Klausur zu Systemnahe Software
9. Juli 2002 (SS 2002)

Bearbeitungszeit: 120 Minuten

NICHT MIT BLEISTIFT SCHREIBEN!

Name:	
Vorname:	
Matrikelnummer:	

Aufgabe	Punkte	Bewertung
1	**20**	
(a)	4	
(b)	2	
(c)	8	
(d)	6	
2	**18**	
(a)	10	
(b)	8	
3	**10**	
(a)	4	
(b)	4	
(c)	2	
4	**8**	
(a)	5	
(b)	3	

Aufgabe	Punkte	Bewertung
5	**12**	
(a)	3	
(b)	3	
(c)	3	
(d)	3	
6	**8**	
7	**6**	
(a)	2	
(b)	4	
8	**8**	
Summe	**90**	

Bitte benutzen Sie für die Lösungen den freigelassenen Platz nach der jeweiligen Angabe oder die Rückseite unter Angabe der Aufgabe. Nennen Sie möglichst alle Annahmen, die Sie für die Lösung einer Aufgabe treffen!

Viel Erfolg!

Aufgabe 1 **(20 Punkte)**

(a) 4 Punkte

Nennen Sie die 4 wichtigsten Unterschiede zwischen den beiden Protokollen TCP und UDP.

Lösung:

- TCP ist verbindungsorientiert, UDP ist Paketorientiert
- TCP ist zuverlässig
- TCP behält die Reihenfolge der Daten bei.
- UDP ist schneller, wenn Zuverlässigkeit nicht unbedingt notwendig ist.

(b) 2 Punkte

Sie wollen je einen TCP und einen UDP Socket erzeugen (noch kein Verbindungsaufbau etc.). Welchen Systemaufruf verwenden Sie dazu jeweils und welche Parameter müssen übergeben werden?

Lösung:

TCP: `socket (PF_INET, SOCK_STREAM, 0);`

UDP: `socket (PF_INET, SOCK_DGRAM, 0);`

(c) 8 Punkte

Welche fünf Angaben bestimmen eine Internet-Verbindung zwischen zwei Rechnern eindeutig? Geben Sie jeweils an, durch welchen Systemcall diese Angaben in einem Client bzw. in einem Server bei einer TCP-Verbindung festgelegt werden.

Lösung:

Angabe	Systemcall Server	Systemcall Client
Protokoll	socket	socket
Lokale Adresse	bind	connect (oder bind)
Lokaler Port	bind	connect (oder bind)
Entfernte Adresse	accept	connect
Entfernter Port	accept	connect

(d) 6 Punkte

In einem Server finden Sie folgenden Code (sie können davon ausgehen, daß die Systemcalls nicht fehlschlagen):

```
listen (socket, 5);
while (1) {
   connfd = accept (socket, NULL, NULL);
   write (connfd, "HI", 3);
   sleep (200);   /* Ersatz fuer richtige Arbeit */
   close (connfd);
}
```

Der Server wartet gerade auf eingehende Verbindungen in accept. Nun versuchen 10 verschiedene Clients im Abstand von einer Sekunde eine Verbindung zu diesem Server aufzubauen. Geben Sie für jeden Client an ob und wann (auf 20s genau) der Text "HI" bei ihm ankommt. Begründen Sie Ihre Antwort.

Lösung:

Die erste Verbindung wird sofort akzeptiert und beantwortet, allerdings dauert es dann 200s bis zum nächsten accept. Da der "backlog" bei listen auf 5 beschränkt ist, werden nur 5 der weiteren 9 in diesem Zeitraum ankommenden Verbindungen gehalten. Die Verbindungen 7-10 werden abgelehnt. Alle 200 Sekunden wird je eine der gehaltenen Verbindungen akzeptiert. Damit ergibt sich folgende Lösung:

- Client 1: 0s
- Client 2: 200s
- Client 3: 400s
- Client 4: 600s
- Client 5: 800s
- Client 6: 1000s
- Client 7-10: Nie

Aufgabe 2 **(18 Punkte)**

(a) 10 Punkte

Schreiben Sie einen TCP-Server, der auf eingehende Verbindungen auf Port 12345 wartet. Sobald eine Verbindung hergestellt ist, antwortet der Server jedes Mal auf den Text “PING” vom Client mit dem Text “PONG”. Dieses Ping-Pong-Spiel geht so lange hin und her, bis es vom Client durch Schließen der Verbindung beendet wird, oder bis der Client einen anderen Text als “PING” schickt. Im letzteren Fall soll der Server die Verbindung beenden. Sie können bei der Lösung auf die `#include`-Anweisungen verzichten.

Lösung:

```
#include <unistd.h>
#include <sys/socket.h>
#include <string.h>

int main ()
{
        struct sockaddr_in addr;
        int sock, conn;
        char buf[5];

        memset (&addr, 0, sizeof (addr));
        addr.sin_family = AF_INET;
        addr.sin_port = htons (12345);
        addr.sin_addr.s_addr = htonl (INADDR_ANY);

        sock = socket (PF_INET, SOCK_STREAM, 0);
        if (sock < 0)
                return 1;
        if (bind (sock, (struct sockaddr *)&addr,
            sizeof (struct sockaddr_in)) < 0)
                return 1;
        listen (sock, 5);
        while (1) {
                conn = accept (sock, NULL, NULL);
                while (1) {
                        if (read (conn, buf, 4) != 4)
                                break;
                        if (strncmp (buf, "PING", 4) != 0)
                                break;
                        if (write (conn, "PONG", 4) != 4)
                                break;
                }
                close (conn);
```

```
        }
        return 0;
}
```

(b) 8 Punkte

Schreiben Sie einen TCP-Client, der eine TCP-Verbindung zu einem Server wie in Teilaufgabe (a) aufbaut. Gehen Sie davon aus, daß der Server auf dem Rechner mit der Internetadresse 10.11.12.13 läuft. Nach dem Verbindungsaufbau soll Ihr Programm einmal den Text “PING” an den Server schicken und auf den Text “PONG” vom Server warten. Anschließend soll die Verbindung geschlossen werden. Der Exitstatus des Programms soll 0 sein, falls der Text “PONG” erfolgreich empfangen wurde. In allen anderen Fällen soll der Exitstatus 1 sein. Auch hier kann auf die `#include`-Anweisungen verzichtet werden.

Lösung:

```
#include <unistd.h>
#include <sys/socket.h>
#include <string.h>
#include <arpa/inet.h>

int main ()
{
        struct sockaddr_in addr;
        int sock;
        char buf[4];

        sock = socket (PF_INET, SOCK_STREAM, 0);
        if (sock < 0)
                return 1;
        memset (&addr, 0, sizeof (struct sockaddr_in));
        addr.sin_family = AF_INET;
        addr.sin_port = htons (12345);
        addr.sin_addr.s_addr = inet_addr ("134.60.66.5");
        if (connect (sock, (struct sockaddr *)&addr,
              sizeof (struct sockaddr_in)) < 0) {
                return 1;
        }
        if (write (sock, "PING", 4) != 4)
                return 1;
        if (read(sock, buf, 4) != 4)
                return 1;
        if (strncmp (buf, "PONG", 4) != 0)
                return 1;
        close (sock);
        return 0;
}
```

Aufgabe 3 **(10 Punkte)**

(a) 4 Punkte
Die folgenden Zeilen wurden auf der Thales aufgezeichnet:

```
thales$ ypcat passwd | head -1
slc:*noway*:117:134:SLC Administrator:/home/theseus/slc:/bin/true
Broken pipe.
```

Erklären Sie bitte genau, was hier passiert und welches der beteiligten Programme die Meldung `Broken pipe` ausgibt.
Lösung:
ypcat schreibt die (lange) Passwortdatei in eine Pipe. Head greift sich nur die 1.Zeile heraus und terminiert dann sofort. Dadurch schreibt ypcat in eine Pipe, an der kein lesender Prozess mehr hängt. Deshalb erhält ypcat vom Kernel das SIGPIPE-Signal und terminiert unfreiwillig. Die Shell hatte ypcat gestartet, wartet auf die Beendigung, merkt, dass ypcat via SIGPIPE terminierte (wait!) und gibt dann Meldung `Broken pipe` aus.

(b) 4 Punkte
Was versteht man unter dem Begriff "Byte-Ordnung"? Geben Sie auch die Namen der beiden wichtigsten Möglichkeiten für die Byte-Ordnung mit einer kurzen Erklärung an.
Lösung:
Die Byte-Ordnung gibt an, in welcher Reihenfolge die Bytes eines Integers im Speicher abgelegt werden. Die beiden Möglichkeiten sind:

Little Endian Das niederwertigste Byte liegt an niedrigsten Adresse

Big Endian Das hoechstwertige Byte liegt an der niedrigsten Adresse

(c) 2 Punkte
Was muß bei der Netzwerkprogrammierung im Zusammenhang mit der Byte-Ordnung beachtet werden?
Lösung:
Die Byteordnung des Netzwerks kann sich von der auf dem lokalen Rechner unterscheiden. Bei der Angabe von Adressen, Ports usw. ist es Aufgabe des Programmierers, diese in die Netzwerkbyteordnung zu konvertieren.

Aufgabe 4 **(8 Punkte)**

Gegeben sei folgender Auszug aus einem Programm:

```
struct sigaction act;

act.sa_sigaction = myfunc;
act.sa_flags = SA_SIGINFO;

sigemptyset (&act.sa_mask);
sigaddset (&act.sa_mask, SIGTERM);
sigaddset (&act.sa_mask, SIGINT);
sigaddset (&act.sa_mask, SIGKILL);

sigaction (SIGCHLD, &act, NULL);
```

(a) 5 Punkte

Erklären Sie, was der Aufruf von sigaction hier *tatsächlich* alles bewirkt. Gehen Sie dabei genau darauf ein, wie sich der Inhalt von act.sa_flags, act.sa_sigaction und act.sa_mask tatsächlich auswirkt.

Lösung:

Die Funktion myfunc wird als Signalhandler für das Signal SIGCHLD installiert. Der Signalhandler wird beim Aufruf mit erweiterter Signalinformation versorgt (da SA_SIGINFO gesetzt ist). Während der Signalhandler ausgeführt wird sind die Signale SIGTERM, SIGINT und SIGCHLD blockiert. SIGKILL kann nicht blockiert werden und SIGCHLD wird blockiert, weil es das Signal ist, für das ein Handler installiert wird (Ausnahme: flag SA_NODEFER wäre gesetzt).

(b) 3 Punkte

Geben Sie eine passende Deklaration für die Funktion **`myfunc`** an.

Lösung:

Deklaration:

`void myfunc (int sig, siginfo_t * info, void * context);`

Bei einer forward Deklaration können die Namen der Parameter entfallen, statt “siginfo_t *” ist auch “struct siginfo *” möglich.

Aufgabe 5 **(12 Punkte)**

(a) 3 Punkte
Geben Sie 3 Dinge aus dem Kontext eines Prozesses an, in denen sich Vater und Kind nach einem Aufruf von `fork` unterscheiden.
Lösung:
Möglich waren folgende Antworten:

- Process ID
- Parent Process ID
- Das Filedeskriptor-Array im kernel wird kopiert
- Zeit bis zu einem Alarm ist beim Kind 0

(b) 3 Punkte
Geben Sie 3 Dinge aus dem Kontext eines Prozesses an, in denen Vater und Kind nach einem Aufruf von `fork` übereinstimmen.
Lösung:
Möglich waren folgende Antworten:

- Reale und effektive User ID
- Reale und effektive Group ID
- Prozessgruppe
- Session ID
- Root Verzeichnis
- Aktuelles Verzeichnis
- Einstellungen für Signalhandling
- Umask

(c) 3 Punkte
Geben Sie 3 Dinge aus dem Kontext eines Prozesses an, die über einen Aufruf von `exec` hinweg immer erhalten bleiben.
Lösung:
Möglich waren folgende Antworten:

- Process ID
- Parent Process ID
- Terminalgruppe
- Einstellungen für einen Alarm
- Root Verzeichnis und aktuelles Verzeichnis
- Umask

- Reale User- und Group-ID

(d) 3 Punkte

Geben Sie 3 Dinge aus dem Kontext eines Prozesses an, die sich bei einem Aufruf von `exec` ändern können.

Lösung:

Möglich waren folgende Antworten:

- Effektive User-ID, falls Setuid-Bit gesetzt
- Effektive Group-ID, falls Setgid-Bit gesetzt
- Code und Daten des Programms.
- Einstellungen für Signalhandling: Signale mit Handler werden auf default gesetzt.

Aufgabe 6 **(8 Punkte)**

Schreiben Sie ein C-Programm `xls`, das das Kommando "`/bin/ls`" aufruft. Dabei sollen alle Argumente aus der Kommandozeile von `xls` an `ls` weitergegeben werden. Außerdem sollen weitere Argumente von der Standardeingabe (durch "\n" getrennt!) für den Aufruf von "`/bin/ls`" eingelesen werden. Die Argumente aus der Standardeingabe werden hinter die Argumente der Kommandozeile gehängt. Darüberhinaus sollen die Umgebungsvariablen von `ls` die selben sein, wie diejenigen von `xls`.
Sie können davon ausgehen, daß keine der Zeilen in der Standardeingabe aus mehr als 1000 Zeichen besteht. Außerdem können Sie davon ausgehen, daß die Standardeingabe nicht mehr als 1000 Zeilen enthält.
Beispielsitzung:

```
thales$ cat xls_input
/bin/cat
/usr/bin/cat
/usr/local/bin/cat
thales$ cat xls_input | xls -l
-r-xr-xr-x   1 root     bin        10088 Jan  6  2000 /bin/cat
-r-xr-xr-x   1 root     bin        10088 Jan  6  2000 /usr/bin/cat
-rwxr-xr-x   1 root     root       56200 Oct 14  1999 /usr/local/bin/cat
thales$ cat xls_input | xls
/bin/cat            /usr/bin/cat        /usr/local/bin/cat
thales$
```

Lösung:

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


int main (int argc, char * argv[], char * envp[])
{
        int i;
        char ** nargv;
        char buf[1000];

        nargv = malloc ((1001+argc) * sizeof (char *));
        nargv[0] = "ls";
        for (i=1; i<argc; ++i) {
                nargv[i] = argv[i];
        }
        while (i < 1000+argc) {
                if (fgets (buf, 1000, stdin) == NULL)
                        break;
                buf[strlen(buf)-1] = '\0';
```

```
                nargv[i++] = strdup (buf);
        }
        nargv[i] = NULL;
        execve ("/bin/ls", nargv, envp);
        return 1;
}
```

Aufgabe 7 **(6 Punkte)**
Gegeben sei folgendes Programm:

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

int main ()
{
        printf ("%d %d\n", (int)geteuid(), (int)getuid());
        return 0;
}
```

Sie übersetzen dieses Programm und führen es aus. Dabei erhalten Sie folgende Ausgabe:

```
thales$ gcc -Wall uid.c -o uid
thales$ ./uid
642 642
thales$
```

(a) 2 Punkte
Benennen Sie, was dieses Programm ausgibt.
Lösung:
Die effektive und die reale User-ID mit der das Programm ausgeführt wird.

(b) 4 Punkte
Anschließend stellen Sie fest, daß Ihr freundlicher Superuser das Programm bereits für Sie übersetzt und unter dem Namen `/tmp/uid` wie folgt installiert hat:

```
thales$ ls -l /tmp/uid
-rws--x--x   1 root     root         6104 Jun 28 15:46 /tmp/uid
```

Geben Sie an, welche Ausgabe Sie erhalten, wenn Sie `/tmp/uid` ausführen. Begründen Sie Ihre Antwort.
Lösung:
Ausgabe: `0 642`
Begründung: Durch das setuid-Bit (zu erkennen an dem Buchstaben s statt x bei den Rechten fuer den Benutzer) wird die effektive User-ID beim Ausführen des Programms auf die User-ID des Eigentümers der Datei /tmp/uid gesetzt. In unserem Fall ist das root (entspricht dem numerischen Wert 0). Die real User-ID wird vom Vater geerbt.

Aufgabe 8 **(8 Punkte)**

Das folgende Programm implementiert eine Shell-Kommandozeile:
(sämtliche fork()-Aufrufe seien erfolgreich!):

```
int main()
{     int p1[2], p2[2];

      pipe(p1);
      if (!fork()) {
            close(0); dup(p1[0]);
            close(p1[0]); close(p1[1]);
            execlp("grep","grep","n U", NULL);
      }
      pipe(p2);
      if (!fork()) {
            close(0); open("/etc/passwd", O_RDONLY);
            close(1); dup(p2[1]);
            close(p1[0]); close(p1[1]);
            close(p2[0]); close(p2[1]);
            execlp("cat","cat",NULL);
      }
      if (!fork()) {
            close(0); dup(p2[0]);
            close(1); dup(p1[1]);
            close(p2[0]); close(p2[1]);
            close(p1[0]); close(p1[1]);
            execlp("uniq","uniq","-c", NULL);
      }
      close(p2[0]); close(p2[1]);
      close(p1[0]); close(p1[1]);
      wait(NULL); wait(NULL);
      return 0;
}
```

Geben Sie bitte die ursprüngliche Shell-Kommandozeile an, die diesem Programm zugrunde liegt. (Genaue Syntax!)

Lösung:

Die Kommandozeile ist cat </etc/passwd | uniq -c | grep ň U¨.

Häufige Fehler:

- Die Eingabeumlenkung (“¡”) bei cat wurde vergessen. “cat /etc/passwd würde aber bedeuten, daß nicht die Shell sondern erst cat die Datei /etc/passwd öffnet. In diesem Falle wäre das öffnen von /etc/passwd entfallen und der Aufruf von cat hatte “execlp (čat¸ čat¸ ̈/etc/passwd¸ NULL);” lauten müssen.

- Die Anführungszeichen um das Argument von grep sind notwendig, würden diesen in der Kommandozeile fehlen, dann hätte der Aufruf von grep "`execlp (ğrep, ğrep, ň, Ü, NULL);`" lauten müssen.
- Die Reihenfolge der Kommandos auf der Kommandozeile ergibt sich daraus, wie die Pipes aufgebaut werden.