1	2	3	4	5	6	7	8	9			Σ

Diese Probeklausur hat im Vergleich zur richtigen Klausur reduzierten Umfang. Bitte bearbeiten Sie alle Aufgaben. Die Gesamt-Punktezahl entspricht der Gesamt-Bearbeitungszeit.

Tipp: Lesen Sie zunächst alle Aufgaben durch und entscheiden Sie, welche Fragen Sie am leichtesten beantworten können; starten Sie dann mit diesen Aufgaben.

Durchgestrichene Aufgaben sind beispielhaft für Aufgaben, die *nicht* in der Online-Klausur vorkommen werden.

Viel Erfolg!

1. Bedienung der Shell (2 Punkte)

a) Welches der folgenden Zeichen weist im Shell-Prompt darauf hin, dass Sie mit normalen Benutzerrechten (also nicht als `root`) arbeiten?

- ☐ `:`
- ☐ `$`
- ☐ `&`
- ☐ `#`

b) Wie zeigen Sie in der Shell das aktuelle Arbeitsverzeichnis an?

- ☐ `showwd`
- ☐ `echo $PWD`
- ☐ `pwd`
- ☐ `PWD`

2. Jobs (7 Punkte)

a) Welche der folgenden Aussagen sind korrekt?

- ☐ Mit der Tastenkombination Strg+Z können Sie (in der Regel) ein Programm unterbrechen.
- ☐ Unterbrochene Jobs können Sie mit `bg` im Hintergrund fortsetzen.
- ☐ Hintergrund-Jobs werden beim Verlassen der Shell immer beendet.
- ☐ Gibt es mehrere Jobs, können Sie mit `fg` gezielt einen davon ansprechen, indem Sie als Parameter für `fg` dessen Prozess-ID angeben.
- ☐ Das Tool `jobs` zeigt zu jedem Job auch die Prozess-ID an.
- ☐ Mit `nohup` können Sie ein Programm so starten, dass es sich nicht gewaltsam abbrechen lässt.
- ☐ Das Programm `top` zeigt standardmäßig die am meisten CPU-Zeit verbrauchenden Prozesse des Systems an.
- ☐ Im Verzeichnis `/proc/jobs` finden Sie für jeden in der aktuellen Shell gestarteten Job einen Unterordner, der in Pseudodateien Informationen über den Job enthält.

b) Wie starten Sie einen neuen Job, der mit Nice-Wert 5 läuft? Als Name für das zu startende Programm verwenden Sie `beispiel`.

~~**c)** Was macht das Kommando `bg` mit einem Job?~~

3. Prozesse (5 Punkte)

a) Mit welchem Signal können Sie einen Prozess so beenden, dass er noch Gelegenheit hat, offene Dateien zu schließen und sich somit „ordentlich" zu beenden? Als Referenz finden Sie nebenstehend die Liste der ersten 28 Signale.

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
```

_____

b) Mit welchem Kommando ändern Sie den Nice-Wert des Prozesses mit der ID 12345 auf -5?

_____

c) Sie haben aus einer Shell heraus mit Root-Rechten und dem Kommando `nice -n 5 bash` eine neue Shell gestartet, aus dieser heraus starten Sie mit `nice -n -5 daemon &` einen Daemon-Prozess im Hintergrund. Mit welchem Nice-Wert läuft dieser Prozess?

_____

d) Sie starten einen Prozess mit dem Kommando `programm &`. Was bewirkt das &-Zeichen?

4. Scheduler (4 Punkte)

Welche der folgenden Aussagen sind korrekt?

☐ Kernel Level Threads und User Level Threads unterscheiden sich dadurch, dass der Scheduler (im Betriebssystem) User Level Threads nicht „kennt", also auch nicht auswählen kann.

☐ **Prioritätsinversion** bedeutet, dass ein mangelhaft arbeitender Scheduler Prozesse mit niedriger Priorität gegenüber solchen mit hoher Priorität bevorzugt.

☐ **Lotterie-Scheduler** entscheiden durch Ziehen eines Loses, welcher Prozess als nächster laufen darf.

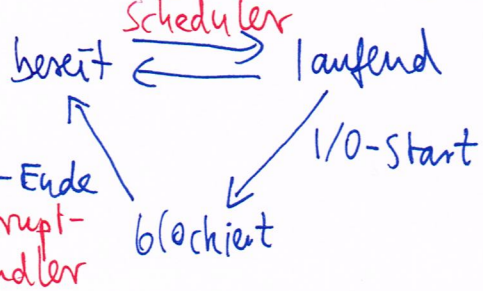
☐ Ein **präemptiver** Scheduler ist nicht in der Lage, laufende Prozesse zu unterbrechen – er muss warten, bis der Prozess von sich aus die CPU „abgibt", also in den Scheduler springt.

☐ **Aging** bedeutet, dass ein bereiter Prozess dauerhaft vom Scheduler nicht ausgewählt wird, wodurch er ständig „altert".

☐ Der Begriff **Burst** bezeichnet eine CPU- bzw. eine I/O-Phase, also z. B. für CPU-Phasen den Zeitraum vom Aktivieren des Prozesses bis zum Deaktivieren durch den Scheduler oder Einleiten einer I/O-Aktion durch den Prozess.

☐ Der Scheduler legt fest, wann die I/O-Blockade eines Prozesses aufgehoben wird (Übergang vom Zustand „blockiert" in den Zustand „bereit").

☐ SJF ist eine präemptive Variante von SRT.



"SJF" < "SRT"

5. System calls (5 Punkte)

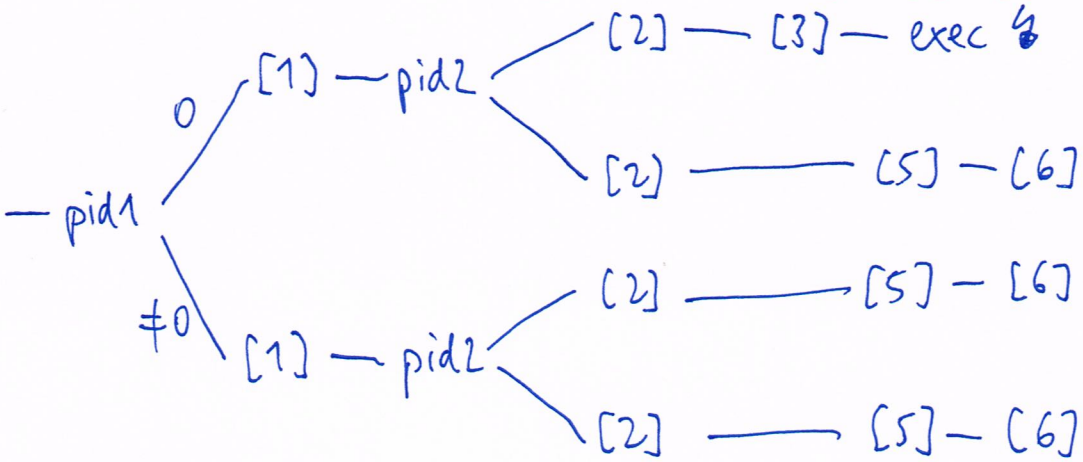
Betrachten Sie den folgenden Programmausschnitt:

```
...
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.");
};
printf ("%s\n","[6] Nach der if-Abfrage endet das Programm.");
...
```

Wie oft und warum erscheinen die mit [1] bis [6] durchnummerierten Ausgaben? Schreiben Sie zu jeder Ausgabe die Anzahl und begründen Sie Ihre Antwort stichwortartig.

(Es reicht auch aus, den Prozessbaum aufzumalen und dann die Ausgaben durchzuzählen.)

Zeichnen Sie den Prozessbaum, um zu prüfen, wie oft die Ausgaben [1] bis [6] erscheinen.



Antworten:

[1] _____ mal

[2] _____ mal

[3] _____ mal

[4] _____ mal

[5] _____ mal

[6] _____ mal

6. Scheduling-Verfahren (9 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 angegebenenen 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	0 1 2 3 4 5 6 7
	~~10~~	10 ~~20~~	20
FCFS	**P P P P P P P P P P**	**Q Q Q Q Q Q Q R R**	**R R R R R S T T**
SJF	P P P P P P P P P P	S R R R R R R R T T	Q Q Q Q Q Q Q Q
RR (q=6)	P P P P P P Q Q Q Q	Q Q R R R R R R S P	P P P T T Q Q R

Ergänzen Sie für SJF und RR hier in der Tabelle die Ausführreihenfolgen. Für RR nehmen Sie ein Zeitquantum von q=6 Zeiteinheiten an. (Neue Jobs werden bei RR im Moment ihrer Ankunft hinten an die aktuelle Warteschlange angehängt – noch bevor ggf. ein aktuell laufender Prozess unterbrochen und in die Warteschlange eingereiht wird.)

b) Das Round-Robin-Verfahren (**RR**) ist unfair gegenüber I/O-lastigen Prozessen. Erklären Sie kurz (in Stichworten), woran das liegt und wie **VRR** (Virtual Round Robin) die Situation verbessert.

RR(6):
0: P []
4: P [Q]
5: P [Q, R]

6: P [Q, R, S]
→ Q [R, S, P]
12: Q [R, S, P, T]
→ R [S, P, T, Q]

18: S [P, T, Q, R]
19: P [T, Q, R]
23: T [Q, R]
25: Q [R]
27: R []

28: _ []

7. Interrupts und System Calls (4 Punkte)

a) Erklären Sie (stichwortartig), warum es besser ist, Gerätetreiber mit Interrupts (statt über Polling) zu implementieren.

- aktives / passives Warten

b) Multiple choice: Kreuzen Sie die korrekten Aussagen an:

☐ Linux-Prozesse führen System Calls durch, indem sie die System-Call-Nummer sowie eventuelle Argumente auf den Stack schreiben und dann einen Software-Interrupt (`int 0x80`) auslösen.

☐ Linux unterteilt Interrupt-Handler in eine *top half* und eine *bottom half*. Die top half ist der eigentliche Interrupt-Handler, und er erzeugt die bottom half. Letztere ist einfach ein neuer Prozess, der die verbleibenden Arbeiten erledigt.

☒ Damit ein Betriebssystem einen präemptiven (unterbrechenden) Scheduler haben kann, muss der Prozessor Interrupts unterstützten und der Rechner einen Timer-Baustein besitzen.

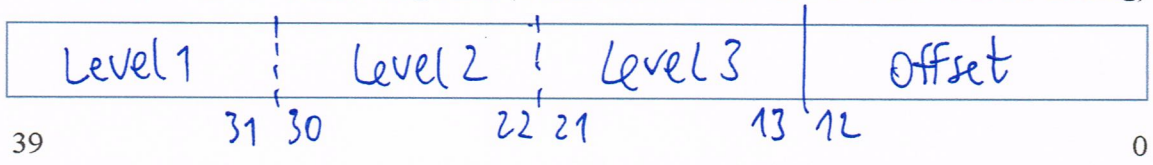
☐ Wenn es mehrere CPU-lastige und mehrere I/O-lastige Prozesse gibt, ist es eine gute Strategie, erst alle CPU-lastigen und danach alle I/O-lastigen Prozesse zu verarbeiten (oder anders herum, jedenfalls getrennt).

8. Virtuelle Adressen (5 Punkte)

Eine CPU arbeitet mit folgenden Werten:

- Seitengröße: 8 KByte = $2^3 \cdot 2^{10}$ Byte = 2^{13} Byte ⇒ 13 Bit Offset
- 40 Bit lange virtuelle Adressen ⇒ Seiten-Nr.: (40−13) Bit = 27 Bit
- 3-stufiges Paging; alle Seitentabellen sind gleich groß 27/3 = 9 Bit pro Teil
- Seitentabelleneinträge sind 8 Byte lang

a) Wie ist eine virtuelle Adresse aufgebaut (welche Bits der Adresse haben welche Bedeutung)?



Zeichnen Sie die die Unterteilung hier ein und beschriften Sie die Abschnitte geeignet.

b) Wie viele Seitentabellen der verschiedenen Stufen gibt es? Wie groß sind diese Tabellen?

\# Tabellen:
Stufe 1: 1
Stufe 2: 2^9 ($2^{\wedge}9$)
Stufe 3: $2^9 \cdot 2^9 = 2^{18}$ ($2^{\wedge}18$)

Größe:
\# Einträge · Größe Eintrag
$= 2^9 \cdot 8$ Byte
$= 2^9 \cdot 2^3 B = 2^{12} B = 4$ KB

9. Deadlocks (4 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,

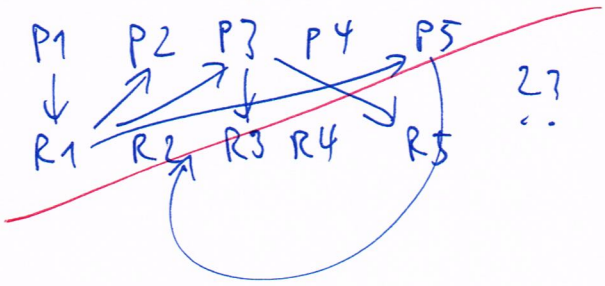
P_2 belegt R_1 und fordert R_2 und R_4 an,

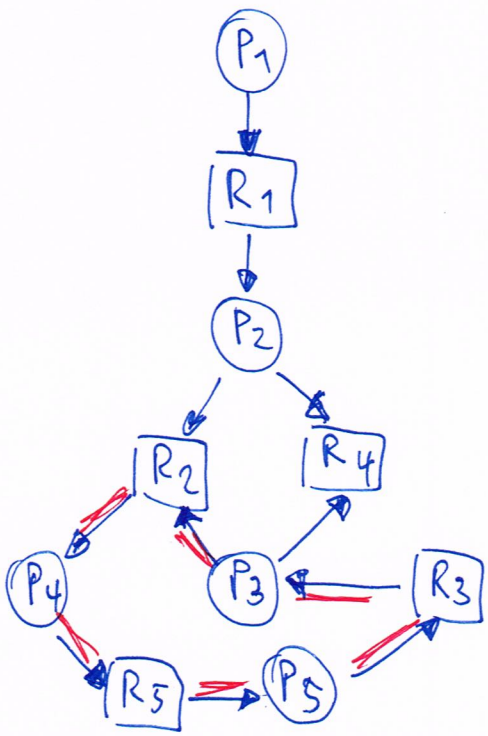
P_3 belegt R_3 und fordert R_2 und R_4 an,

P_4 belegt R_2 und fordert R_5 an,

P_5 belegt R_5 und fordert R_3 an.

Überprüfen Sie mit dem Ressourcen-Zuordnungs-Graph, ob ein Deadlock vorliegt, und falls ja, welche Threads an dem Deadlock beteiligt sind.





Kreis!
Deadlock P3 P4 P5

NEUE AUFGABENTYPEN (BEISPIELE)

Auf der Folgeseite finden Sie eine **neue** Beispielaufgabe. Weitere Aufgaben reiche ich eventuell vor dem Prüfungstermin nach.

10. Semaphor-Emulation (8 Punkte)

Betrachten Sie den folgenden unvollständigen Pseudocode:

```
01 boolean verfuegbar[1..4] = { True, True, True, True };
02 mutex_t mutex = True;
03 boolean ok;
04 int ressourcen_id;
05
06 ok = False;
07 ressourcen_id = -1;     // keine Ressource genutzt
08
09 // ANFORDERN / wait()
10 while (! ok) {
11   lock (mutex);
12   for ( i in [1, 2, 3, 4] ) {
13     if (verfuegbar[i] == True) {
14       verfuegbar[i] = False;
15       ressourcen_id = verfuegbar;i    // merken, welche wir nutzen
16       ok = True;
17       break;                          // Erfolg: Schleife beenden
18     }
19   }
20   unlock (mutex);
21 }
22
23 // ZUGRIFF
24 ...
25
26 // FREIGABE / signal()
27 lock (mutex);
28 verfuegbar[ressourcen_id] = True;
29 unlock (mutex);
30 ressourcen_id = -1;
```

Hier wird das Verhalten eines auf 4 initialisierten Semaphors mit Hilfe eines Mutex `mutex` und einer Liste von vier Ressourcen-Zuständen `verfuegbar[]` simuliert. Threads, die die Ressource nutzen wollen, führen den Code in Zeilen 10-21 aus, nach der Nutzung der Ressource führen sie den Code in Zeilen 27-30 aus.

a) Stellt das Verfahren sicher, dass nie mehr als vier Threads gleichzeitig auf die 4-fach-Ressource zugreifen? Begründen Sie Ihre Antwoirt, indem Sie erläutern, was passiert, wenn ein 5. Thread zugreifen will.

b) Kann es irgendwo im Programmablauf zu **aktivem** Warten kommen? Falls ja, nennen Sie alle Stellen. z. 10-21 (ok == False)

c) Kann es irgendwo im Programmablauf zu **passivem** Warten kommen? Falls ja, nennen Sie alle Stellen. z. 11, z. 27

d) Erläutern Sie einen Vorteil und einen Nachteil dieses Verfahrens im Vergleich zur Verwendung eines auf 4 initialisierten Semaphors. Würden Sie dieses Verfahren nutzen wollen?