Prof. Dr. Michael Jäger FB MNI

Klausur “Betriebssysteme I” – 14.3.2008

Bitte bearbeiten Sie die Aufgaben auf den Aufgabenblättern.
Die Benutzung von Unterlagen oder Hilfsmitteln ist nicht erlaubt.
Die Bearbeitungszeit beträgt 90 Minuten.

Nachname: _____

Vorname: _____

Studiengang/Prüfungsordung: _____

Matrikelnummer: _____

Klausurvoraussetzungen (Hausübungen usw.) aus Semester:
(Zutreffendes ankreuzen)

WS 08/09	SS 08	WS 07/08	Sonstiges	

Unterschrift: _____

Aufgabe	Punktzahl maximal	Punktzahl erreicht
1	10	
2	4	
3	4	
4	6	
5	4	
6	4	
7	9	
8	9	
9	8	
Summe	58	

Aufgabe 1 2+2+2+2+2

Punkte ____ *von* **10**

a) Beschreiben Sie kurz, wie in einem monolithischen Betriebssystem ein Systemaufruf funktioniert.

b) Kann die Ausführung des Systemaufrufs $execlp$ zu einem Kontextwechsel führen? Gilt dies ebenso für $read$ und $getpid$? Begründen Sie die Antwort!

c) In einem UNIX-Multiprozessorsystem sollen mehrere Threads eines C-Programms echt parallel ausführbar sein. Die Parallelität wird durch fork()-Operationen implementiert. Welche andere Lösung kennen Sie? Welche Nachteile hat die fork()-Lösung?

d) Ein UNIX-Prozess will eine Datei zum Schreiben öffnen. Beschreiben Sie kurz, wie das Betriebssystem überprüft, ob der Prozess das darf?

e) Wie kommt es, dass in einer Prozesstabelle manchmal Prozessdeskriptoren von schon beendeten Prozessen stehen?

Aufgabe 2 4 Punkte

Punkte ____ *von* **4**

Sie testen unter UNIX ein selbstentwickeltes Programm. Der Prozess wird mit der Meldung „Division durch Null“ terminiert. Sie stellen fest, dass ihr Programm eine solche Division versucht hat. Allerdings haben Sie für diesen Fall weder die Ausgabe der Fehlermeldung noch den Abbruch der Ausführung programmiert.

Offenbar führt der Prozessor Befehle aus, die nicht im Programm stehen!
Wie kommt es dazu? Beschreiben Sie den Mechanismus!

Was ist daran UNIX-spezifisch?

Aufgabe 3 4 Punkte

Punkte ____ *von* **4**

Welche möglichen Ausgaben hat das folgende C++-Programm, wenn der Elternprozess die PID 5 und der Kindprozess die PID 6 hat?

```
#include <iostream>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <cstdlib>

using namespace std;
int main() {
  int i=0;

  cout << getpid() << endl;
  i=fork();
  cout << getpid() << ":" << getppid() << ":" << i << endl;
  waitpid(i,NULL,0);
  if (i==0)
    cout << getpid() << endl;
  exit(0);
}
```

Aufgabe 4 6 Punkte

Punkte ____ *von* **6**

Ein Prozeß hat zwei Subprozesse, die beide zuerst Funktion f, danach Funktion g aufrufen. Der Aufruf von g soll aber in beiden Subprozessen erst dann erfolgen, wenn der Geschwisterprozess aus dem Aufruf von f zurückgekehrt ist.

Geben Sie eine C++-Implementierung an, die Pipes zwischen den Geschwisterprozessen für die Synchronisation verwendet. Die Pipes sind „klassische“ Einweg-Kommunikationskanäle, keine Stream-Pipes !

Vervollständigen Sie dazu das folgende Programm.
(Fehlerbehandlung und include-Anweisungen lassen Sie dabei weg.)

```
using namespace std;

void f() { cout << "f aufgerufen von " << getpid() << endl; }
void g() { cout << "g aufgerufen von " << getpid() << endl; }

int main() {









  switch( fork() ) {
  case 0:


    f();


    g();


    exit(0);
  }

  switch( fork() ) {
  case 0:

    f();


    g();


    exit(0);
  }
}
```

Aufgabe 5 2+2

Punkte ☐ *von* **4**

Mehrere Prozesse greifen konkurrierend auf Dateien zu und benutzen dabei Dateisperren für Exklusivzugriff.

Die Notation $< PID >: lock(D)$ soll bedeuten, dass ein Prozess $< PID >$ eine Datei D exclusiv sperrt, nachdem er ggf. auf die Freigabe durch einen anderen Prozess gewartet hat.

a) Geben Sie für den Zustand, der durch die nachfolgende Sequenz von Lock-Operationen entsteht, ein Ressourcenzuordnungsdiagramm an.

1 $< PID1 >: lock(datei1)$
2 $< PID1 >: lock(datei2)$
3 $< PID2 >: lock(datei3)$
4 $< PID1 >: lock(datei3)$
5 $< PID2 >: lock(datei1)$

b) Liegt eine Verklemmung vor (Begründung)?

Aufgabe 6 2+2

Punkte ☐ *von* **4**

a) Der Scheduler eines Systems arbeitet mit Verdrängung bei einem festen Zeitquantum vom 1 Millisekunde. Wie bewerten Sie diese Quantumsgröße?

b) Ein Lesezugriff auf die Tastatur wird vom Scheduler mit einem Prioritäts-Bonus für den Prozess/Thread „belohnt“. Macht das Sinn (Begründung)?

Aufgabe 7 2+2+1+2+2

Punkte ____ *von* **9**

a) Eine virtuelle Adresse mit 32 Bit ist in eine Seitennummer S von 19 Bit und eine Distanz D von 13 Bit aufgeteilt.

Wird wird die Distanz der zugehörigen realen Adresse berechnet und welche Rolle spielt der TLB („translation lookaside buffer“) dabei?

b) Wie groß sind die Seitenrahmen für das obige Beispiel?

c) Die Umrechnung von virtuellen in reale Speicheradressen in einem System mit Paging kostet Zeit.

Ist ein System mit 2-stufigen Seitentabellen im Schnitt deutlich schneller oder langsamer als ein anderes System mit 3-stufigen Seitentabellen (Begründung)?

d) Ist ein TLB von einem Kontextwechsel betroffen (Begründung)?

e) Was ist ein „weicher“ Seitenfehler („soft page fault“)?

Aufgabe 8 3+3+3

Punkte ____ *von* **9**

a) In der Regel werden Dateien auf einer Platte nicht physikalisch zusammenhängend abgespeichert. Warum nicht? Nennen Sie die Vorteile einer zusammenhängenden Abspeicherung?

b) Beschreiben Sie die einzelnen Schritte der *unlink*-Operation zum Löschen einer Datei bei einem UNIX-Dateiystem. Warum ist die Reihenfolge wichtig?

c) Das Anfügen eines einzigen Bytes an eine vorhandene große Datei in einem FAT-Dateisystem kann eine ganze Reihe von Plattenzugriffen erfordern. Erläutern Sie, warum das so ist!

Aufgabe 9 4+4

Punkte ____ *von* **8**

Zur zentralen Überwachung eines Netzwerks läuft auf jedem Rechner ein Programm *statusserver*, das an seine Clients verschiedene Statusinformationen liefert. Die Client-Server-Kommunikation ist mit Stream-Sockets realisiert.

a) Betrachten Sie nachfolgenden Code-Ausschnitt aus dem nebenläufigen Server. Welche wesentlichen Systemaufrufe fehlen hier oder sind falsch? Ergänzen und verbessern Sie.

```
int main(...){
  int mysocket=socket(PF_INET, SOCK_STREAM, 0);

  struct sockaddr_in myAddress;
  ...  // Initialisierung der Serveradresse
  int res=bind(mysocket, ...);

  while(1){
    struct sockaddr_in clientAddress;

    // auf Client-Aufrtag warten:
    int newSocket=connect(mysocket, &clientAddress, sizeof (struct sockaddr_in));
    switch(fork()){
    case -1: ... // Fehlerbehandlung
    case 0: {
      // Subprozess bearbeitet Auftrag
      int status=processRequest(newSocket);
      exit(status);
      }
    }
  }
}
```

b) Geben Sie eine Serverfunktion *benutzerListeSenden* an, die eine Liste der aktuellen Benutzer über einen als Parameter übergebenen Socket an den Client versendet. Benutzen Sie das Programm „finger“, das eine solche Benutzerliste auf seine Standardausgabe schreibt.

```
int benutzerListeSenden(int clientSocket){









 }
```