6. Übung

Ausgabe	Abgabe	Diskussion
22.11.13	29.11.13	02.12.-06.12.13

Bitte bei der Abgabe Name/Matr.Nr. der Mitglieder einer Gruppe, Nummer der Übung/Teilaufgabe und Datum auf den Lösungsblättern nicht vergessen! Darauf achten, dass die Lösungen beim richtigen Tutor abgegeben werden. Achten Sie bei Programmieraufgaben außerdem darauf, dass diese im Linuxpool kompilierbar sind.
Zu spät abgegebene Lösungen werden nicht berücksichtigt!

Aufgabe 1: Scheduling-Strategien (2+2+2=6 Punkte)

Sie haben in der Vorlesung nicht-preemptive Scheduling Strategien kennengelernt. In der Vorlesung wurden die Strategien jeweils für eine CPU vorgestellt. Sie sind aber leicht so erweiterbar, dass sie ankommende Prozesse auf mehreren CPUs verteilen können.

Betrachten wir als Beispiel die Kassen eines Supermarktes. Jede Kasse entspricht einer CPU und jeder Kunde, der an die Kasse kommt, stellt einen Prozess dar. Die Zahl der Artikel, die ein Kunde eingekauft hat, entspricht der Zeit, die ein Prozess zur Ausführung benötigt. Wir nehmen dabei an, dass die Zeit zum Registrieren eines Artikels an der Kasse konstant ist und setzen diese mit einer Zeiteinheit an, den eigentlichen Bezahlvorgang vernachlässigen wir.

Da es Kunden im Supermarkt nicht zugemutet werden kann, andere Kunden vorzulassen, kommt üblicherweise die FIFO-Strategie zum Einsatz, d.h. wer zuerst die Kasse erreicht, wird zuerst abkassiert. Da die Kunden eines Supermarktes auf ihren Vorteil bedacht sind, stellen sie sich immer an der Kasse an, an der die Wartezeit am geringsten ist, d.h. an der vor ihnen die wenigsten Artikel registriert werden müssen.

a) Ein Supermarkt habe drei Kassen (K1, K2 und K3). An keiner der Kassen wartet ein Kunde. Nun treffen (fast gleichzeitig) Kunden mit folgenden Artikelanzahlen (in dieser Reihenfolge) ein:

   6, 15, 23, 10, 3, 13, 15, 7, 20, 40, 19, 4, 6, 21 (14 Kunden)

   O.B.d.A. stellt sich der erste Kunde an K1, der zweite an K2, der dritte an K3 an.

   - Dokumentieren Sie die sich ergebenden Warteschlangen an den Kassen K1, K2 und K3.
   - Wie viele Zeiteinheiten muss ein Kunde im Durchschnitt warten, bis er an der Kasse ankommt?

b) Der Leiter des Supermarktes ist mit dieser mittleren Wartezeit für seine Kunden nicht zufrieden und führt eine Kasse ein, an der nur Kunden bedient werden, welche weniger oder gleich 11 Artikel eingekauft haben (K1 wird in diesem Sinn ausgezeichnet). Kunden mit wenigen Artikeln dürfen sich natürlich nach wie vor auch an den anderen Kassen anstellen, sofern sie eine geringere Wartezeit erwarten.

   - Dokumentieren Sie die sich ergebenden Warteschlangen (K1 ist die Kasse für Kunden mit weniger als 12 Artikeln; die ersten Kunden stellen sich an die Kassen K1, K2, K3 - in dieser Reihenfolge).

- Berechnen Sie die mittlere Wartezeit der Kunden.
- Was für eine Strategie versucht der Supermarktleiter näherungsweise zu erreichen?
- Was für einen Nachteil hat die neue Regelung?

c) Der Supermarktleiter ist nicht zufrieden mit der unter b) eingeführten Lösung. Er entscheidet, die Kasse K1 nur noch für Kunden mit 20 oder mehr Artikeln zu erlauben.

- Dokumentieren Sie die sich ergebenden Warteschlangen (K1 ist die Kasse für Kunden mit mehr als 19 Artikeln; die ersten Kunden stellen sich an die Kassen K2, K3, K1 - in dieser Reihenfolge).
- Berechnen Sie die mittlere Wartezeit der Kunden.
- Wird der Supermarktleiter bei dieser Lösung bleiben?

Aufgabe 2: Scheduling Strategien (5+1=6 Punkte)

Gegeben sei folgende Scheduling-Strategie mit vier Prioritätsklassen. Jeder Klasse ist eine eigene FIFO-Warteschlange und ein Quantum zugeordnet:

Klasse	Quantum	Priorität
0	1	höchste
1	4	
2	16	
3	∞	niedrigste

Es wird *Process Aging* verwendet, d.h. wenn das Zeitquantum eines Prozesses abgelaufen ist, wird er an das Ende der Warteschlange mit nächstniedriger Priorität gestellt. Neuen Prozessen ist eine bestimmte Priorität zugeordnet. Sie werden an das Ende der Warteschlange ihrer Prioritätsklasse angehängt. Es wird am Ende jeder Zeiteinheit überprüft welcher Prozess am Anfang der nicht leeren Warteschlange mit der höchsten Priorität steht und dieser dann im nächsten Schritt bearbeitet.

Folgende Prozesse sollen betrachtet werden:

Prozess	Ankunftszeit	Priorität	Laufzeit
A	0	0	4
B	0	1	7
C	1	0	1
D	2	1	4
E	5	3	20
F	10	1	3
G	13	0	2
H	15	1	3
I	16	1	3

Ein Prozess, der zum Zeitpunkt t ankommt, wird erst ab dem (t+1)-ten Zeitpunkt berücksichtigt, d.h. er kann frühestens eine Zeiteinheit nach seiner Ankunft die CPU verwenden.

a) Geben Sie für die ersten 20 Zeiteinheiten an, welchem Prozess Rechenzeit zugeteilt wird.

b) Geben Sie für alle Prioritätsklassen an, welche Prozesse sich nach Ablauf der ersten 20 Zeiteinheiten noch in der Warteschlange befinden (in der korrekten Reihenfolge), wie viele Zeiteinheiten jeder einzelne Prozess noch laufen muss und wie viele Zeiteinheiten von seinem Quantum noch übrig sind.

Aufgabe 3: Linux-Scheduler (2 Punkte)

Erklären Sie die Ansätze und Funktionsprinzipien des aktuellen Linux-Schedulers und erläutern sie insbesondere dessen Komplexitätsklasse, dessen Datenstrukturen und dessen Granularität.

Aufgabe 4: C Programmierung (4 Punkte)

a) Wählen sie sich eine shell (bash, csh, etc.) und machen Sie sich mit den Grundlagen, wie z.B. Dateien erzeugen, kopieren, verschieben, Anzeigen von Dateiinhalten und Auflisten von Ordnerinhalten, vertraut.

b) Machen Sie sich mit Low-Level-IO Systemaufrufen unter Linux vertraut. Für diesen Teil der Aufgabe benötigen Sie nur `open()`, `close()`, `read()` und `write()`. Verwenden sie **nicht** die Standardbibliotheksfunktionen für IO! Schreiben Sie eine Funktion

```
int copy(char *sourcename, char *targetname);
```

Die Funktion soll den Inhalt einer Datei kopieren und zwar nur dann, wenn es noch keine Datei mit dem Namen der Zieldatei existiert. Eine Datei darf nicht einfach überschrieben werden. Verwenden Sie zum kopieren einen Puffer (char buffer[BUFFSIZE]). (Experimentieren Sie wenn sie wollen mit unterschiedlichen Puffergrößen und vergleichen sie die benötigte Zeit.)

c) Verwenden Sie Ihre copy-Funktion um einen einfachen Papierkorb zu implementieren. Der Papierkorb soll drei Befehle unterstützen:

```
./trashcan -p filename
```

PUT: Eine Datei innerhalb des Verzeichnisses in dem trashcan ausgeführt wird, soll in einen versteckten Ordner `".ti3_trash"` verschoben werden.

```
./trashcan -g filename
```

GET: Eine Datei innerhalb von `".ti3_trash"` soll wiederhergestellt werden, im selben Verzeichnis, in dem trashcan ausgeführt wird.

```
./trashcan -r filename
```

REMOVE: Die Datei filename in dem Ordner `".ti_trashcan"` wird endgültig gelöscht.

Das Verzeichnis soll automatisch beim ersten Aufruf von `trashcan` erzeugt werden. Finden Sie zum Erstellen eines Ordners sowie für das Löschen von Dateien entsprechende Systemaufrufe. In keinem Fall darf eine existierende Datei überschrieben werden. Überlegen Sie sich sinnvolle Fehlerbehandlungen. Sie können das Grundgerüst `trashcan_framework.c` von der Veranstaltungsseite verwenden.