Lösungsvorschlag zur Klausur „Betriebssysteme“ vom 1.10.2014

Blau gekennzeichnete Textstellen sind beispielhafte Lösungen bzw. Antworten zu den Aufgaben.
Rot gekennzeichnete Textstellen sind Anmerkungen, die nicht zu den Antworten gehören, sondern auf typische Fehler oder den Lösungsweg hinweisen.

Aufgabe 1 (4 Punkte)

Ein Benutzer ruft ein ausführbares Maschinenprogramm auf. Das Betriebssystem findet die Programmdatei und stellt sicher, dass die Zugriffsrechte für den Aufruf vorhanden sind. Beschreiben Sie detailliert, wie der Rest der Verarbeitung des Programmaufrufs durch das Betriebssystem erfolgt. Geben Sie insbesondere auch an, welche logischen Hauptspeicherbereiche für das neue Programm benötigt werden und wie dafür gesorgt wird, dass die CPU das Programm bearbeitet.

- BS erzeugt Prozessdeskriptor u. reserviert Programmspeicher für Kontext, Code, statische Daten, Stack, Heap
- Code und initialisierte Daten werden aus der Programmdatei in den Hauptspeicher kopiert
- Befehlszähler wird mit 0 initialisiert
- Deskriptor wird in die Liste zum Zustand „bereit“ (oder „neu“) eingefügt

Aufgabe 2 (1+1+1+2+1 Punkte)

In einem Zustandsübergangsdiagramm für Prozesse gibt es einen Zustand $terminiert$ (auch „Zombie“ genannt).

a) Was bedeutet dieser Zustand?

Prozess ist terminiert, aber Deskriptor noch in Prozesstabelle

b) Warum gibt es einen solchen Zustand überhaupt?

Elternprozess kann später Terminierungsinfo aus dem Kind-Deskriptor abfragen

c) Von welchen der Zustände $neu, bereit, blockiert, ausfuehrend$ gibt es Übergänge in diesen Zustand $terminiert$?

Von allen

d) Welche Ereignisse können zu einem Übergang von $ausfuehrend$ in den Zustand $terminiert$ führen?

Normales Ende (exit) und synchrone und asynchrone Signal-Ereignisse, z.B.

- Asynchron: Benutzer bricht Prozess ab (STRG-C, kill-Kommando, Sitzungsende)
- Synchrone Ereignisse: unzulässiger Speicherzugriff, Division durch 0, ...

e) Welche Ereignisse können zu einem Übergang von neu in den Zustand $terminiert$ führen?

Asynchrone Signale, z.B.

- Benutzer bricht Prozess ab (STRG-C, kill-Kommando, Sitzungsende)
- Rechner wird heruntergefahren, Power-Fail-Signal

Aufgabe 3 (2+2+2+2 Punkte)

a) Betrachten Sie zwei Threads eines C-Programms. Welche Variablen bzw. Hauptspeicherbereiche können diese Threads ohne weiteres gemeinsam nutzen und welche nicht?

Gemeinsam: Statische Daten und Heap (auch Kontext und Maschinencode), Separat: Laufzeitstack

b) Warum darf ein Prozess nicht ohne weiteres auf den Programmspeicher eines anderen Prozesses zugreifen?

BS muss sicherstellen, dass ein Programm (eines Benutzers) unabhängig von anderen Programmen (andererer Benutzer) abläuft. Dazu gehört ein Abschottungskonzept, das auch den Programmspeicher umfasst.

c) Wie verhindert ein modernes Betriebssystem mit virtuellem Speicherkonzept, dass ein Prozess die Daten eines anderen Prozesses manipuliert?

Der logische Adressraum eines Prozessen wird durch Seitentabellen auf einen für diesen Prozess exklusiv reservierten realen Adressraum abgebildet.

d) Was ist ein Kontextwechsel und funktioniert er?

Bei einem Kontextwechsel wird ein Prozessor von der Bearbeitung eines Prozesses (Threads) A zur Bearbeitung eines anderen Prozesses (Threads) B wechseln. Dazu muss i.d.R. der Maschinenzustand von A (Registerinhalte) gerettet (in den Hauptspeicher kopiert) und der alte Maschinenzustand von B restauriert werden.

Ein Kontextwechsel muss nicht mit einer Verdrängung zusammenhängen, sondern erfolgt häufig durch Blockieren oder Terminieren.

Aufgabe 4 (2+2+2+2+4 Punkte)

Betrachten Sie den UNIX-Shell-Befehl `ls -l /mydir | wc -c > outfile`

a) Welche Prozesse sind an der Ausführung beteiligt und wie ist deren Verwandschaftsverhältnis?

Der Shellprozess als Elternprozess, für wc und ls je ein Kindprozess

b) Bei der Ausführung kommen verschiedene Synchronisationsmaßnahmen zwischen den beteiligten Prozessen zur Anwendung. Nennen Sie alle Blockade-Situationen, die im konkreten Beispiel möglich sind. (Welcher Prozess wartet auf welches Ereignis?)

- ls muss ggf. im $write$ warten auf wc , falls die Pipe voll ist.
- wc muss ggf. im $read$ warten auf ls , falls die Pipe leer ist.
- Die Shell wartet auf die Terminierung beider Kinder.
- wc kehrt aus dem $read$ zurück, sobald ls terminiert

c) Was passiert, wenn es die Datei `/mydir` nicht gibt? Beschreiben Sie (in Stichworten), wie sich die beiden Programme „ls“ und „wc“ dann verhalten.

- ls bekommt Fehler beim $open$, schreibt Fehlermeldung auf $stderr$ und terminiert mit $exit(1)$
- wc blockiert im $read$ bis zur Terminierung von ls und gibt 0 in die Datei $outfile$ aus, weil seine Standardeingabe (Pipe) leer bleibt.
- Shell siehe (b).

ls schreibt nichts in die Standardausgabe (Pipe). wc erhält kein Signal. Der $read$-Aufruf in wc liefert einfach 0 (Bedeutung: „End of File“) zurück.

d) Welche Gründe könnten sonst noch dazu führen, dass die Befehlsausführung scheitert?

- Zugriffsrechtsproblem, z.B. wc darf $outfile$ nicht erzeugen oder ls darf $/mydir$ nicht lesen.
- Hauptspeichermangel, z.B. beim $fork$ oder $exec$
- Zu wenig Speicher im BS-Kern für Pipe
- Sonstiger Ressourcenmangel, z.B. maximale Prozessanzahl des Benutzers erreicht, Deskriptortabelle der Shell voll
- Besondere sonstige (asynchrone) Ereignisse, z.B. Benutzereingriff, System-Halt

Wichtig sind die Situationen, die das Programm abfangen muss, d.h. Fehler bei Systemaufrufen. Natürlich kann auch mitten in der Ausführung der Blitz einschlagen und der Rechner abrauchen! Auch wenn dadurch die Befehlsausführung scheitert, gibt es dafür keine Punkte!

e) Geben Sie ein C-Programm an, dass diesem Shell-Befehl entspricht. Das Programm soll zum Aufruf von `ls` und `wc` den Systemaufruf *execlp* verwenden. Vervollständigen Sie dazu das nachfolgende Programmskelett. Lassen Sie dabei jegliche Fehlerbehandlung und die `#include` -Anweisungen weg.

```
int main(){
    pid_t lspid, wcpid;
    int pipefd[2], outfilefd;

    pipe(pipedf); /* Pipe erzeugen, oeffnen */

    switch(lspid=fork()){
    case 0:
       /* Ausgabeumleitung in Pipe */
       dup2(pipefd[1], 1);
       close(pipefd[0]);
       close(pipefd[1]);

       execlp("ls", "ls", "/mydir", NULL);
       exit(1);
    }

    switch(wcpid=fork()){
    case 0:
       /* Eingabeumleitung auf Pipe */
       dup2(pipefd[0], 0);
       close(pipefd[0]);
       close(pipefd[1]);

       /* Ausgabeumleitung auf outfile */
       outfilefd=open("outfile", O_WRONLY|O_CREAT|O_TRUNC, 0644);
       dup2(outfilefd, 1);
       close(outfilefd);

       execlp("wc", "wc", "-c", NULL);
       exit(1);
    }

    close(pipefd[0]);
    close(pipefd[1]);

    waitpid(lspid,NULL,0);
    waitpid(wcpid,NULL,0);
}
```

Details bei den $open$-Parametern spielen keine Rolle für die Bewertung.

Typische Fehler: Pipe nicht geöffnet, outfile nicht geöffnet, Ausgabeumlenkung nach outfile fehlend/fehlerhaft, Verklemmung durch unnötig offene Pipe-Deskriptoren (im Shell-Elternprozess oder im wc).

Aufgabe 5 (3+2 Punkte)

Der Befehl

```
kill -TERM 1234
```

bewirkt die Generierung des Signals SIGTERM für den Prozess mit der PID 1234. Die Wirkung des Befehl hängt von verschiedenen Einstellungen ab.

a) Welche Konsequenzen sind möglich und wovon hängt die Wirkung ab?

Wirkung hängt von Benutzer-definierter Signalbehandlung (sigaction) ab:

- keine: Default-Behandlung beendet Prozess 1234
- Ignorieren: Es passiert nichts
- Signal-Handler registrieren: Der Signal-Handler wird aufgerufen

(Die Signalauslieferung kann außerdem durch Signalmaske verzögert werden.)

b) Wenn der Signalempfänger das Signal erhält, könnte er gerade Anwendungscode oder aber einen Systemaufruf ausführen. Spielt das für die Signalbehandlung eine Rolle (Begründung)?

Ja, weil ein Systemaufruf nie durch eine Signalbehandlung unterbrochen wird. Ein „langsamer“ Systemaufruf wird abgebrochen (und später ggf. neu gestartet), ein „schneller“ wird erst beendet.

Aufgabe 6 (2+2+2 Punkte)

Ein Programm will auf das 2. Cluster einer Datei zugreifen. Aus einer logischen Clusternummer kann die physikalische Adresse berechnet werden. Wie und wo findet das System die Clusternummer des 2. Clusters

a) bei einem FAT-Dateisystem

Sei n die Nummer des ersten Clusters der Datei. n steht im Verzeichnis. Die Nummer des 2. Clusters der Datei steht dann im n-ten Eintrag der FAT

b) bei einem klassischen UNIX-Dateisystem

Die Nummern der ersten 10 Datei-Cluster stehen direkt im I-Node der Datei.

c) bei Windows NTFS

Im MFT-Eintrag der Datei gibt es eine Liste von Fragmenten der Datei (Run-Liste). Für jedes Fragment ist die erste Clusternummer und die Clusteranzahl eingetragen. Daraus kann die Adresse berechnet werden. Im Beispiel ist das 2. Dateicluster entweder im selben Fragment wie das erste, oder es ist das erste Cluster im 2. Fragment.

Aufgabe 7 (1+1+2+2+1+2+2 Punkte)

Ein System verwendet virtuellen Speicher mit Paging. Für die Berechnung der 48 Bit großen realen Adressen wird eine dreistufige Seitentabelle benutzt. Eine virtuelle Adresse ist 30 Bit groß und von der Form | p_1 | p_2 | p_3 | D | mit folgender Aufteilung:

p_1: 8 Bit für die Seitentabelle der 1. Stufe
p_2: 8 Bit für die Seitentabelle der 2. Stufe
p_3: 4 Bit für die Seitentabelle der 3. Stufe
D: 10 Bit für die Distanz

Angenommen, die virtuelle Adresse *v* adressiert das 4. Byte der 5. Seite eines Prozesses, die im 13. Rahmen des Hauptspeichers steht. Zur Ermittlung der realen Adresse *r* werden 3 Seitentabellen benötigt.

a) Wo steht die Adresse der 3. benötigten Seitentabelle?

In der Seitentabelle der 2. Stufe im Eintrag p_2

b) In welcher der Seitentabellen steht die Distanz?

In keiner

Die Seitentabellen dienen zur Umsetzung logischer in reale Adressen. Die Distanz wird nicht umgerechnet.

c) Geben Sie die virtuelle Adresse dieses Bytes hexadezimal an:

0x1003

Die Zählung beginnt immer bei 0: Das 1. Byte hat die Distanz 0, das 4. Byte also die Distanz 3. Für die 5. Seite ist p_3 4. Für die Berechnung muss man die Adressen binär darstellen:
Seitennummer=100, Distanz(10-Bit)=00 0000 0011,
v(30 Bit)=00 0000 0000 0000 0001 0000 0000 0011

d) Geben Sie die reale Adresse dieses Bytes hexadezimal an:

0x3003

e) Wieviele Bit groß muss ein TLB-EIntrag (Schlüssel und Wert) sein?

(30-10) Bit + (48-10) Bit = 58 Bit

Die Distanzen stehen nicht im TLB, nur die Seiten- und Rahmennummern!

f) Geben Sie an, wie der zugehörige TLB-Eintrag aussieht:

(Schlüssel=Seitennummer $\rightarrow$ Wert=Rahmennummer) also (4$\rightarrow$12) oder (0x4$\rightarrow$ 0xC)

g) Kann ein Programmierer die TLB-Hit-Rate beeinflussen? Wenn ja, wie? (Begründung)

Ja. Speziell beim Durchlaufen sehr großer, über viele Seiten gehender Felder ist darauf zu achten, dass die Elemente gemäß ihrer physikalischen Anordnung im Speicher verarbeitet werden. Damit vermeidet das Programm unnötige Seitentabellenzugriffe.

Aufgabe 8 (2+2 Punkte)

Betrachten Sie folgende UNIX-Shell-Befehle und deren Ausgaben:

```
$ mkdir /tmp/v /tmp/v/v1           # Verzeichnisse erzeugen
$ ls -di /tmp /tmp/v /tmp/v/v1     # I-Node-Nummern anzeigen
881 /tmp  844 /tmp/v  846 /tmp/v/v1
```

a) Geben Sie möglichst exakt den Inhalt der Verzeichnisdatei `/tmp/v` an.

Verzeichnisdatei enthält Liste von Paaren (Dateiname, I-Node-Nummer). Inhalt von `/tmp/v` :

`"."`	844
`".."`	881
`"v1"`	846

b) Welche Hardlinks gibt es zum I-Node 844 ?

Ein Hardlink ist ein Verzeichniseintrag. Im Beispiel existieren 3 Hardlinks zur Datei 844:

Verzeichnis	**Hardlink zu 844 in diesem Verzeichnis**	
`/tmp`	`"v"`	844
`/tmp/v`	`"."`	844
`/tmp/v/v1`	`".."`	844