Klausur zu Systemnahe Software
05. Juli 2005 (SS 2005)

Bearbeitungszeit: 120 Minuten
NICHT MIT BLEISTIFT SCHREIBEN!

Name:	
Vorname:	
Matrikelnummer:	
Studiengang:	

Nr	Max	Bewertung	
1	**4**	xxxxx	
2	**12**	xxxxx	
(a)	4		xxxxx
(b)	2		xxxxx
(c)	2		xxxxx
(d)	2		xxxxx
(e)	2		xxxxx
3	**10**	xxxxx	
4	**6**	xxxxx	
5	**12**	xxxxx	
(a)	3		xxxxx
(b)	3		xxxxx
(c)	3		xxxxx
(d)	3		xxxxx

Nr	Max	Bewertung	
6	**10**	xxxxx	
7	**22**	xxxxx	
(a)	12		xxxxx
(b)	10		xxxxx
8	**14**	xxxxx	
(a)	2		xxxxx
(b)	1		xxxxx
(c)	4		xxxxx
(d)	2		xxxxx
(e)	2		xxxxx
(f)	1		xxxxx
(g)	2		xxxxx
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! Bei den vorgegebenen Programmen wurden die `include`-Zeilen der Übersicht halber weggelassen.

Viel Erfolg!

Aufgabe 1 **(4 Punkte)**

Das folgende C-Programm nutzt die beiden Unix-Kommandos

```
NAME
    cd - change working directory
SYNOPSIS
    /usr/bin/cd [directory]

NAME
   pwd - return working directory name
SYNOPSIS
   /usr/bin/pwd

===========================================================

int main() {
     int pid = fork();
     if (pid == 0 ) {
        printf("Working Directory: "); fflush(stdout);
        execlp("pwd", "pwd", NULL);
     };
     wait(NULL);
     pid = fork();
     if ( pid == 0 ) execl("/usr/bin/cd", "cd", NULL);
     wait(NULL);
     pid = fork();
     if (pid == 0 ) {
        printf("Working Directory: "); fflush(stdout);
        execlp("pwd", "pwd", NULL);
     };
     wait(NULL);
     exit( 0 );
}
```

Das Programm wird im Arbeitskatalog

`/home/thales/swg/soft2.05/progs` gestartet, der Heimatkatalog des Benutzers (`cd` ohne Argumente) sei `/home/thales/swg`.

Geben Sie exakt an, was obiges Programm an die Standardausgabe liefert!

Lösungsvorschlag:

```
Working Directory: /home/thales/swg/soft2.05/progs
Working Directory: /home/thales/swg/soft2.05/progs
```

Aufgabe 2 **(12 Punkte)**

(a) 4 Punkte
Welche Signale können von einem Prozess nicht abgefangen werden? Weshalb ist dies sinnvoll?

Lösungsvorschlag:

Die Signale SIGKILL und SIGSTOP können nicht abgefangen werden. Es muss immer eine Möglichkeit geben, einen loopenden Prozess zu beenden. Außerdem sollte es auch immer möglich sein, einen Prozess zwischenzeitlich „schlafen zu legen“, damit die bisher verbrauchte Rechenzeit nicht verloren geht.

(b) 2 Punkte
Schildern Sie, wie ein Zombie-Prozess entstehen kann.

Lösungsvorschlag:

Ein Prozess „forkt“, wobei ein Sohn-Prozess entsteht, welcher sogleich terminiert. Der Erzeuger-Prozess wartet jedoch nicht via wait() oder waitpid() auf den terminierten Sohn sondern beschäftigt sich anderweitig (oder er terminiert sogar selbst). Nach der Termination des Sohn-Prozesses wird dieser also zu einem „Zombie“.

(c) 2 Punkte
Was bleibt von einem Prozess im System übrig, wenn er zum Zombie wird?

Lösungsvorschlag:

„Nur“ der Eintrag in der Prozess-Tabelle bleibt übrig.

(d) 2 Punkte
Was passiert mit einem Zombie, wenn sein Erzeuger terminiert?

Lösungsvorschlag:

Dann erbt der init-Prozess den Zombie. Dieser räumt ihn (z. B. via wait()) ab.

(e) 2 Punkte
Wie kann ein Erzeuger-Prozess verhindern, dass ein von ihm erzeugter Prozess zum Zombie wird? Skizzieren Sie kurz **zwei** Möglichkeiten.

Lösungsvorschlag:

a) Er wartet mit wait() / waitpid()

b) Er installiert für das Signal SIGCHLD einen Handler mit einem wait() darin

c) Er ignoriert das Signal SIGCHLD

Aufgabe 3 **(10 Punkte)**

Schreiben Sie ein Programm, das ein anderes Programm mit dem Namen `doit` aufruft; dieses befindet sich im aktuellen Suchpfad. Ihr Programm liest alles, was `doit` an die Standardausgabe schreibt, und schreibt es selbst an die Diagnoseausgabe. Es kann sein, dass `doit` Diagnoseausgaben produziert; sorgen Sie dafür, dass diese "verschwinden"! Die Bibliotheksfunktion `popen()` darf **nicht** benutzt werden. Mit Ausnahme der SystemCalls `read()`, `write`, `close()` und `dup()` ist bei allen anderen SystemCalls eine entsprechende Fehlerbehandlung (Fehlerausgabe plus Termination) vorzusehen! Die `include`-Anweisungen können weggelassen werden! Der `doit` ausführende Prozess soll natürlich nicht verwaisen, seine Terminationsdaten interessieren Sie aber nicht!

Lösungsvorschlag:

```
int main() {
     char buf[32]; int n; int p[2];
     pipe(p);
     int pid = fork();
     if (pid < 0 ) { perror("fork"); exit(1); }
     if (pid == 0) {
        close(p[0]);
        close(1); dup(p[1]); close(p[1]);
        close(2);
        if (open("/dev/null", O_WRONLY) < 0 )
           perror("open"), exit(2);
        execlp("doit", "doit", NULL);
        perror("execlp"), exit(3);
     } else {
       close(p[1]);

       write(1,buf,strlen(buf));
       while ( (n = read(p[0],buf,sizeof(buf)) ) > 0){
           write(1,buf,n);
       }
       close(p[0]);
       wait(NULL);
     }
     exit( 0 );
}
```

Aufgabe 4 **(6 Punkte)**

Gegeben ist folgendes Programm:

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>

int main() {
int pid, status;
if ( (pid = fork()) < 0 ) perror("fork"), exit(1);
if (pid == 0 ) {
  char buf[32] = "Hello, World!\n"; int p[2];
  pipe(p);
  close(p[0]);
  if (write(p[1],buf,strlen(buf)) <= 0 )
     perror("write"), exit(2);
  exit( 0 );
} else {
  wait(&status);
  if ( status & 0x7F )
     printf("Terminated by signal %d\n", status & 0x7F);
  else
     printf("Terminated with exit value %d\n", (status >> 8) & 0xFF);
}
exit(0);
}
```

Was gibt dieses Programm aus? Erklären Sie bitte genau, was hier passiert!

Lösungsvorschlag:

Der Kindprozess schreibt in eine Pipe, an der keiner zum Lesen hängt. Deshalb erhält dieser vom Kernel das SIGPIPE-Signal und terminiert unfreiwillig. Er Erzeuger-Prozess fragt den Exit-Status an und gibt die Meldung aus “Terminated by signal 13”

Aufgabe 5 **(12 Punkte)**

(a) 3 Punkte

Ein Prozess **P** erzeugt via `fork()` einen neuen Prozess **Ch-P**. Geben Sie 3 Dinge an, in denen sich der Kontext von **P** und **Ch-P** unterscheidet!

Lösungsvorschlag:

pid, ppid, Rückgabewert von fork(), UFDT wird kopiert, Zeit bis zu einem Alarm ist beim Kind 0

(b) 3 Punkte

Ein Prozess **P** erzeugt via `fork()` einen neuen Prozess **Ch-P**. Geben Sie 3 Dinge an, in denen der Kontext von **P** und **Ch-P** übereinstimmt!

Lösungsvorschlag:

real-uid, real-gid, euid, egid, pgid, sid, root directory, current directory, signal handler settings, umask

(c) 3 Punkte

Geben Sie 3 Dinge aus dem Kontext eines Prozesses an, die über einen Aufruf von `exec` hinweg immer erhalten bleiben.

Lösungsvorschlag:

pid, ppid, tty-group, alarm, root-directory, cwd, umask, ruid, rgid

(d) 3 Punkte

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

Lösungsvorschlag:

euid (falls suid-Bit gesetzt), egid (falls sgid-Bit gesetzt), das geladene Image (code, daten etc.), Einstellungen fuer Signalhandling.

Aufgabe 6 **(10 Punkte)**

Schreiben Sie eine Prozedur `socket-info` nach folgender Spezifikation:

- `socket-info` bekommt als einzigen Parameter den Filedeskriptor einer Internet-Socketverbindung
- Ausgabe nach *stdout*:
  - lokale Adresse und lokaler Port
  - entfernte Adresse und entfernter Port (**peer**)
  - Adressen sind in "dotted decimal form" auszugeben
- Rückgabewert: 0 bei Fehler, 1 bei Erfolg

Beispiel für eine Ausgabe:

```
Local: 134.60.66.5:25
Remote:  134.60.166.1:37945
```

Lösungsvorschlag:

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int socket-info (int sock) {
   struct sockaddr_in addr;
   int len = sizeof (addr);
   if (getsockname (sock, (struct sockaddr * )&addr, &len) < 0) {
      perror ("getsockname"); return 0;
   }
   printf ("Local:  %s:%d\n", inet_ntoa (addr.sin_addr),
           ntohs (addr.sin_port));
   len = sizeof (addr);
   if (getpeername (sock, (struct sockaddr * )&addr, &len) < 0) {
      perror ("getpeername");
      return 0;
   }
   printf ("Remote: %s:%d\n", inet_ntoa (addr.sin_addr),
           ntohs (addr.sin_port));
```

```
   return 1;
}

int main ()
{
   struct sockaddr_in addr;
   int sock;

   memset (&addr, 0, sizeof (addr));
   addr.sin_port = htons (25);
   addr.sin_family = AF_INET;
   addr.sin_addr.s_addr = inet_addr ("134.60.66.5");
   sock = socket (PF_INET, SOCK_STREAM, 0);
   socket-info (sock);
   connect (sock, (struct sockaddr * )&addr, sizeof (addr));
   socket-info (sock);
   socket-info (sock);
   socket-info (0);
   return 0;
}
```

Aufgabe 7 **(22 Punkte)**

(a) 12 Punkte

Schreiben Sie ein UDP-basiertes Server-Programm, das auf eingehende Pakete auf Port 11011 wartet. Enthält das Paket den Text „PING“, so wird an den sendenden Client mit dem Text „PONG “ geantwortet.

Lösungsvorschlag:

```
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#define SERVPORT   11011
#define MAXLINE 32

int main() {
   struct sockaddr_in addr, client_addr;
   size_t client_addr_len = sizeof(client_addr);
   char recvline[MAXLINE];
   int sfd;
   memset(&addr, 0, sizeof(addr));
   addr.sin_family = AF_INET;
   addr.sin_port = htons(SERVPORT);
   addr.sin_addr.s_addr = htonl(INADDR_ANY);

   if ((sfd = socket(AF_INET, SOCK_DGRAM, 0)) <0)
       perror("socket"),  exit(1);

   if (bind(sfd, (struct sockaddr *) &addr, sizeof(addr)) <0)
       perror("bind"), exit(2);

   while (1) {
      if ( (recvfrom(sfd, recvline,MAXLINE,0,
              (struct sockaddr *) &client_addr,
              & client_addr_len)) < 0) {
         perror("Server -- recfrom"); exit(3);
      }
      if (strncmp(recvline, "PING", 4) != 0) break;
```

```
        if (sendto(sfd, "PONG", 4, 0,
                    (struct sockaddr *) & client_addr,
                    sizeof(client_addr)) < 4 ) {
            perror("Server -- sendto"); exit(4);
        }
    }
    exit(0);
}
```

(b) 10 Punkte

Schreiben Sie einen UDP-basierten Client, der an den Server (Teilaufgabe (a)) Datenpakete mit dem Text „PING“ schickt. Gehen Sie davon aus, dass der Server auf dem Rechner mit der Internetadresse 127.0.0.1 läuft. Antwortet der Server mit „PONG“, so wiederholt der Client das Spiel – maximal werden 5 „PING“s geschickt, der Exit-Status soll dann 0 sein. Antwortet der Server nicht mit „PONG“, so terminiert der Client sofort mit Exit-Status 1.

Lösungsvorschlag:

```
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <arpa/inet.h>
#define SERVPORT    11011

int main() {
    struct sockaddr_in serv_addr, cli_addr;
    int fd, i, n; char buf[BUFSIZ];
    memset(&serv_addr,0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(SERVPORT);
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

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

    if ((fd = socket(AF_INET, SOCK_DGRAM, 0)) <0)
         perror("socket"), exit(2);

     for(i = 0; i < 5; i++) {
       if ( sendto(fd, "PING", 4, 0,
                (struct sockaddr *) & serv_addr,
                sizeof(serv_addr)) < 0) {
          perror("Client: sendto"); exit(2);
       }
       printf("PING-");
       if (( n = recvfrom(fd, buf, sizeof(buf),0,
                    (struct sockaddr *) 0, (int *) 0)) < 0 ) {
```

```
            perror("Client: recvfrom"); exit(2);
        }
        if (strncmp(buf, "PONG", 4) != 0)
                exit(1);

        buf[n] = '\0'; printf("%.*s\n", n, buf);
      }

    close(fd);
    exit(0);
}
```

Aufgabe 8 **(14 Punkte)**

(a) 2 Punkte
Sie wollen eine Client/Server-Anwendung schreiben. Client und Server müssen nicht unbedingt auf demselben Rechner laufen. Außerdem soll sichergestellt sein, dass alle Daten in genau derselben Weise ankommen, wie sie gesendet werden (sofern das Netz mitspielt ;-)). Welchen Verbindungstyp benötigen Sie also? Begründen Sie Ihre Antwort.

Lösungsvorschlag: TCP/IP-Verbindung – Begründung:

1. Internet-Domain, da Client und Server nicht unbedingt auf dem selben Rechner laufen (also AF_INET und nicht AF_UNIX).
2. Verlässliche Verbindung, weil alles genau so ankommen soll, wie es gesendet wurde (also SOCK_STREAM und nicht SOCK_DGRAM).

(b) 1 Punkte
Welchen Verbindungstyp würde man wohl wählen, wenn es in erster Linie auf die Geschwindigkeit der Übertragung ankommt und es dabei keine Rolle spielt, ob ein Paket verlorengeht? (Client und Server müssen – wie in der vorigen Teilaufgabe – nicht unbedingt auf demselben Rechner laufen.)

Lösungsvorschlag:

UDP/IP-Verbindung

(c) 4 Punkte

Welche Systemaufrufe macht ein Server, der eine TCP/IP-Verbindung auf der Basis von Sockets realisiert? Geben Sie die Systemaufrufe in der richtigen Reihenfolge an und beschreiben Sie kurz, was jeder einzelne Systemaufruf macht.

Lösungsvorschlag:

socket()	Endpunkt für eine Kommunikation via Sockets erzeugen.
bind()	Endpunkt mit einem Port verknüpfen.
listen()	Warteschlange für eine bestimmte Anzahl von Anfragen, die gleichzeitig warten können, anlegen.
accept()	Solange warten, bis sich mindestens eine Anfrage in der Warteschlange befindet und die erste dieser Anfragen annehmen.

(d) 2 Punkte
Welche Systemaufrufe macht ein Client, der eine TCP/IP-Verbindung auf der Basis von Sockets realisiert? Geben Sie die Systemaufrufe in der richtigen Reihenfolge an und beschreiben Sie kurz, was jeder einzelne Systemaufruf macht.

Lösungsvorschlag:

socket()	Endpunkt für eine Kommunikation via Sockets erzeugen.
connect()	Verbindung herstellen zu einem Port auf dem angegebenen Rechner. (Auf dem lokalen Rechner wird ein Zufallsport alloziiert.)

(e) 2 Punkte

Was macht die Funktion `htonl()`? Wieso muss sie verwendet werden?

Lösungsvorschlag:

`htonl()` wandelt eine Long-Integer von Host- in Network-Byte-Order.

Nicht auf allen Rechnern werden die Bytes in derselben Reihenfolge abgelegt (Big-Endian vs. Little-Endian). Auf dem Netz muss aber die Reihenfolge der Bytes eindeutig sein. Somit ist die Funktion `htonl()` bei einigen Rechnern die identische Abbildung und bei anderen Rechnern dreht sie die Reihenfolge der Bytes um. Aus Portabilitätsgründen muss die Funktion `htonl()` daher immer verwendet werden.

(f) 1 Punkte

Mit welcher Funktion kann man in einem C-Programm aus dem Namen eines Rechners dessen IP-Adresse gewinnen?

Lösungsvorschlag:

`gethostbyname()`

(g) 2 Punkte

Die Bearbeitung der Anfragen durch den Server kann evtl. länger dauern. Die Wartezeit für eine Anfrage soll aber nicht zu groß sein. Wie sollte der Server also implementiert sein? Wie nennt man einen solchen Server?

Lösungsvorschlag:

Für jede Anfrage erzeugt der Server einen (Kind-)Prozess, der diese Anfrage bearbeitet. Er selbst kann dann sofort die nächste Anfrage entgegennehmen. Ein solcher Server heißt Concurrent Server.