

System-Programmierung (syspr), 14. Januar 2019, <u>CC BY-SA</u> thomas.amberg@fhnw.ch

Assessment II

vorname:	Punkte: / 76, Note:
Name:	Frei lassen für Korrektur.
Klasse: □ Klasse 3ia □ Klasse 3ib	
Hilfsmittel:	
- Die vom Dozenten ausgeteilte C Referenzkarte.	
- Lösen Sie die Aufgaben direkt auf den Prüfungsblä	ittern.
- Zusatzblätter, falls nötig, mit Ihrem Namen und F	ragen-Nr. auf jedem Blatt.
Nicht erlaubt:	
- Unterlagen (Slides, Bücher,).	
- Computer (Laptop, Smartphone,).	
- Kommunikation mit anderen Personen.	
Bewertung:	
- Multiple Response: \square <i>Ja</i> oder \square <i>Nein</i> ankreuzen,	, +1/-1 Punkt pro richtige/falsche Antwort,
d.h. Antworten "raten" lohnt sich nicht (pro Frage	gibt es aber nie weniger als 0 Punkte).
- Multiple Choice: Eine $oxtimes Antwort$ pro Frage ankre	euzen, 4 Punkte pro richtige Antwort.
- Offene Fragen: Max. 6 Punkte pro Frage für Korre	ktheit, Vollständigkeit und Kürze.
- Programme: Max. 12 Punkte pro Frage für Idee un	nd Umsetzung des Programms.
Fragen zur Prüfung:	
- Während der Prüfung werden vom Dozent keine F	Fragen zur Prüfung beantwortet.

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

Threads & Synchronisation

1) Welche der folgenden Eigenschaften hat ein (Pthread) Mutex? Punkte: ____ / 6

Zutreffendes ankreuzen:

```
Ja | □ Nein Hat zwei Zustände, "zu" oder "offen".
□ Ja | □ Nein Kann "named" oder "unnamed" sein.
□ Ja | □ Nein Ermöglicht gegenseitigen Ausschluss.
□ Ja | □ Nein Erlaubt N > 1 gleichzeitige "Besitzer".
□ Ja | □ Nein Synchronisiert (mehrere) Prozesse.
□ Ja | □ Nein Kann Race Conditions verhindern.
```

2) Gegeben diesen Code, implementieren Sie die Funktion *consumer_start()*. Punkte: ___ / 12

Der Code ist Teil eines Programms, das zufällige Items thread-safe produziert / konsumiert.

Konsumieren wird durch buf remove item() simuliert, Vorbedingung: buf n items() > 0.

```
pthread_mutex_t buf_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t buf_not_full = PTHREAD_COND_INITIALIZER;
pthread_cond_t buf_not_empty = PTHREAD_COND_INITIALIZER;
void *producer_start(void *arg) {
    while (1) {
        pthread_mutex_lock(&buf_mutex);
        while (buf_n_items() == BUF_MAX_ITEMS) {
            pthread_cond_wait(&buf_not_full, &buf_mutex);
        }
        buf_insert_item(random()); // "produce"
        pthread_cond_signal(&buf_not_empty);
        pthread_mutex_unlock(&buf_mutex);
    }
}
void *consumer_start(void *arg);
    // TODO: implement this function, call
    // int buf_remove_item(); to "consume"
    // with precondition: buf_n_items() > 0.
int main () ... // ignore, calls consumer_start, producer_start in Threads
```



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

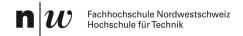
```
int pthread_cond_signal(pthread_cond_t *cond); // signal a condition
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
// wait on a condition, blocking, until another thread calls signal

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

// the following buf_ functions provide a simple, custom, circular buffer
int buf_n_items(); // number of items in buf, 0 <= n <= BUF_MAX_ITEMS
int buf_remove_item(); // precondition: buf_n_items() > 0; returns item
void buf_insert_item(int item); // pre: buf_n_items() < BUF_MAX_ITEMS</pre>
```

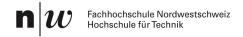
Idee (kurz) und Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:

	//	Annahme:	Dieser	Code	ist	in	derselben	Datei	wie	der	Code	von	Aufgabe	2
L														



IPC mit Pipes

3) Schreiben Sie ein Programm, das eine Pipe von Child zu Parent erstellt. Punkte: $___/$ 12
Der Child Prozess soll die Nachricht "hello" an den Parent Prozess schicken, über diese Pipe.
Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:
<pre>int close(int fd); // close a file descriptor 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] 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</pre>
Idee (kurz) und Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:



Sockets

) Nennen Sie drei Unterschiede zwischen Stream- und Datagram-Sockets.	Punkte:	/6
5) Nennen Sie drei Unterschiede zwischen Internet Sockets und Pipes.	Punkte:	/6

6) Welche Reihenfolge haben die folgenden Calls bei einem Web Server? Punkte: ____ / 4

```
int accept(int sock_fd, struct sockaddr *addr, socklen_t *addrlen);
 int bind(int sock_fd, const struct sockaddr *addr, socklen_t addrlen);
 int listen(int sock_fd, int backlog);
 ssize_t read(int fd, void *buf, size_t count);
 int socket(int domain, int type, int protocol);
 ssize_t write(int fd, const void *buf, size_t count);
Eine Antwort ankreuzen:
 □ socket(); accept(); bind(); listen(); read(); write();
 □ socket(); accept(); listen(); bind(); read(); write();
 □ socket(); bind(); accept(); listen(); read(); write();
 □ socket(); bind(); listen(); accept(); read(); write();
 □ socket(); listen(); accept(); bind(); read(); write();
 □ socket(); listen(); bind(); accept(); read(); write();
Terminals
7) Was sind drei Unterschiede von canonical und raw Mode bei Terminals? Punkte: ____ / 6
```

POSIX IPC

8) Gegeben diesen Code, implementieren Sie die vier *parking_* Funktionen. Punkte: ____ / 12 Das Programm ist eine einfache Simulation, mit Autos, die ein Parkhaus besetzen / verlassen. Ein Semaphor soll dabei garantieren, dass nie mehr als *N_SPACES* Autos im Parkhaus sind.

```
#include ... // ignore
#define N_CARS 7
#define N_SPACES 3
void parking_garage_open(); // TODO: implement
void parking_garage_enter(); // TODO: implement
void parking_garage_leave(); // TODO: implement
void parking_garage_close(); // TODO: implement
void *car_start(void *arg) {
    parking_garage_enter();
    int r = random() % 3;
    sleep(r); // 0-2s
    parking_garage_leave();
    pthread_exit(0);
}
void main() {
    pthread_t car_threads[N_CARS];
    parking_garage_open();
    for (int i = 0; i < N_CARS; i++) {
        pthread_create(&car_threads[i], NULL, car_start, NULL);
    for (int j = 0; j < N_CARS; j++) {
        pthread_join(car_threads[j], NULL);
    parking_garage_close();
}
```

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

```
int sem_destroy(sem_t *sem); // destroy an unnamed semaphore
int sem_getvalue(sem_t *sem, int *value); // get the value of a semaphore
int sem_init(sem_t *sem, int pshared, unsigned int value); // pshared = 0
int sem_post(sem_t *sem); // increment/unlock a semaphore
int sem_wait(sem_t *sem); // decrement/lock a semaphore, blocking if <= 0</pre>
```



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

//	Annahme:	Dieser	Code	ist	in	derselben	Datei	wie	der	Code	von	Aufgabe	8

Zeitmessung

9) Schreiben Sie ein Programm, das seine Laufzeit (Echt- & User-Zeit) misst. Punkte: ___ / 12 Die Messung soll so lange laufen, bis CTRL-C gedrückt wurde bzw. SIGINT angekommen ist.

```
$ ./my_time
press CTRL-C ...
<CTRL-C gedrückt, Kernel sendet SIGINT Signal>
real 3.920000s
user 0.000000s
```

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

```
int printf(const char *format, ...); // format int %d, double %lf

typedef void (*sighandler_t)(int); // handler signature
sighandler_t signal(int signum, sighandler_t handler); // install handler
int pause(void); // wait for signal

clock_t times(struct tms *buf); // returns real time; buf = CPU time
long sysconf(int name); // name = _SC_CLK_TCK => clock_t ticks per second
struct tms { // CPU time
    clock_t tms_utime; // user time
    clock_t tms_stime; // system time
};
```

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



Zusatzblatt zu Frage Nr	von (Name)