

Betriebssysteme im Wintersemester 2008/2009

Übungsblatt 6

Abgabetermin: 19.11.2008, 13:30 Uhr

Aufgabe 22: (H) CPU-Scheduling: Theorie

(10 Pkt.)

- a. Welche beiden Varianten des Scheduling haben Sie kennengelernt?
Beschreiben Sie jeweils kurz einen Vorteil.
Nennen Sie jeweils ein Beispiel für ein Betriebssystem, in dem diese Art von Scheduler eingesetzt wird.
- b.
 - (i) In welchem Modus befindet sich der Prozessor, während der Scheduler-Prozess verarbeitet wird?
 - (ii) Begründen Sie mit zwei Argumenten, warum dieser Modus für die Ausführung sinnvoll ist.
- c.
 - (i) Erläutern Sie den Begriff des Quantums im Zusammenhang mit CPU-Scheduling.
 - (ii) Was passiert, wenn man das Quantum zu groß wählt?
 - (iii) Was passiert, wenn man das Quantum zu klein wählt?
- d. Wie kann ein Betriebssystem dafür sorgen, dass kein Thread aufgrund niedriger Priorität verhungert?
- e. Gegeben sei ein Mehrprozessorsystem bei dem im Moment alle CPUs belegt sind. Nun kommt ein Thread aus dem Zustand `Blocked` in den Zustand `Ready`. Der Scheduler muss jetzt entscheiden, ob einer der rechnenden Threads pausiert wird, damit dieser neue Thread verarbeitet werden kann.
Aufgrund welcher Kriterien sollte diese Entscheidung gefällt werden?

Aufgabe 23: (H) CPU-Scheduling: Praxis

(10 Pkt.)

Folgende Prozesse sollen betrachtet werden (die Zeiten seien in beliebigen Zeiteinheiten gegeben, die Prioritäten von 0 bis 2, wobei 0 die höchste Priorität bezeichne):

Prozess	Ankunftszeitpunkt	Bedienzeit	Priorität
A	0	4	1
B	0	5	0
C	1	2	1
D	3	6	1
E	4	2	2
F	4	3	2
G	6	3	0
H	9	5	0
I	10	1	1

Der Ankunftszeitpunkt t bezeichnet auch den Zeitpunkt, zu dem der Prozess in die Warteschlange eingereiht wird. Kommen zwei Prozesse zur gleichen Zeit, so wird die Ordnung auf den Prozessnamen herangezogen (Prozess A vor Prozess B, usw.). Wird ein Prozess vor seinem Terminieren zum Zeitpunkt t' unterbrochen, so reiht er sich hinten in die Warteschlange mit Ankunftszeit t' wieder ein.

Geben Sie für die Strategien FIFO (First In First Out), SJF (Shortest Job First), Round Robin mit Quantum $t = 3$, sowie für preemptives und nicht-preemptives Priority-Scheduling jeweils in Form eines Gantt-Charts für die ersten 20 (0 bis 19) Zeiteinheiten an, wann welchem Prozess Rechenzeit zugeteilt wird, und wann die Prozesse ggf. terminieren.

Hinweis: Ein Gantt-Diagramm zeigt die zeitliche Abfolge von Aktivitäten als Balken auf einer Zeitachse. Tragen Sie die Zeiteinheiten auf der x-Achse auf und zeichnen Sie dann Balken für die einzelnen Prozesse im Zeitverlauf.

Aufgabe 24: (K) Multiple Choice: Threads

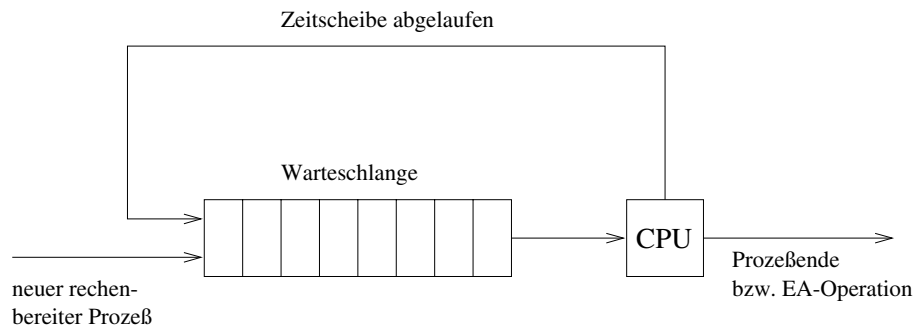
(3 Pkt.)

- a. Sind die folgenden Aussagen zu Multithreading wahr oder falsch?
 - (i) Eine Unterbrechung ist das Maschinenkonzept für die Behandlung nicht deterministischer, also unvorhergesehener Abläufe.
 - (ii) Werden einem Thread mehrere Prozesse zugeordnet, so spricht man von Multithreading.
 - (iii) Die Java Virtual Machine ist ein Beispiel für den Einsatz von User-Level-Threads.
- b. Sind die folgenden Aussagen über User- und Kernel-Level-Threads wahr oder falsch?
 - (i) User-Level-Threads sind aus Sicht des Betriebssystem-Kerns nicht als solche sichtbar.
 - (ii) Reine User-Level-Threads können in einer Multiprozessor-Umgebung nicht parallel ausgeführt werden.
 - (iii) Bei User-Level-Threads ist das Thread-Management Aufgabe der Anwendung.
 - (iv) Beim Thread-Wechsel zwischen User-Level-Threads wird kein Moduswechsel erforderlich.
 - (v) Beim Thread-Wechsel zwischen Kernel-Level-Threads wird kein Moduswechsel erforderlich.

Aufgabe 25: (T) CPU-Scheduling: Round-Robin

(8 Pkt.)

Beim Round-Robin-Scheduling wird eine Warteschlange von Prozessen verwaltet, wobei jeweils dem in der Schlange ersten Prozess für eine feste Zeitdauer Q (die sog. *Zeitscheibe*) die CPU zugeteilt wird. Prozesse, deren Zeitscheibe abgelaufen ist, werden am Ende der Warteschlange wieder eingereiht.



Nehmen Sie an, empirische Messungen hätten ergeben, dass ein Prozess im Mittel für eine Zeitspanne T die CPU benutzt, bevor er das obige Warteschlangensystem aufgrund einer angestoßenen E/A-Operation bzw. bei Prozessende verlässt. Ein Prozesswechsel benötige eine Zeitdauer S , die als Overhead verloren geht.

- Geben Sie für Round-Robin-Scheduling mit einer Zeitscheibe Q eine Formel für die CPU-Auslastung an.
- Welche CPU-Auslastung ergibt sich in den folgenden Fällen?
 - $Q > T$,
 - $Q = S \ll T$ und
 - $Q \approx 0$.
- Leiten Sie daraus Kriterien für den Wert von Q ab. Dabei soll vorausgesetzt werden, dass stets rechenbereite Prozesse zur Verfügung stehen.

Aufgabe 26: (P) Scheduler-Simulation: Grundlagen

(12 Pkt.)

Sie haben nun verschiedene Arten von Schemlern (preemptiv, nicht-preemptiv) sowie unterschiedliche Scheduling-Strategien (Prioritäten, FCFS, Round Robin usw.) in der Theorie kennen gelernt. Jetzt sollen Sie selbst einen Scheduler in der Programmiersprache Java implementieren. Es ist klar, dass es sich dabei nur um eine Simulation, nicht um einen realen und funktionsfähigen Scheduler handeln kann. Beantworten Sie die folgenden Fragen zunächst mit der Zielsetzung, einen preemptiven Priority-Scheduler zu simulieren.

- Bei preemptivem Scheduling müssen laufende Prozesse nach bestimmten, festgelegten Zeitintervallen unterbrochen und die Kontrolle an den Scheduler abgegeben werden. Dieser entscheidet dann nach seiner Strategie, ob der gerade abgebrochene Prozess fortgesetzt wird oder ein anderer Prozess zur Ausführung kommt. Wie werden diese Unterbrechungen in realen Systemen realisiert, und wie können sie in der Programmiersprache Java simuliert werden? Welche Probleme können dabei auftreten?
- In realen Systemen besitzt jeder Prozess einen Prozesskontrollblock, der alle Statusinformationen (Prozess-ID, Priorität usw.) für diesen Prozess enthält. Wie würden Sie diesen Umstand in Java realisieren?

- c. Alle aktuellen Prozesse müssen in geeigneter Weise im Speicher gehalten, also verwaltet werden. Welche Datenstruktur würden Sie dazu einsetzen? Begründen Sie Ihre Antwort, und nennen Sie die wichtigsten Eigenschaften und Operationen auf dieser Datenstruktur.
- d. Manche Prozess-Attribute werden bei allen Strategien benötigt (Beispiel: Prozess-ID), andere hingegen nicht (Beispiel: Prozess-Priorität). Wie lässt sich dies bei der Implementierung berücksichtigen? Wie können Sie Ihren Programmcode auf diese Weise für Erweiterungen um neue Scheduling-Strategien offen halten, ohne dass größere Code-Fragmente mehrfach in verschiedenen Klassen vorkommen?