System-Programmierung (syspr) 07. Juni 2022

thomas.amberg@fhnw.ch

Assessment II

vorname:	Punkte: / 90,	Note:
Name:	Frei lassen für Korrek	tur.
Klasse: 4ibb1		
Hilfsmittel:		
- Ein A4-Blatt handgeschriebene Zusammenfassung.		
- Lösen Sie die Aufgaben jeweils direkt auf den Prüfung	sblättern.	
- Zusatzblätter, falls nötig, mit Ihrem Namen und Frage	en-Nr. auf jedem Blatt.	
Nicht erlaubt:		
- Unterlagen (Slides, Bücher,).		
- Computer (Laptop, Smartphone,).		
- Kommunikation mit anderen Personen.		
Bewertung:		
- Programme: Bewertet wird die Idee/Skizze und Umse	tzung des Programms.	
Fragen zur Prüfung:		
- Während der Prüfung werden vom Dozent keine Frag	en zur Prüfung beantwo	rtet.

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

Threads und Synchronisation

1) Gegeben den folgenden Code, implementieren Sie die Funktion *make()*, welche den *stock* zufällig um je *o-4* Einheiten erhöht, und die Funktion *ship()*, die wartet, bis wieder ein Batch von *10* Einheiten versandt werden kann (*stock -= 10*). Der *stock_mutex* soll dabei den Zugriff auf die *stock* Variable schützen, und *batch_ready* soll das Warten optimieren. Punkte: _ / 18

```
#include ... // ignore

volatile int stock;
pthread_mutex_t stock_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t batch_ready = PTHREAD_COND_INITIALIZER;

void *ship(void *arg); // TODO: implement
void *make(void *arg); // TODO: implement

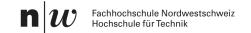
int main() {
    pthread_t thread;
    pthread_create(&thread, NULL, ship, NULL);
    pthread_detach(thread); // ship thread runs forever
    make(NULL); // runs forever, on the main thread
}
```

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_lock(pthread_mutex_t *mutex); // lock a mutex
int pthread_mutex_unlock(pthread_mutex_t *mutex); // unlock a mutex

pthread_cond_t cond = PTHREAD_COND_INITIALIZER; // init. a cond. variable
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex); //
wait on a condition; returns 0 on success, like most pthread_... calls
int pthread_cond_signal(pthread_cond_t *cond); // signal a condition

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



IPC mit Pipes

2) Schreiben Sie ein Programm *procrace*, welches für jeden von n übergebenen Namen einen Child Prozess startet, und dann mit *getch()* auf die ENTER Taste wartet. Daraufhin sollen alle Prozesse *per Pipe* das Zeichen bekommen, den Namen auf die Konsole auszugeben. P.: _ / 18

```
$ ./procrace bat cat dog
Press ENTER to start:
dog
bat
cat
```

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

```
int <code>getchar(void);</code> // blocks until ENTER is pressed, returns a character int <code>printf(const char *format, ...);</code> // format string %s, char %c, int %d pid_t <code>fork(void);</code> // create a child process, returns 0 in child process void <code>exit(int status);</code> // cause process termination; does not return int <code>pipe(int pipe_fd[2]);</code> // create a pipe, <code>from pipe_fd[1] to pipe_fd[0] ssize_t read(int fd, void *buf, size_t count);</code> // read from a file descr, returns 0 (EOF) if reading a pipe which has been closed on the other end ssize_t <code>write(int fd, const void *buf, size_t count);</code> // write to a file int <code>close(int fd);</code> // close a file descriptor, returns 0 on success
```

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

Sockets

3) Gegeben den folgenden Code, implementieren Sie in *main()* einen HTTP Server, der Web

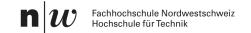
Requests < *MAX_REQ_LEN* beantwortet und auf *STDOUT_FILENO* ausgibt. Punkte: / 18

```
#define MAX_REQ_LEN 4096
struct sockaddr_in addr;
char request_buf[MAX_REQ_LEN];
char response_buf[] =
    "HTTP/1.1 200 OK\r\n"
    "Connection: close\r\n"
    "Content-Length: 0\r\n"
    "\r\n";
void initAddr() {
    char *host = "0.0.0.0"; // any
    int port = 8080;
    struct in_addr ip_addr;
    inet_pton(AF_INET, host, &ip_addr);
    size_t addr_size = sizeof(struct sockaddr_in);
    memset(&addr, 0, addr_size);
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr = ip_addr;
}
int main(void); // TODO: implement
```

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

```
int accept(int sock_fd, struct sockaddr *addr, socklen_t *addrlen); //
  accept a connection on a socket, returns a (client) socket descriptor
int bind(int sock_fd, const struct sockaddr *addr, size_t addr_size); //
  bind a local address to a socket, returns 0 on success
int listen(int sock_fd, int backlog); // listen for connections on a
  socket, backlog e.g. 5, nr of pending connections, returns 0 on success
int socket(int domain, int type, int pr); // domain = AF_UNIX or AF_INET,
  type = SOCK_STREAM or SOCK_DGRAM, pr = 0, returns a socket 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
int close(int fd); // close a file descriptor, returns 0 on success
```



POSIX IPC

4) Schreiben Sie ein Programm cookies, welches p Personen simuliert, die parallel aus einer Packung mit total k Keksen essen. Wenn alle Kekse weg sind, meldet jede Person, wie viele Kekse sie gegessen hat. Nutzen Sie dazu Threads und POSIX Semaphoren.

Punkte: -/ 18

```
$ ./cookies 2 7
1: got 4 cookies
2: got 3 cookies
```

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 pthread_create(pthread_t *thread, const pthread_attr_t *attr,
   void *(*start) (void *), void *arg); // starts a thread; attr = NULL
   int pthread_detach(pthread_t thread); // detach a thread, 0 on success
   pthread_t pthread_self(void); // returns the ID of the calling thread
   void pthread_exit(void *retval); // terminate calling thread, no return

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>
```

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

Zeitmessung

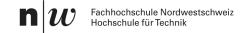
5) Schreiben Sie ein Programm drift, das misst wie viel zu spät das Signal für einen Timer von n Sekunden ankommt. Der Typ des Timers soll $ITIMER_VIRTUAL$ sein, es geht also um CPU Zeit im User Space. Der struct itimerval kann mit $\{\{o, o\}, \{n, o\}\}$ initialisiert werden, wobei n per Command Line übergeben wird, wie in diesem Beispiel.

Punkte: $_/$ 18

```
$ ./drift 60
timer of 60s took 59.950001s, drift = -0.049999s
```

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

```
int printf(const char *frmt, ...); // frmt int %d, float %f, double %lf
typedef void (*sighandler_t)(int); // type of a signal handler function
sighandler_t signal(int sig, sighandler_t handler); // set SIG_IGN,
SIG_DFL, or a programmer-defined function to handle the signal sig.
int pause(void); // causes the calling process to sleep until a signal
terminates the process or causes invocation of a handler function.
int setitimer(int which, struct itimerval *val, struct itimerval *old);
// set the value of an interval timer; returns 0 on success; which =
ITIMER_VIRTUAL decrements in process virtual time. It runs only when the
process is executing. A SIGVTALRM signal is delivered when it expires.
struct itimerval {
                                      struct timeval {
  struct timeval it_interval;
                                        time_t tv_sec; // seconds
  // interval for periodic timer
                                        suseconds_t tv_usec; // microsec.
  struct timeval it_value;
                                      };
  // time until next expiration
};
clock_t times(struct tms *buf); // get process times; returns the number
of clock ticks that have elapsed since an arbitrary point in the past;
number of clock ticks per sec can be obtained using sysconf(_SC_CLK_TCK);
struct tms { clock_t tms_utime; clock_t tms_stime; ... } // user & sys time
```



Zusatzblatt zu Aufgabe Nr	von (Name)	