Klausur zu Systemnahe Software
17. Juli 2006 (SS 2006)

Bearbeitungszeit: 120 Minuten
NICHT MIT BLEISTIFT SCHREIBEN!

Name:	
Vorname:	
Matrikelnummer:	
Studiengang:	

Nr	Max	Bewertung	
1	**12**	xxxxx	
2	**12**	xxxxx	
(a)	2		xxxxx
(b)	2		xxxxx
(c)	4		xxxxx
(d)	4		xxxxx
3	**12**	xxxxx	
(a)	8		xxxxx
(b)	4		xxxxx
4	**14**	xxxxx	
(a)	4		xxxxx
(b)	4		xxxxx
(c)	6		xxxxx

Nr	Max	Bewertung	
5	**10**	xxxxx	
6	**10**	xxxxx	
(a)	4		xxxxx
(b)	4		xxxxx
(c)	2		xxxxx
7	**10**	xxxxx	
8	**10**	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!

Viel Erfolg!

Aufgabe 1 **(12 Punkte)**

Schreiben Sie ein C-Programm `compileall`, das als Kommandozeilenargumente mehrere C-Quelldateien erhält. Jede dieser Dateien soll einzeln mit Hilfe von `gcc` (ein Aufruf von `gcc` pro Datei) übersetzt werden. Dabei soll die Zahl der Zeilen, die insgesamt von allen Aufrufen des `gcc` auf `stderr` geschrieben werden, mit Hilfe des Kommandos `wc` gezählt werden.

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


int main (int argc, char * argv[])
{
        int i, p[2];
        pid_t pid;

        if (pipe (p) < 0) {
                perror ("pipe"); exit (1);
        }
        pid = fork ();
        if (pid < 0) {
                perror ("fork"); exit (1);
        }
        if (pid == 0) {
                close (0);
                dup (p[0]);
                close (p[0]);
                close (p[1]);
                execlp ("wc", "wc", "-l", NULL);
                perror ("execlp"); exit (1);
        }
        close (p[0]);
        for (i=1; i<argc; ++i) {
              pid_t pid2 = fork ();
              if (pid2 < 0) {
                      perror ("fork"); exit (1);
              }
              if (pid2 > 0) {
                      while (wait(0) != pid2);
```

```
                continue;
            }
            /* Child */
            close (2);
            dup (p[1]);
            execlp ("gcc", "gcc", "-c", argv[i], NULL);
            perror ("execlp");
            exit (1);
        }
        close (p[1]);
        while (pid != wait (0));
        return 0;
}
```

Aufgabe 2 **(12 Punkte)**

Geben Sie für jedes der folgenden kleinen Programme und alle weiteren von ihm ggf. erzeugten Prozesse an, ob sie terminieren und falls ja auf welche Art und Weise (Exitstatus bzw. Signal). Achten Sie bei mehreren Prozessen darauf, dass aus Ihrer Lösung klar wird, welcher Prozess von welchem erzeugt wurde. **Alle Antworten sind zu begründen!**

(a) 2 Punkte

```
#include <unistd.h>
int main () {
        int p[2];
        pipe (p);
        close (p[0]);
        write (p[1], "hiho", 4);
        return 0;
}
```

```
Das Leseende der Pipe wird geschlossen und dann
in das Schreibende geschrieben, daher terminiert
der Prozess mit SIGPIPE.
```

(b) 2 Punkte

```
#include <unistd.h>
int main () {
        *(char*)NULL = 0;
        return 0;
}
```

```
Beim Schreibe an die Adresse 0 kommt es zu einem
Segmentation Fault, das Programm terminiert also
durch SIGSEGV.
```

(c) 4 Punkte

```
#include <unistd.h>
int main () {
        int p[2];
        char buf[10];
        pid_t pid;
        pipe (p);
        pid = fork ();
        if (pid) {
                close (p[0]);
                write (p[1], "hiho", 4);
                return 1;
        }
        while (read (p[0], buf, 10) > 0) ;
        return 0;
}
```

```
Der Vater schreibt erfolgreich hiho in die pipe
und terminiert dann mit Exitcode 1.
Der Kindprozess liest die geschriebenen Daten
aus der pipe und blockiert dann, weil er selbst
die pipe noch zum Schreiben offen hat.
```

(d) 4 Punkte

```
#include <unistd.h>
int main ()
{
        int p1[2], p2[2];
        char buf[10];
        pipe (p1); pipe (p2);
        write (p2[1], "hiho", 4);
        dup2 (p2[0], p1[1]);
        while (read (p1[0], buf, 10) > 0);
        return 3;
}
```

```
hiho kann erfolgreich in die Pipe p2 geschrieben
```

werden. Anschliessend fuehrt dup2 dazu, dass das Schreibende der Pipe p1 geschlossen wird. Damit liefert das erste Lesen bereits EOF (Rueckgabewert 0 und das Programm terminiert mit Exitcode 3.

Aufgabe 3 **(12 Punkte)**

(a) 8 Punkte

Kreuzen Sie jeweils an, ob sich die angegebenen Eigenschaften aus dem Kontext eines Prozesses

- bei einem erfolgreichen Aufruf von `fork` zwischen Vater und Kindprozess unterscheiden
- bei einem erfolgreichen Aufruf von `exec` ändern.

	Änderung bei fork			Änderung bei exec		
	immer	machmal	nie	immer	manchmal	nie
Prozess-ID	X					X
Aktuelles Verzeichnis			X			X
Effective User-ID			X		X	
Signalbehandlung			X		X	

Bewertung: Ein Punkt für jedes richtige Kreuz, ein Punkt Abzug für jedes falsche. Insgesamt gibt es in dieser Teilaufgabe keine negativen Punkte.

(b) 4 Punkte

Beschreiben Sie in den Fällen, in denen Sie in Teilaufgabe (a) **machmal** angekreuzt haben, unter welchen Umständen eine Änderung stattfindet und unter welchen Umständen nicht.

```
Effektive Used-ID aendert sich, wenn das setuid-Bit
   gesetzt ist und die ausgefuehrte Datei einem
   anderen User gehoert.
Signalbehandlung aendert sich, wenn eigene Funktionen
   als Signalhandler installiert wurden.
```

Aufgabe 4 **(14 Punkte)**

(a) 4 Punkte
Gegeben sei das folgende Programm:

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

int main (int argc, char ** argv)
{
        assert (argc == 2);
        printf ("%s\n", argv[1]);
        fflush (stdout);
        if (argv[1][0] == 'F')
                return 0;
        argv[1][0]++;
        execvp ("testprog", argv);
        perror ("exec");
        return 1;
}
```

Geben Sie an, was dieses Programm ausgibt, wenn es übersetzt den Namen `testprog` hat und als `testprog A` aufgerufen wird.

```
A
B
C
D
E
F
```

(b) 4 Punkte
Jetzt wird der Aufruf von `fflush` gestrichen und das Programm wird mit `testprog A >OUT` aufgerufen. Welchen Inhalt hat anschließend die Datei `OUT`? **Begründung?**

```
Inhalt von OUT ist ein Zeile:
F
Begruendung: Ohne fflush wird wegen der IO-Umlenkung
blockweise gepuffert. Diese Puffer gehen bei exec
verloren, lediglich der letzte Aufruf, der mit einem
normalen return 0 terminiert fuehrt tatsaechlich zu
einer Ausgabe.
```

(c) 6 Punkte
Gegeben sei das folgende Programm:

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <signal.h>
int main ()
{
        pid_t pid;

        pid = fork ();
        pid = fork ();
        pid = fork ();
        printf ("pid=%ld ppid=%ld\n",
                getpid (), getppid ());
        fflush (stdout);
        sleep (1);
        if (pid == 0)
                kill (getppid(), SIGKILL);
        sleep (1);
        printf ("Still alive (pid=%ld)\n", getpid());
        fflush (stdout);
        return 0;
}
```

Geben Sie eine plausible Ausgabe dieses Programms an. Gehen Sie dabei davon aus, dass die Shell die Prozess-ID 10 hat und dass weitere Prozess-IDs bei 20 beginnen und fortlaufend vergeben werden.

```
pid=20 ppid=10
pid=21 ppid=20
pid=22 ppid=20
pid=23 ppid=21
pid=24 ppid=20
pid=25 ppid=21
pid=26 ppid=22
pid=27 ppid=23
Still alive (pid=24)
Still alive (pid=25)
Still alive (pid=26)
Still alive (pid=27)
```

Aufgabe 5 **(10 Punkte)**

Schreiben Sie ein Programm, das nach folgenden Rechenregeln Operationen durchführt:

- Der Startwert für die Operationen ist 1.
- Das Ergebnis einer Operation dient immer als Eingabe für die nächste.
- Beim Eintreffen eines SIGTERM-Signals wird das Programm beendet und der aktuelle Wert ausgegeben.
- Beim Eintreffen eines SIGUSR1-Signals wird der aktuellen Wert um 100 erhöht.
- Beim Eintreffen eines SIGUSR2-Signals wird der aktuelle Wert
  - durch 2 dividiert, falls er gerade ist
  - mit 3 multipliziert und noch um 1 erhöht, falls er ungerade ist.

Die Rechenoperationen sollen im Signalhandler durchgeführt werden, und es muss sichergestellt sein, dass die Signalhandler für SIGUSR1 und SIGURS2 nicht durch eines dieser Signale unterbrochen werden können.

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

int result = 1;
volatile sig_atomic_t done = 0;
void termhandler (int sig) {
        done = 1;
}

void handler (int sig)
{
        if (sig == SIGUSR1) {
                if (result % 2 == 0) {
                        result /= 2;
                } else {
                        result = 3*result+1;
                }
        } else {
                result += 100;
```

```
        }
}

int main ()
{
        struct sigaction act;
        act.sa_flags = 0;
        act.sa_handler = &termhandler;
        sigemptyset (&act.sa_mask);

        if (sigaction (SIGTERM, &act, NULL) < 0) {
                perror ("sigaction");
                exit (1);
        }
        act.sa_handler = handler;
        sigaddset (&act.sa_mask, SIGUSR1);
        sigaddset (&act.sa_mask, SIGUSR2);
        if (sigaction (SIGUSR1, &act, NULL) < 0 ||
            sigaction (SIGUSR2, &act, NULL) < 0) {
                perror ("sigaction");
                exit (1);
        }
        while (!done)
                sleep (10);
        printf ("result: %d\n", result);
        return 0;
}
```

Aufgabe 6 **(10 Punkte)**

(a) 4 Punkte

Eine 16-Bit Zahl besteht im Speicher aus den zwei Bytes `0x0A` und `0x10` (hexadezimal geschrieben) in dieser Reihenfolge. Welchen Wert (dezimal) hat diese Zahl, wenn man Sie als 16-Bit Zahl in Big-Endian-Darstellung bzw. als 16-Bit-Zahl in Little-Endian-Darstellung interpretiert? **Der Rechenweg muss aus der Lösung hervorgehen.**

```
0x0A entspricht Dezimal 10
0x10 entspricht Deziaml 16
Big Endian:       0x0A10 = 0x0A*256+0x10
                = 10*256+16 = 2576
Little Endian:    0x0A10 = 0x10*256+0x0A
                = 16*256+10 = 4106
```

(b) 4 Punkte

Bei einer (zivilisierten) Diskussion zwischen verschiedenen Menschen in einem Raum muss der Zugriff auf das gemeinsame Übertragunsmedium Luft ganz ähnlich geregelt werden, wie bei Netzwerken. Welches der in der Vorlesung vorgestellten Zugriffsverfahren kommt hierbei in aller Regel zum Einsatz? Beschreiben Sie, wie die einzelnen Komponenten des Verfahrens bei einer Diskussion zwischen Menschen funktionieren.

```
Verfahren: CSMA/CD
Carrier Sense: Man redet nur, wenn gerade niemand
   anderes rededt
Multiple Access: Alle koennen im Prinzip
   gleichzeitig reden
Collission Dectection: Fangen mehrere Personen
   gleichzeitig an zu reden, brechen alle ab und
   einer faengt nach kurzer Pause wieder an.
```

(c) 2 Punkte

Jede Ethernet-Karte hat eine sogenannte MAC- oder Hardware-Adresse. Diese ist für alle Ethernet-Netzwerkkarten weltweit eindeutig. Warum eignet sich diese Adresse dennoch nicht als Ersatz für die Internet-Adresse eines Rechners?

```
Es existiert keinerlei Sortierung nach Netzen
=> explodierende Routingtabelle oder keine
Moeglichkeit eine bestimmte Adresse im weltweiten
Netz zu finden.
Nicht alle Netze basieren auf Ethernet (siehe
Token Ring).
```

Aufgabe 7 **(10 Punkte)**

Schreiben Sie ein Programm, das als einzigen Parameter den Namen eines Rechners übergeben bekommt. Das Programm versucht dann, zu jedem TCP-Port zwischen 1 und 200 auf diesem Rechner eine Verbindung aufzubauen. Diejenigen Ports, bei denen dies erfolgreich ist, werden ausgegeben. Fehlgeschlagene Verbindungen sollen nicht zu einer Ausgabe führen.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <assert.h>
#include <netdb.h>
#include <unistd.h>
#include <errno.h>

int main (int argc, char ** argv)
{
    int port;
    struct sockaddr_in saddr;
    struct hostent * h;
    int s;
    assert (argc == 2);
    h = gethostbyname (argv[1]);
    if (!h) {
        herror ("gethostbyname");
        exit (1);
    }
    memcpy (&saddr.sin_addr, h->h_addr_list[0],
            h->h_length);
    saddr.sin_family = AF_INET;
    for (port = 1; port < 200; ++port) {
        int ret;
        saddr.sin_port = htons (port);
        s = socket (PF_INET, SOCK_STREAM, 0);
        if (s < 0) {
            perror ("socket");
        }
        ret = connect (s, (struct sockaddr*)&saddr,
```

```
                        sizeof (saddr));
        if (ret < 0) {
            if (errno != ECONNREFUSED)
                perror ("connect");
            continue;
        }
        close (s);
        printf ("Port%4d: Open\n", port);
    }
    return 0;
}
```

Aufgabe 8 **(10 Punkte)**

Schreiben Sie einen Server, der auf Port 5381 auf eingehende TCP-Verbindungen wartet und jedem Client, der eine Verbindung aufbaut, eine Seriennummer zuteilt, diese (als Text) an den Client schickt und die Verbindung zum Client dann beendet. Es soll während der gesamten Laufzeit des Servers sichergestellt sein, dass eine Seriennummer höchstens einmal verwendet wird.

```
#include <stdio.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <string.h>

int main ()
{
        struct sockaddr_in saddr;
        int s;
        int serial = 1;

        saddr.sin_family = AF_INET;
        saddr.sin_port = htons (5381);
        saddr.sin_addr.s_addr = INADDR_ANY;

        s = socket (PF_INET, SOCK_STREAM, 0);
        if (s < 0) {
                perror ("socket");
                exit (1);
        }
        if (bind (s, (struct sockaddr*)&saddr,
                  sizeof (saddr)) < 0) {
                perror ("bind");
                exit (1);
        }
        if (listen (s, 10) < 0) {
                perror ("listen");
                exit (1);
        }
        while (1) {
                int s2 = accept (s, 0, 0);
                int off;
```

```
		char buf[20];
		if (s2 < 0) {
			perror ("accept");
			continue;
		}
		serial++;
		sprintf (buf, "%d\r\n", serial);
		off = 0;
		while (off < strlen (buf)) {
			int ret = write (s2, buf+off,
			           strlen (buf+off));
			if (ret < 0) {
				perror ("write");
				break;
			}
			off += ret;
		}
		close (s2);
	}
	return 0;
}
```