

System-Programmierung (syspr)

10. Januar 2023

thomas.amberg@fhnw.ch

Assessment II

Vorname:	Punkte: /	90, Note:	
Name:	Frei lassen für Korrektur.		
Klasse: 3ib			
Hilfsmittel:			
- Ein A4-Blatt handgeschriebene Zusammenfassung.			
- Lösen Sie die Aufgaben jeweils direkt auf den Prüfun	gsblättern.		
- Zusatzblätter, falls nötig, mit Ihrem Namen und Frag	ge-Nr. auf jedem B	latt.	
Nicht erlaubt:			
- Unterlagen (Slides, Bücher,).			
- Computer (Laptop, Smartphone,).			
- Kommunikation mit anderen Personen.			
Bewertung:			
- Offene Fragen: Bewertet wird Korrektheit, Vollständ	igkeit und Kürze d	er Antwort.	
- Programme: Bewertet wird die Idee/Skizze und Ums	setzung des Progra	mms.	
Fragen zur Prüfung:			
- Während der Prüfung werden vom Dozent keine Fra	gen zur Prüfung be	eantwortet.	

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

Threads und Synchronisation

1) Schreiben Sie ein Programm *sum*, das die String-Längen seiner Command Line Argumente parallel berechnet, in *je einem Thread*, und dann die *korrekte* Summe ausgibt. Punkte: _ / 18

```
$ ./sum it was all a dream
14
```

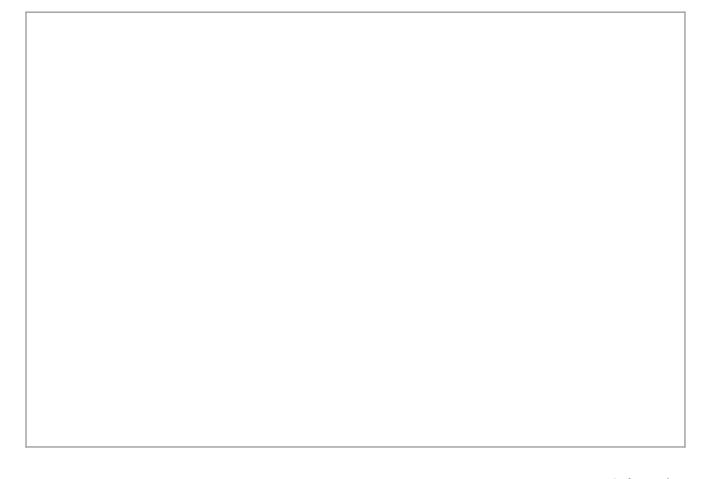
Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:

```
int printf(const char *format, ...); // format string %s, char %c, int %d
size_t strlen(const char *s); // calculate the length of a string

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
    void *(*start) (void *), void *arg); // starts a thread; attr = NULL
int pthread_join(pthread_t thread, void **retval); // retval = NULL

int pthread_mutex_lock(pthread_mutex_t *mutex); // lock a mutex
int pthread_mutex_unlock(pthread_mutex_t *mutex); // unlock a mutex
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // initialize a mutex
```

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



IPC mit Pipes

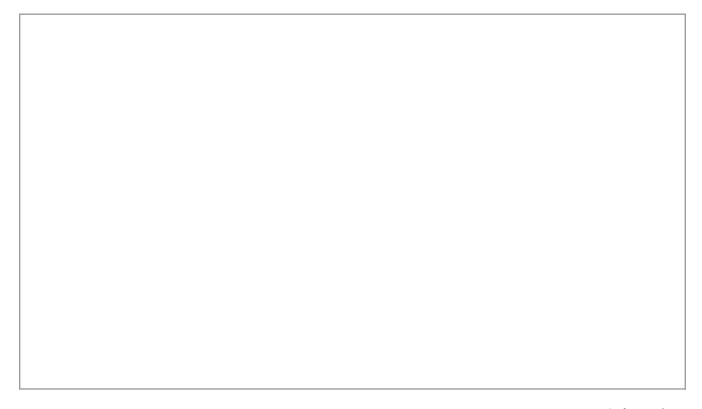
2) Schreiben Sie ein Programm *sd*, welches die maximale *Tiefe des Stacks* bestimmt, indem es eine rekursive Funktion aufruft, in einem *Child Prozess*, welche in jeder Iteration die aktuelle Stack-Tiefe per *Pipe* an den Parent schickt, der dann den letzten Wert ausgibt. Punkte: _ / 18

```
$ ./sd
524119 recursive calls before stack overflow
```

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

```
int atoi(const char *nptr); // convert a string to an integer int printf(const char *format, ...); // format string %s, char %c, int %d int sprintf(char *s, const char *format, ...); // like printf() to string pid_t fork(void); // create a child process, returns 0 in child process int pipe(int pipe_fd[2]); // create a pipe, from pipe_fd[1] to pipe_fd[0] ssize_t read(int fd, void *buf, size_t count); // read from a file descr, returns 0 (EOF) if reading a pipe which has been closed on the other end ssize_t write(int fd, const void *buf, size_t count); // write to a file int close(int fd); // close a file descriptor, returns 0 on success
```

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



Sockets

3) Nennen Sie drei wesentliche Unterschiede von *Internet Stream Sockets* zu *Internet Datagram Sockets*. Formulieren Sie jeweils für beide Seiten einen kurzen Satz. Punkte: _ / 6

Internet Stream Sockets (TCP)	Internet Datagram Sockets (UDP)

(Fortsetzung siehe nächste Seite)

POSIX IPC

4) Gegeben den folgenden Code, implementieren Sie die *eat()* Funktion, welche ein Kind beim Fondue-Essen simuliert. Eine Portion Fondue umfasst 3 Brotstücke, die nacheinander in eine einzige Pfanne getaucht werden. *Semaphoren* sollen garantieren, dass max. 2 von insgesamt 4 Gabeln gleichzeitig in der Pfanne sind. Kinder hat's 7, wenn eins fertig ist, soll "kid *id* is done" ausgegeben werden. Gabeln werden weitergereicht. Brot hat's genug für alle. Punkte: __/ 12

```
#include ... // ignore
#define N_BREAD 3 // per kid
#define N FORKS 4
#define N_SPOTS 2
#define N_KIDS 7
sem_t sem_spots;
sem_t sem_forks;
void *eat(void *arg); // TODO: implement
int main() {
    sem_init(&sem_forks, 0, N_FORKS);
    sem_init(&sem_spots, 0, N_SPOTS);
    for (int i = 0; i < N_KIDS; i++) {
        pthread_t t;
        pthread_create(&t, NULL, eat, (void *) i);
        pthread_detach(t);
    pthread_exit(NULL);
}
```

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

```
int printf(const char *format, ...); // format string %s, char %c, int %d
int sem_init(sem_t *sem, int pshared, unsigned int value); // initialize
an unnamed semaphore, pshared = 0, returns 0 on success
int sem_wait(sem_t *s); // decrement a semaphore, blocking if value <= 0,
returns 0 on success; the below non-blocking variant works the same, but
int sem_trywait(sem_t *s); // returns -1, EAGAIN instead of blocking
int sem_post(sem_t *s); // increment a semaphore, returns 0 on success</pre>
```

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

5) Gegeben den folgenden Code, implementieren Sie *my_sem_init()*, _*wait()* und _*post()* mit dem *Mutex* in *my_sem_t*. Die Semantik soll *POSIX Semaphoren* entsprechen. Punkte: _ / 17

```
#include ... // ignore
typedef struct my_sem {
   pthread_mutex_t m;
   int n; // value
} my_sem_t;
int my_sem_init(my_sem_t *s, int n); // TODO: implement
my_sem_t s;
void *start(void *arg) { // runs in a thread
   my_sem_wait(&s); // blocking
   assert(s.n >= 0);
   my_sem_post(&s);
}
int main(int argc, char *argv[]) {
   int n = atoi(argv[1]); // initial value
   my_sem_init(&s, n);
   assert(s.n == n);
    ... // code to start 2 * n threads, details don't matter
}
```

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

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // initialize a mutex
int pthread_mutex_init(pthread_mutex_t *m, pthread_mutexattr_t *attr); //
alternative to initialize a mutex, attr can be NULL; returns 0 on success
int pthread_mutex_lock(pthread_mutex_t *m); // lock a mutex
int pthread_mutex_unlock(pthread_mutex_t *m); // unlock a mutex

Zeitmessung

6) Schreiben Sie ein Programm *ago*, das "jetzt vor *n* Tagen" als Datum ausgibt. Punkte: _ / 10 Output wie im Beispiel, Datum im default Format, *n* wird per Command Line übergeben:

```
$ date
Tue Jan 3 17:40:28 CET 2023
$ ./ago 10
Sat Dec 24 17:40:32 2022
```

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

```
int atoi(const char *nptr); // convert a string to an integer
int printf(const char *format, ...); // format string %s, char %c, int %d
time_t time(time_t *t); // get local time in seconds since Epoch
char *ctime(const time_t *t); // convert t to ASCII, default date format
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 normalized
struct tm {
  int tm_sec; // Seconds (0-60)
                                     int tm_year; // Year - 1900
                                     int tm_wday; // Day of the week
  int tm_min; // Minutes (0-59)
  int tm_hour; // Hours (0-23)
                                                  // (0-6, Sunday = 0)
  int tm_mday; // Day of the
                                     int tm_yday; // Day in the year
               // month (1-31)
                                                  // (0-365, 1 Jan = 0)
  int tm_mon; // Month (0-11)
                                     int tm_isdst; // Daylight saving time
                                   };
```

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

Terminals

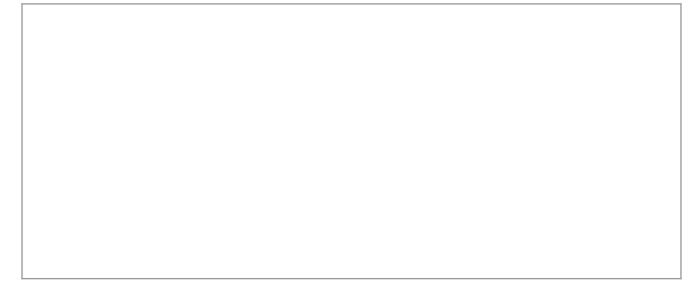
7) Gegeben das folgende Programm, welche *ASCII Zeichenfolge* (inkl. \n) steht im Terminal, wenn nach dem Start die Tasten \n 0, \n 0, \n 0 und \n 2. \n 1 gedrückt wurden? P.kte: \n 1 9

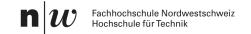
```
01 #include ... // ignore
02
03 int main() {
       struct termios attr;
04
05
       tcgetattr(STDIN_FILENO, &attr);
       attr.c_lflag &= ~ICANON;
06
       attr.c_lflag &= ECHO;
07
       tcsetattr(STDIN_FILENO, TCSANOW, &attr);
80
       char buf[32];
09
       int r = read(STDIN_FILENO, buf, 32);
10
       while (r > 0) {
11
           write(STDOUT_FILENO, buf, r);
12
           write(STDOUT_FILENO, "\n", 1);
13
           r = read(STDIN_FILENO, buf, 32);
14
15
       }
16 }
```

Hier ein Auszug aus der Doku, Fehlerbehandlung und #includes sind hier nicht relevant:

```
int tcgetattr(int fd, struct termios *t); // get parameters of a terminal
int tcsetattr(int fd, int opt, struct termios *t); // set parameters of a
terminal, if opt is TCSANOW, the change shall occur immediately
struct termios: ICANON Enable canonical mode; ECHO Echo input characters
```

Output des Programms (inkl. alle \n) und Schritt für Schritt Begründung hier eintragen:





Zusatzblatt zu Aufgabe Nr	von (Name)	