



## 1. Prozesse & Threads

(10 Punkte)

- a) Erklären Sie den Unterschied zwischen Prozessen und Threads. [4 P.]

Der wesentliche Unterschied ist, dass Prozesse über einen eigenen Speicherbereich verfügen, während sich alle (zum gleichen Prozess gehörenden) Threads den (Prozess-) Speicherbereich teilen.

- 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? [2 P.]

Neue Prozesse erzeugt man durch „Forken“, also mit `fork()`.

Parameter: keine

Rückgabewert: Eine Prozess-ID `pid`; der Vaterprozess erhält die PID des neuen (Sohn-) Prozesses zurück, der Sohn-Prozess erhält 0 zurück. Im Fehlerfall (Prozesserzeugung gescheitert) ist der Rückgabewert -1.

- 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. [2 P.]

```
void *helloworld () {  
    printf ("Hello World \n");  
}  
pthread_t thread[2];  
for (i=0; i<2; i++) { pthread_create (&thread[i], NULL, helloworld, NULL); }  
for (i=0; i<2; i++) { pthread_join (&thread[i], NULL); };
```

Threads, die mit `pthread_create` erzeugt werden, starten direkt - es gibt keine Funktion namens `pthread_start()`, `pthread_run()` etc.

- d) Wie unterscheiden sich allgemein User- und Kernel-Level-Threads und welche Vor- und Nachteile sind damit jeweils verbunden? [2 P.]

Kernel-Level-Threads sind dem Kernel bekannt, genau wie Prozesse laufen sie über den Scheduler des Kernels. Im Gegensatz dazu kennt der Kernel User-Level-Threads nicht; hier sind (Thread-) Bibliotheksfunktionen für Erzeugen und Scheduling der Threads verantwortlich.

KL-Threads verursachen hohen Aufwand bei Kontext-Switches (Umschalten in den Kernel-Mode), dafür ermöglichen sie aber, dass etwa ein Thread mit I/O blockiert ist, während andere Threads (des gleichen Prozesses) weiter laufen können.

UL-Threads müssen beim Thread-Wechsel nicht in den Kernel-Mode schalten (der OS-Scheduler hat damit nichts zu tun), dafür blockiert im I/O-Fall aber der gesamte Prozess.



## 2. Prozess-Zustände

(5 Punkte)

- a) Nennen und erklären Sie die drei wichtigsten Zustände, zwischen denen Prozesse (und auch Threads) hin und her wechseln. [3 P.]

running / laufend: Prozess läuft gerade (hat eine CPU erhalten, die seinen Code ausführt)

ready / bereit: Prozess ist ausführbereit - es fehlt ihm nur eine CPU

blocked / blockiert: Prozess ist blockiert. Um wieder bereit zu werden, muss ein Ereignis eintreten (z. B. Abschluss einer I/O-Operation, nach der er aufgeweckt wird). (User-Level-Threads können natürlich nicht schlafen; Kernel-Level-Threads schon.)

- 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? [2 P.]

Einen Thread auszulagern, *würde* bedeuten, seine Speicherseiten auf Platte zu schreiben und aus dem Speicher zu entfernen - Threads haben aber keinen eigenen Speicher, es müsste also der Prozessspeicher ausgelagert werden. Dann wäre der Prozess im Zustand „ausgelagert“ und nicht im Zustand „ready“.

## 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. [4 P.]

Die Alternative heißt „Polling“: Das Betriebssystem wartet nicht auf einen Interrupt vom Gerät, sondern fragt den Gerätestatus regelmäßig ab und handelt dann entsprechend der erhaltenen Statusinformationen.

Beim Polling befindet sich das OS in einer ständigen Schleife, die viele unnütze Anfragen an die Hardware stellt; diese Rechenzeit geht den Prozessen verloren. Jeder Gerätetreiber muss eine weitere solche Endlosschleife im System etablieren.

- 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. [4 P.]

1. „verschachtelte“ Interrupts nicht zulassen; Interrupts landen dann in einer Warteschleife und werden erst abgearbeitet, wenn der aktuell behandelte Interrupt komplett verarbeitet wurde. Vorteil: Das ist einfach. Nachteil: Manche Interrupts sind wichtiger als andere, müssen aber trotzdem auf die (vielleicht viel Zeit benötigende) Fertigstellung der letzten Interrupt-Bearbeitung warten.

2. Interrupt-Prioritäten: Jedem Interrupt wird eine Priorität zugeordnet. Solche mit höherer Priorität können aktuell bearbeitete Interrupts mit niedrigerer Priorität unterbrechen. Nach Abarbeitung des höher priorisierten Interrupts geht es mit dem unterbrochenen Interrupt mit niedrigerer Priorität weiter. Vorteil: Wichtige Dinge werden sofort erledigt. Nachteil: Aufwendigere Implementierung, Interrupt-Handler müssen mit Unterbrechungen umgehen können.



- 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 P.]

Die top half ist der eigentliche Interrupt Handler, der vom OS beim Auftreten eines Interrupts aktiviert wird. Er führt nur die wichtigsten (zeitkritischen) Dinge durch, etwa Abfragen eines Gerätestatus und Rücksetzen des Geräts („Rückmeldung: Ich habe Deinen Interrupt gesehen und bearbeitet“).

Die bottom half (Tasklet) führt die restlichen Schritte durch, die nicht zeitkritisch sind und darum nicht in der top half stattfinden.

Der Sinn der Trennung ist, möglichst schnell den eigentlichen Interrupt Handler zu beenden, um (im Fall der Deaktivierung von weiteren Interrupts) Interrupts wieder zuzulassen.

#### 4. System calls

(6 Punkte)

- a) Was ist ein Dateideskriptor und wie „erzeugen“ Sie ihn? [2 P.]

Dateideskriptor ist ein Identifizier für eine geöffnete Datei, der von allen Dateioperationen (Lesen, Schreiben, Schließen, Seek etc.) benutzt wird.

Erzeugen durch Öffnen der Datei: `fd = open (...);` `fd = creat (...);`

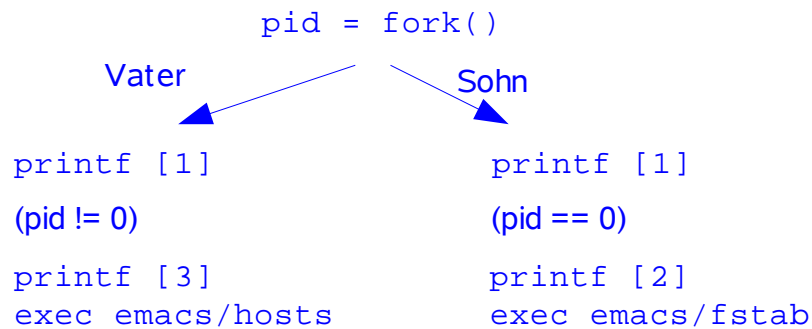
- b) Betrachten Sie den folgenden Programmausschnitt:

```
...
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.) [4 P.]



Das Programm startet zwei Editoren:



Und es gibt nur [1], [1], [2], [3] aus. Der Teil ab Ausgabe [4] (also auch der zweite fork) wird nie ausgeführt, weil nach dem ersten fork() sowohl Sohn als auch Vater nach den ersten Ausgaben mit exec() ein anderes Programm nachladen.

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

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

Es gebe die folgenden vier Prozesse mit den angegebenen 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	0 1 2 3 4 5 6 7
		10	20
<b>FCFS</b>	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
<b>SJF</b>	<b>P</b> P P P P P P P P P	<b>S</b> Q Q Q Q Q <b>R</b> R R R R	R R R R R R
<b>SRT</b>	<b>P</b> P P P Q Q <b>S</b> Q Q Q	<b>P</b> P P P P P <b>R</b> R R R R	R R R R R R

Ergänzen Sie für SJF und SRT (hier auf dem Blatt) die Ausführreihenfolgen. [2 P.]

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? [2 P.]

Weil dann (die Reaktion auf) eine Interaktion komplett in einem Quantum abgearbeitet werden kann und das System schnell auf die Eingabe etc. reagiert. Ist das Quantum kleiner, wird der interaktive Prozess unterbrochen.

c) Erläutern Sie den Begriff *Prioritätsinversion*, der im Zusammenhang mit Prioritäten-Schedulern auftaucht. Wie kann man verhindern, dass es zu Prioritätsinversion kommt? [4 P.]

Prioritätsinversion bedeutet: Ein Prozess mit hoher Priorität ist blockiert, wartet auf einen Prozess mit niedrigerer Priorität - letzterer kommt aber nie dran, weil immer höhere Prozesse laufen.



Eine Möglichkeit, das zu verhindern, ist die dynamische Anpassung der Prioritäten: Je länger ein Prozess (auf die CPU) warten muss, desto höher wird seine Priorität.

d) Wie unterscheiden sich Round Robin und Virtual Round Robin? [2 P.]

Round Robin hat den Nachteil, I/O- und CPU-lastige Prozesse ungleich zu behandeln: I/O-lastige Prozesse laufen immer nur ganz kurz und unterbrechen dann für die nächste I/O-Phase - müssen sich aber wieder ganz hinten in der Warteschlange anstellen. Virtual RR „merkt“ sich die nicht verbrauchte CPU-Zeit und ordnet den durch I/O unterbrochenen Prozess in eine Extrawarteschlange ein.

## 6. „Tape-Paging“

(2 Punkte)

Wenn Sie einen externen Speicher (etwa ein Tape-Roboter, der Daten auf eine Reihe von Datenbändern sichern und zurücklesen kann) hätten, der erheblich langsamer als Festplatten ist, könnten Sie eine Art „Tape-Paging“ implementieren und bei voller Festplatte Daten auf diesen externen Speicher auslagern.

Würden Sie eher einzelne Cluster oder ganze Dateien auslagern? Warum?

Es ist besser, ganze Dateien auszulagern, weil es unwahrscheinlich ist, dass von einer Datei nur Fragmente benötigt werden; wenn das komplette Lesen einer auf Tape ausgelagerten Datei dann gleich mehrere „Tape-Faults“ verursacht, entsteht erheblicher Zusatzaufwand.

## 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. [2 P.]

Die optimale Strategie ist: Die Seite ersetzen, die am längsten nicht neu angefordert wird. Das ist nicht implementierbar, weil die Zukunft nicht bekannt ist - es gibt keine Möglichkeit, herauszufinden, welche Seite das sein wird.

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? [3 P.]

LRU verursacht i.d.R. weniger Page Faults. Das liegt an dem Lokalitätsprinzip: Nach dem Zugriff auf eine Seite ist die Wahrscheinlichkeit hoch, dass kurzfristig nochmals darauf zugegriffen wird.

## 8. Virtuelle Adressen

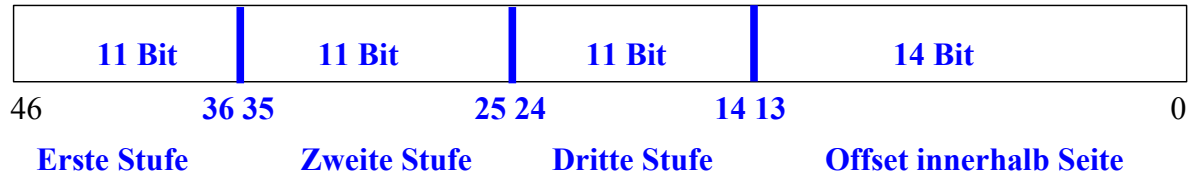
(6 Punkte)

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)?



16 KByte sind  $2^{14}$  Byte, also ist der relative Offset innerhalb einer Seite 14 Bit.

47 Bit - 14 Bit = 33 Bit, darum  $33 \text{ Bit} / 3 = 11 \text{ Bit pro Seitentabelle}$

Zeichnen Sie 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?

Auf der ersten Stufe gibt es eine Tabelle mit  $2^{11} = 2048$  Einträgen.

Auf der zweiten Stufe gibt es  $2^{11} = 2048$  Tabellen mit je 2048 Einträgen.

Auf der dritten Stufe gibt es  $2^{22} = 2048^2 = 4194304$  Tabellen mit je 2048 Einträgen.

Jede Tabelle belegt  $2048 * 8 \text{ Byte} = 16 \text{ KByte}$ , also genau eine Seite.

## 9. Synchronisation

(14 Punkte)

- a) Was ist ein Mutex? [2 P.]

Ein Mittel zur Prozess-/Thread-Synchronisation, das gegenseitigen Ausschluss (mutual exclusion) garantiert. Man schützt damit kritische Bereiche, die stets nur von einem Prozess/Thread betreten werden dürfen.

- 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? [4 P.]

Da Spin Locks nicht schlafen, darf man sie in Interrupt-Handlern benutzen. Interrupt-Handler sind keine Prozesse und dürfen sich darum nicht schlafen legen - es gäbe danach keine Möglichkeit, sie zu wecken.

- c) Beschreiben Sie das Erzeuger-Verbraucher-Problem und skizzieren Sie mit Hilfe von Pseudo-Programm-Code eine Semaphore-basierte Lösung. (Verwenden Sie wahlweise eine an C oder an Python erinnernde Syntax; exakte Befehlsnamen etc. sind irrelevant.) [8 P.] [zu viele Punkte]

```
const N=4;           // maximale Anzahl Speicherplätze
int sem_write=N;    // Initialisierung der Semaphore
int sem_read=0;

while(1) {          // Prozess Producer
    wait (sem_write);
    write (value);  // Schreibzugriff auf Speicher
```



```
signal (sem_read);  
}  
  
while(1) { // Prozess Consumer  
    wait (sem_read);  
    read (value);  
    signal (sem_write);  
}
```

## 10. Dateisysteme

(7 Punkte)

- a) Ein Dateisystem arbeite mit Ketten-Allokation (*Chained Allocation*) der Cluster, und eine Datei in diesem Dateisystem bestehe aus  $N$  Clustern. Wie viele Plattenzugriffe sind erforderlich, um
- den ersten Cluster
  - den mittleren Cluster (Nummer  $N/2$ , falls  $N$  gerade ist)
  - den letzten Cluster (Nummer  $N$ )
- zu lesen? [2 P.]

Es sind 1,  $N/2$  bzw.  $N$  Zugriffe nötig.

- 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? [2 P.]

Es sind nur Schreibzugriffe betroffen; beim Lesen fällt kein Overhead für Journaling an.

Man nimmt den Performance-Verlust hin, weil die Dateisystem-Integrität gesichert wird.

- c) Beschreiben Sie in Stichworten, wie der Shortest Service Time First (SSTF) Disk Scheduler funktioniert. [3 P.]

Der SSTF-Disk Scheduler wählt die Anforderung aus, die die kleinste Bewegung des Schreib-/Lesekopfes von der aktuellen zur Zielposition erfordert.

## 11. Speicherzuteilung vs. Cluster-Allokation

(7 Punkte)

- a) Sowohl bei der Speicherverwaltung als auch bei den Dateisystemen basieren die einfachsten Verfahren darauf, Prozessspeicher bzw. die Cluster einer Datei zusammenhängend anzuordnen. Erklären Sie in beiden Themengebieten die Begriffe „externe Fragmentierung“ und „Kompaktierung / Defragmentierung“ (also insgesamt vier Begriffserklärungen).

- Externe Fragmentierung (Speicher): Der Speicher wird zusammenhängend verwaltet und dynamisch in Partitionen zerlegt. Durch Vergeben und Wieder-Freigeben entstehen immer mehr kleine unbenutzte Bereiche, die theoretisch für Anforderungen genutzt werden könnten - in der Praxis aber zu klein sind und ohne Verwendung bleiben.

- Externe Fragmentierung (Dateisystem): Dateien belegen im Regelfall aufeinander folgende Cluster. Nach Löschen von Dateien entstehen freie Cluster-Bereiche, die für die nächste erzeugte Datei verwendet werden. Ist diese größer als der zusammenhängende freie Bereich, werden Cluster an anderen Stellen verwendet. Eine Datei verteilt sich damit über mehrere voneinander getrennte Bereiche.



- Kompaktierung (Speicher): Benutzte Speicherbereiche werden zusammengeschoben, so dass es zwischen ihnen keine unbenutzten Lücken mehr gibt. Danach liegt der ganze freie Speicher zusammenhängend am Ende und kann auch für Anforderungen nach großen Speicherbereichen genutzt werden.

- Defragmentierung (Dateisystem): Die Cluster der Dateien werden so umgeordnet, dass jede Datei einen zusammenhängenden Cluster-Bereich nutzt.

- b) Viele Konzepte der Speicherverwaltung gibt es auch bei den Dateisystemen. Die beliebige Verteilung von Speicherseiten in den Seitenrahmen des Hauptspeichers lässt sich (sehr grob) mit der indizierten Allokation (*indexed allocation*) von Clustern vergleichen. Warum verursacht dies bei Dateisystemen (gegenüber zusammenhängender Anordnung) stärkere Performance-Verluste als bei der Speicherverwaltung? [1 P.]

RAM lässt sich direkt ansprechen, beim Zugriff auf einen Festplattensektor muss erst der Schreib-/Lesekopf an die richtige Position bewegt werden. Diese Seek-Zeiten sind bei zusammenhängender Anordnung der Cluster einer Datei vernachlässigbar (minimale Positionskorrektur), bei indizierter Allokation kommen größere Seek-Zeiten zu den eigentlichen Lesezeiten hinzu.