Sonstige Aufgaben

X1. Welche Verwaltungsstrukturen benötigt das Betriebssystem für die Verwaltung von
 a) Prozessen
 b) Hauptspeicher
 c) Dateien?
 Geben Sie jeweils einen möglichen Inhalt für jede dieser Strukturen an und begründen Sie Ihre Auswahl! 94/2

X2*.Beim Löschen einer Datei werden in der Regel die Plattenblöcke in die Freispeicherliste aufgenommen, ihr Inhalt bleibt aber erhalten. Diskutieren Sie die Konsequenzen, die das sofortige Zerstören (z.B. Überschreiben mit einem bestimmten Bitmuster) der Blöcke beim Löschen hätte! 98/2

X3. Erläutern Sie die Begriffe
 capability – i-node – lock() – Seitentabelle – Verweilzeit
 jeweils unter den Gesichtspunkten Einordnung in das Gebiet Betriebssysteme – Erklärung – Bedeutung (wozu, Vor-/Nachteile, Umfeld)! 99/2

X4. An eine vorhandene, bereits geöffnete Datei sollen 10 Byte angefügt werden. Beschreiben Sie die Aktionen des Dateisystems, die erforderlich sind, um diesen Schreibzugriff auszuführen! Gehen Sie dabei davon aus, daß sich die Dateien auf einer Festplatte mit einem Unix-Dateisystem befinden. 00/2

X5. Erläutern Sie die Begriffe (im Sinne eines Glossars)
 Prozeßsteuerblock (PCB) – i-node – RPC
 jeweils unter den Gesichtspunkten Einordnung in das Gebiet Betriebssysteme (***ein*** Stichwort) und Erklärung (Bedeutung, Struktur, Anwendungsbereich o.ä.)! 00/8

X6. Stellen Sie NFS und AFS in Bezug auf die Cache-Architektur gegenüber. Gehen sie dabei auf die im Cache gehaltenen Einheiten und die Strategie zur Konsistenzerhaltung ein. 02/8

X7. Einige Systeme stellen einen Systemaufruf *rename* zur Umbenennung einer Datei innerhalb eines Verzeichnisses zur Verfügung. Nennen und erläutern Sie drei wesentliche Unterschiede, die einerseits zwischen diesem Aufruf und andererseits dem Kopieren der Datei in eine Datei mit neuem Namen und anschließendem Löschen der urspünglichen Datei bestehen! 03/3

X8. Erläutern Sie den Begriff „Mounting“ im Sinne eines Glossars: Einordnung in das Gebiet Betriebssysteme – Begriffserklärung – Bedeutung (Nutzen, vergleichbare Vorgehensweisen)! 03/3

X9. In heutigen Betriebssystemen wie Unix ist ein Systemaufruf (system call) nicht als gewöhnlicher Prozeduraufruf realisiert. Was unterscheidet einen solchen Systemaufruf von einem Prozeduraufruf? Warum ist dies notwendig? 04/3

X10. Warum können unter Unix ein Parent-Prozeß und der durch `fork()` erzeugte Child-Prozeß nicht ohne weiteres über gemeinsame globale Variable miteinander kommunizieren? 04/3

X11. Im folgenden Unix-Programm wird mittels `pipe()` eine Pipe erzeugt, in die der Parent-Prozeß den String "`hello world!`" schreibt; der Parent-Prozeß wartet dabei mit `waitpid()` auf das Ende des Child-Prozesses. Der durch `fork()` erzeugte Child-Prozeß liest den String aus dem anderen Ende der Pipe und gibt ihn mit `printf()` auf dem Bildschirm aus. Auf die Angabe der Header-Dateien und eine Fehlerbehandlung wird der Einfachheit halber verzichtet.

```
 1  int main (void) {
 2    char buffer[16];
 3    pid_t pid;
 4    int fd[2];
 5
 6    switch (pid = fork()) {
 7      case  0:          /* Child */
 8        read (fd[0], buffer, sizeof (buffer));
 9        printf("%s", buffer);
10        break;
11      default:          /* Parent */
12        pipe (fd);
13        waitpid (pid, 0, 0);
14        write (fd[1], "hello world!", 13);
15    }
16    exit (0);
17  }
```

a) Erklären Sie, aufgrund welcher zwei Fehler das Programm nicht korrekt funktioniert!
(Das Programm enthält keine syntaktischen Fehler, und auch die Parameterangaben sind korrekt. Die links stehenden Zeilennummern dienen nur zur Bezugnahme für die Lösung und gehören nicht zum Programm.)

b) Was muß an dem Programm geändert werden, damit es das gewünschte Verhalten hat? 04/3

X12. Das nachfolgende C-Programm bestimmt das Maximum einer umfangreichen Folge von nichtnegativen `int`-Werten, die in einer Datei `binary.dat` abgelegt sind, und druckt diesen Wert aus. Auf Fehlerbehandlung wurde aus Gründen der Übersichtlichkeit verzichtet.

```
#include <stdio.h>
#include <fcntl.h>

main()
{ int f;
  int i, max;

  f = open("binary.dat", O_RDONLY));
  max = 0;
  while (read(f, &i, sizeof(int)) != 0)
    if (max < i)
      max = i;

  printf("Maximum: %d\n",max);
  close(f);
  return 0;
}
```

a) Handelt es sich bei der `open`-Funktion um einen Kernaufruf, eine Funktion der Standard Library oder ein Nutzerprogramm? Wofür wird die Integer-Variable `f` verwendet?

b) Angenommen, die `read`-Operation sei so implementiert, daß bei jedem Aufruf die Daten direkt von der Platte übertragen werden. Warum führt dies zu schlechter Laufzeiteffizienz? Erläutern Sie zwei Möglichkeiten, wie diese Effizienz deutlich verbessert werden kann!

c) Was versteht man generell unter dem Öffnen einer Datei? Was geschieht dabei allgemein (unabhängig von Unix)? Nennen Sie wenigstens ein grundlegendes Ziel des Betriebssystem-Entwurfs, dem diese Dateioperation dient! 05/2

X13. In einem verteilten System gebe es einen Dateiserver und mehrere Clients, die nach folgendem Prinzip arbeiten:
- Dateien werden blockweise übertragen;
- Dateiblöcke werden beim Client zum Lesen im Hauptspeicher als Cache zwischengespeichert;
- Schreiboperationen eines Client erfolgen nach dem write-through-Prinzip direkt beim Server.

Ein Client möchte nun eine komplette Datei vom Server holen, von der einige Blöcke in seinem Cache stehen, andere nicht. Erklären Sie das wichtigste Problem, das dabei auftreten kann, und geben Sie eine Lösungsmöglichkeit an! 07/8, 05/8

X14. Geben Sie die Ausgabe an, die bei der Abarbeitung des nachfolgend aufgeführten Programms entsteht! Nehmen Sie dabei an, daß 1 die Prozeßnummer (`PID`) des zugehörigen Prozesses ist und weitere Prozeßnummern bei `fork` anschließend fortlaufend vergeben und nicht wiederverwendet werden.

Hinweis. Die `printf`-Anweisung in dem Programm bewirkt eine zeilenweise Ausgabe der Form `PID 17 -> i = 31`
falls der ausdruckende Prozeß 17 als `PID` hat und 31 der aktuelle Wert von `i` ist. Der Parameter `0` bei `wait(0)` bedeutet, daß der exit-Status ignoriert wird. 07/8, 06/8

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

unsigned i = 3;

int main (void)
{
    while (i > 0)
    {
        i = i-1;
        printf ("PID %u -> i = %u\n", getpid(), i);
        if (fork() > 0)
            wait (0);
    }

    return 0;
}
```

X15. Betrachtet werde in einem Unix-System ein Programm, das folgenden Code ausführt:

```
sleep(20);
fork();
sleep(20);
if (fork() > 0) wait();
sleep(20);
exit(0);
```

Dabei bewirkt der Systemaufruf `sleep(20)`, daß der aufrufende Prozeß für 20 Sekunden schlafengelegt (blockiert) wird. Der Einfachheit halber werde angenommen:

– Prozeßnummern (PIDs) werden fortlaufend vergeben.
– In dem System gibt es keine weiteren Prozesse.

a) Wieviel Pozesse werden insgesamt erzeugt (einschl. dem Prozeß mit PID = 1, der zu obigem Programm gehört)?

b) Tragen Sie in nachstehender Tabelle ein, welche Prozesse sich zu den angegebenen Zeitpunkten nach Start des Programms in dem System befinden, und kennzeichnen Sie ihren jeweiligen Zustand durch ein nachgestelltes S bzw. W für sleeping bzw. waiting!

09/2

Zeit/s	PID
10	
30	
50	
70	
90	
110	