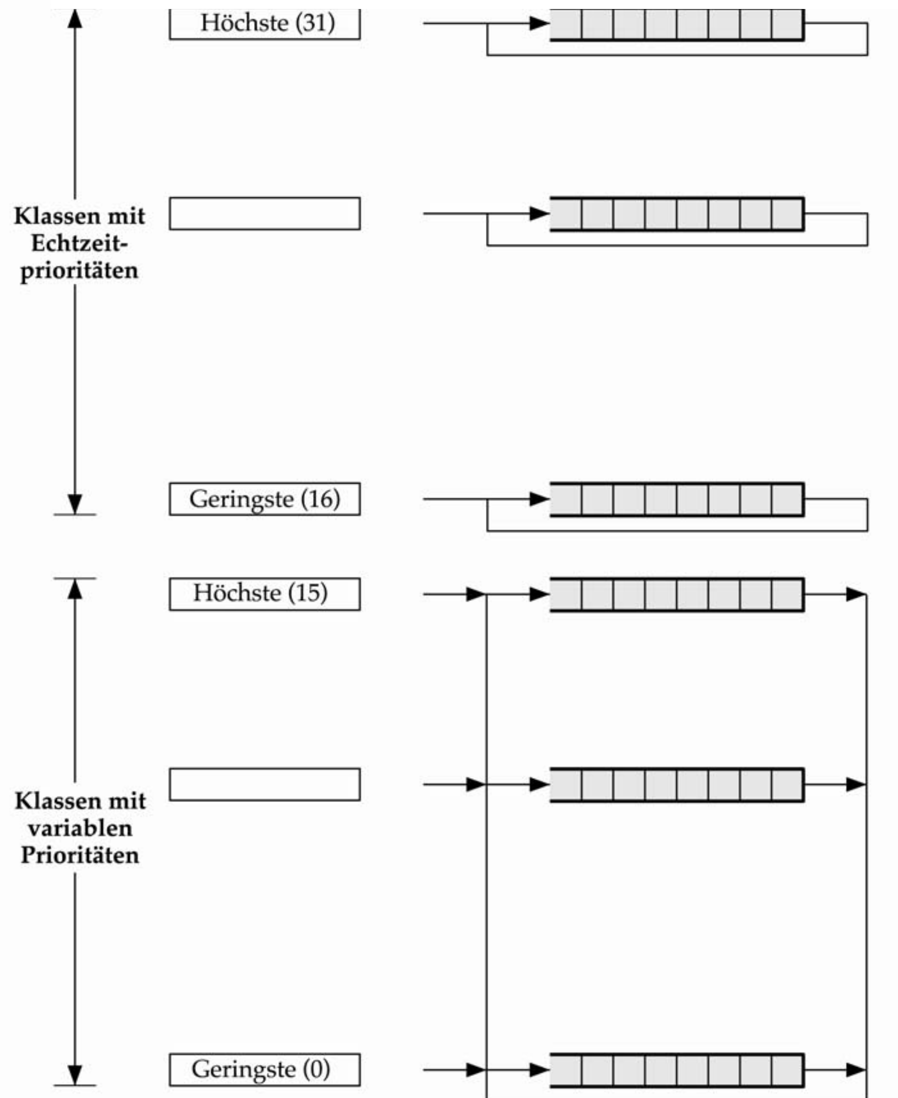
Echtzeit-Scheduling

1. Echtzeit-Scheduling

2. Scheduling in Linux

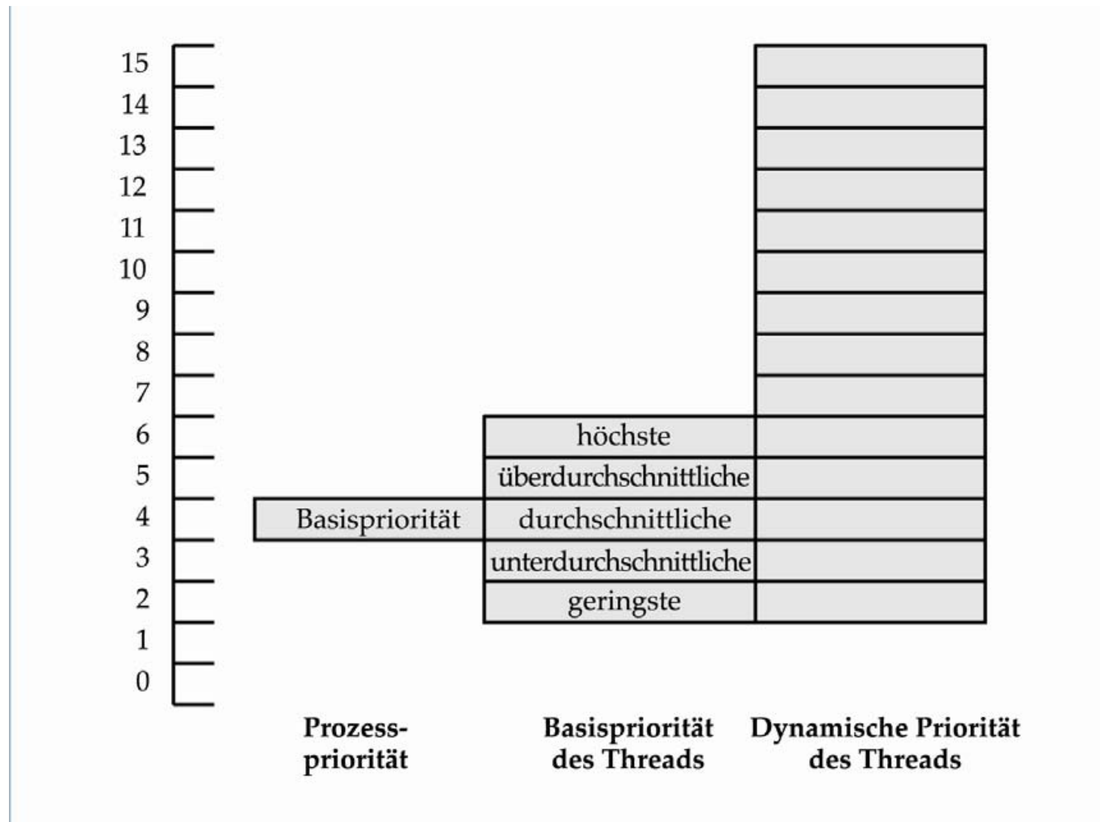
3. Scheduling in Windows 2000

Scheduling in Windows 2000



- unterbrechender Scheduler
- flexibles Prioritätensystem
- Innerhalb Prioritätenstufen RR-Prinzip
- In einigen Stufen: dynamsiche Änderung der Prioritäten entsprechend aktueller Thread-Aktivität
- Basispriorität des Prozesses
- Threadpriorität relativ zur Basispriorität des zug. Prozesses

Prozess/Thread-Priorität bei Windows 2000



- Threadunterbrechung durch Ablauf Zeitquantum: Priorität wird herabgesetzt
- Threadunterbrechung durch E/A-Blockierung: Priorität wird erhöht