Technische Universität Berlin Fachgebiet Komplexe und Verteilte IT-Systeme Sommersemester 2015	**Aufgabenblatt 3** zu – Systemprogrammierung – Prof. Dr. Odej Kao, Dr. Peter Janacik, Tutoren
Abgabetermin:	[1] – 25.05.2015 [2] – 01.06.2015

Aufgabe 3.1: Theorie (1 Punkt) (Theorie[1])

In einem Ein-Prozessor-System werden Prozesse, wie in folgender Tabelle dargestellt, gestartet:

Prozess	A	B	C	D	E
Ausführungszeitpunkt	0	2	5	6	9
Laufzeitdauer	6	1	8	12	3

Abbildung 1: Prozesse eines Ein-Prozessor-Systems

a) Simulieren Sie die Prozessorzuteilung für die Scheduling-Verfahren

- SJN,
- LCFS-PR,
- Round Robin (jeweils mit $\tau = 1$ und $\tau = 3$) sowie
- Multilevel-Feedback (mit $\tau_i = 2^i$ $(i = 0, 1, \dots)$).

Geben Sie für jeden Zeitpunkt den Inhalt der Warteschlange mit an.

Die Lösung soll in der Form der im Folgenden dargestellten Tabelle abgegeben werden:

Zeit	0	1	2	3	…	30	31
CPU	A	…	…	…	…	…	…
Warteschlange		…	…	…	…	…	…
		…	…	…	…	…	…

b) Berechnen Sie jeweils

- die Warte- und Antwortzeit für *jeden* Prozesses sowie
- die mittlere Warte- und Antwortzeit des Systems

für alle unter a) verwendeten Verfahren.

Aufgabe 3.2: Periodische Prozesse (1 Punkt) (Theorie[1])

Auf dem Bordcomputer eines Autos sollen Prozesse periodisch ausgeführt werden.

- Prozess A aggregiert Sensordaten über die Umdrehungszahl der Räder.
- Diese Datensätze werden vom Antiblockiersystem (Prozess B) verarbeitet.
- Prozess C überprüft Kollisions-Sensoren und löst, wenn nötig, den Airbag aus.

Die Prozess-Charakteristika können Sie der folgenden Tabelle entnehmen:

Prozess	Startzeitpunkt	Laufzeit	Periode
A	0	1	3
B	1	4	15
C	2	2	5

Abbildung 2: Prozess-Charakteristika

Die Frist (*Deadline*) eines Prozesses entspricht in diesem Fall seiner Periode.

a) Existiert für diese Prozesse ein zulässiger Schedule? Wenn ja: Wie könnte dieser aussehen? Wie groß ist die Hyperperiode?

b) Ändert sich daran etwas, wenn zu den Prozessen ein weiterer (D: 4, 2, 6) hinzugefügt wird?

c) Was ist der Unterschied zwischen Hard- und Softlimits bei Echtzeitsystemen? Geben Sie für beides Beispiele an.

Aufgabe 3.3: Scheduling-Verfahren (Tafelübung)

Benennen Sie die aus der Vorlesung bekannten Scheduling-Verfahren und ordnen Sie diese nach

a) Strategiealternativen:

- ohne/mit Verdrängung,
- ohne/mit Prioritäten und
- unabhängig/abhängig von der Bedienzeit.

b) Betriebszielen:

- Effizienz/Durchsatz,
- Antwortzeit und
- Fairness.

Begründen Sie Ihre Entscheidungen für die Betriebsziele!

Aufgabe 3.4: Prozessorausnutzung (Tafelübung)

Die Prozessorausnutzung sei als Quotient aus der minimal erforderlichen und der tatsächlich benötigten Zeit zur Ausführung anstehender Prozesse definiert. Dabei soll die Laufzeit eines Prozesses T Zeiteinheiten betragen und ein Prozesswechsel S Zeiteinheiten kosten (mit $S \ll T$, also S wesentlich kleiner als T).

a) Geben Sie eine alternative Formel zur Berechnung der Prozessorausnutzung für das Round-Robin-Verfahren unter Verwendung der Zeitscheibenlänge τ und der Prozesszahl n an.

b) Stellen Sie die Abhängigkeit von Effizienz und Zeitscheibenlänge grafisch dar.

c) Berechnen Sie anhand der Formel aus a) die Grenzwerte für folgende Fälle:

- $\tau \to 0$,
- $\tau = S$ und
- $\tau \to \infty$.

Aufgabe 3.5: Scheduler (3 Punkte) (Praxis^2)

In dieser Aufgabe sollen verschiedene Schedulervarianten implementiert werden.

Jeder Scheduler `XX` soll dabei als Interface die folgenden Funktionen anbieten:

- `int init_XX()`
  Wird vor dem Start der Tests aufgerufen und soll den Scheduler in einen Startzustand versetzen. Hier können Sie bei Bedarf auch benötigte Datenstrukturen initialisieren. Die Funktion soll 1 zurückgeben, wenn der Scheduler bereits implementiert wurde, sonst 0.

- `void free_XX()`
  Gibt durch den Scheduler eventuell allozierten Speicher frei.

- `void arrive_XX(int id, int length)`
  Wird jedesmal aufgerufen, wenn ein neuer Prozess hinzukommt. Es wird für jeden Prozess eine Kennung und die Gesamtlaufzeit in Zeiteinheiten übergeben. Prozesse kommen dabei immer am Anfang einer Zeiteinheit an, d.h. noch bevor `tick_XX()` aufgerufen wird.

- `void tick_XX()`
  Wird bei Zeitscheiben-basierten Algorithmen vor Beginn jeder Zeiteinheit einmal aufgerufen.

- `void finish_XX()`
  Wird jedesmal am Ende der Zeiteinheit, in der ein Prozess seine Arbeit beendet hat, aufgerufen.

Im Scheduler sollen die Prozesse verwaltet und zu passenden Zeitpunkten aus `arrive()`, `tick()` oder `finish()` heraus Prozesswechsel durchgeführt werden. Um einen Prozesswechsel durchzuführen steht ihnen (über die Datei `task.c`) folgende Funktion zur Verfügung:

- `void switch_task(int id)`
  Wechselt zum Prozess mit der übergebenen ID. Falls alle Prozesse abgearbeitet wurden, soll dies mit einem Wechsel zu der vordefinierten ID `IDLE` signalisiert werden.

Achtung: Das oben beschriebene Interface soll von der vorgegebenen Testumgebung in der Datei `main.c` aus aufgerufen werden. Es ergibt im Allgemeinen keinen Sinn, wenn Sie die entsprechenden Funktionen direkt innerhalb ihrer Scheduler-Implementierung aufrufen.

Implementieren Sie einen ...

a) ... Scheduler, der nach dem LCFS-Verfahren arbeitet.

b) ... Scheduler, der nach dem SJN-Verfahren (nicht präemptiv) arbeitet.

c) ... Scheduler, der nach dem SRTN-Verfahren arbeitet.

d) ... Multilevel-Feedback-Scheduler unter Verwendung von Queues mit folgenden Eigenschaften:

- Die `init`-Funktion erwartet die zwei Parameter `time_step` (Zeitscheibe der ersten Queue) und `num_queues` (Anzahl der Queues, siehe nächster Punkt).
- Dabei gibt `num_queues` die Anzahl der Queues *inklusive* der FIFO-Queue am Ende an. Bsp.: `num_queue = 4` bedeutet es gibt 3 Zeitscheiben-Queues und eine FIFO-Queue.
- Die Zeitscheiben für die erste Queue ergibt sich aus dem Parameter `time_step`. Für die Zeitscheibe der Queue n gilt: Zeitscheibe = `time_step` $\cdot$ n für n = $[1, $`num_queues`$]$.
- Am Beispiel für `time_step = 2` und `num_queues = 4`:
  - Queue 1: `time_step = 2`
  - Queue 2: `time_step = 4`
  - Queue 3: `time_step = 6`
  - Queue 4: FIFO

Hinweise:

- **Datenstrukturen:** Sie werden für die Implementierung (nicht vorgegebene) Datenstrukturen benötigen. Greifen Sie hier z.B. auf ihre Ergebnisse des ersten Aufgabenblattes zurück.
- **Start des ersten Tasks:** Beachten Sie, dass beim LCFS- und SJN-Verfahren ein an die `arrive`-Funktion übergebener Task sofort starten soll, falls nur der `IDLE`-Task läuft. Dagegen startet ein Task beim SRTN- und MLFB-Verfahren *nur* nach Verwendung der `tick`-Funktion.
- **Ungenutze Funktionen:** Die Verfahren aus a) und b) verwenden keine Zeitscheiben, daher ist die `tick`-Funktion dort nicht zu implementieren. Die Verfahren aus c) und d) können des Weiteren auch ohne die `finish`-Funktion vollständig abgebildet werden.
- **Makefile 1:** Wenn Sie das `Makefile` ausführen, sollte eine ausführbare Datei mit dem Namen `scheduler` erstellt werden, die bei Ausführung die in der Datei `main.c` beschriebene Testumgebung ausführt. Für einen vollständigen Test müssen Sie diese selbst erweitern.
- **Makefile 2:** Falls Sie bei der Implementierung zusätzliche Datenstrukturen erstellen, müssen sie diese wie alle anderen Dateien im Makefile unter `SRC` eintragen. Beachten Sie, dass sie jeweils nur die Quelldatei, jedoch nicht die dazugehörige Headerdatei, eintragen müssen.