Technische Universität Berlin Fachgebiet Komplexe und Verteilte IT-Systeme Sommersemester 2020	**Aufgabenblatt 3** zu – Systemprogrammierung – Prof. Dr. Odej Kao
Abgabetermin: **14.6.2020 23:55 Uhr**	

Aufgabe 3.1: 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 (BZ).

b) Betriebszielen:

- Effizienz/Durchsatz,
- Antwortzeit und
- Fairness.

Begründen Sie Ihre Entscheidungen für die Betriebsziele!

Aufgabe 3.2: Scheduling-Handsimulation (Tafelübung)

Es werden in einem Ein-Prozessor-System Prozesse wie in Abbildung 1 beschrieben gestartet:

Prozess	A	B	C	D
Ankunftszeitpunkt	0	3	4	8
Dauer	5	3	8	4
Priorität	4	1	2	3

Abbildung 1: Prozesse eines Systems mit einer CPU und einem Thread

a) Simulieren Sie folgende Scheduling-Verfahren für die Prozesse aus Abbildung 1 unter Verwendung des SysprogInteract Tools (https://citlab.github.io/SysprogInteract/):

- FCFS,
- PRIO-NP,
- SRTN,
- RR mit $\tau = 2$.

b) Simulieren Sie das MLF Scheduling-Verfahren für die Prozesse aus Abbildung 1 an der Tafel mit $\tau_i = 2^i \ (i = 0, 1, \dots)$. Gehen Sie dabei von einer vereinfachten Variante ohne Verdrängung aus. Welchen Vorteil hätte Verdrängung?

c) Berechnen Sie für jedes der verwendeten Verfahren

- die Warte- und Antwortzeit *jedes* Prozesses sowie
- die mittlere Warte- und Antwortzeit des gesamten Systems.

Aufgabe 3.3: Prozessorausnutzung (Tafelübung)

Die Prozessorausnutzung ρ 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 (es gilt: $S \ll T$).

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

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

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

Aufgabe 3.4: Scheduling-Theorie (2 Punkte) (Theorie)

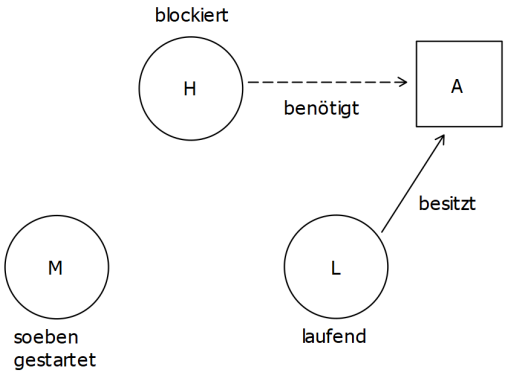

Abbildung 2: Prozesse eines Systems unter Verwendung eines verdrängenden Scheduling-Verfahrens mit Prioritäten

a) Zwischen welchen zwei Schedulingzielen bildet das HRRN-Verfahren einen Kompromiss? **(0,5 Punkte)**

b) Erklären sie das Phänomen der *Prioritätsinvertierung*. Gehen Sie dabei davon aus, dass drei Prozesse *H*, *M* und *L* mit respektive hoher, mittlerer und geringer Priorität scheduled werden sollen. *H* und *L* benötigen das exlusive Betriebsmittel *A*, welches derzeit von *L* besetzt ist. Nun wird der Prozess *M* gestartet (momentane Situation wie in Abbildung 2). **(0,3 Punkte)**

c) Was ist ein möglicher Lösungsansatz für dieses Problem? Erläutern Sie diesen kurz.**(0,3 Punkte)**

d) Was sind die Unterschiede zwischen *Online-* und *Offline-Scheduling*? Gehen sie dabei auch auf die benötigten Voraussetzung für beide ein. **(0,4 Punkte)**

e) Was sind die Unterschiede und Gemeinsamkeiten von *Hard-* und *Soft-real-time-Systems*. Nennen sie jeweils ein Beispiel. **(0,5 Punkte)**

Aufgabe 3.5: Scheduling-Handsimulation (2 Punkte) (Theorie)

Es werden in einem Ein-Prozessor-System Prozesse wie in Abbildung 3 beschrieben gestartet:

Prozess	A	B	C	D	E
Ankunftszeitpunkt	0	2	3	6	8
Dauer	7	4	3	2	4

Abbildung 3: Prozesse eines Systems mit einer CPU und einem Thread

a) Simulieren Sie folgende Scheduling-Verfahren für die Prozesse aus Abbildung 3:

- SRTN,
- HRRN,
- MLF mit $\tau_i = 2^i$ $(i = 0, 1, \dots)$. (nicht unterbrechend)

Geben Sie für *jeden* Zeitpunkt den Inhalt der Warteschlange und den Prozess auf der CPU an.

Die Lösung soll in Form der dargestellten Tabelle abgegeben werden, wobei anzumerken ist, dass für Multilevel-Feedback mehrere Warteschlangen benötigt werden:

Zeit	0	1	2	3	…	18	19
CPU	A	…	…	…	…	…	…
Warteschlange		…	…	…	…	…	…
		…	…	…	…	…	…

b) Berechnen Sie für *jedes* der in a) verwendete Verfahren

- die Warte- und Antwortzeit *jedes* Prozesses sowie
- die mittlere Warte- und Antwortzeit des gesamten Systems.

Aufgabe 3.6: Scheduler (5 Punkte) (Praxis)

In dieser Aufgabe sollen verschiedene Scheduler implementiert werden.

- Last Come First Served ohne Verdrängung (LCFS)
- Round-Robin (RR)
- Priority mit Verdrängung (PRIOP)
- Shortest Remaining Time Next mit Verdrängung (SRTNP)
- Highest Response Ratio Next (HRRN)
- Multilevel Feedback (mit 4 Levels (Level $i = 0...3$) mit $\tau_i = 2^i$ für die Level 0 bis 2 und FCFS auf dem letzten Level, nicht unterbrechend)

Als Orientierungshilfe ist dafür die Implementierung eines First Come First ServedSchedulers in der Datei *FCFS.c* bereits vorgegeben. Ihre Lösungen sollen in den Dateien *RR.c*, *PRIOP.c*, *LCFS.c*, *MLF.c*, *HRRN.c* und *SRTNP.c*, die sich im Verzeichnis /src/ befinden, erfolgen. Entsprechende Stellen sind mit dem Kommentar //TODO"gekennzeichnet. Sie können natürlich eigene Funktionen auch außerhalb der mit //TODO"gekennzeichneten Funktionen erstellen. Außerdem können Sie auch die entsprechenden Headerfiles *RR.h*, *PRIOP.h* usw. im Ordner */lib/*) erweitern (siehe Hinweise in entsprechenden Files).

Programmaufruf

Beim Aufruf des Programms soll als ersten Parameter die Abkürzung eines Namens einer Schedulingstrategie übergeben werden (also RR, PRIOP, FCFS, ...). Danach folgen immer Gruppen von drei Zahlen, die die Eigenschaften der Prozesse bestimmen: Ankunftszeit, Dauer, Priorität. Bei Round-Robin kann zusätzlich als letzen Parameter noch die Länge der Zeitscheibe angegeben werden. Der Standardwert ist ansonsten 2.

Beispiele

Wir betrachten folgende drei Prozesse:

- A: Ankunft 0, Dauer 4, Priorität 2
- B: Ankunft 3, Dauer 3, Priorität 5
- C: Ankunft 5, Dauer 1, Priorität 1

Beispiel 1
Aufruf dieser drei Prozesse mit First Come First ServedScheduler:

```
$ ./scheduler FCFS 0 4 2 3 3 5 5 1 1
```

Dies führt zu Ausgabe:

```
Starting FCFS scheduler
| A | A | A | A | B | B | B | C |
```

Beispiel 2
Aufruf dieser drei Prozesse mit Round-RobinScheduler mit Zeitscheibe $\tau = 3$

```
$ ./scheduler RR 0 4 2 3 3 5 5 1 1 3
```

Dies führt zur Ausgabe:

```
Starting RR scheduler
| A | A | A | B | B | B | A | C |
```

Achten Sie darauf, nicht zwei Prozesse zum gleichen Zeitpunkt ankommen zu lassen!

Programmablauf

Die Vorgabe verarbeitet bereits die übergeben Parameter, kümmert sich um die Ausgabe der Prozesse und startet ihren Ablauf. Sie müssen nur noch um die Funktionalität der Scheduler kümmern. Dabei sollen sie zu jedem Scheduler die vier Funktionen bearbeiten. Informationen, was diese Funktionen tun sollen, sind jeweils in den header-files vermerkt. Es handelt sich jeweils um äquivalente Funktionen, die aber je nach scheduler unterschiedlich aussehen können.

Zuerst wird immer die Funktion `int xx_startup()` aufgerufen. In dieser können sie alles tun, was einmalig notwendig ist, um die nachfolgenden Funktionen auszuführen, zum Beispiel Variablen initialisieren oder Ähnliches. Beim Round-Robin-Verfahren wird hier zusätzlich die Länge der Zeitscheibe übergeben. Dann werden in einer Schleife die Funktionen

```
process* xx_new_arrival(process* arriving_process, process* running_process)
```

und `process* xx_tick (process* running_process)` aufgerufen. Jeder Aufruf von `xx_tick` stellt das Verstreichen einer Zeiteinheit dar.

Wenn zu einem bestimmten Zeitpunkt ein neuer Prozess ankommt, dann wird ein Pointer auf ein process-Objekt (siehe *process.h* für weitere Informationen) in `x_new_arrival` als Parameter *arriving_process* übergeben. Andernfalls ist dieser Parameter *NULL*. Der Pointer auf *running_process*, der in diesen beiden Funktionen übergeben wird ist jeweils der zuletzt aktive Prozess (oder *NULL*, falls kein Prozess aktiv war). Die Funktion `xx_new_arrival` soll, falls es einen neu ankommenden Prozess gibt, diesen verarbeiten (zum Beispiel einer Queue hinzufügen).
Der Rückgabewert soll wieder ein Pointer auf den jetzt aktuellen zu bearbeitenden Prozess sein (falls dieser getauscht wurde) oder auf den alten *running_process.*

Danach wird die Funktion `xx_tick` aufgerufen. In dieser soll geprüft werden, ob der aktuelle Prozess gewechselt werden muss (zum Beispiel, weil seine Bearbeitungsdauer abgeschlossen ist). Es soll ein Pointer zum nun aktuell zu bearbeitenden Prozess zurückgegeben werden. Dies ist dann der Prozess, der auch in der Ausgabe auf dem Bildschirm erscheint.

Nachdem alle Prozesse bearbeitet wurden, wird einmalig die Funktion `void xx_finish()` aufgerufen. Hier kann alles erledeigt werden, was es noch zu erledigen gibt (Speicher freigeben etc.).

Die Datenstruktur *process*, die Sie in der Datei *lib/process.h* finden repräsentiert die einzelnen Prozesse, die Sie beim Programmstart definieren. In diesem *process* sind die Ankunftszeit (start_time), die Ausführdauer (time_left), die Priorität (priority), sowie eine ID gespeichert.

Hilfen/Hinweise

Textausgabe auf der Konsole

Achten Sie darauf, keine zusätzlichen Zeichen auf der Standardausgabe im fertigen Programm auszugeben, da dies das automatische Testen erschwert. Zusätzliche Ausgaben werden dementsprechend mit Punktabzug bewertet.

Queue

Es steht ihnen eine Implementierung einer FIFO-Queue zur Verfügung. Die nutzbaren Funktionen dieser Queue sind in *lib/queue.h* beschrieben. Diese Queue wird auch in der vorgegeben implementierung des FCFS-Schedulers verwendet.

Kompiliertes Programm

Die Vorgabe enthält das Programm *scheduler_vorgabe*. In diesem sind bereits alle Scheduler einprogrammiert. Sie können dieses Programm nutzen um die Ausgaben mit Ihrem Programm zu vergleichen. Falls sich das Programm nicht ausführen lässt, müssen sie es möglicherweise vorher ausführbar machen. Geben Sie dazu folgendes in die Konsole ein:

```
$ chmod +x ./scheduler_vorgabe
```

Valgrind

Achten Sie darauf, dass dynamisch allokierter Speicher wieder freigegeben wird. Der allokierte Speicher für die *process*-Objekte werden schon von der Vorgabe freigegeben. Geben sie also nur selbst allokierten Speicher frei! Um zu prüfen, ob der Speicher komplett freigegeben wurde kann das Programm *valgrind* genutzt werden. Beispielaufruf:

```
$ valgrind ./scheduler RR 0 4 2 3 3 5 5 1 1 3
```

Makefile

In der Vorgabe wird ein makefile mitgeliefert. Um das Projekt zu kompilieren rufen sie einfach im Ordner, in dem sich das makefile befindet

```
$ make
```

auf. Sie müssen diese Datei nicht verändern. Sollten Sie allerdings weitere .c oder .h Files einbinden wollen, dann passen Sie das makefile entsprechend an. Es ist allerdings nicht nötig weitere Datein hinzuzufügen. **Ändern Sie nicht die Files main.c, process.h, colors.c und colors.h**.

Farbausgabe

Die bunte Ausgabe soll Ihnen helfen, schneller einen Überblick über die Ausgabe zu bekommen. Falls Sie jedoch Probleme mit der Farbausgabe auf Ihrem Terminal haben, dann können Sie das Flag -c mitübergeben, um die farbige Ausgabe zu unterbinden. Beispiel:

```
$./scheduler FCFS 0 4 2 3 4 2 -c
```

Ihr Projekt sollte ohne Warnungen und Fehlern auf den Uni-Rechnern mit Ubuntu 18 kompilieren und ausführbar sein. Testen Sie die Funktionalität, indem Sie per SSH auf einen Unirechner zugreifen und Ihr Programm dort testen. Informationen zur Nutzung von SSH finden Sie auf der Webseite der Freitagsrunde. Auf der Kursseite auf ISIS wurde außerdem eine Videoanleitung zum Verbinden per SSH bereitgestellt.