E 2 CPU-Scheduling und Multithreading

Lernziele

- Die quasiparallele Ausführung von Prozessen genauer untersuchen.
- Die Eigenschaften verschiedener Prozessorzuteilungsstrategien anhand von Beispielen ermitteln.
- Multithreading unter Windows praktisch durchführen.
- Wichtige Regeln bei der Arbeit mit Threads kennen lernen.
- Mit den Beschreibungen von Systemfunktionen arbeiten lernen.

Übungsumgebung

- PC mit Windows XP
- Minimalist GNU for Windows (MinGW) installiert; alternativ Cygwin-Umgebung möglich
- Optional: MSDN installiert (Ausweichmöglichkeit **`http://msdn.microsoft.com`**)
- Vorlagedateien (C Quellcode): **`TwoThreads.c`**

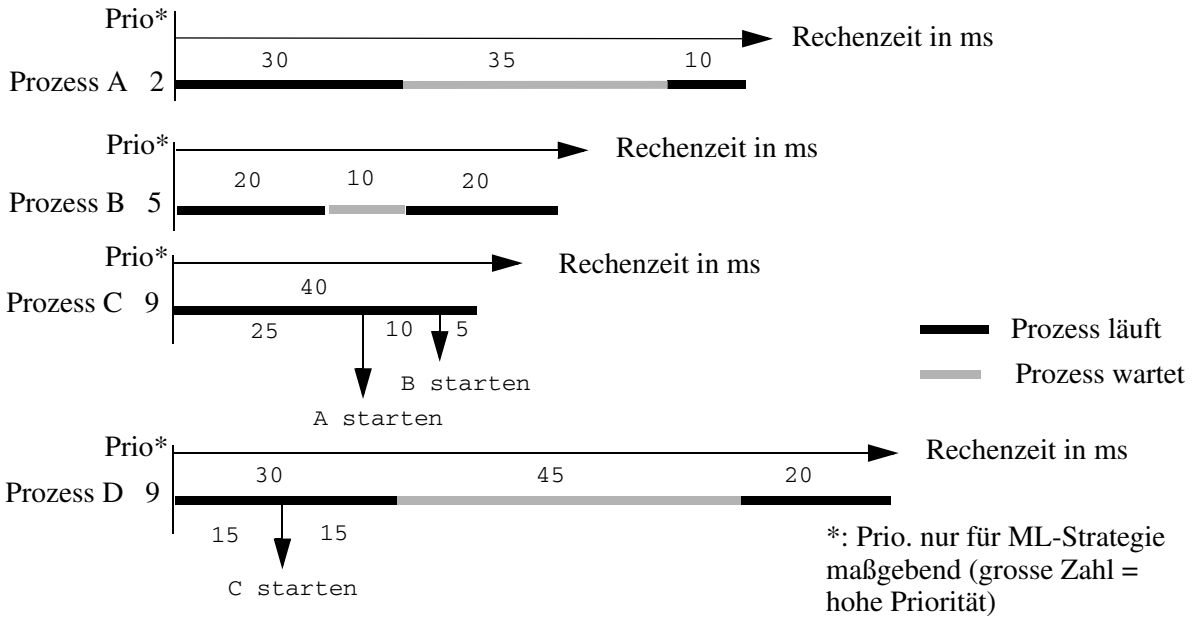
Allgemeine Hinweise

- Diese Übung besteht aus zwei theoretischen und zwei praktischen Aufgaben.
- Für diese Übung ist ein Windows-VMware-Image mit dem Namen *"Betriebssysteme-Windows"* vorbereitet. Der Benutzername ist: "hsr" und das Kennwort: "welcome".
- Musterlösungen der praktischen Aufgaben finden Sie im Übungsverzeichnis unter **`loesungen`**.

Aufgaben

E 2.1 Scheduling-Verfahren

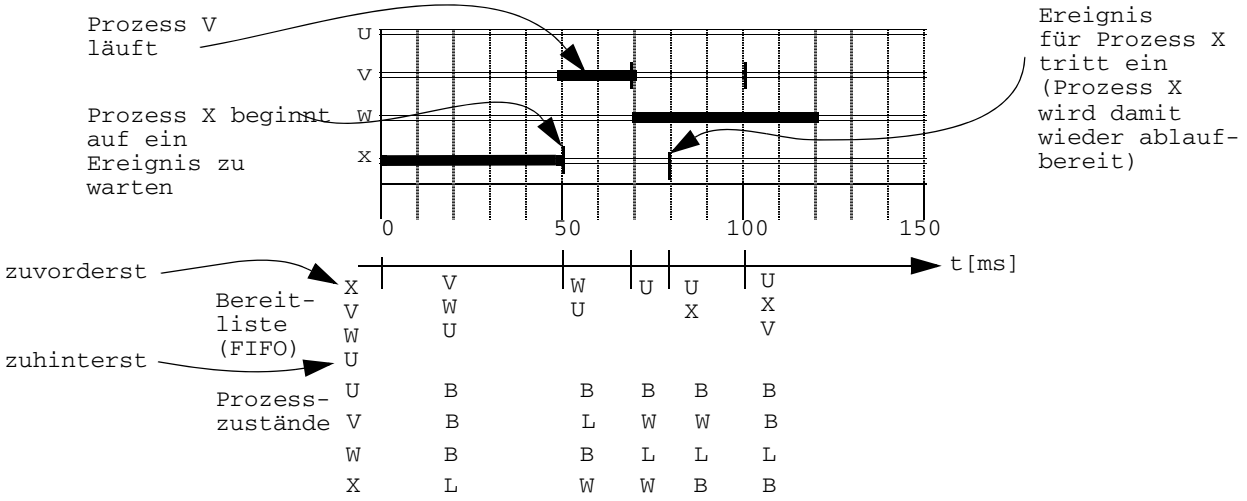
In einem Multitasking-Einprozessorsystem werden vier Prozesse A, B, C, D ausgeführt, die für sich alleine wie folgt ablaufen würden: Zum Zeitpunkt t = 0 wird Prozess D gestartet, der seinerseits nach 15 ms Rechenzeit Prozess C startet. Die Prozesse A und B werden durch Prozess C jeweils nach 25 bzw. 35 ms Rechenzeit gestartet.



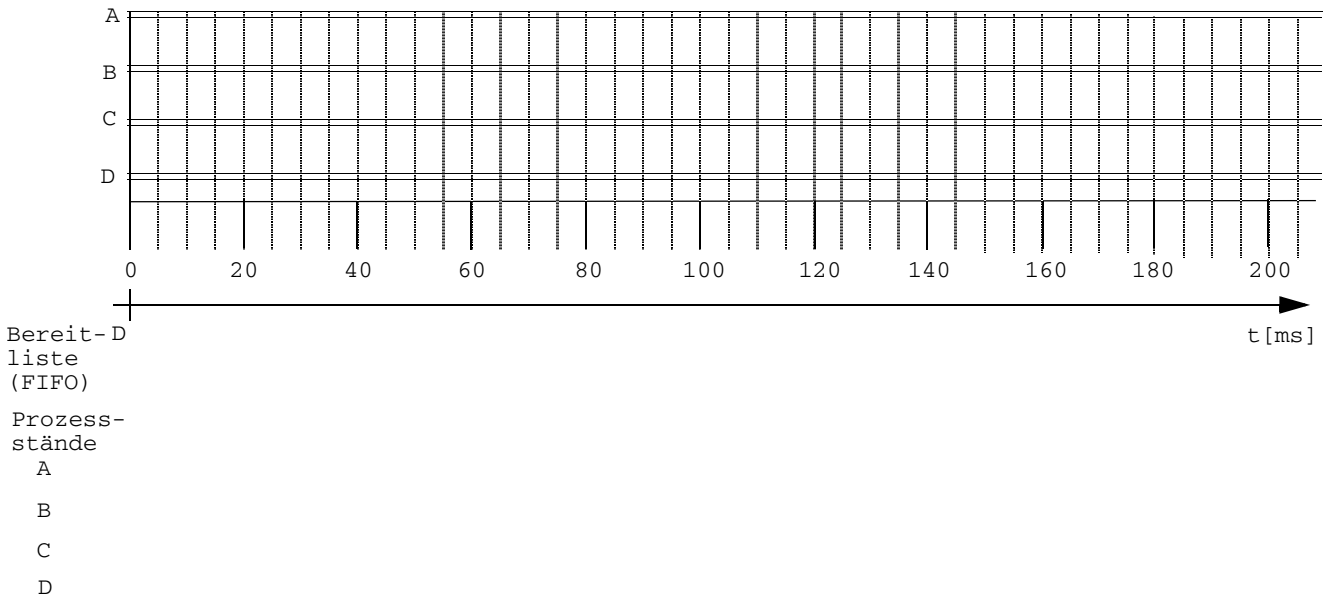
In den folgenden Teilaufgaben sollen die Abläufe für verschiedene Prozessorzuteilungsstrategien für die quasiparallele Ausführung auf einem Einprozessorsystem aufgezeichnet werden.

Dies beinhaltet (s. auch Muster unten):

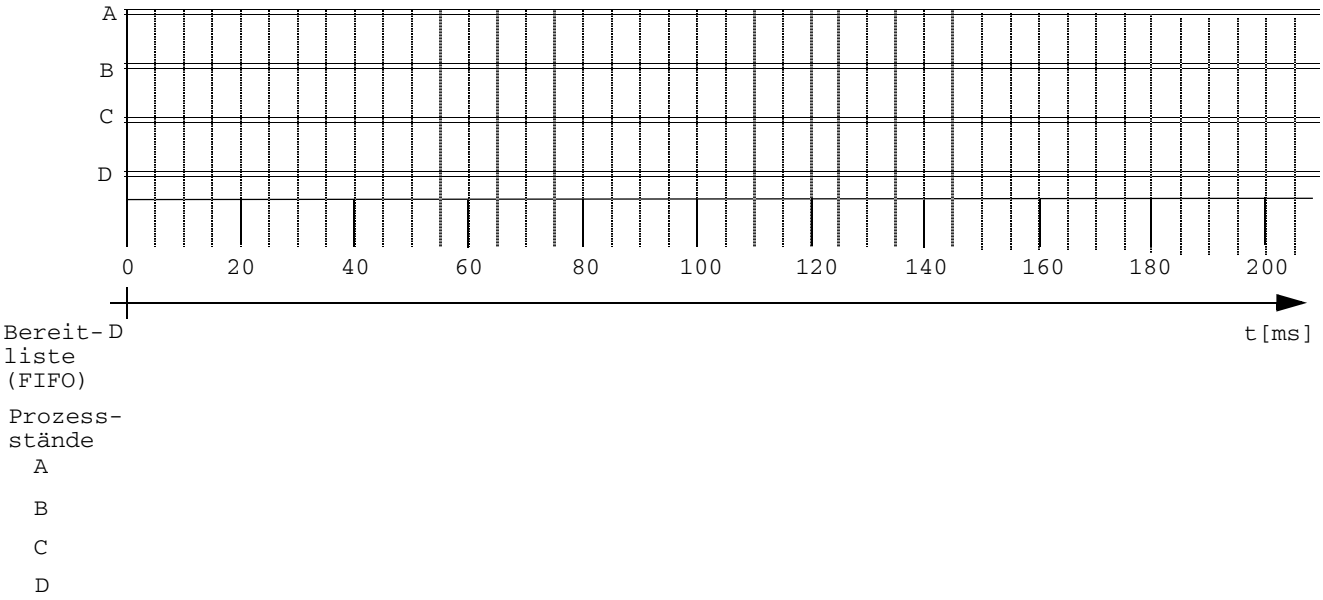
- Eintragen des jeweils laufenden Prozesses mit einem schwarzen Balken.
- Eintragen der einzelnen Ereignisse und des Wartebeginns auf diese (mit kleinem senkrechtem Strich). Die Warteereignisse treten jeweils nach der oben bezeichneten Zeit, gerechnet *ab Beginn des Wartens* durch den Prozess, ein.
- Eintragen der aktuellen Inhalte der Bereitliste (Ready-Liste) des Betriebssystems.
- Eintragen der aktuellen Prozesszustände (L für "Laufend", B für "Bereit", W für "Wartend", R für "Ruhend"). (s. Beispiel)
- Hinweis: Bei der ML-Strategie wird die Bereitliste *primär* nach Prioritäten und *sekundär* nach FCFS (First Come First Served) organisiert.



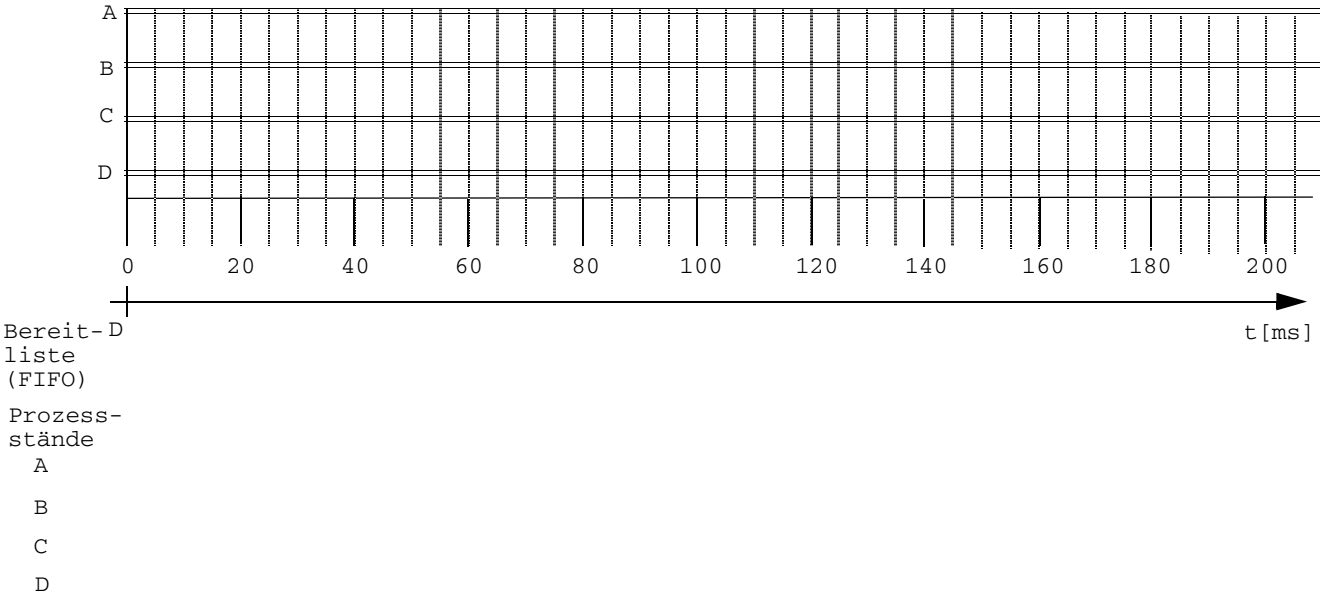
a) FIFO-Strategie (*FCFS, nonpreemptive*). Ein ablaufbereiter Prozess erhält erst dann den Prozessor zugeteilt, wenn er in der Bereitliste ist und der laufende Prozess den Prozessor selbst freigibt. Wird ein Prozess ablaufbereit und der Prozessor ist nicht belegt, so kann er sofort zugeteilt werden.



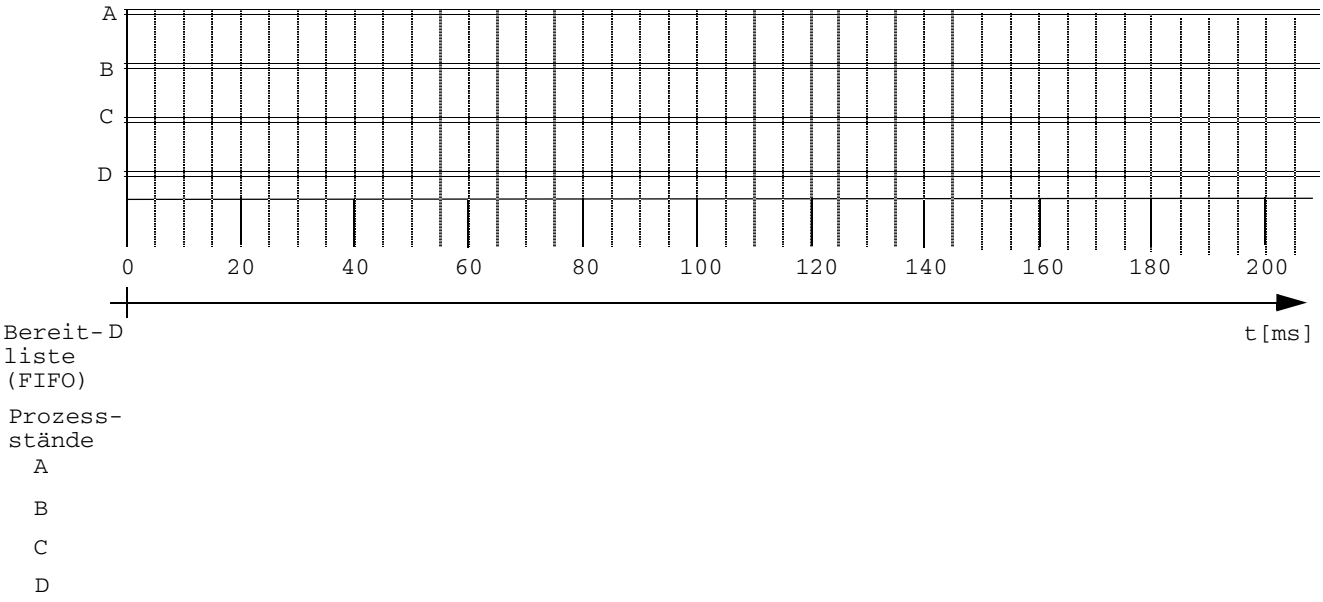
b) RR-Strategie (*round-robin, preemptive*). Die Zeitscheiben haben eine einheitliche Größe von 10 ms in einem festen Zeitraster. Beginnt ein Prozess vor Ende einer Zeitscheibe zu warten, so findet eine sofortige Neuzuteilung statt. Jeder Prozess erhält ein Zeitquantum von 2 Zeitscheiben. Beim Zeitscheibenende wird das Zeitquantum des laufenden Prozesses um 1 verkleinert. Erreicht das Zeitquantum 0, so wird dem Prozess der Prozessor entzogen. Dem vordersten Prozess in der Bereit-liste wird anschließend ein Zeitquantum von 2 und der Prozessor zugeteilt.



c) ML-Strategie (*priority based, preemptive*). Auf der höchsten Prioritätsstufe ablaufbereiter Prozesse komme *zusätzlich* noch die RR-Strategie mit einem fixen Zeitraster von 10 ms zum Tragen (Zeitquantum = 2 Zeitscheiben).



d) Es stehe ein Multiprozessorsystem mit vier Prozessoren zur Verfügung. Wie sieht der Ablauf aus? Warum spielt es keine Rolle, welche der Strategien aus Teilaufgabe a)..c) gewählt wird?

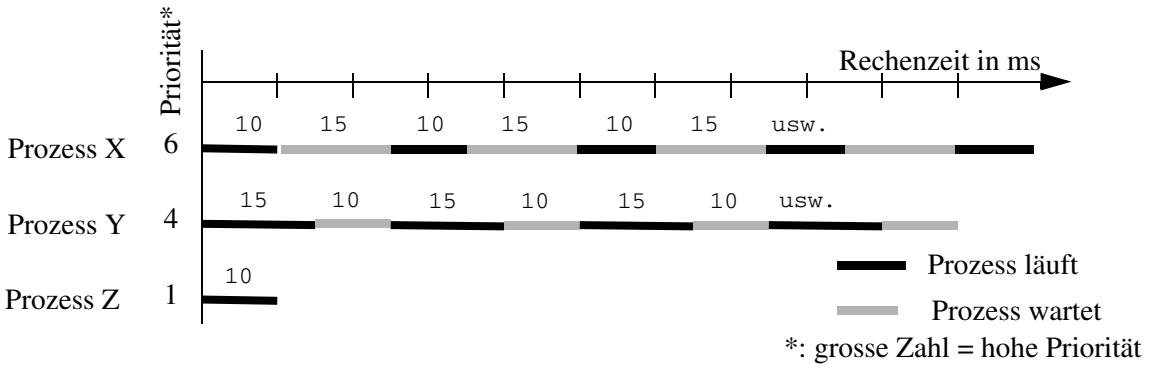


e) Diskutieren Sie die Lösungen hinsichtlich zeitgerechter Ausführung der zwei wichtigsten Prozesse C und D und bezüglich des gesamten Zeitbedarfs.

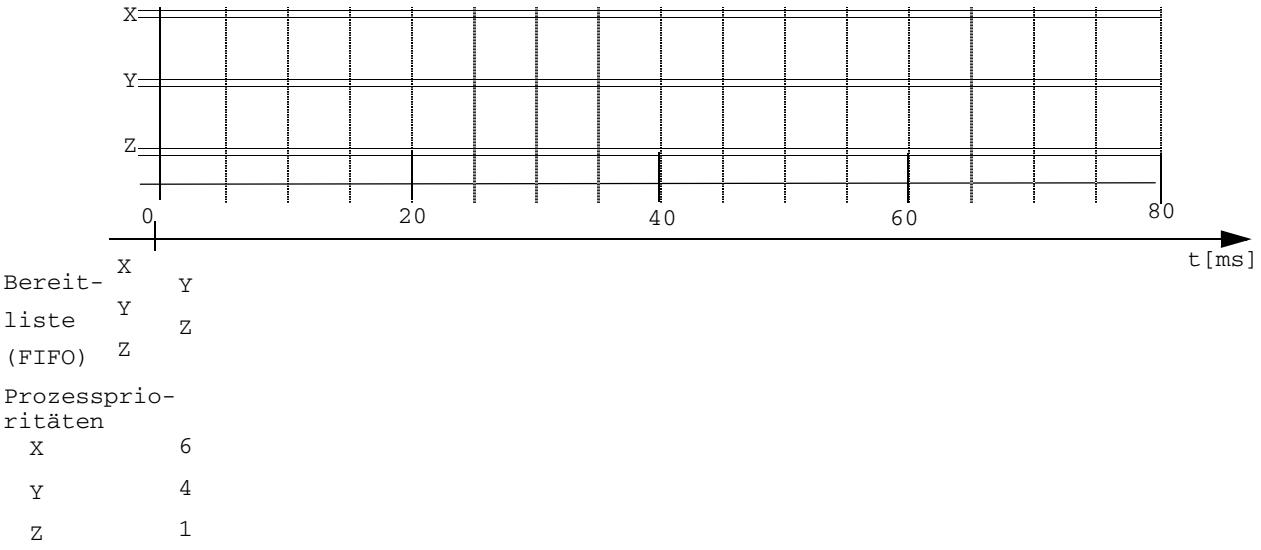
f) Berechnen Sie die mittlere Prozessorauslastung (in Prozent) für die Teilaufgaben a)..c) für den Zeitraum 100..200 ms.

E 2.2 Verköstigung der Armen

Gewisse Betriebssysteme benutzen *dynamische Prioritätsanhebungen*, damit Prozesse tiefer Priorität nicht verhungern, d.h. irgendwann auch einmal den Prozessor erhalten.
Es gelte die Strategie nach c) aus Aufgabe 2 mit folgender Erweiterung: Die Priorität eines Prozesses wird für eine neue Zeitscheibe (Zeitscheibengröße 10 ms) dann um 1 erhöht, wenn der Prozess in der alten Zeitscheibe bereits "bereit" war und in der neuen Zeitscheibe immer noch "bereit" bleibt. Bekommt ein Prozess den Prozessor zugeteilt, so wird seine Priorität beim nächstfolgenden Zeitscheibenende auf den Anfangswert zurückgesetzt.
Ermitteln Sie den Ablauf für die drei Prozesse X, Y und Z in einem Multitasking-Einprozessorsystem, die für sich alleine wie folgt ablaufen würden:



Resultierender Ablauf (nur Bereich 0..80 ms):



E 2.3: Programm mit zwei Threads

Für diese Aufgabe gehen Sie von der Vorlagedatei **`TwoThreads.c`** aus. Diese enthält eine noch nicht vollständige Vorlage für die Thread-Erzeugung unter Windows. Daraus soll eine Windows Konsolenapplikation mit zwei Threads entstehen.

Eine Windows-Konsolenapplikation benutzt für die Ein-/Ausgabe eine *Textkonsole (Kommandofenster)* und kein *GUI (Graphical User Interface)*. Wird eine Konsolenapplikation gestartet, so wird eine Ausführungsumgebung vom Betriebssystem bereitgestellt und die C *main()*-Funktion als *primary thread* gestartet.

Hinweis: die Signatur der Windows **`CreateThread()`**-Funktion ist in der MSDN beschrieben. Die MSDN stellt eine Informationssammlung zur Windows-Programmierung dar. Diese Informationssammlung kann im Web über den URL **`http://msdn.microsoft.com`** angesprochen werden. Beachten Sie, dass es sich um eine Vielzahl von Informationen handelt, von denen die Beschreibung der in dieser Übung benutzen Systemfunktion nur ein kleiner Teil darstellt.

Die Beschreibungen zu den Win32-Systemfunktionen finden Sie in der MSDN unter:

MSDN->MSDN Library->Win32 and COM Development->System Services

Informationen zu Funktionen für die Prozess- und Thread-Erzeugung finden Sie dort unter:
->DLLs, Processes, and Threads->Processes and Threads

a) Erweitern Sie das C-Codefragment zu einer vollständigen Konsolenapplikation mit zwei Threads (d.h. *primary thread* und ein weiterer Thread). Der weitere Thread kann eine Ausgabe auf das Kommandofenster machen oder sonst etwas (wählen Sie selbst; z.B. Aufruf von **`Sleep()`**). Wenn Sie das Programm vervollständigt haben, können Sie es auf der Kommandozeile übersetzen mit:
`gcc TwoThreads.c -o TwoThreads`
Damit wird das ausführbare Programm **`TwoThreads.exe`** erzeugt, sofern keine Fehler aufgetretenn sind. Das Programm **`TwoThreads.exe`** können Sie von der Kommandozeile aus starten oder über den *windows file explorer.*
Achtung: wenn der *primary thread* sich beendet, dann werden auch alle weiteren Threads terminiert. Beugen Sie dieser Situation beim Programmentwurf vor!

b) Dem zweiten Thread soll zur Datenverarbeitung ein Ganzzahlwert (integer) bereitgestellt werden. Welche grundsätzlichen Möglichkeiten bestehen dazu?

.....

.....

E 2.4: Warten auf Thread-Terminierung

Vermutlich haben Sie in der Aufgabe 1 eine nicht sehr elegante Methode gewählt, um die Beendigung des *primary threads* zu verzögern, damit der zweite Thread zu Ende laufen kann.

Hier nun die Methode, wie man dies besser lösen kann: bei der Threaderzeugung wird ein Deskriptorwert (sog. Handle) an den Aufrufer von **`CreateThread()`** returniert. Dieser Handle identifiziert ein dem Thread zugeordnetes Systemobjekt (=systeminterne Datenstruktur), das zwei Zustände kennt: *signalisiert* bzw. *nicht signalisiert*. Nun das Wichtige für uns: ein Threadobjekt wechselt erst dann in den signalisierten Zustand, wenn der zugehörige Thread terminiert hat.

Mit der Systemfunktion **`WaitForSingleObject()`** können wir nun also im *primary thread* genau solange warten, bis der zweite Thread terminiert hat. Testen Sie dies aus!

Wie kann das gleiche Problem gelöst werden, wenn auf die Beendigung mehrerer Threads gewartet werden muss?

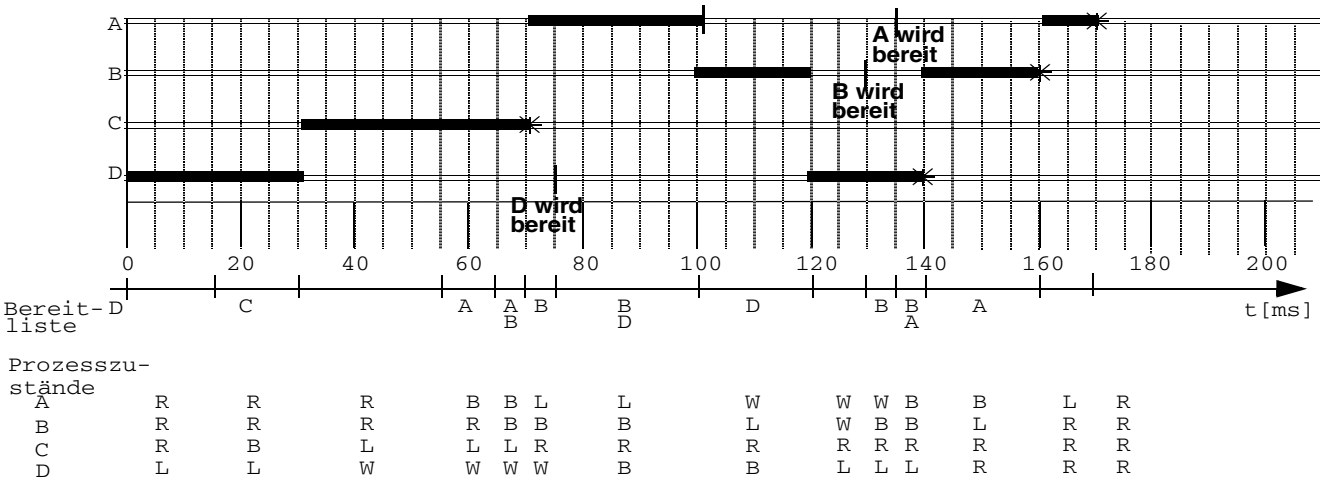
.....

.....

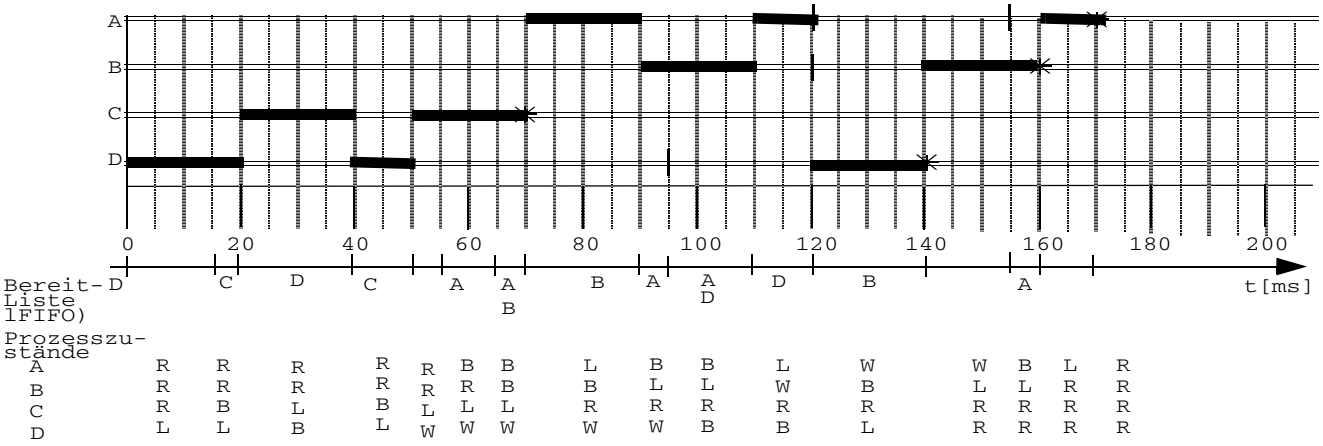
E 2 Musterlösung: CPU-Scheduling

E 2.1: Scheduling-Verfahren

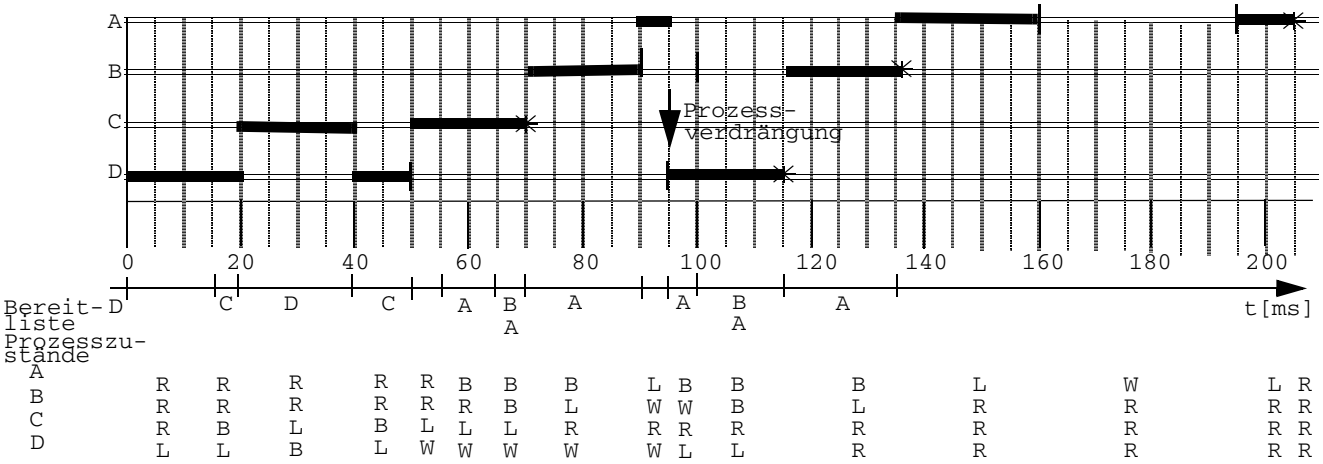
a) FIFO-Strategie (*FCFS, nonpreemptive*)



b) RR-Strategie (*round-robin, preemptive*)

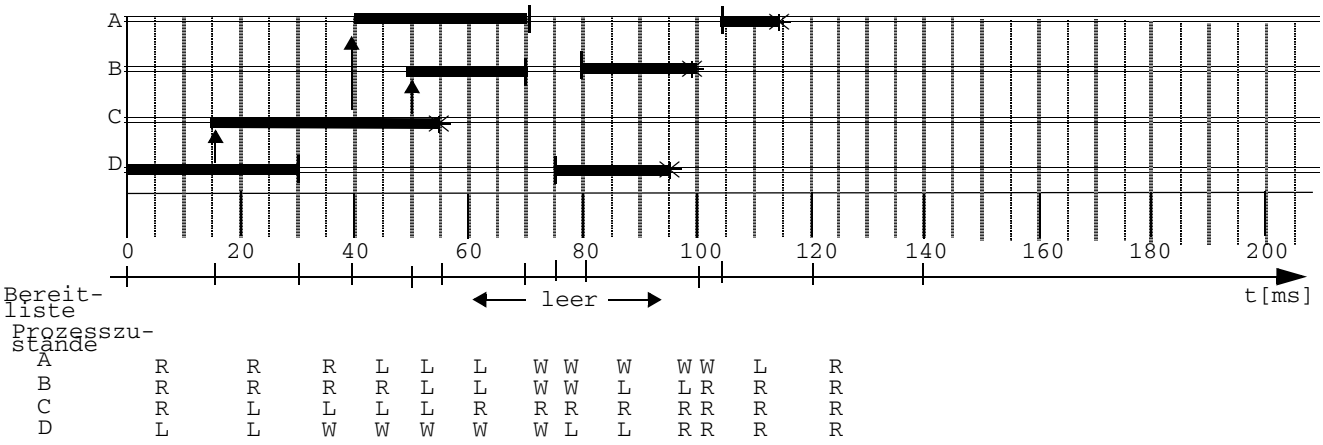


c) ML-Strategie (*priority based, preemptive*)



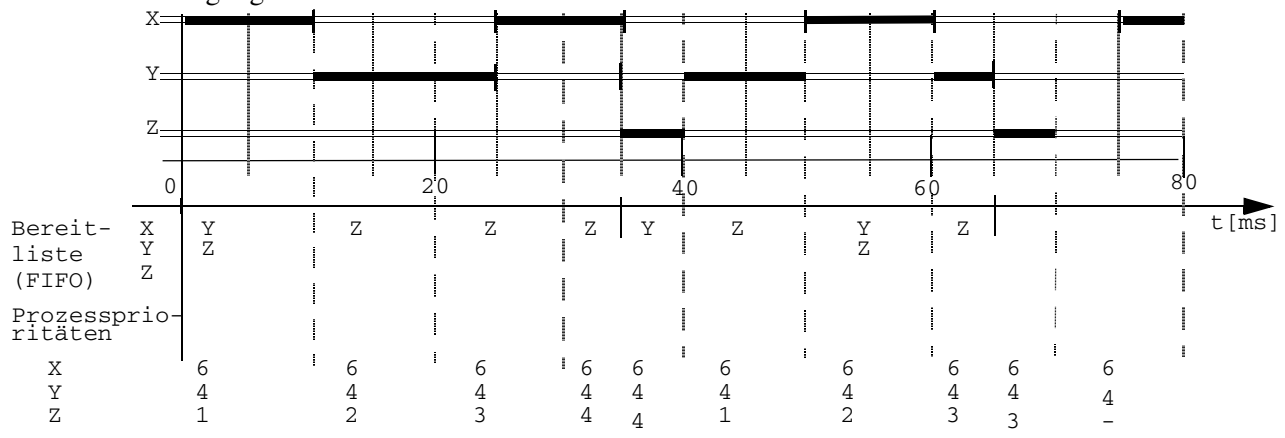
Hinweis: Bei diesem Verfahren wird die Bereitliste *primär* nach Prioritäten und *sekundär* nach FCFS (First Come First Served) organisiert.

d) Multiprozessorsystem:



f) Prozessorauslastung a) 70% b) 70% c) 65%

E 2.2: Verköstigung der Armen



E 2.3: Programm mit zwei Thread

a) siehe Lösungsmuster **`aufg1.c`** unter **`/loesungen`**

b) - Bereitstellung als Initialparameter beim Aufruf der Funktion **`CreateThread()`**
- Bereitstellung in globaler Variable (diese Lösung ist nicht funktionsfähig, wenn z.B. ein Thread zur Bearbeitung einer Anfrage gestartet wird und dies evtl. so schnell erfolgt, dass mehrere Anfragen parallel bearbeitet werden müssen)

E 2.4: Warten auf Thread-Terminierung

Unter Benutzung der Systemfunktion **`WaitForMultipleObjects()`**. Erlaubt das gleichzeitige Warten auf mehrere Threads (z.Zt. limitiert auf max. 64)