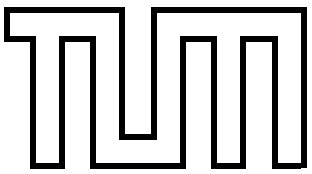

Technische Universität München

Fakultät für Informatik
Forschungs- und Lehreinheit Informatik VI

Übung zur Vorlesung Echtzeitsysteme

Aufgabe 10 – Scheduling Algorithms

Nadine Keddis
keddis@fortiss.org

Dominik Sojer
sojer@in.tum.de

Stephan Sommer
sommerst@in.tum.de

Michael Geisinger
geisinge@in.tum.de

Wintersemester 2011/12

Aufgabe 10: Scheduling

Allgemeines zu Scheduling

Der Scheduler (siehe Abbildung 1) ist ein Modul eines Betriebssystems, das die Rechenzeit an die unterschiedlichen Prozesse verteilt. Der ausgeführte Algorithmus wird als Scheduling-Algorithmus bezeichnet.

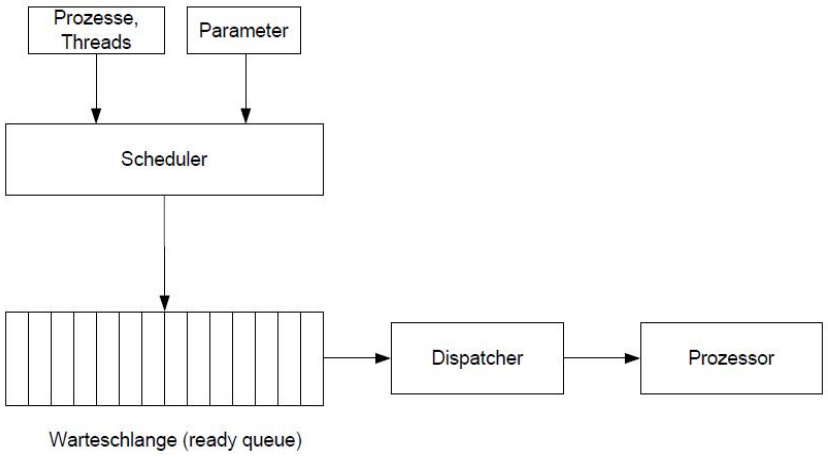

Abbildung 1: Scheduler

In dieser Übung werden folgende Scheduling Algoritmen betrachtet:

- Least Slack Time nicht-präemptiv;
- Earliest Deadline First nicht-präemptiv;
- Least Slack Time präemptiv;
- Earliest Deadline First präemptiv;

Die Scheduling Algorithmen unterscheiden sich durch mehrere Faktoren, unter anderem können sie präemptiv oder nicht-präemptiv sein. Präemptives (bevorrechtigt, entziehend) Scheduling zeichnet sich dadurch aus, dass bei jedem Auftreten eines relevanten Ereignisses die aktuelle Ausführung eines Prozesses unterbrochen wird und eine neue Schedulingentscheidung getroffen wird.

EDF nicht-präemptiv Scheduling

Wenn zu einem bestimmten Zeitpunkt mehr als ein Prozess zur Ausführung bereit ist, bekommt der Prozess mit der frühesten Deadline den Prozessor und wird ausgeführt. Wenn seine Ausführung zu Ende ist, wird nach dem gleichen Prinzip die nächste Scheduling Entscheidung getroffen. Ein Beispiel ist im Bild 2 dargestellt.

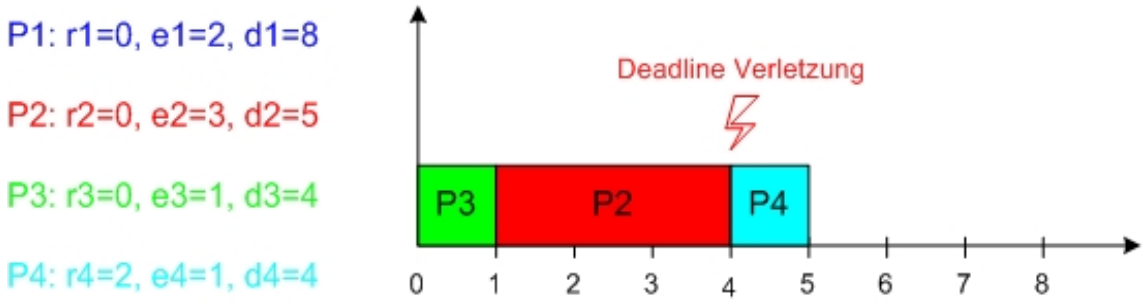

Abbildung 2: Scheduler

LST nicht-präemptiv Scheduling

Wenn zu einem bestimmten Zeitpunkt mehr als ein Prozess zur Ausführung bereit ist, bekommt der Prozess mit der kleinsten *SlackTime* den Prozessor und wird ausgeführt. Wenn seine Ausführung zu Ende ist, wird nach dem gleichen Prinzip die nächste Scheduling Entscheidung getroffen. Ein Beispiel kann man dem Bild 3 entnehmen.

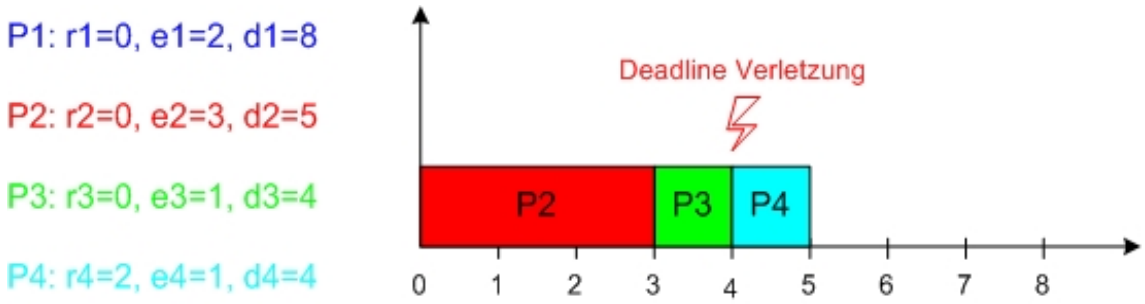

Abbildung 3: Scheduler

EDF präemptiv Scheduling

Wenn zu einem bestimmten Zeitpunkt mehr als ein Prozess zur Ausführung bereit ist, bekommt der Prozess mit der frühesten Deadline den Prozessor und wird ausgeführt. Er wird mindestens für einen Zeitabschnitt *Delta* (ein Parameter, im Beispiel *Delta* ist gleich 1) ausgeführt. Anschliessend wird geprüft, ob ein anderer Prozess mit einer früheren Deadline auf die Ausführung wartet. Wenn ja, wird der ausgeführte Prozess unterbrochen und der unterbrechende Prozess ausgeführt. Nach einem weiteren Zeitintervall *Delta* wird die nächste Scheduling Entscheidung nach dem gleichen Prinzip getroffen. (Siehe Beispiel in Bild 4).

LST präemptiv Scheduling

Wenn zu einem bestimmten Zeitpunkt mehr als ein Prozess zur Ausführung bereit ist, bekommt der Prozess mit der kleinsten Slack Time den Prozessor und wird ausgeführt. Er wird mindestens eine Zeit *Delta* (ein Parameter, im Beispiel *Delta* ist gleich 1) ausgeführt.

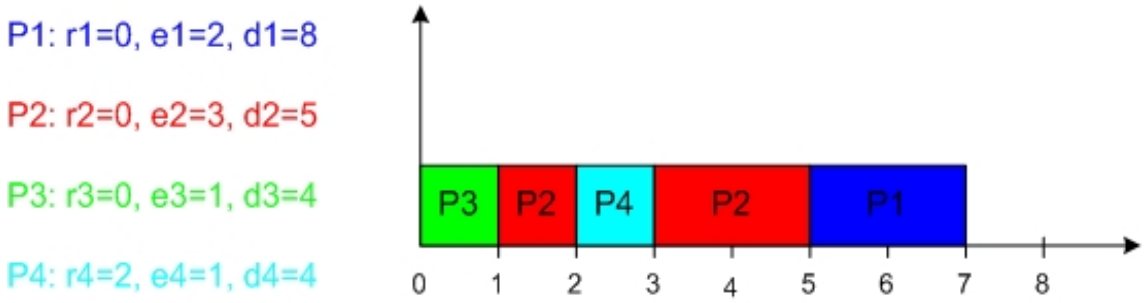

Abbildung 4: Scheduler

Dann wird geprüft, ob ein anderer Prozess mit einer kleineren Slack Time auf die Ausführung wartet. Wenn ja, wird der ausführende Prozess unterbrochen und der unterbrechende Prozess ausgeführt. Nach einem Zeitintervall *Delta* wird die nächste Scheduling Entscheidung nach dem gleichen Prinzip getroffen. Ein Beispiel kann man dem Bild 5 entnehmen.

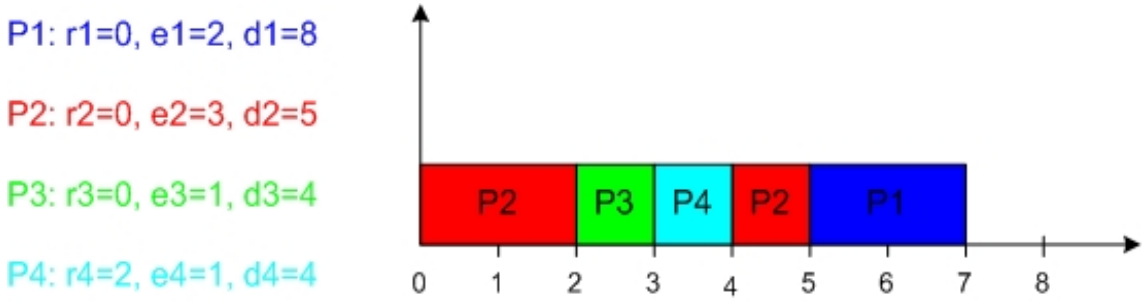

Abbildung 5: Scheduler

Allgemeines zu dem Rahmenprogramm

Im Rahmenprogramm ist ein Task als folgende Struktur definiert:

```
typedef struct task{
        int taskID;
        int readyTime;
        int executionTime;
        int deadline;
        int processed;
        int slackTime;
        struct task *next;
} task_t;
```

Die Tasks werden in einer verketteten Liste organisiert, wobei *task_t *taskList* ein Pointer auf das erste Element ist. Alle notwendige Funktionen zur Verwaltung der Liste werden bereitgestellt:

- task_t *getTaskWithLeastSlackTime(int currentTimestamp);
- task_t *getTaskWithEarliestDeadline(int currentTimestamp);
- void addToTaskList(int taskID, int readyTime, int executionTime, int deadline);
- int deleteFromList(int taskID);
- void clearTaskList();

Ausserdem gibt es folgende nütziche Funktionen:

- void printTasksInfo();
- void printTaskIDs();
- void printReadyTaskIDs(int currentTimestamp);
- void calculateSlackTimes(int currentTimestamp);
- void printSlackTimes(int currentTimestamp);
- void printDeadlines(int currentTimestamp);

Aufgabe 10.1-10.3

Nehmen Sie den nicht-präemptiven EDF Scheduler (ist in der Funktion scheduleEDFnon-Priemptive() realisiert) als Beispiel und realisieren Sie dann ...

- (Aufgabe 10.1) einen nicht-präemptiven LST Scheduler in der Funktion scheduleLST-nonPriemptive().
- (Aufgabe 10.2) einen päemptiven EDF Scheduler in der Funktion scheduleEDFpri-emptive().
- (Aufgabe 10.3) einen päemptiven LST Scheduler in der Funktion scheduleLSTpri-emptive().

Nutzen Sie dazu folgende Funktionen:

- task_t *getTaskWithEarliestDeadline(int currentTimeStamp);
- task_t *getTaskWithLeastSlackTime(int curruntTimeStep);

Als Testset nutzen Sie die 4 fogneden Prozesse:

P1: r1=0, e1=2, d1=8
P2: r2=0, e2=3, d2=5
P3: r3=0, e3=1, d3=4
P4: r4=2, e4=1, d4=4

Die Prozesse werden zu der Liste der zu bearbeitenden Prozesse mit der Funktion

```
void addToTaskListSet1();
```

hinzugeügt.

Die Ergebnisse Ihrer schedulers sollen mit dem übereinstimmen, was im Abschnitt 'Allgemeines zum Scheduling' auf den Bildern dargestellt ist.

Aufgabe 10.4

Wann und wie kann feststellt werden, dass ein Deadline verletzt wird? Schauen Sie sich die folgenden Funktionen an:

```
int checkDeadlineViolationEDF(int currentTimeStamp);
int checkDeadlineViolationLST(int curruntTimeStep);
```

die bei dem EDF und LST Scheduling festlegen, ob ein Deadline verletzt wurde. Vergessen Sie nicht, mittels diese Funktionen beim Scheduling zu überprüfen, ob ein Deadline verletzt wurde.

Führen Sie jetzt Ihr Programm mit dem zweiten Testset von Prozessen aus:

P1: r1=0, e1=2, d1=6
P2: r2=0, e2=3, d2=5
P3: r3=0, e3=1, d3=4
P4: r4=2, e4=1, d4=4

Sie werden zu der Liste der zu bearbeitenden Prozesse mit der Funktion

```
void addToTaskListSet2();
```

eingefügt.

Warum kommt es mit **jedem** Verfahren zu einer Deadline Verletzung?