Name: _____

Arbeitszeit: 45 min.
Hilfsmittel: Taschenrechner

1. Prozesse & Threads (10 Punkte)

a) Erklären Sie den Unterschied zwischen Prozessen und Threads.

b) Unix-artige Systeme (darunter auch Linux) haben eine Standardmethode für das Erzeugen neuer Prozesse. Beschreiben Sie den Start eines neuen Prozesses: Welchen System Call verwenden Sie, welche Parameter und Rückgabewerte hat er, und welche Bedeutungen haben diese?

c) 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.

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

2. Prozess-Zustände (5 Punkte)

a) Nennen und erklären Sie die drei wichtigsten Zustände, zwischen denen Prozesse (und auch Threads) hin und her wechseln.

b) Neben den drei Standard-Zuständen gibt es noch weitere. Beispielsweise „ausgelagert / swapped“. Ein Prozess habe drei (POSIX-) Threads erzeugt. Der Prozess ist im Zustand „ready“, alle drei Threads sind im Zustand „ausgelagert“ – warum kann das nicht sein?

3. Interrupts (12 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?

4. System calls (6 Punkte)

a) Was ist ein Dateideskriptor und wie „erzeugen“ Sie ihn?

b) Betrachten Sie den folgenden Programmausschnitt:

```
...
int pid = fork();
printf ("%s\n","[1] Zeit für eine Fallunterscheidung");
if (pid) {
  printf ("%s\n","[2] Ich starte jetzt emacs/fstab");
  execl ("/bin/emacs", "/etc/fstab", (char *)NULL);
} else {
  printf ("%s\n","[3] Ich starte jetzt emacs/hosts");
  execl ("/bin/emacs", "/etc/hosts", (char *)NULL);
};
printf ("%s\n","[4] Zwei Editoren erfolgreich gestartet");
int pid2 = fork();
printf ("%s\n","[5] Einer geht noch...");
execl ("/bin/emacs", "/etc/resolv.conf", (char *)NULL);
printf ("%s\n","[6] Jetzt läuft auch der dritte.");
...
```

Wie viele Editoren startet dieses Programm? Welche Meldungen gibt es – wie oft – auf der Konsole aus? (Die Ausgaben sind mit [1] bis [5] durch nummeriert; geben Sie nur die Nummern der Meldungen aus, die auf der Konsole erscheinen.)

5. Scheduling-Verfahren (Uni-Prozessor) (10 Punkte)

a) Aus der Vorlesung kennen Sie die Schedulung-Verfahren FCFS (First Come First Served), Shortest Job First (SJF) und Shortest Remaining Time Next (SRT).

Es gebe die folgenden vier Prozesse mit den angegebenenen Ankunftszeiten und Gesamtrechenzeiten:

Prozess	**Ankunft**	**Rechenzeit**
P	0	10
Q	4	5
R	5	10
S	6	1

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 R R R R R	R R R R R S
SJF			
SRT			

Ergänzen Sie für SJF und SRT (hier auf dem Blatt) die Ausführreihenfolgen.

b) Beim *Round-Robin*-Scheduling-Verfahren gibt es die Empfehlung, das Zeitquantum (nach dem ein Prozesswechsel erfolgt) etwas größer zu wählen als die typische Zeit, die zum Abarbeiten einer Interaktion nötig ist. Warum?

c) Erläutern Sie den Begriff *Prioritätsinversion*, der im Zusammenhang mit Prioritäten-Schedulern auftaucht. Wie kann man verhindern, dass er zu Prioritätsinversion kommt?

d) Was ist die Grundidee des *Fair-Share-Scheduling*?

6. Scheduling-Verfahren (Multi-Prozessor) (4 Punkte)

a) Erläutern Sie das Konzept des *Gang-Scheduling*.

b) Linux verwendet seit Kernel-Version 2.6 einen O(1)-Scheduler. Erläutern Sie, worauf sich hier die Bezeichnung O(1) bezieht. Wie findet der Linux-Scheduler den nächsten auszuführenden Prozess?

7. Seitenersetzung (5 Punkte)

a) In der Vorlesung haben Sie verschiedene Seitenersetzungsstrategien kennen gelernt. Eine davon wurde als „optimale Strategie“ vorgestellt, aber als nicht implementierbar verworfen. Beschreiben Sie die optimale Strategie und begründen Sie, warum diese nicht implementierbar ist.

b) Wenn Sie die Strategien LRU (least recently used) und FIFO (first in, first out) vergleichen, führt eine davon in der Regel zu weniger Page Faults. Warum?

8. Virtuelle Adressen (6 Punkte)

Eine CPU arbeitet mit folgenden Werten:

- Seitengröße: 16 KByte
- 47 Bit lange virtuelle Adressen
- 3-stufiges Paging; alle Seitentabellen sind gleich groß
- 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 auf dem Blatt ein und beschriften Sie die Abschnitte geeignet.

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

9. Speicher / Dateisysteme (5 Punkte)

In den Kapiteln zu Speicherverwaltung und zu Dateisystemen haben Sie mehrere Ähnlichkeiten in den Problemstellungen und Lösungen gesehen, beispielsweise sind die Konzepte mehrstufiger Indirektionsblöcke (Dateisystem) und mehrstufigen Pagings (Speicherverwaltung) verwandt.

a) Erläutern Sie kurz die Konzepte „Indirektionsblöcke“ und „mehrstufiges Paging“ und begründen Sie, warum es sich um verwandte Ansätze handelt.

b) Geben Sie zwei weitere Beispiele für die Verwandtschaft der beiden Themen.

10. Synchronisation (14 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.)

11. Dateisysteme (7 Punkte)

a) Welche Aufgabe hat unter Linux das Virtual Filesystem?

b) Journaling verlangsamt den Zugriff auf ein Dateisystem, weil zusätzlicher Aufwand betrieben wird. Sind davon Lese- oder Schreibzugriffe oder beide betroffen? Warum nimmt man den Performance-Verlust hin?

c) Freie Blöcke eines Dateisystems muss man sich in einer Datenstruktur merken. Skizzieren Sie kurz zwei Möglichkeiten, diese Informationen effizient zu speichern. (Effizienz bedeutet dabei u.a., dass aus Geschwindigkeitsgründen für den Zugriff i.d.R. ein Zugriff auf Daten im Hauptspeicher ausreichen sollte.)

12. Segmentierung (7 Punkte)

a) Erklären Sie, was Segmentierung ist. Wie funktioniert im Fall von Segmentierung die Adressübersetzung? Zeichnen Sie eine Skizze, aus der man ableiten kann, wie die Adresse in eine physikalische Adresse übersetzt wird.

b) Nennen Sie zwei Vorteile, die sich aus der Verwendung von Segmenten ergeben.

c) Warum kann ein Betriebssystem, das mit Segmentierung (ohne Paging) arbeitet, keine Programme ausführen, deren Speicherbedarf größer als der physikalisch vorhandene Speicher ist?