Scheduling

Rechenprozess(Task)-Scheduling,

Gegeben:

- **Menge von Paaren (Anforderung, Zeitpunkt)**

Gesucht:

- **Reihenfolge, in der Anforderungen bearbeitet werden, so dass für alle die Rechtzeitigkeitsbedingung erfüllt ist**

Scheduling

Die Lösung: Scheduling

- **Strategie zur Festlegung der Reihenfolge von zu bearbeitenden Rechenprozessen (statisches Scheduling)**
- **Strategie zur Festlegung des nächsten zu bearbeitenden Rechenprozesses (dynamisches Scheduling)**
- **Ermöglicht die (quasi-)parallele Bearbeitung mehrerer Rechenprozesse**

Scheduling

Statisches und dynamisches Scheduling

Statisches Scheduling

- Festlegung eines „Fahrplans“ (im vorhinein), nach dem die einzelnen Rechenprozesse in einem festen Schema abzuarbeiten sind.
- Einsatz: zyklische, sicherheitskritische Anwendungen mit fixen Zeitpunkten für auftretende Anforderungen (z.B. Flugzeug-Steuerungen)
  → formaler Nachweis der Einhaltung von Realzeitbedingungen möglich
- Speicherprogrammierbare Steuerungen

Dynamisches Scheduling

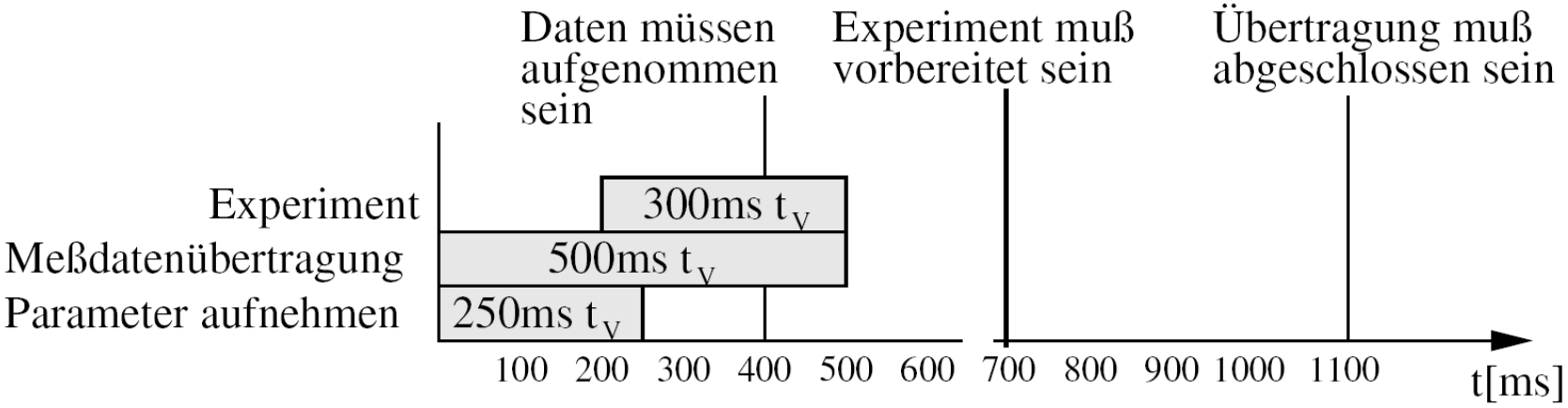
- Im Betriebssystem enthaltener Scheduler zur Laufzeit:
  - Zuteilung des Prozessors an Rechenprozesse aufgrund der jeweils aktuellen Bedarfssituation
- Rechenprozesse müssen unterbrechbar (*präemptiv*) sein.

Scheduling

Ein Beispiel

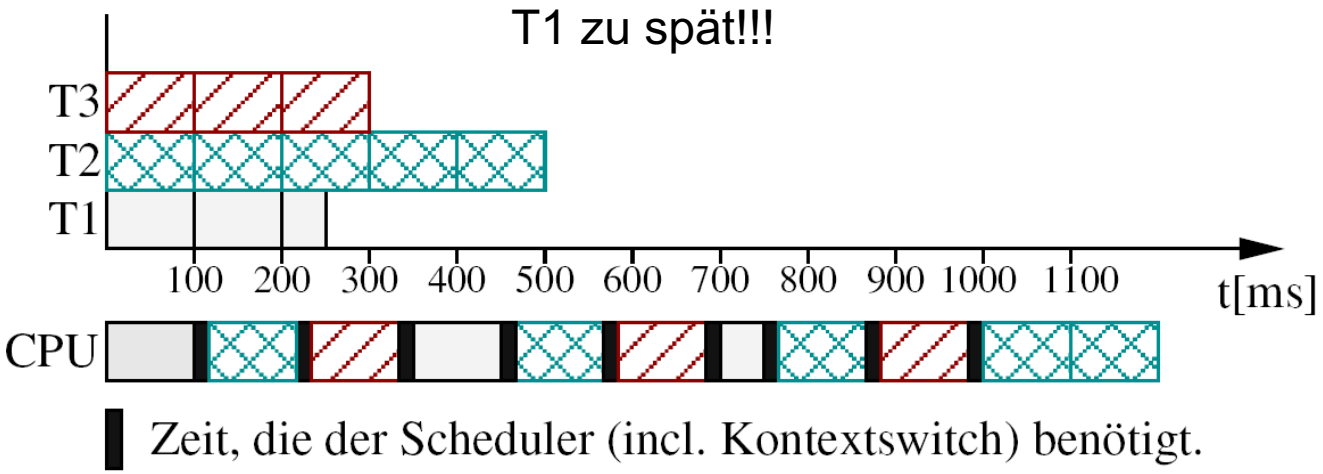
Realzeitsystem (Single Core CPU) steuert Experiment an Bord eines Satelliten

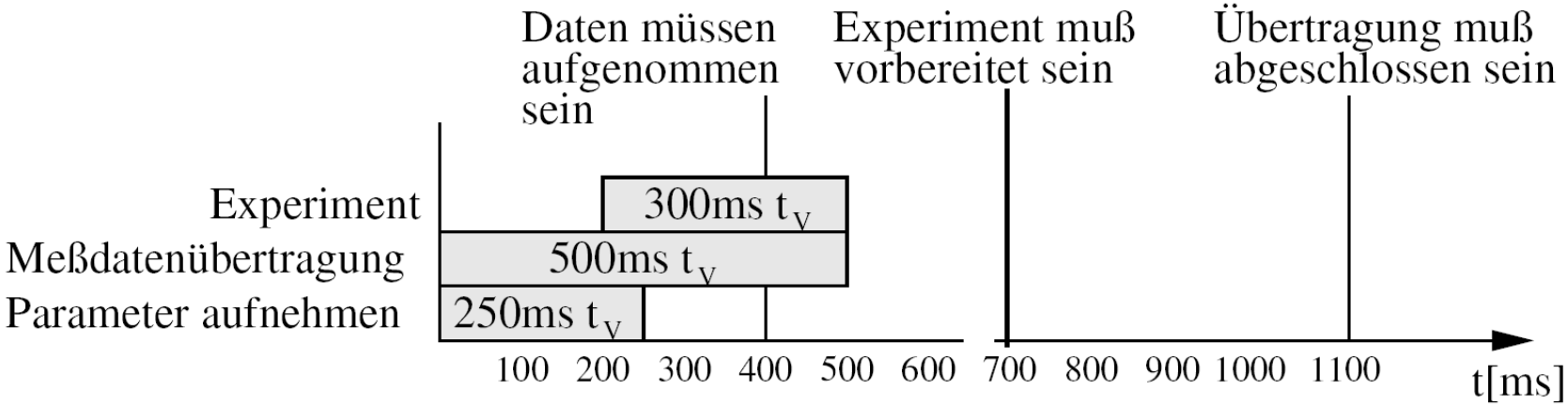
- **(T1) Alle 1500ms (Zeitpunkt t_0) Meßdaten aufnehmen (t_P=1500 ms).**
- **(T1) Die Messwertaufnahme muss nach 400 ms abgeschlossen sein (t_{Zmax}=400 ms).**
- **(T3) Alle 200 ms nach Start der Messdatenaufnahme t_0: Experiment anstoßen.**
- **(T3) Die Vorbereitungen müssen 500 ms später abgeschlossen sein (also 700 ms nach dem Start der Messwertaufnahme t_0, t_P=1500 ms, t_{Zmax}=500 ms).**
- **(T2) Messergebnisse müssen innerhalb von 1100 ms (t_{Zmax}=1100 ms) zur Erde weitergeleitet werden (alle 60s, t_P=60000 ms).**
- **(T1)Task 1: t_V = 250 ms; (T2)Task 2: t_V = 500 ms; (T3)Task 3: t_V = 300 ms**



Scheduling

Ein Beispiel: Versuch eines Schedulings





Scheduler

- Einheit eines Betriebssystems, die für die Zuteilung von Rechenzeit an Prozesse/ Threads zuständig ist
- Bei Multi-Prozessor-Systemen verteilt der Scheduler die Prozesse/Threads zusätzlich auch auf Prozessoren

Scheduling

Scheduling Points

Scheduler überprüft, ob ein anderer Prozess die CPU erhalten sollte und veranlasst ggf. einen Kontextwechsel

- **Ende einer Systemfunktion (Übergang Kernel/User Mode):**
  - **Die Systemfunktion hat den aktiven Prozess blockiert (z.B. Warten auf Ende von I/O)**
  - **In der Systemfunktion sind die Scheduling-Prioritäten geändert worden**
- **Interrupts**
  - **Timer-Interrupt: aktiver Prozess hat sein Quantum verbraucht (Round Robin)**
  - **I/O signalisiert das Ende einer Wartebedingung (höher priorer Prozess wird „bereit“)**

Scheduling

Bewertungskriterien

Gerechtigkeit

- Jeder Prozess soll einen fairen Anteil der CPU-Zeit erhalten

Effizienz

- Die CPU soll möglichst gut ausgelastet werden

Durchlaufzeit

- Ein Prozess soll so schnell wie möglich abgeschlossen sein

Durchsatz

- Es sollen so viele Jobs wie möglich pro Zeiteinheit ausgeführt werden

Antwortzeit

- Die Reaktion auf Ereignisse soll möglichst schnell erfolgen

Determinismus

- Das Scheduling als solches soll berechenbar sein (Identisches Verhalten bei identischen Bedingungen)

Scheduling

Bewertungskriterien

Wichtig bei Standard-BS

- Gerechtigkeit
- Effizienz
- Durchlaufzeit
- Durchsatz

Wichtig bei Echtzeit-BS

- Antwortzeit
- Determinismus

Scheduling

Schedulingstrategien: First Come First Serve (FCFS)

Auch: First In First Out (FIFO)

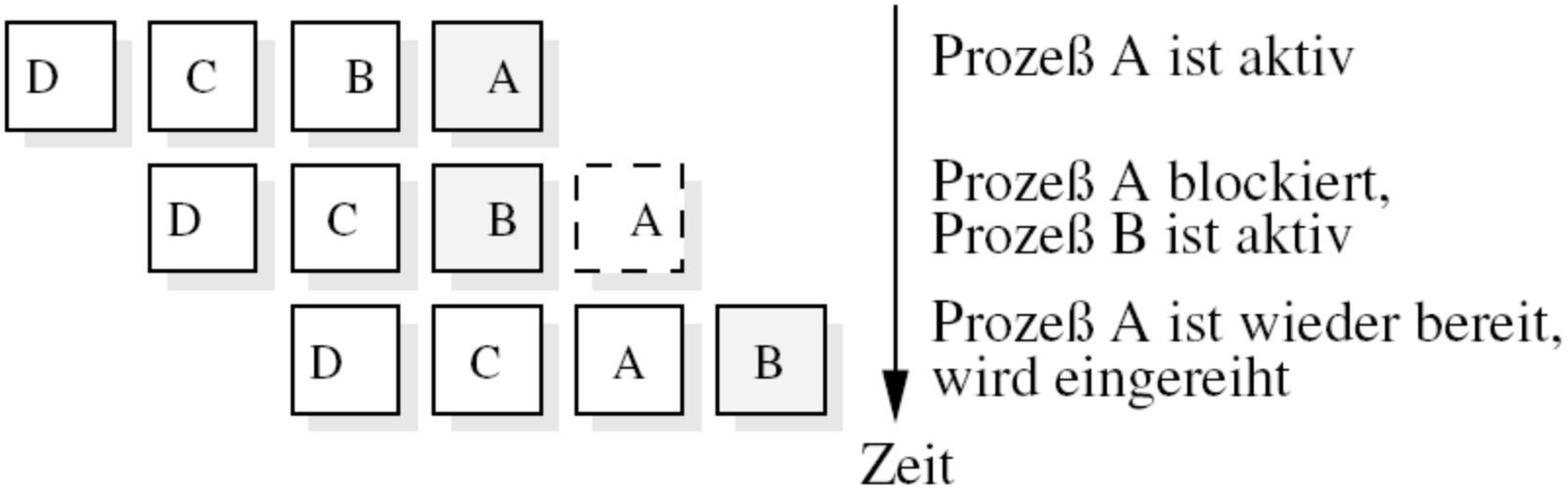
Prinzip

- Die bereiten Prozesse sind in einer Warteschlange nach ihrem Erzeugungszeitpunkt geordnet.
- Jeder Prozess darf bis zu seinem Ende laufen, außer er geht in den Zustand „Blockiert“ über.
- Geht ein Prozess vom Zustand „Blockiert“ in den Zustand „Bereit“ über, wird er entsprechend seinem Erzeugungszeitpunkt wieder in die Warteschlange eingereiht, unterbricht aber den laufenden Prozess nicht.

Anwendungen

- Batch-Systeme

Schedulingstrategien: First Come First Serve (FCFS)

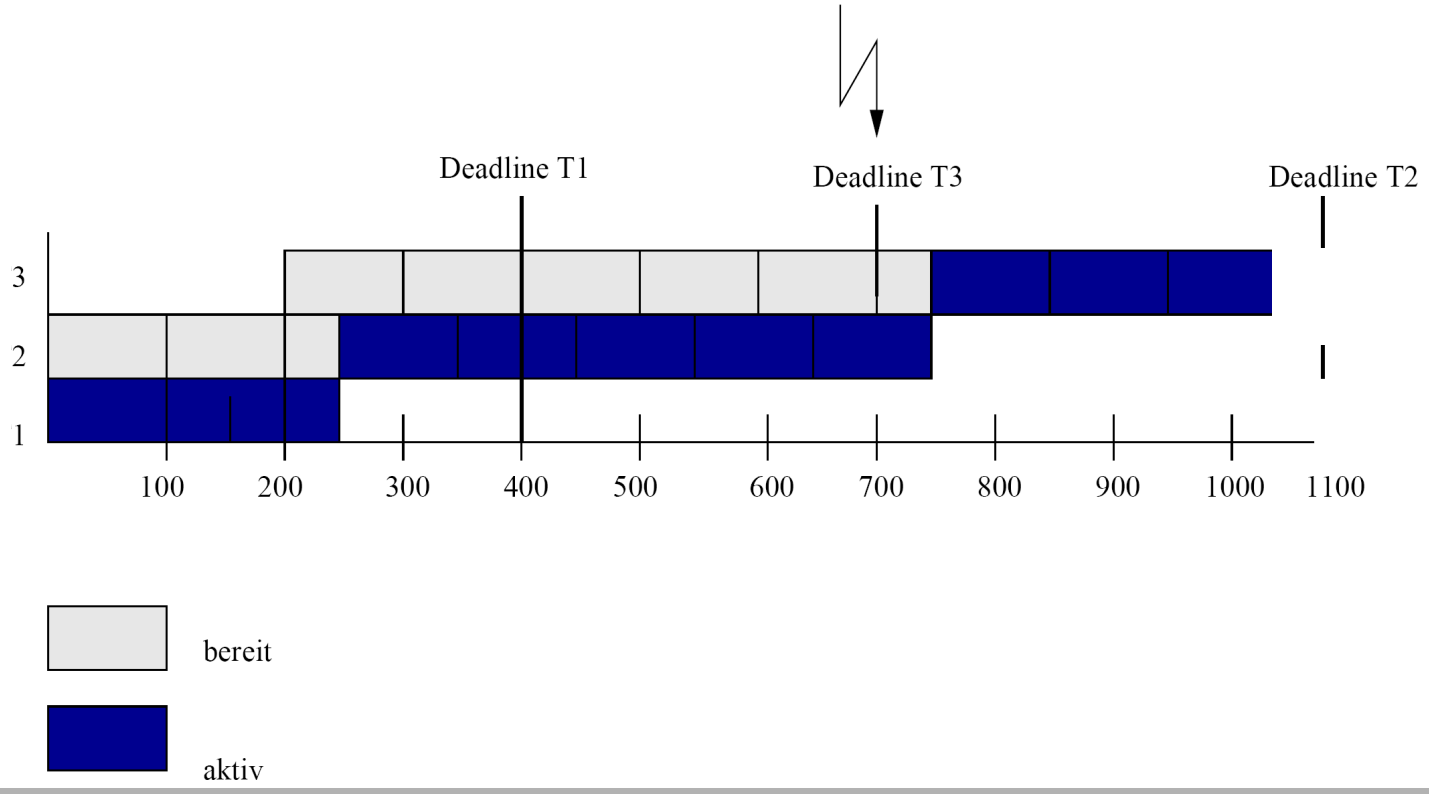


Scheduling

Schedulingstrategien: First Come First Serve (FCFS)

Realzeiteigenschaften:

- Nicht für Realzeitsysteme geeignet, da ein Prozess alle anderen blockieren kann!



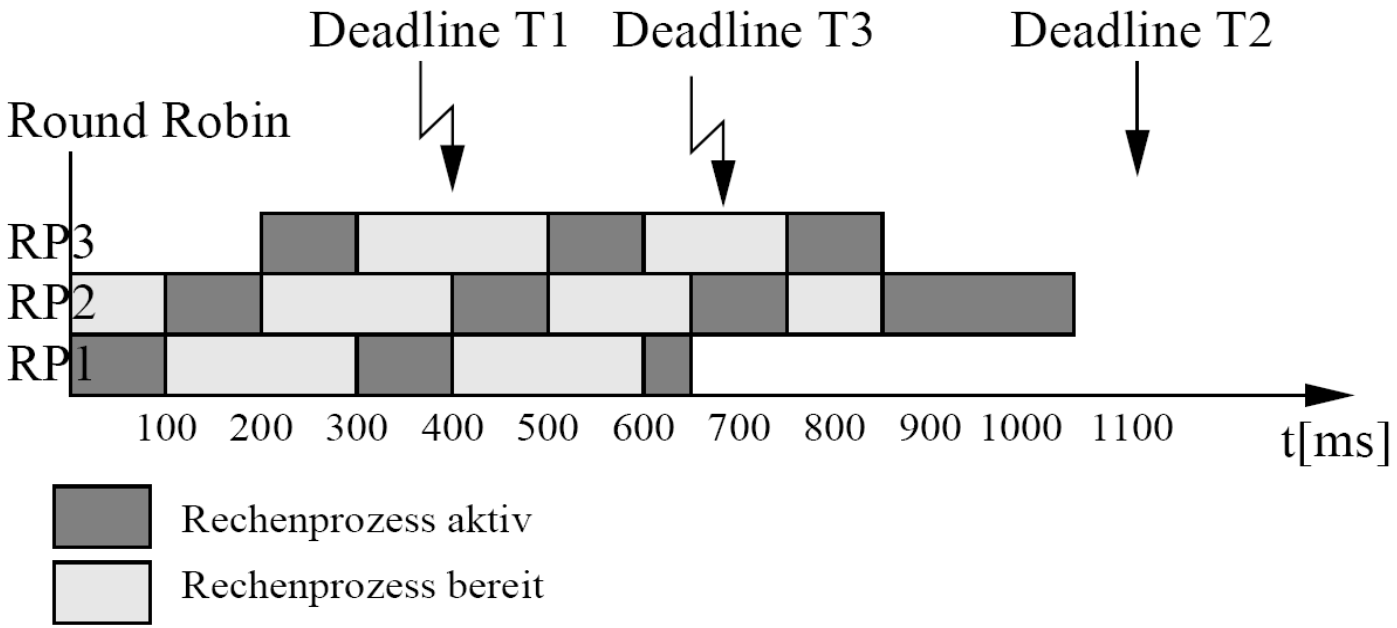
Scheduling

Schedulingstrategien: Round Robin Scheduling

Prinzip

- Alle Prozesse werden in eine Warteschlange eingereiht
- Jedem Prozess wird eine Zeitscheibe (time slice, quantum) zugeteilt
- Ist ein Prozess nach Ablauf seines Quantums noch im Zustand „Aktiv“,
  - wird ihm die CPU entzogen, d.h. er erhält den Zustand „lauffähig“
  - wird der Prozess am Ende der Warteschlange eingereiht
  - wird dem ersten Prozess in der Warteschlange die CPU zugeteilt
- Geht ein Prozess vom Zustand „wartend“ in den Zustand „lauffähig“ über, so wird er am Ende der Warteschlange eingereiht.

Schedulingstrategien: Round Robin Scheduling



Schedulingstrategien: Round Robin Scheduling

Kriterien für die Wahl des Quantums

- **Das Verhältnis zwischen Quantum und Kontextwechselzeit muss vernünftig sein.**
- **Großes Quantum: effizient, aber lange Verzögerungszeiten und Wartezeiten möglich.**
- **Kleines Quantum: kurze Antwortzeiten, aber großer Overhead durch häufige Prozessumschaltung.**

Scheduling

Schedulingstrategien: Round Robin Scheduling

Realzeiteigenschaften:

Dynamisches Scheduling

- Anzahl der bereiten Rechenprozesse unbekannt:

  Abarbeitungszeitpunkt eines Prozesses nicht vorhersagbar (nicht deterministisch)
- Keine sofortige Reaktion auf asynchrone Anforderungen
- Fazit: Für Realzeitsysteme nicht geeignet

Statisches Scheduling

- Abwandlung des Verfahrens (TDMA=Time Division Multiple Access): Zeitscheiben sind fest
- Anzahl der Prozesse im System von vornherein bekannt:

  Verfahren kann sich abhängig von der Aufgabenstellung (z.B. SPS, GSM) für Realzeitsysteme eignen

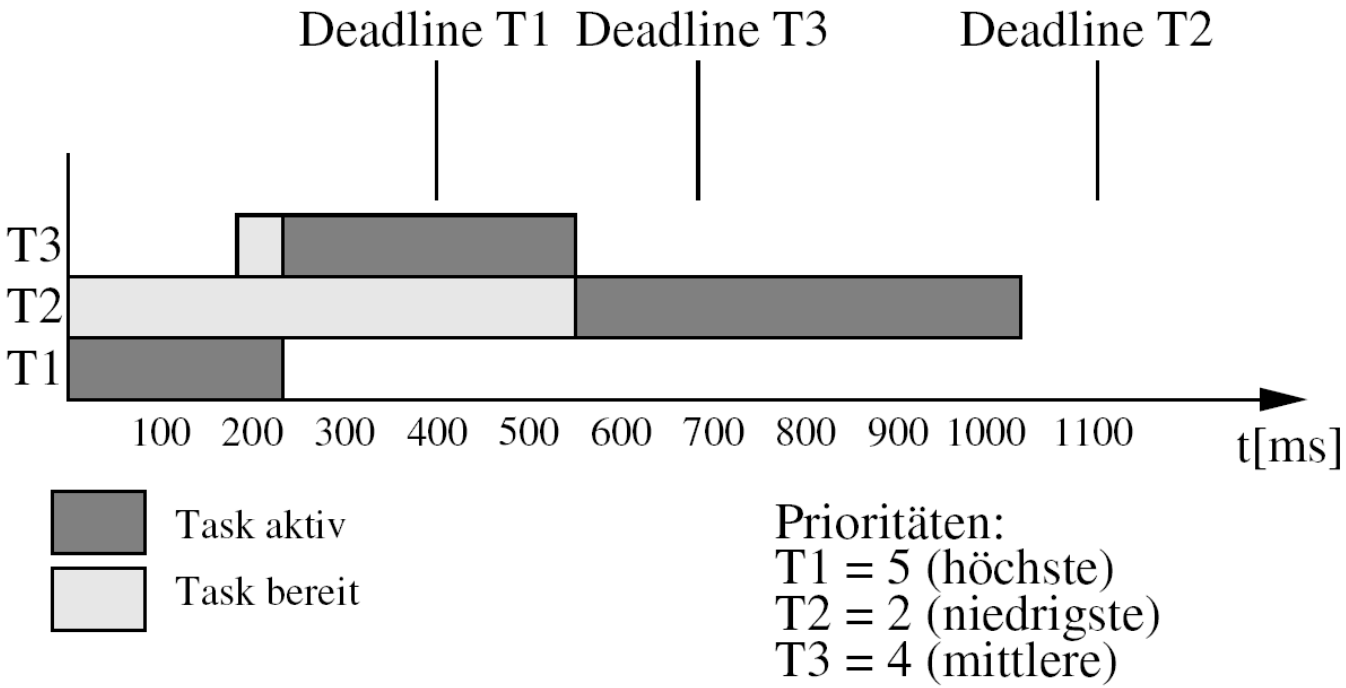
Scheduling

Schedulingstrategien: Prioritätengesteuertes Scheduling

Prinzip

- **Für jeden Prozess wird eine Priorität vergeben.**
- **Der Prozess mit der höchsten Priorität bekommt die CPU.**

Schedulingstrategien: Prioritätengesteuertes Scheduling



Schedulingstrategien: Prioritätengesteuertes Scheduling

Realzeiteigenschaften:

Für Realzeitsysteme geeignet

- **insbesondere wenn keinerlei Informationen bezüglich der maximal zulässigen Reaktionszeiten zur Verfügung stehen**

Problem: Wahl der Prioritäten

Scheduling

Schedulingstrategien: Deadline-Scheduling

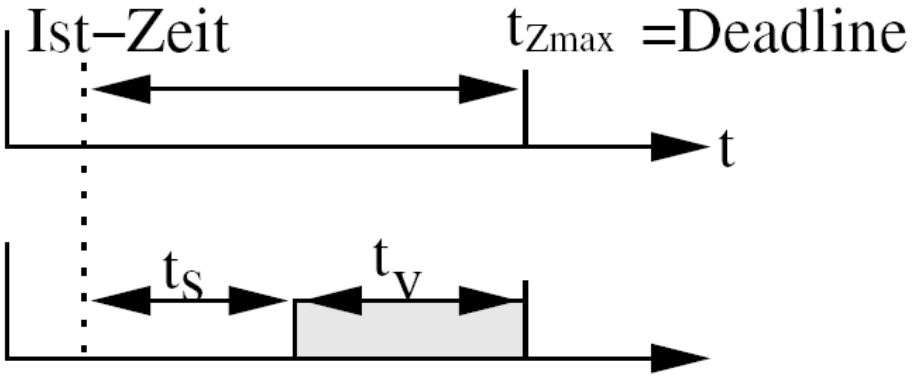
EDF (*earliest deadline first*)

Prinzip

- **Der Rechenprozess mit der am nächsten gelegenen Deadline (maximal zulässige Reaktionszeit) bekommt die CPU zugeteilt.**
- **Prozesse müssen dem Scheduler ihre Deadlines bekannt geben**

Scheduling

Schedulingstrategien: Deadline-Scheduling

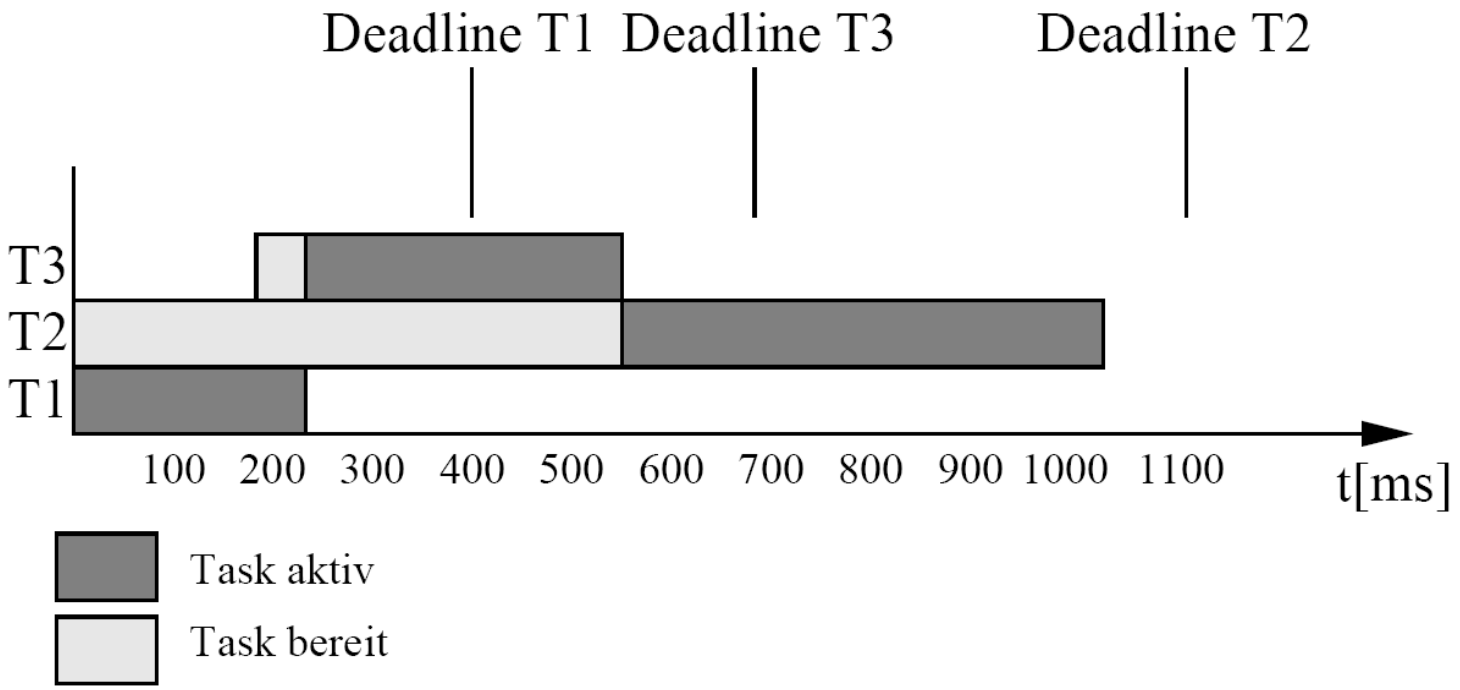


t_{Zmax}	zu diesem Zeitpunkt muß die Task bearbeitet sein
t_V	Verarbeitungszeit (ohne Wartezeit)
t_S	Zeit–Spielraum bis zum spätestmöglichen Start der Verarbeitung (laxity)

Alternative: LLF (Least Laxity First)

Problem: Hochfrequente Kontextswitches, wenn gleicher Spielraum erreicht ist

Schedulingstrategien: Deadline-Scheduling



Schedulingstrategien: Deadline-Scheduling

Realzeiteigenschaften:

Das Verfahren führt zur Einhaltung der maximalen Reaktionszeiten, wenn dies überhaupt möglich ist (optimales Verfahren)!

Nachteil:

Deadlines (sprich: die maximal zulässigen Reaktionszeiten) müssen angegeben werden, sind aber nicht immer bekannt.

„Domino“ Effekt in der Praxis: Wenn eine Deadline verfehlt wird, werden meist alle anderen Deadlines auch verfehlt.

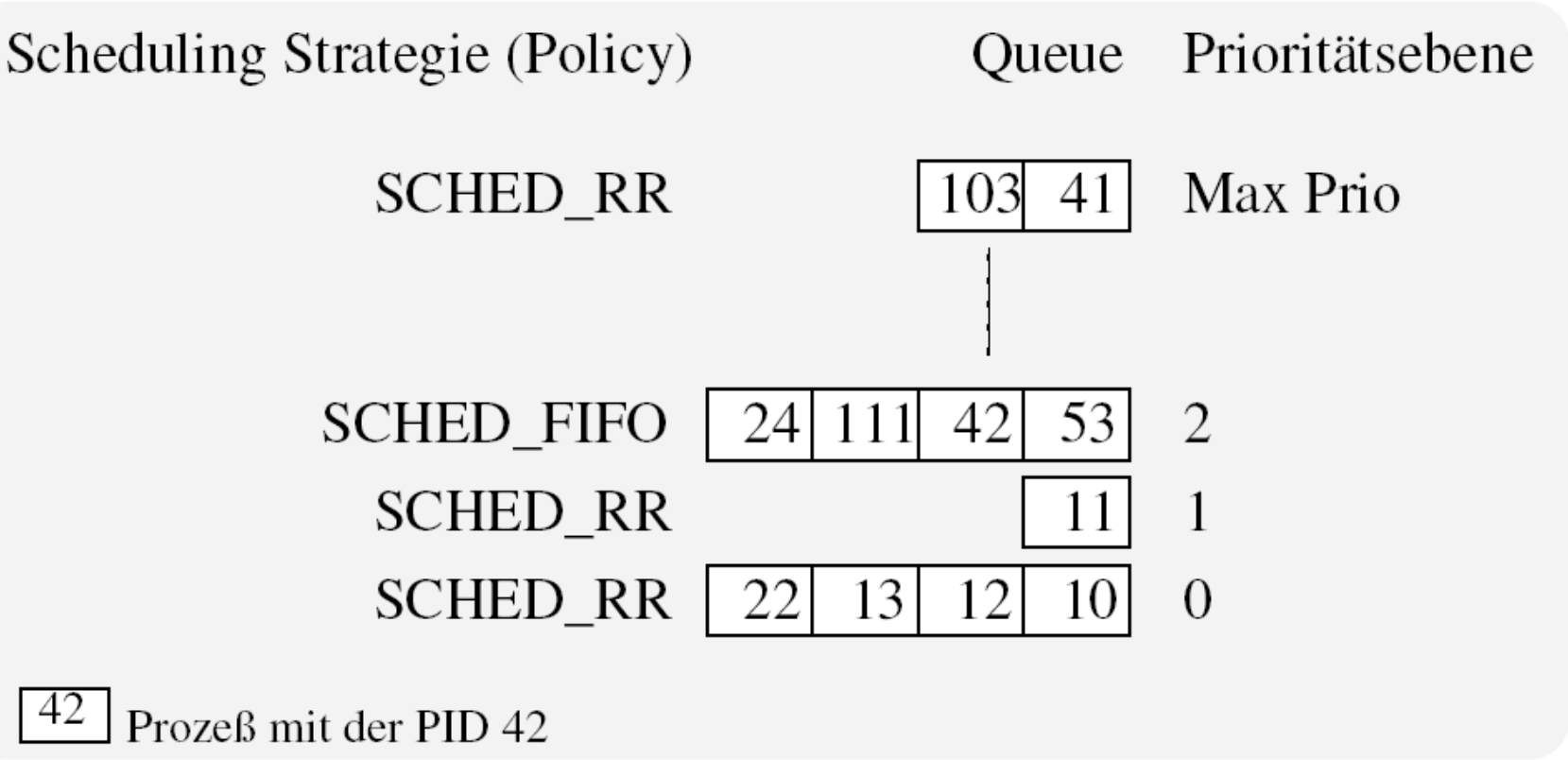
Scheduling

Schedulingstrategien: POSIX.1-2017

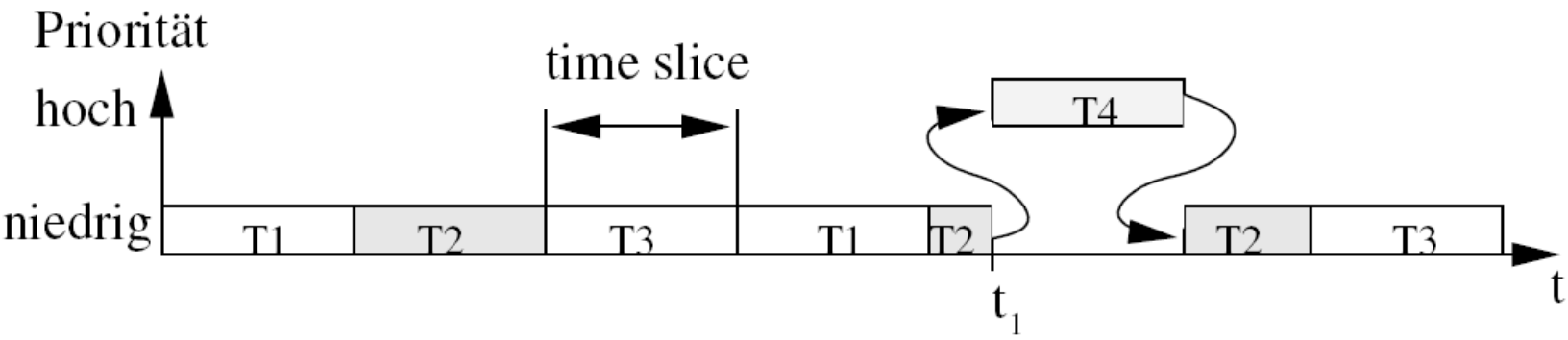
Prinzip

- Prioritätengesteuertes Scheduling
- Auf jeder Prioritätsebene können sich mehrere Prozesse befinden.
- Innerhalb einer Prioritätsebene werden Prozesse gescheduled nach:
  - First In First Out (First Come First Serve)
  - Round Robin
  - Prioritätsebene 0 besitzt die niedrigste Priorität.

Schedulingstrategien: POSIX.1-2017



Schedulingstrategien: POSIX.1-2017



Schedulingstrategien: POSIX.1-2017

Zuteilung von statischen Prioritäten

- **Faustregel („Rate Monotonic Scheduling“ (RMS)):**
  - **Je kürzer die Prozesszeit t_{pi},**
  - **desto kleiner im Regelfall die Verarbeitungszeit t_{vi} und**
  - **desto höher die fixe! Priorität Task i**
- **Nach Prozessart: interaktiv, Batch, Netzwerk etc.**
- **Nach Prozessaktivität**
  - **CPU intensive Prozesse bekommen niedrige Priorität**
  - **I/O intensive Prozesse bekommen hohe Priorität**
- **Benutzerdefiniert**

Schedulingstrategien: Ergänzung

Zuteilung von statischen Prioritäten

- **Faustregel („Deadline Monotonic Scheduling (DMS)“):**
  - **Je kürzer die maximal erlaubte Reaktionszeit t_{zmaxi},**
  - **desto kleiner im Regelfall die Verarbeitungszeit t_{vi} und**
  - **desto höher die fixe! Priorität Task i**

- **DMS = RMS wenn $t_{pi} = t_{zmaxi}$**
- **DMS ist mächtiger als RMS**
- **DMS umfasst RMS**

Scheduling in VxWorks

- Wind-Scheduling (ähnlich POSIX)
- POSIX-Scheduling

Wind Scheduling

- Präemptives Prioritäten-Scheduling
  - 256 Prioritätsebenen (0=höchste, 255=niedrigste Priorität)
- Innerhalb einer Prioritätsebene: Round Robin Scheduling

Unterschiede zu POSIX

- Prioritätenbezeichnungen (hoch, niedrig) invers.
  - POSIX hohe Zahl (z.B. 255) bedeutet hohe Priorität,
  - Wind-Scheduling hohe Zahl bedeutet niedrige Priorität
- Scheduling innerhalb einer Prioritätsebene immer RR

Scheduling in VxWorks

API

Einstellung der RR-Parameter

`kernelTimeSlice` (globale Variable ☹)

Modifikation der Taskpriorität

`taskPrioritySet(...)`

Scheduling ausschalten

`taskLock()`

Scheduling einschalten

`taskUnlock()`

Scheduling in OS9

- Prioitätsscheduling für Realtimetasks (Startprio>maxage)
- Timesharing - „Ageing“-Verfahren für Nicht-RealtimeTasks: Startprio <maxage
- Präemptives Prioritäten-Scheduling
  - 65535 Prioritätsebenen (65535=höchste, <minage>=niedrigste Priorität)
  - Höchstprio rechenwillige Task (Nonrealtime/Realtime) erhält CPU und
- Realtimetasks: keine zwei Tasks gleicher Prio sinnvoll
- Non-Realtimetasks:
  - minage < Startprio < maxage, Tasks < minage werden startbereit gemacht aber nicht schedult
  - Je eingestelltem Timeslice alle: n*TIC z.B. n=2 → 2TICS
    - Wartepriobonus für Tasks Startprio <maxage + 1 wenn sie nicht dran waren, Maximales Limit pro Task:= (Startprio + Summe seiner Warteboni) < maxage
    - Erhält eine Task mit Warteboni die CPU so fällt sie sofort zurück auf ihre Startprio, behält aber für einen Timeslice die CPU

Scheduling in OS9

- ***OS-9 Timesharing Algorithmus***

Es gibt auch Mischformen in denen sowohl zyklische als auch ereignisgesteuerte prioritätsgeteuerte Scheduling-Mechanismen implementiert sind.

OS9: Laborsysteme: Maxage 255, Minage 0, Defaultprio = 128

Beispiel OS9 Scheduling, gegeben:

Timeslice soll zwei Tics a 1ms entsprechen

- **pro Timeslice ´= 2ms altert´ (aging) ein wartender Prozess um 1**
- **Maximal-Aging-Priorität = 255 (Maxage),**
- **Erhält ein Prozess die CPU wird er auf seine Startpriorität zurückgestuft UND**
- **alle anderen erhalten gleichzeitig einen Wartebonus von 1 Prioritätspunkt**

Task A	**Startpriorität 130**
Task B	**Startpriorität 128**
Task C	**Startpriorität 128**

Zu Beginn hat kein Prozess die CPU. Alle drei werden gleichzeitig gestartet.

Scheduling

Scheduling in OS9

Process Scheduling: A Simple Case

Iteration	CPU	Active Queue		
pre	<none>	A_{130}	B_{128}	C_{128}
1	A^{130}	A_{130}	B_{129}	C_{129}
2	A^{130}	B_{130}	C_{130}	A_{130}
3	B^{128}	C_{131}	A_{131}	B_{128}
4	C^{128}	A_{132}	B_{129}	C_{128}
5	A^{130}	B_{130}	A_{130}	C_{129}
6	B^{128}	A_{131}	C_{130}	B_{128}
7	A^{130}	C_{131}	A_{130}	B_{129}
8	C^{128}	A_{131}	B_{130}	C_{128}
9	A^{130}	B_{131}	A_{130}	C_{129}
10	B^{128}	A_{131}	C_{130}	B_{128}
11	A^{130}	C_{131}	A_{130}	B_{129}
12	C^{128}	A_{131}	B_{130}	C_{128}

Scheduling

POSIX-Scheduling-API

Setzt die Task-Priorität

`sched_setparam (OS9: _os_setpr)`

Liest die Schedulingparameter einer Task

`sched_getparam`

Liest die Schedulingparameter einer Task

`sched_setscheduler`

Gibt die CPU frei (erzwingt Scheduling)

`sched_yield`

Liest die aktuelle Schedulingstrategie

`sched_getscheduler`

Liest die maximal mögliche Priorität bzgl. einer Schedulingstrategie

`sched_get_priority_max`

Liest die minimale Priorität bzgl. einer Schedulingstrategie

`sched_get_priority_min`

Ergibt die Dauer des Zeitintervalls bei Round Robin Scheduling

`sched_rr_get_interval`

POSIX-Scheduling-API, Codefragment

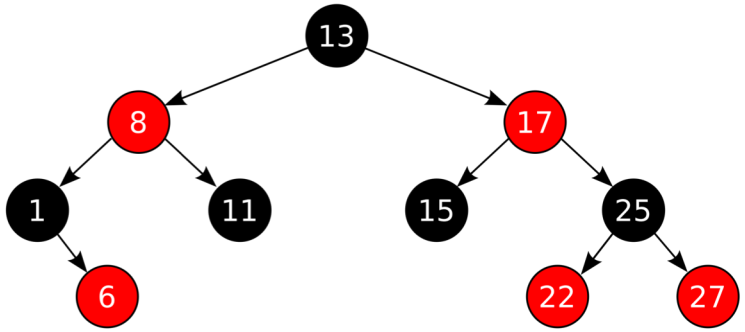
```
int main( int argc, char **argv )
{
  struct sched_param SchedulingParameter;
  sched_getparam( pthread_self(), &SchedulingParameter );
  printf("Priority: %d\n", SchedulingParameter.sched_priority);

  // change scheduling policy and priority to realtime priority
  SchedulingParameter.sched_priority = PRIORITY_OF_THIS_TASK;
  if( sched_setscheduler( pthread_self(), SCHED_RR,
      &SchedulingParameter )!= 0 ) {
    perror( "Set Scheduling Priority" ); exit( -1 );
  }

  sched_getparam( pthread_self(), &SchedulingParameter );
  printf("Priority: %d\n", SchedulingParameter.sched_priority);
}
```

Scheduling

Completely Fair Scheduler (CFS) ab Kernel 2.6.43 bis heute (Erweiterung CFS Task Groups)

Prinzipiell:

- **Der CFS garantiert eine faire Aufteilung der Prozessorzeit**
- **jedem Prozess ein *vruntme* genannter Wert zugeordnet (virtual runtime), der seine tatsächliche Laufzeit auf Nanosekunden bestimmt abstrahiert und eine Bewertung darüber erlaubt, wie lange der Prozess schon gelaufen ist.**
- **Derjenige Prozess mit der jeweils geringsten *vruntime* wird gewählt. Als Struktur wird dafür ein nach der *vruntime* sortierter rot-schwarz Baum (Binärbaum) verwendet.**



Earliest Deadline First – Scheduling in RTLinux

- **Angabe der jeweils nächsten Deadline durch den Prozess zur Laufzeit**
- **API zum Scheduler**

```
int pthread_setdeadline_np(pthread_t thread, hrtime_t deadline)
int pthread_getdeadline_np(pthread_t thread, hrtime_t *deadline)
```

`hrtime_t` **64-Bit unsigned int, Nanosekunden seit nicht näher spezifiziertem Zeitpunkt in der Vergangenheit**

Scheduling

Scheduling: POSIX.1-2017 , Sporadic Scheduling

Problem:

- Annahme, dass alle Tasks exakt zyklisch sind, ist meist unrealistisch (sonst: statisches Scheduling, wenn es gemeinsame Grundfrequenz gibt, oder Raten Monotonic Analysis(RMS))
- Spezifikation von Deadlines alleine berücksichtigt die Auslastung nicht (beliebige Prozesszeiten möglich)
- Priorisierung allein (bis auf RMS) nur mit Faustregeln „beherrschbar“, kein Echtzeitnachweis (bis auf triviale Fälle, z.B. 1 harte Echtzeitanforderung)

Lösung: Schöner Kompromiss: Sporadic Scheduling

Prinzip:

- Zyklische Tasks mit minimaler Prozesszeit und Verarbeitungszeit spezifizieren
- Tasks mit unbekannter Prozesszeit (z.B. „gelegentlich“, „manchmal Bursts“) als zyklische Tasks modellieren und bei Prozesszeitverletzung (Überschreiten der spezifizierten Auslastung) in der Priorität herabsetzen
- Folge: „Saurer Apfel“: Für sporadische Tasks Reaktionszeitverletzung, wenn CPU bei 100% Last

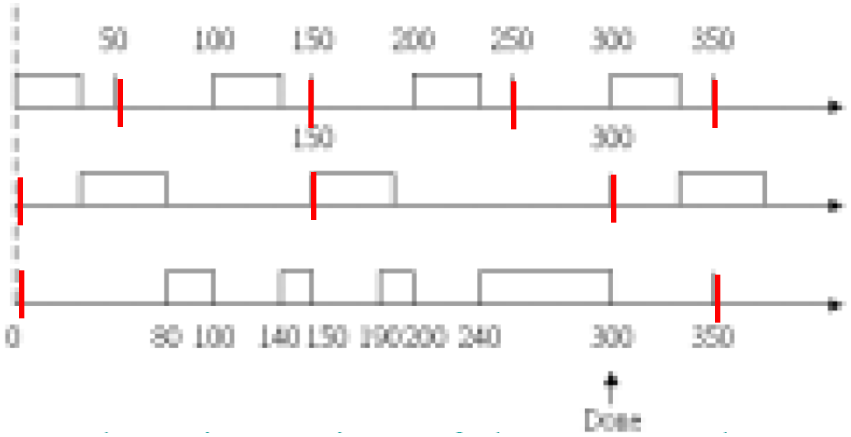
Scheduling: POSIX.1-2017 , Sporadic Scheduling

Was ist RMS???

- Ganz einfach!!! Aber stark in der Aussage!!!
- Rate Monotonic Analysis

Annahmen:

- Alle Tasks sind zyklisch
- Maximale Reaktionszeit = Prozesszeit
- Verarbeitungszeit bekannt



Aussage:

2. Echtzeitbedingung erfüllt, wenn

- Tasks Prioritäten entsprechend ihrer Prozesszeiten erhalten:
  Hochfrequente Tasks höherprior
  als niederfrequente („rate monotonic“)

- Auslastung bei n Tasks <

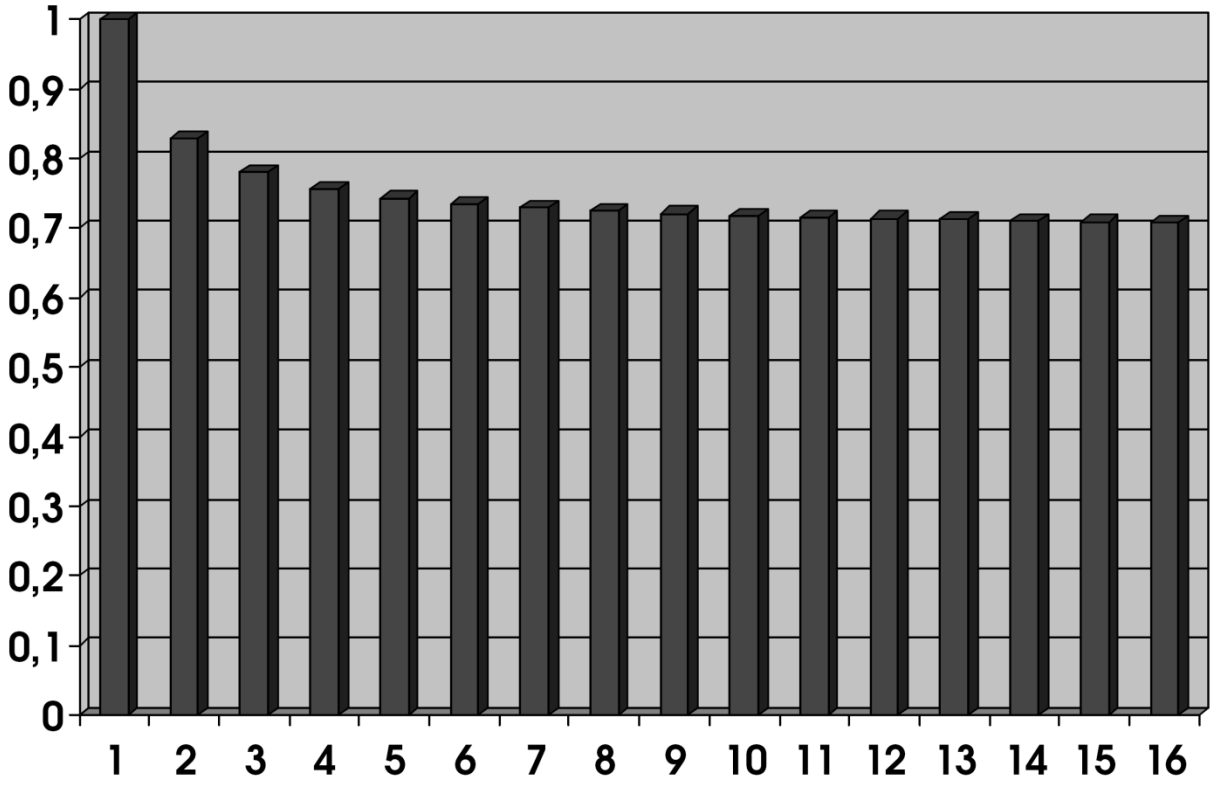
$$U(n) = n(2^{1/n} - 1)$$

Scheduling

IEEE POSIX.1-2017 , Sporadic Scheduling

Was ist RMS???

Auslastung bei n Tasks < $U(n) = n(2^{1/n} - 1)$



Scheduling

Scheduling: POSIX.1-2017 , Sporadic Scheduling

Und wie lässt man jetzt azyklische (sporadische) Tasks aussehen wie zyklische???

Mit Sporadic Scheduling!!!

- **Rahmen: Ganz normales, prioritätsgesteuertes präemptives Scheduling (in dem auch zyklische Tasks laufen)**
- **Pro (sporadischer) Task anzugeben:**
  - **Priorität**
  - **(angenommene) Prozesszeit**
  - **(Erlaubte) Verarbeitungszeit in einem Prozess-Zeittakt**
  - **„Hintergrundpriorität“, auf die die Task (vorübergehend) bei Verarbeitungszeit-Überschreitung zurückfällt**

POSIX.1-2017, Sporadic Scheduling

Folgen:

- **Rechtzeitigkeit wird für alle Tasks erfüllt, die sich an ihre Spec halten!!!**
- **Falls eine Task versucht, ihr erlaubtes Kontingent zu überschreiten:**
  - **Auslastung < 100% : Task erhält zusätzliche Rechenzeit mit Hintergrundpriorität**
  - **Auslastung = 100%: Task verletzt Rechtzeitigkeitsbedingung**
- **Task, die versucht, ihr erlaubtes Kontingent zu überschreiten, beeinflusst das Gesamtsystem in keiner Weise negativ!!!**

Scheduling

POSIX.1-2017, Sporadic Scheduling

Wie funktioniert‘s im Detail:

Parameter in der IEEE Spec:

Replenishment Period T (Auffüll-Periode, entspricht Prozesszeit)

Dieser Wert bestimmt das Zeitintervall, innerhalb dessen der Thread seine Ausführungszeit C verbrauchen kann.

Initial budget C (Anfängliches Budget, Verarbeitungszeit pro Takt)

Ausführungszeit, die ein Thread innerhalb seiner Replenishment Period T unter normaler Priorität N läuft, bevor er zu einer geringeren Priorität L wechselt

Priority N (Normale Priorität)

Low Priority L (Hintergrundpriorität)

Priorität, auf die der Thread nach Verbrauch seiner Ausführungszeit C fällt

Maximal number of pending replenishments

Begrenzt Anzahl der Replenishments und somit den Systemoverhead, den das Schedulingverfahren verursacht.

Scheduling

POSIX.1-2017, Sporadic Scheduling

Im Detail: Replenishment-Strategie

Wichtig: In jeder Zeitperiode T (Prozesszeit), egal wann sie beginnt, darf C nicht überschritten werden!!!

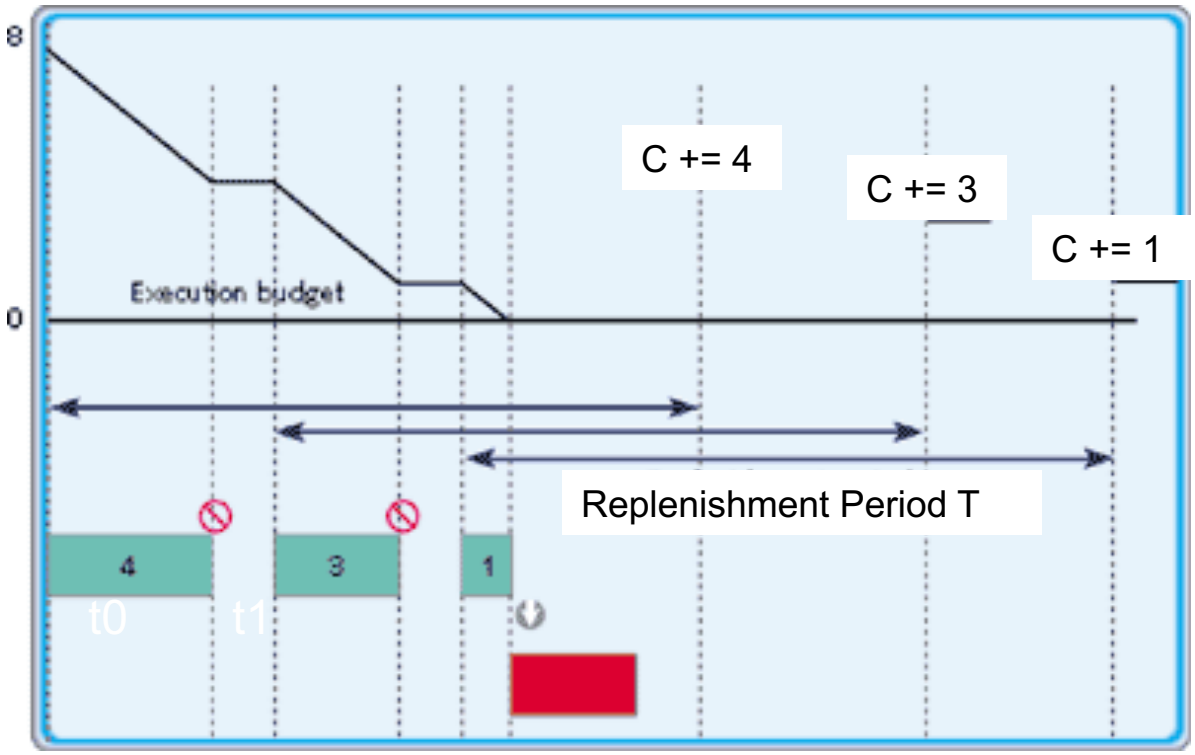
Daher folgende Strategie:

t0: Task erhält CPU

t1: CPU wird entzogen

Für den Zeitpunkt t0 + T wird geplant,

C um t1 – t0 (verbrauchte Rechenzeit) aufzufüllen



Scheduling

Schedulingverfahren in der Anwendung

Der Ingenieur/Informatiker ist mit Scheduling konfrontiert bei:

- **Projektierung eines Realzeitsystems (Auswahl des Verfahrens)**
- **Zerlegung einer Aufgabe (Applikation) in Prozesse/Threads**
- **Festlegung von Prioritäten**
- **Evaluierung von Scheduling-Parametern (z.B. Deadlines)**
- **Implementierung der Rechenprozesse**