

System-Programmierung (syspr) 15. Juni 2021

thomas.amberg@fhnw.ch

Assessment II

Vorname:	Punkte:	/ 90,	Note:
Name:	Frei lassen	ı für Korrek	ctur.
Klasse: 4ibb1			
Hilfsmittel:			
- Ein A4-Blatt handgeschriebene Zusammenfassung.			
- Lösen Sie die Aufgaben jeweils direkt auf den Prüfungs	sblättern.		
- Zusatzblätter, falls nötig, mit Ihrem Namen und Frager	n-Nr. auf jed	lem Blatt.	
Nicht erlaubt:			
- Unterlagen (Slides, Bücher,).			
- Computer (Laptop, Smartphone,).			
- Kommunikation mit anderen Personen.			
Bewertung:			
- Multiple Response: \square Ja oder \square $Nein$ ankreuzen, +1/-	-1 Punkt pro	richtige/fa	lsche Antwort,
beide nicht ankreuzen ergibt +0 Punkte; Total pro Fraș	ge gibt es nie	e weniger al	s 0 Punkte.
- Offene Fragen: Bewertet wird Korrektheit, Vollständig	keit und Küı	rze der Antv	vort.
- Programme: Bewertet wird die Idee/Skizze und Umset	zung des Pr	ogramms.	
Fragen zur Prüfung:			
- Während der Prüfung werden vom Dozent keine Frage	n zur Prüfu	ng beantwo	rtet.

- Ist etwas unklar, machen Sie eine Annahme und notieren Sie diese auf der Prüfung.



Prozess Lebenszyklus

1) Was ist der Output dieses Programms und wieso?	Punkte: / 6
---	-------------

```
01: int main() {
02:     while(fork() == 0) {
03:         printf("%d\n", getpid());
04:     }
05:     return 0;
06: }
```

Output und Begründung hier eintragen; Annahme: #includes sind vorhanden.

2) Welche dieser Aussagen sind korrekt?

Punkte: ___ / 6

Zutreffendes ankreuzen:

□ Ja □ Nein	Rückgabewert von <i>fork()</i> ist immer eine Prozess ID.
□ Ja □ Nein	Ein Child Prozess teilt den Speicher mit dem Parent.
□ Ja □ Nein	Der Speicher des Parents wird beim fork()-en kopiert.
□ Ja □ Nein	Ein Parent lebt immer länger als seine Child Prozesse.
□ Ja □ Nein	Ein Parent Prozess kann zum Zombie Prozess werden.
□ Ja □ Nein	Ein Child Prozess kann den <i>init</i> Prozess "adoptieren".

Threads und Synchronisation

3) Was ist der Output dieses Programms, und wieso?

```
01: pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
02: pthread_cond_t c = PTHREAD_COND_INITIALIZER;
03:
04: void *start(void *arg) {
        pthread_mutex_lock(&m);
05:
        printf("A\n");
06:
        pthread_cond_signal(&c);
07:
        pthread_mutex_unlock(&m);
08:
09: }
10:
11: int main() {
12:
        printf("B\n");
        pthread_t thread;
13:
        pthread_mutex_lock(&m);
14:
        pthread_create(&thread, NULL, start, NULL);
15:
        printf("C\n");
16:
17:
        pthread_cond_wait(&c, &m);
        printf("D\n");
18:
        pthread_mutex_unlock(&m);
19:
20: }
```

Output und Begründung hier eintragen; Annahme: #includes sind vorhanden.

Output	Begründung

Punkte: ____ / 8

IPC mit Pipes

4) Schreiben Sie ein Programm, das ein per Command Line übergebenes Wort über eine Pipe vom Parent zum Child-Prozess sendet, und dort auf *STDOUT_FILENO* ausgibt. P.kte: _ / 12 *Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:*

```
pid_t fork(void); // create a child process, returns 0 in child process
int pipe(int pipe_fd[2]); // create pipe, from pipe_fd[1] to pipe_fd[0]
int close(int fd); // close a file descriptor
ssize_t read(int fd, void *buf, size_t count); // read from a file descr.
ssize_t write(int fd, const void *buf, size_t count); // write to a file
size_t strlen(const char *s); // calculate the length of a string
```

Idee (kurz) und Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:

Sockets

5) Schreiben Sie ein Programm, das UNIX Domain Datagram Nachrichten unter der Adresse /echo empfängt, und den Inhalt der Nachricht an den Absender zurück schickt. P.kte: ___ / 14

Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:

```
int socket(int domain, int type, int protocol); // create an endpoint for
communication; domain = AF_INET, AF_UNIX; type = SOCK_STREAM, SOCK_DGRAM;
protocol = 0; returns a file descriptor for the new socket or -1 on error

A UNIX domain socket address is represented in the following structure:
struct sockaddr_un {
    sa_family_t sun_family; // AF_UNIX
    char sun_path[108]; // Pathname
};

int bind(int socket, struct sockaddr *addr, socklen_t addrlen); // bind

ssize_t recvfrom(int socket, void *buf, size_t len, int flags, // = 0
    struct sockaddr *address, socklen_t *address_len); // receive message

ssize_t sendto(int socket, void *message, size_t len, int flags, // = 0
    struct sockaddr *dest_addr, socklen_t dest_len); // send message

char *strcpy(char *dest, char *src); // copy a string src to buffer dest
```

Idee (kurz) und Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:

6) Vergleichen Sie UNIX Datagram Sockets mit Internet Datagram Sockets. Punkte: ___ / 6

UNIX Domain Datagram Sockets	Internet Datagram Sockets (UDP)

POSIX IPC

7) Schreiben Sie ein Programm, das immer *n* Child-Prozesse hat, welche zufällig, nach *rand()* Sekunden terminieren. Ein Named Semaphor soll die Anzahl konstant halten. Und ein Signal Handler soll bei SIGCHLD jeweils *wait()* aufrufen, um Zombies zu verhindern. Punkte: _ / 16 *Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:*

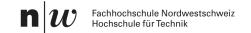
```
int atoi(const char *nptr); // convert a string to an integer
int rand(); // returns a random number [0, RAND_MAX]

void exit(int status); // cause normal process termination
pid_t fork(void); // create a child process; returns 0 in child process;
// the termination signal of the child is always SIGCHLD
pid_t wait(int *wstatus); // wait for process to change state

sem_t *sem_open(char *name, int oflag, mode_t mode, unsigned int value);
// initialize and open a named semaphore; O_CREAT, ...; S_IRUSR, S_IWUSR, ...
int sem_post(sem_t *s); // increment a semaphore
int sem_wait(sem_t *s); // decrement a semaphore, blocking if <= 0

typedef void (*sighandler_t)(int); // handler signature
sighandler_t signal(int signum, sighandler_t handler); // install handler
int sleep(int seconds); // calling thread sleeps for a number of seconds</pre>
```

Fortsetzung auf der nächsten Seite.



Zeitmessung

8) Gegeben die folgenden Zeitmessungen mit *time* für die Programme, a, b und c, was sind die wesentlichen Unterschiede von a, b und c, und wie äussert sich das im Resultat? Punkte: _ / 8

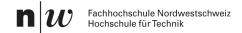
```
$ cat a.c
                        $ cat b.c
                                                  $ cat c.c
int main() {
                        int main() {
                                                  int main() {
    sleep(3);
                             int n =
                                                      time_t t0 = time(NULL);
}
                               320000000;
                                                      time_t t1 = t0;
                             while (n > 0) {
                                                      while (t1 - t0 < 3) {
                                                           t1 = time(NULL);
                                 n--;
                                                      }
                        }
                                                  }
$ time ./a
                        $ time ./b
                                                  $ time ./c
         0m3.013s
                                  0m2.964s
                                                           0m3.008s
real
                        real
                                                  real
         0m0.002s
                                  0m2.909s
                                                           0m0.886s
user
                        user
                                                  user
         0m0.011s
                                  0m0.020s
                                                           0m2.082s
sys
                        sys
                                                  sys
```

Hier ein Auszug aus der Doku:

```
unsigned int sleep(unsigned int seconds); // suspends the execution of
the calling thread; returns remaining seconds if interrupted by signal
time_t time(time_t *tloc); // get time in seconds since the Epoch
```

Unterschiede und Begründung hier eintragen; Annahme: #includes sind vorhanden.

Programm / Resultat (a) Begründung:	Programm / Resultat (b) Begründung:	Programm / Resultat (c) Begründung:



9) Schreiben Sie ein Programm, welches auf Grund der aktuellen Lokalzeit das Datum des ersten Sonntags im nachfolgenden Monat berechnet und wie im Beispiel ausgibt. P.kte: _ / 14

```
$ ./sunday
04.07.2021
```

Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:

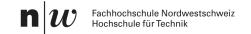
```
int printf(const char *frmt, ...); // frmt int %d or e.g. %02d, string %s

time_t time(time_t *t); // get local time in seconds since Epoch
struct tm *localtime(const time_t *t); // get broken-down local time
time_t mktime(struct tm *tm); // convert broken-down local time to time_t
// ignores tm_wday, tm_yday; values outside valid interval are normalised

struct tm {
  int tm_sec, tm_min, tm_hour; // (0-60), (0-59), (0-23)
  int tm_mday; // Day of the month (1-31)
  int tm_won; // Month (0-11)
  int tm_year; // Year - 1900
  int tm_wday; // Day of the week (0-6, Sunday = 0)
  int tm_yday; // Day in the year (0-365, 1 Jan = 0)
};

size_t strftime(char *s, size_t max, char *format, struct tm *tm);
// format date and time, e.g. "%d.%m.%Y"
```

Idee (kurz) und Source Code hier, und auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:



Zusatzblatt zu Aufgabe Nr	von (Name)	