

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 Prüfungsblättern.
- Zusatzblätter, falls nötig, mit Ihrem Namen und Fragen-Nr. auf jedem Blatt.

Nicht erlaubt:

- Unterlagen (Slides, Bücher, ...).
- Computer (Laptop, Smartphone, ...).
- Kommunikation mit anderen Personen.

Bewertung:

- Multiple Response: ☐ *Ja* oder ☐ *Nein* ankreuzen, +1/-1 Punkt pro richtige/falsche Antwort, beide nicht ankreuzen ergibt +0 Punkte; Total pro Frage gibt es nie weniger als 0 Punkte.
- Offene Fragen: Bewertet wird Korrektheit, Vollständigkeit und Kürze der Antwort.
- Programme: Bewertet wird die Idee/Skizze und Umsetzung des Programms.

Fragen zur Prüfung:

- Während der Prüfung werden vom Dozent keine Fragen zur Prüfung beantwortet.
- Ist etwas unklar, machen Sie eine Annahme und notieren Sie diese auf der Prüfung.

Threads und Synchronisation

1) Welche der folgenden Aussagen über Threads sind korrekt?

Punkte: _ / 4

Zutreffendes ankreuzen

- | | |
|---|--|
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Der <i>main</i> Thread muss auf alle Threads warten, mit <i>pthread_join()</i> . |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Ein beliebiger Thread kann mit <i>exit()</i> sich und alle anderen beenden. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Zwei Threads in einem Prozess führen verschiedene Programme aus. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Jeder neue Thread hat eine eigene Kopie aller globalen Variablen. |

2) Nennen Sie drei wesentliche Eigenschaften von Critical Sections.

Punkte: _ / 6

Schreiben Sie jeweils einen ganzen Satz mit der Eigenschaft und einem kurzen Beispiel.

3) Schreiben Sie ein Programm *len*, das die String-Längen seiner Command Line Argumente parallel berechnet, in *je einem* Thread, und dann die *korrekte* Summe ausgibt. Punkte: / 16

```
$ ./len 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 Fragen-Nr.:

IPC mit Pipes

4) Schreiben Sie ein Programm *pipesync*, das eine Pipe zur Synchronisation von $n > 0$ Child Prozessen verwendet, indem im Parent (blockierend) gelesen wird, bis *EOL* kommt. Dies ist genau dann der Fall, wenn der letzte Prozess das Schreib-Ende geschlossen hat. P.kte: / 16

Die Zahl n wird dabei per Command Line übergeben, der Output soll so aussehen, mit PIDs:

```
$ ./pipesync 2
child 8005: closing write end.
child 8004: closing write end.
parent got EOL.
```

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

pid_t fork(void); // create a child process, returns 0 in child process
pid_t getpid(void); // get process identification
void exit(int status); // cause process termination; does not return

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
int close(int fd); // close a file descriptor
```

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

Sockets

5) Wieso brauchen Stream Sockets ein *connect()* und Datagram Sockets nicht? Punkte: _ / 4

Hier ein Auszug aus der Doku:

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);  
// initiate a connection on a socket, returns 0 on success, -1 on error
```

Komplettieren Sie jeweils den Satz mit einer kurzen Erklärung:

Stream Sockets brauchen `connect()` weil

Datagram Sockets brauchen kein `connect()` weil

6) Welche der folgenden Aussagen über Unix Domain Sockets sind korrekt? Punkte: _ / 4

Hier ein Auszug aus der Doku:

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);  
// bind a name to a socket, returns 0 on success, -1 on error
```

Zutreffendes ankreuzen:

- | | |
|---|---|
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Der <i>bind()</i> Aufruf nimmt sowohl Unix Domain als auch IP Adressen. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Unix Domain Sockets erlauben Kommunikation zwischen Unix Hosts. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Unix Domain Datagram Sockets übertragen Datenpakete zuverlässig. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | File Permissions bestimmen, wer Zugriff auf Unix Domain Sockets hat. |

POSIX IPC

7) Schreiben Sie ein Programm, das ein Parkhaus mit n Plätzen und $2 * n$ Autos simuliert, die dauernd rein und nach einer Sekunde wieder raus fahren. Die Simulation soll Threads nutzen und garantieren, dass zu einem Zeitpunkt maximal n Autos im Parkhaus sind. 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 sleep(int seconds); // calling thread sleeps for a number of seconds

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
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
int sem_wait(sem_t *s); // decrement a semaphore, blocking if <= 0
int sem_post(sem_t *s); // increment a semaphore
```

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

Zeitmessung

8) Schreiben Sie ein Programm *filetime*, das ausgibt, wie lange es in Echtzeit dauert, 1 MB in ein neu kreierte File *f1* bzw. *f2* zu schreiben, zuerst mit, dann ohne *O_SYNC*. Punkte: _ / 16

```
$ ./filetime
f1: real = 0.180000s
f2: real = 0.020000s
```

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

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

int open(const char *pathname, int flags, mode_t mode); // opens the file
specified by pathname; or creates it if O_CREAT is used; returns the file
descriptor; flags include O_APPEND, O_CREAT, O_SYNC, O_RDONLY, O_WRONLY;
modes, which are used together with O_CREAT include S_IRUSR and S_IWUSR.
ssize_t write(int fd, const void *buf, size_t n); // writes up to n bytes
from buf to the file referred to by fd; returns nr. of bytes written ≤ n.

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
```

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

9) Schreiben Sie ein Programm, das mittels *alarm()* nach 3 Sek. "alarm!" ausgibt. P.kte: _ / 8

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

```
int printf(const char *frmt, ...); // frmt int %d, float %f, double %lf  
  
int alarm(int sec); // arranges for a SIGALRM signal to be delivered to  
calling process in sec seconds; returns the number of seconds remaining  
  
int pause(void); // causes the calling process to sleep until a signal  
terminates the process or causes invocation of a handler function.  
sighandler_t signal(int sig, sighandler_t handler); // set a programmer-  
defined function to handle a signal; typedef void (*sighandler_t)(int);
```

Idee (kurz) und Source Code hier. Hinweis: Diese Aufgabe ist kürzer als die vorherigen.

Zusatzblatt zu Aufgabe Nr. ____ von (Name) _____