Vorlesung Betriebssysteme WS 2013/2014
Aufgabenblatt 3 vom 5. Dezember 2013
(Vorstellung der Lösungen bei den Tutoren bis zum 19.12.13)

Aufgabe 3.1: (15 Punkte)
Erläutern Sie Ihrem Tutor den Unterschied zwischen **`preemptiven`** und **`nicht-preemptiven`** Scheduling. Welcher Ansatz ist zu bevorzugen, wenn der Systemdurchsatz maximiert werden soll? Wie versuchen Windows-Serversysteme den Durchsatz zu maximieren?

Aufgabe 3.2: (30 Punkte)
Gegeben sei ein Einprozessor-System welches Round-Robin-Scheduling mit 16 Prioritätsstufen verwendet (0-15, 0 = niedrigste, 15 = höchste Priorität). Die Quantumslänge beträgt 20ms. Die Zeit für einen Kontextwechsel sei vernachlässigbar. Der Scheduler verwaltet laufende Threads und entscheidet ausschließlich nach dem Ablauf eines Quantums welcher Thread als nächstes laufen soll. Das Einfügen in die Warteliste erfolgt nach FIFO-Ordnung.

Es sollen drei Threads mit den folgenden Eigenschaften ausgeführt werden:

Thread ID	**Startzeit t_s (in ms)**	**Ausführungszeit e (in ms)**
1	0	100
2	15	80
3	30	60

- Zeichnen Sie ein Gantt-Diagramm unter der Annahme, dass alle drei Threads mit einer (statischen) Priorität von 8 ausgeführt werden.
- Jetzt soll Th3 mit einer Priorität von 9 ausgeführt werden. Weiterhin erhält Th1 eine Priorität von 7 und tritt nach 16ms in einen I/O-Wartezustand. Die Priorität von Th1 soll nach Beendigung der I/O-Operation um drei erhöht werden („Boost“). Th1 verlässt den Wartezustand bei t=45ms. Die Priorität von Th1 wird um eine Stufe am Quantumsende reduziert, bis wieder die Basispriorität erreicht ist. Zeichnen Sie ein Gantt-Diagramm für den beschriebenen Vorgang!

Bitte bringen Sie die Diagramme zum Tutoriumstermin mit und erläutern Sie ihrem Tutor.

Aufgabe 3.3: (20 Punkte)
- Erklären Sie die Implementierung von Teilen des Windows (WRK)-Schedulers (http://wrk.dcl.hpi.uni-potsdam.de/)!
- Eine neue Priorität wird für einen Thread mit Hilfe der Funktion `KiComputeNewPriority` berechnet. Erläutern Sie zeilenweise den genauen Ablauf der Funktion!
- Ein „Boost“ der Threadpriorität kann beispielsweise mit der Funktion `KiBoostPriorityThread` erzeugt werden. Erläutern Sie zeilenweise den genauen Ablauf der Funktion!

Aufgabe 3.4: (35 Punkte)
In der Übung wurde ein *Multilevel Queue Scheduler* vorgestellt. Dieser verwaltet pro Prioritätsstufe eine Liste von lauffähigen Threads. Bei der Auswahl eines Threads werden die Listen entsprechend ihrer Priorität inspiziert, d.h. die Liste mit einer höheren Priorität wird vor eine Liste mit niedrigerer Priorität durchsucht. Innerhalb einer Liste wird Round Robin

zur Auswahl eines Threads verwendet. Danach wird der ausgewählte Thread durch den Dispatcher für ein Quantum zur Ausführung gebracht werden. Es gibt insgesamt zwei Prioritätsstufen: `SCHED_PRIORITY_LOW` und `SCHED_PRIORITY_HIGH`.

Das vorgestellte Verfahren birgt das Risiko, dass Threads „verhungern“. Implementieren Sie im Scheduler ein Ihnen bekanntes Verfahren, um dieses Problem zu lösen. Modifizieren Sie dazu die Funktion `Schedule` sowie `AddThread` innerhalb der Datei **`scheduler.c`** und erweitern Sie die Datenstruktur `DISPATCHER_TASK` in der Datei **`dispatcher.h`** geeignet.

- Erläutern Sie Ihrem Tutor den Unterschied zwischen Scheduler und Dispatcher!
- Erläutern Sie Ihre Implementierung und demonstrieren Sie das Programm Ihrem Tutor!
- Verwenden sie **`perfmon`** zur Anzeige des Verhaltens Ihrer Implementierung!
- Vergleichen Sie das Verhalten Ihrer Implementierung mit der Theorie (zum Beispiel mit Hilfe eines Gantt-Diagramms)!
- Bewerten Sie die Laufzeitkomplexität Ihres Verfahrens!

Hinweis: Der Programmrahmen (aufgabe34.zip) realisiert einen Scheduler im Usermode.

Verpacken Sie die Source-Code Dateien (**`scheduler.c`** und **`dispatcher.h`**) in eine ZIP-Datei (**`blatt34.zip`**). Die ZIP-Datei muss mindestens 24 Stunden vor dem Tutoriumstermin in das Abgabesystem eingestellt werden.