Datum: 01.07.2011 **Name:** _____ **Vorname:** _____

Arbeitszeit: 50 Minuten Matr.-Nr.: _____

Hilfsmittel: Taschenrechner Unterschrift: _____

wird vom Prüfer ausgefüllt

1	2	3	4	5	6						Σ

Diese Probeklausur hat **reduzierten Umfang** (50 statt 90 Minuten in der Prüfung). Die tatsächliche Prüfung wird eine „**Auswahlklausur**“ sein (es reicht, einen Teil der Aufgaben zu bearbeiten, um die für eine 1,0 nötigen Punkte zu erreichen), in dieser Probeklausur bearbeiten Sie bitte alle Aufgaben (Summe: 52 Punkte).

1. Synchronisation (10/52 Punkte)

a) Was ist ein **Mutex**?

b) Der Linux-Kernel verwendet **Spin Lock** genannte Mutexe. Diese Spin Locks legen sich nicht schlafen. Inwiefern hat das Auswirkungen auf die Verwendbarkeit innerhalb von Interrupt-Handlern?

c) Beschreiben Sie das **Erzeuger-Verbraucher-Problem** und skizzieren Sie mit Hilfe von Pseudo-Programm-Code eine Semaphor-basierte Lösung. (Verwenden Sie wahlweise eine an C oder an Python erinnernde Syntax; exakte Befehlsnamen etc. sind irrelevant.)

2. Prozesse und Threads (8/52 Punkte)

a) Linux verwendet das **POSIX-Thread-Modell** für die Thread-Programmierung. Schreiben Sie in Pseudo-Code (darf wie ein C- oder wie Python-Programm „aussehen“) ein kleines Programm, das zwei Threads erzeugt, die beide „Hello World“ ausgeben. Das Programm soll erst beendet werden, wenn beide Threads beendet sind.

b) Wie unterscheiden sich allgemein **User-** und **Kernel-Level-Threads** und welche Vor- und Nachteile sind damit jeweils verbunden?

3. System calls (10/52 Punkte)

a) Nach dem Öffnen einer Datei mit `fd=open(...)` wollen Sie lesend und schreibend auf die Datei zugreifen. Geben Sie die Syntax der Befehle zum Lesen und Schreiben an, d. h. nennen Sie die Parameter der beiden Funktionen und erklären Sie die Bedeutungen der Parameter. (Die korrekte Reihenfolge der möglichen Parameter ist dabei nicht wichtig.)

b) Betrachten Sie den folgenden Programmausschnitt:

```
int pid1 = fork();
printf ("%s\n","[1] Ein Fork ist durch, einer muss noch.");
int pid2 = fork();
printf ("%s\n","[2] Zeit für eine Fallunterscheidung.");
if ( (pid1==0) && (pid2==0) ) {
  printf ("%s\n","[3] Ich starte jetzt emacs.");
  execl ("/bin/emacs", "/etc/fstab", (char *)NULL);
  int pid3 = fork();
  printf ("%s\n","[4] Nach dem dritten Fork.");
} else {
  printf ("%s\n","[5] Ich gucke nur zu.");
};
```

Wie oft und warum erscheinen die mit [1] bis [5] durchnummerierten Ausgaben? Schreiben Sie zu jeder Ausgabe die Anzahl und begründen Sie Ihre Antwort stichwortartig oder mit einer Skizze.

c) Beim Aufruf des System calls `fork()` erhält der Sohn den Wert 0 und der Vater die Prozess-ID des neu erzeugten Sohnes (≠0) zurück. Für eine reine Unterscheidung („wer bin ich?“) könnte man auch umgekehrt arbeiten, also im Vater den Wert 0 und im Sohn die Prozess-ID des Vaters (≠0) zurückgeben. Warum ist diese Alternative schlecht?

4. Scheduling-Verfahren (Uni-Prozessor) (11/52 Punkte)

a) Aus der Vorlesung kennen Sie die Scheduling-Verfahren **FCFS** (First Come First Served), **SJF** (Shortest Job First) und Round Robin (**RR**).

Es gebe die folgenden fünf Prozesse mit den angegebenen Ankunftszeiten und Gesamtrechenzeiten:

Prozess	Ankunft	Rechenzeit
P	0	10
Q	4	8
R	5	7
S	6	1
T	12	2

Für First Come First Served sieht die Ausführreihenfolge wie folgt aus:

Zeit	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9 10	0 1 2 3 4 5 6 7 20
FCFS	**P** P P P P P P P P P	**Q** Q Q Q Q Q Q Q **R** R	R R R R R **S** **T** T
SJF			
RR (q=4)			
RR (q=10)			

Ergänzen Sie für SJF und RR hier in der Tabelle die Ausführreihenfolgen. Für RR nehmen Sie ein Zeitquantum von q=4 bzw. q=10 Zeiteinheiten an. (Neue Jobs werden bei RR im Moment ihrer Ankunft hinten an die aktuelle Warteschlange angehängt.)

b) Wenn Sie FCFS mit RR(q=4) und RR(q=10) vergleichen, was fällt Ihnen dann auf? Bewerten Sie die Wahl des Zeitquantums q=4 bzw. q=10.

c) Was ist der Unterschied zwischen **I/O-lastigen Prozessen** und **CPU-lastigen Prozessen**?

5. Deadlocks (4/52 Punkte)

a) Auf einem System gebe es fünf Prozesse P_1, P_2, ..., P_5 und fünf exklusive Betriebsmittel R_1, R_2, ..., R_5. Der momentane Zustand sei wie folgt:

P_1 belegt keine Ressource und fordert R_1 an,

P_2 belegt R_1 und fordert R_2 und R_4 an,

P_3 belegt R_3 und fordert R_2 und R_4 an,

P_4 belegt R_2 und fordert R_5 an,

P_5 belegt R_5 und fordert R_3 an.

Überprüfen Sie mit einem der beiden in der Vorlesung behandelten Verfahren, ob ein Deadlock vorliegt, und falls ja, welche Threads an dem Deadlock beteiligt sind.

6. Interrupts (9/52 Punkte)

a) Nennen Sie – im Umgang mit an einen Computer angeschlossener Hardware – eine **Alternative zur Verwendung von Interrupts**, und erläutern Sie, warum Interrupts deutliche Verbesserungen bei Performance und Einfachheit der Programmierung liefern.

b) Während der Bearbeitung eines Interrupts kommt es zu einem **weiteren Interrupt**. Bei der Implementierung eines Betriebssystems haben Sie verschiedene Möglichkeiten, eine solche Situation zu behandeln. Nennen Sie zwei davon und erläutern Sie Vor- und Nachteile.

c) Linux teilt Interrupt-Behandlungsroutinen (Interrupt Handler) in zwei Teile auf, eine **top half** und eine **bottom half** (auch **Tasklet** genannt). Wie unterscheiden sich top und bottom half, und warum führt man diese Trennung ein?

Wenn Sie eine Korrektur Ihrer Probeklausur wünschen, geben Sie diese bitte am Ende des Praktikumstermins ab.