Prof. Dr. Michael Jäger FB MNI

Klausur “Betriebssysteme I” – 8.2.2013

Bitte bearbeiten Sie die Aufgaben auf den Aufgabenblättern. Die Benutzung eines Blattes mit Notizen ist erlaubt, aber nicht die Verwendung sonstiger Unterlagen oder Hilfsmittel. Die Bearbeitungszeit beträgt 90 Minuten.

Nachname: _____

Vorname: _____

Studiengang/Prüfungsordung: _____

Matrikelnummer: _____

Hausübungen abgegeben: (zutreffendes Ankreuzen)

☐ WS 12/13 ☐ SS 12 ☐ WS 11/12 ☐ SS 11 Sonstiges ☐

Unterschrift: _____

Aufgabe	Punktzahl maximal	Punktzahl erreicht
1	6	
2	4	
3	4	
4	4	
5	6	
6	4	
7	6	
8	7	
9	4	
10	5	
Summe	50	

Aufgabe 1 (2+2+2 Punkte)

Punkte ☐ *von* **6**

a) Ein Prozess im Zustand „blockiert“ wartet auf Daten vom Netzwerk. Sobald die Daten vorhanden sind, wird der Prozess „aufgeweckt“. Was bedeutet eigentlich „aufwecken“?

b) Beschreiben Sie in Stichworten, wie in einem monolithischen Betriebssystem ein Systemaufruf ausgelöst und vom Systemkern bearbeitet wird.

c) Ein Betriebssystem unterstützt Threads. Welche Ressourcen muss der Systemkern bei der Erzeugung eines neuen Thread bereitstellen?

Aufgabe 2 (4 Punkte)

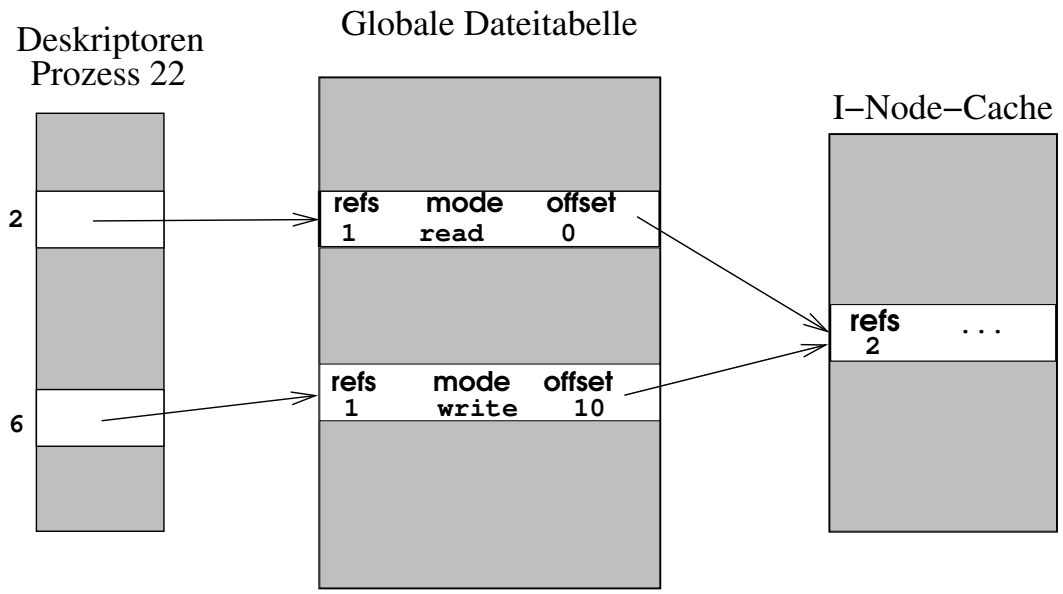
Punkte ☐ *von* **4**

Ein Benutzer eines UNIX-Systems betätigt die Tastenkombination Strg-C (Control-C) um ein Programm abzubrechen. Beschreiben Sie, was daraufhin passiert. Kommt es in jedem Fall zum Abbruch eines Programms? Wenn ja, welches Programm ist betroffen? Wie kann überhaupt das Drücken von Tasten zur Terminierung eines Programms führen?

Aufgabe 3 (4 Punkte)

Punkte ____ *von* **4**

Die Skizze unten enthält Informationen zu Dateien, die Prozess 22 zu einem bestimmten Zeitpunkt offen hat. Was passiert, wenn dieser Prozess nun den Systemaufruf `dup2(2,6)` ausführt. Wie ist der Zustand nach dem Aufruf?



Aufgabe 4 (2+2 Punkte)

Punkte ____ *von* **4**

a) Das Working-Set-Prinzip garantiert jedem Prozess ein Mindestkontingent an Seitenrahmen. Was passiert aber, wenn zu viele Programme gestartet werden, so dass gar nicht genug Rahmen für alle da sind?

b) Beschreiben Sie kurz, wie die Seitenauslagerungsstrategie „Second Chance“ funktioniert? Warum ist „Second Chance“ besser als „FIFO“?

Aufgabe 5 (1+1+2+2 Punkte)

Punkte ____ *von* **6**

Ein System verwaltet seine virtuellen Adressen mit 2-stufigen Seitentabellen. Eine Adresse v=(p1,p2,D) hat 32 Bit, davon p1 und p2 je 10 Bit und D 12 Bit.

Der virtuellen Adresse V=0140B704 ist eine reale Adresse im Rahmen 3 zugeordnet.

a) Geben Sie p1, p2 und D hexadezimal an.

p1= p2= D=

b) Geben Sie die reale Adresse hexadezimal an.

c) Geben Sie den TLB-Eintrag dazu hexadezimal an.

d) Wo genau steht die Adresse der Seitentabelle der zweiten Stufe zu V?

Aufgabe 6 (4 Punkte)

Punkte ____ *von* **4**

Von einer Shell mit der Prozessnummer 7 wird das nachfolgende Programm aufgerufen. Das Programm erhält die PID 8, der Kindprozess des Programms die PID 9. Welche Daten erscheinen in welcher Reihenfolge in der Standardausgabe?

```
#include <stdio.h>
#include <wait.h>
#include <unistd.h>

int main(){
   int p=fork();
   if(p) {
     printf("p=%d\n", p);
     printf("ppid=%d\n", getppid());
     }
   else
      printf("ppid=%d\n", getppid());

   waitpid(p, NULL, 0);
   printf("pid=%d\n", getpid());
 }
```

Aufgabe 7 (6 Punkte)

Punkte ____ *von* **6**

Die folgende Implementierung von vereinfachten Shell-Pipelines ist lückenhaft. Ergänzen Sie die Eingabeumlenkung im Leser und die Ausgabeumlenkung im Schreiber.

Auch im Hauptprozess fehlt noch etwas wichtiges. Erklären Sie warum. Ergänzen Sie das Fehlende.

```
int kommando::pipeline(){
    pid_t leser, schreiber;

    int pipefd[2];
    if(pipe(pipefd)){ perror("Fehler bei pipe"); exit(1); }

    switch(leser=fork()){
    case -1:  perror("Fehler bei fork"); return -1;
    case 0:
      // Pipe-Leser, Umlenkung der Eingabe




      execlp(args[1], args[1], NULL);
      perror("exec-Fehler beim Pipe-Leser");
      exit(1);
    }

    switch(schreiber=fork()){
    case -1: perror("Fehler bei fork"); exit(1);
    case 0:
      // Schreiber: Ausgabeumlenkung




      execlp(args[0], args[0], NULL);
      perror("exec-Fehler beim Pipe-Schreiber");
      exit(1);
    }




    // Warten auf Ende der beiden Pipeline-Prozesse
    waitpid(leser,NULL,0);
    waitpid(schreiber,NULL,0);
}
```

Aufgabe 8 2+2+3 Punkte

Punkte ____ *von* **7**

a) Ein Round-Robin-Scheduler verdrängt den ausführenden Prozess nach Ablauf seines Quantums von 1ms. Welche Konsequenzen hat das?

b) Wie könnte ein CPU-Scheduler dafür sorgen, dass das System immer schnell auf Mausklicks des Benutzers reagiert?

c) Welche Vorteile könnte es haben, wenn der Scheduler in einem Multiprozessorsystem immer versuchen würde, alle Threads eines Programms parallel auszuführen? Begründen Sie die Antwort.

Aufgabe 9 (4 Punkte)

Punkte ____ *von* **4**

In einem FAT-Dateisystem gibt es 10 Cluster mit den Nummern 2-11. Es gibt 2 Dateien, beide im Wurzelverzeichnis: Datei x mit den Clustern 2 und 5, und Datei y mit den Clustern 3,6 und 4. Geben Sie als Skizze den Inhalt des Wurzelverzeichnisses und der FAT an.

Aufgabe 10 (5 Punkte)

Punkte ____ *von* **5**

Ein Prozess P soll zwei nebenläufige Subprozesse erzeugen. Beide Subprozesse verhalten sich gleich: Zuerst wird die Funktion *f1*, danach die Funktion *f2* aufgerufen. Keiner darf aber *f2* aufrufen, bevor der andere den Aufruf von *f1* beendet hat. Beide Subprozesse von P müssen also ggf. auf ihre „Bruder"-Prozesse warten.

Geben Sie eine C-Implementierung an, die Pipes zur Synchronisation verwendet.

Vervollständigen Sie dazu das Programm:

```
#include <stdio.h>
#include <wait.h>
#include <stdlib.h>
#include <unistd.h>

void f1 () { printf("hier f1, aufgerufen von %d\n", getpid()); }
void f2 () { printf("hier f2, aufgerufen von %d\n", getpid()); }

int main(){
```