1	2	3	4	5	6	7	8	9			Σ

Die Bearbeitungszeit der Probeklausur ist 80 Minuten; für die richtige Klausur haben Sie 120 Minuten Zeit. Entsprechend hat diese Probeklausur reduzierten Umfang (2/3). Bitte bearbeiten Sie alle Aufgaben. Es sind insgesamt 80 Punkte zu erreichen (richtige Klausur: 120 Punkte).

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

Viel Erfolg!

1. Bedienung der Shell (6 / 80 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`

d) Welches Verzeichnis verwendet der Benutzer `mueller` als Home-Verzeichnis?

/home/mueller

f) Wie findet die Shell ein Programm, wenn Sie als Befehl einen Programmnamen (z. B. `vi`) eingeben?

– Durchsuchen der Shell-Variable $PATH
– enthält mehrere, durch „:“ getrennte Verzeichnisse
– erste Datei, die passt, wird ausgeführt

2. Arbeiten mit Verzeichnissen (5 / 80 Punkte)

a) Sie befinden sich in Ihrem Home-Verzeichnis `/home/user` und wollen darin eine Hierarchie von Unterverzeichnissen `Daten`, `Daten/Briefe` und `Daten/Briefe/privat` erzeugen. Welche der folgenden Befehlszeilen erledigen die Aufgabe korrekt?

- ☐ `mkdir Daten/Briefe/privat`
- ☒ `mkdir -p Daten/Briefe/privat`
- ☒ `mkdir Daten; mkdir Daten/Briefe; mkdir Daten/Briefe/privat`
- ☐ `mkdir Daten; mkdir Briefe; mkdir privat`

b) Geben Sie ein `cd`-Kommando an, das eine **absolute** Pfadangabe verwendet, um aus dem Ordner `/usr/local/src` in den Ordner `/usr/share/doc` zu wechseln.

cd /usr/share/doc

c) Mit welchem Kommando löschen Sie rekursiv das im aktuellen Verzeichnis sichtbare Unterverzeichnis `Muell` (mit allen enthaltenen Dateien und Unterordnern)?

rm -r Muell

3. Jobs (11 / 80 Punkte)

a) Welche der folgenden Aussagen sind korrekt?

- [x] Mit der Tastenkombination Strg+Z können Sie (in der Regel) ein Programm unterbrechen.
- [x] 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.
- [x] Das Tool `jobs` zeigt zu jedem Job auch die Prozess-ID an. (nur über -l)
- [ ] Mit `nohup` können Sie ein Programm so starten, dass es sich nicht gewaltsam abbrechen lässt.
- [x] 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`.

nice -5 beispiel oder nice -n 5 beispiel

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

Es macht ihn zu einem Hintergrund-Job (background); dafür muss er vorher suspendiert gewesen sein.

4. Prozesse (10 / 80 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.

15 (SIGTERM)

```
 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) Zwei Prozesse haben die Nice-Levels 0 (Prozess A) und 10 (Prozess B). Welcher von beiden hat die höhere Priorität?

Prozess A. Der andere ist „nicer“.

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

renice -5 12345

d) 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?

0 (= 5 + (–5))

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

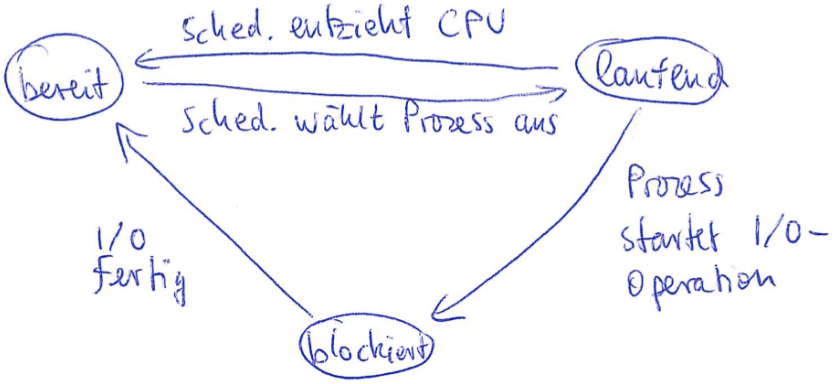
Der Job startet als Hintergrund-Job, die Shell ist direkt wieder benutzbar.

f) Sie wollen ein Dateisystem aushängen (unmounten), erhalten aber eine Fehlermeldung. Woran kann das liegen? Nennen Sie zwei Möglichkeiten (Stichworte reichen).

1) Ein Prozess nutzt eine Datei auf dem FS
2) Ein Prozess hat einen Ordner auf dem FS als aktuelles Arbeitsverzeichnis

5. Prozess-Zustände (10 / 80 Punkte)

a) Die drei wichtigsten Prozesszustände sind **bereit**, **laufend** und **blockiert**. Beschreiben Sie, welche Übergänge zwischen diesen Prozessen möglich sind und warum es zu diesen Übergängen kommt. (Es reicht aus, die Zustände und Übergänge in einer Grafik zu zeichnen und die Übergangspfeile mit Stichworten zu erläutern.)



b) **Threads** und **Prozesse** wechseln zwischen unterschiedlichen Zuständen hin und her. Nennen Sie zwei Zustände, die es nur bei Prozessen gibt, und begründen Sie jeweils kurz, warum es nicht sinnvoll ist, diese Zustände für Threads zu definieren.

„swapped“: Das bedeutet ja, dass der Speicher auf Platte ausgelagert ist. Hier können sich die Threads eines Prozesses aber nicht unterscheiden, weil sie denselben Speicher nutzen.

„stopped“ (von Benutzer angehalten), zumindest unter Linux lassen sich einzelne Threads nicht per kill-Befehl anhalten und fortsetzen.

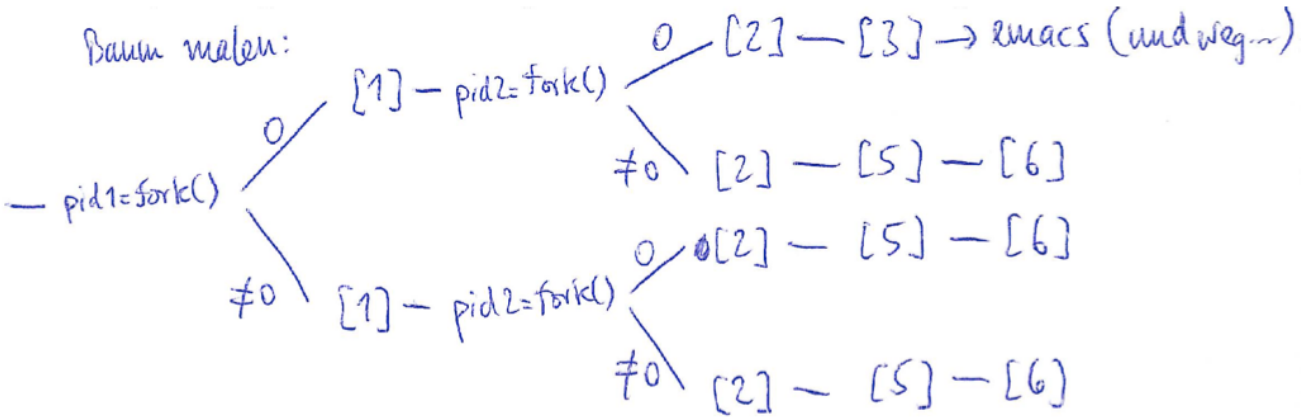
6. System calls (5 / 80 Punkte)

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.");
...
```

} wird nie ausgeführt!

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.



7. Scheduler (9 / 80 Punkte)

Welche der folgenden Aussagen sind korrekt?

- [X] 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.
- [X] **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“.
- [X] 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“).

☐ FCFS ist ein präemptives Scheduling-Verfahren.

☐ SJF ist eine präemptive Variante von SRT.

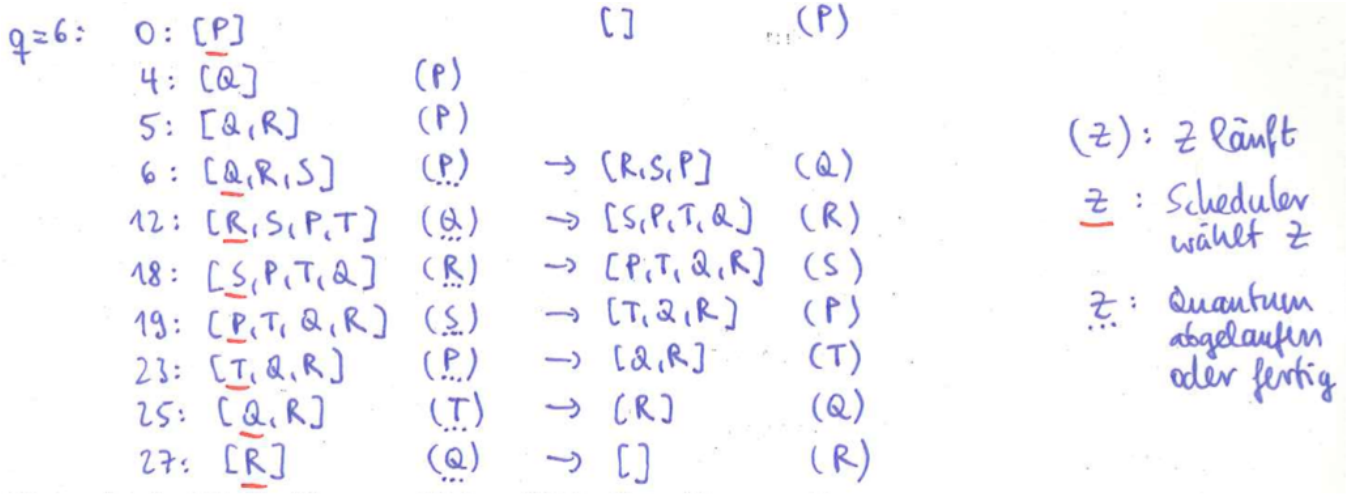
8. Scheduling-Verfahren (15 / 80 Punkte)

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

Zeit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
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	**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**
RR (q=15)	**P**	P	P	P	P	P	P	P	P	P	**Q**	Q	Q	Q	Q	Q	Q	Q	**R**	R	R	R	R	R	**S**	**T**	T

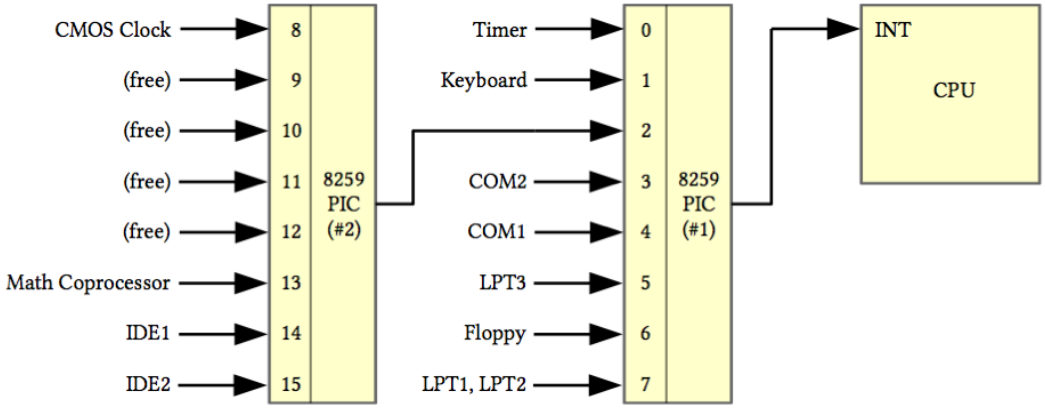


b) Laut Vorlesung ist das Round-Robin-Verfahren (**RR**) unfair gegenüber I/O-lastigen Prozessen. Erklären Sie kurz (in Stichworten), woran das liegt und wie **VRR** (Virtual Round Robin) die Situation verbessert.

I/O-lastige Prozesse laufen nur kurz, weil sie immer wieder mit I/O blockieren; sie verbrauchen dabei nur einen kleinen Teil ihres Quantums. Nach Aufhebung der Blockade müssen sie sich wieder hinten in der Warteschlange anstellen.

9. Interrupts, Faults und System Calls (9 / 80 Punkte)

a) Erklären Sie (stichwortartig oder via Skizze), wie bei der klassischen Intel-Architektur die Kaskadierung der zwei PICs funktioniert. Gehen Sie auch auf das Bestätigen eines Interrupts ein – abhängig davon, ob dieser vom ersten oder zweiten PIC kommt.



Bei Interrupts vom zweiten PIC müssen **beide** PICs ein ACK erhalten.

b) ULIX: Interrupt-, Fault- und System-Call-Handler bauen jeweils mit PUSH-Befehlen (im Assembler-Quellcode) den Stack so auf, dass dort die Datenstruktur `context_t` liegt. Alle Handler erhalten beim Aufruf einen Pointer auf diese Struktur als Argument. Wofür können die Elemente der Struktur innerhalb der Handler genutzt werden? Geben Sie zwei Beispiele.

Über die context_t-Struktur sind die auf den Stack gepushten Register-Inhalte (darunter auch EAX, EBX usw.) **erreichbar**, und sie können auch aus den Handlern heraus **verändert** werden – die Stack-Inhalte werden ja später wieder zurück in die Register geschrieben (mit den diversen POP-Befehlen)

Praktischer Nutzen: u. a.
– Parameterübergabe für System Calls,
– Rückgabewert aus System Call,
– Ändern der Rücksprungadresse (bei Rückkehr in User Mode) → nicht in Vorlesung besprochen

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

☐ Faults werden *asynchron* verarbeitet. ← nein, synchron. Interrupts sind asynchron.

☐ 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. ← kein Prozess! (der Rest stimmt)

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

☐ ULIX: In die Tabelle `void *interrupt_handlers[]` werden die Adressen spezifischer Interrupt-, Fault- und System-Call-Handler eingetragen. ← nur Interrupts!