

# Assessment II

Vorname: \_\_\_\_\_

Punkte: \_\_\_\_ / 90, Note: \_\_\_\_

Name: \_\_\_\_\_

*Frei lassen für Korrektur.*

Klasse: 4ibb1

Hilfsmittel:

- Ein A4-Blatt handgeschriebene Zusammenfassung.
- Lösen Sie die Aufgaben jeweils direkt auf den Prüfungsblättern.
- Zusatzblätter, falls nötig, mit Ihrem Namen und Frage-Nr. auf jedem Blatt.

Nicht erlaubt:

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

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*

- ☐ Ja | ☐ Nein      Zwei Threads in einem Prozess führen verschiedene Programme aus.
- ☐ Ja | ☐ Nein      Der Main-Thread kann andere Threads mit `pthread_join()` beenden.
- ☐ Ja | ☐ Nein      Ein beliebiger Thread kann sich und alle anderen mit `exit()` beenden.
- ☐ Ja | ☐ Nein      Jeder neue Thread sieht eine eigene Kopie aller globalen Variablen.

2) Welche der folgenden Aussagen über Critical Sections sind korrekt?

Punkte: \_ / 4

*Zutreffendes ankreuzen*

- ☐ Ja | ☐ Nein      Eine Critical Section muss immer atomar ausgeführt werden.
- ☐ Ja | ☐ Nein      Eine Critical Section garantiert den gegenseitigen Ausschluss.
- ☐ Ja | ☐ Nein      Ein einzelnes C Statement kann eine Critical Section sein.
- ☐ Ja | ☐ Nein      Kritisch ist die Ausführungsreihenfolge der Instruktionen.

3) Was sind drei wesentliche Unterschiede zwischen Mutex und Semaphoren?

Punkte: \_ / 6

*Unterschiede hier eintragen, jeweils beide Seiten in einem kurzen Satz beschreiben:*

Mutex	Semaphore

## IPC mit Pipes

4) Schreiben Sie ein Programm, das zwei Child-Prozesse via Pipe verbindet, ein per Command

Line übergebenes Wort von Child 1 zu Child 2 sendet und dort auf *stdout* ausgibt. P.kte: \_ / 18

Verwenden Sie die folgenden Calls (soweit sinnvoll), ohne *#includes* und Fehlerbehandlung:

```
void exit(int status); // cause process termination; does not return  
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]
```

```
int close(int fd); // close a file descriptor  
ssize_t read(int fd, void *buf, size_t n); // attempts to read up to n  
bytes from file descriptor fd into buf; returns number of bytes read ≤ n  
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
```

```
size_t strlen(const char *s); // calculate the length of a string
```

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

## Sockets

5) Schreiben Sie ein Programm, welches UNIX Domain Byte-Streams unter der Adresse */log* empfängt, und die gelesenen, max. 1024 Bytes auf `STDOUT_FILENO` schreibt. Punkte:    / 16

Verwenden Sie die folgenden Calls (soweit sinnvoll), ohne `#includes` und Fehlerbehandlung:

```
int accept(int socket, struct sockaddr *addr, socklen_t *addrlen); //  
accept a connection on a socket; returns the accepted client socket  
int bind(int socket, struct sockaddr *addr, socklen_t addrlen); // bind  
int listen(int socket, int backlog); // listen for connections on socket  
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
```

```
struct sockaddr_un { // UNIX domain socket address structure  
    sa_family_t sun_family; // AF_UNIX  
    char sun_path[108]; // Pathname  
};
```

```
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
```

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

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

6) Welche Aussagen zu Internet Datagram Sockets treffen im Allgemeinen zu? Punkte: \_ / 4

*Zutreffendes ankreuzen:*

- ☐ Ja | ☐ Nein      Datagram Sockets sind verbindungslos, es braucht daher kein *connect()*.
- ☐ Ja | ☐ Nein      Zu jeder Message gibt es zwei IP-Adressen, Empfänger und Absender.
- ☐ Ja | ☐ Nein      Der Zugriff auf Internet Datagram Sockets ist via Filesystem geregelt.
- ☐ Ja | ☐ Nein      Die gängigere Bezeichnung für Internet Datagram Sockets ist TCP.

## POSIX IPC

7) Was sind drei wesentliche Unterschiede von POSIX Message Queue zu Pipes? Punkte: \_ / 6

*Unterschiede hier eintragen, jeweils beide Seiten in einem kurzen Satz beschreiben:*

POSIX Message Queue	Pipe

|  
|  
|  
|  
|  
|

*Fortsetzung auf der nächsten Seite ...*

8) Schreiben Sie *ein* Programm, das 9 Personen beim Ausverkauf simuliert, die gleichzeitig versuchen, eins von 5 Shirts zu erhalten. Nutzen Sie *fork()* und eine Named Semaphore */sem*. Personen, die eins erhalten, geben ihre *PID* und "yes" aus, sonst *PID* und "no". Punkte:    / 14  
Verwenden Sie die folgenden Calls, ohne *#includes*, mit Fehlerbehandlung, soweit wie nötig:

```
int printf(const char *format, ...); // format string %s, char %c, int %d

void exit(int status); // cause process termination; does not return
pid_t getpid(void); // returns the process ID (PID) of the calling proc.
pid_t fork(void); // create a child process, returns 0 in child process

sem_t *sem_open(char *name, int oflag, mode_t mode, unsigned int val); //
create, initialize and open a named semaphore; returns a pointer to the
semaphore; use oflag = O_CREAT; mode = S_IRUSR|S_IWUSR and set its value
int sem_post(sem_t *s); // increment a semaphore
int sem_wait(sem_t *s); // decrement a semaphore, blocks if value <= 0
int sem_trywait(sem_t *sem); // returns 0 on success, -1 if value <= 0
```

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

## Zeitmessung

9) Schreiben Sie ein Programm *yesterday*, welches das korrekte Kalenderdatum von gestern, 00:00:00, in Lokalzeit, im default Format ausgibt, wie hier im Beispiel gezeigt. Punkte: \_ / 10

```
$ date
Sat May 31 19:22:40 CEST 2025
$ ./yesterday
Fri May 30 00:00:00 2025
```

Verwenden Sie die folgenden Calls (soweit sinnvoll), ohne #includes und Fehlerbehandlung:

```
int printf(const char *format, ...); // format string %s, char %c, int %d

char *ctime(const time_t *t); // t to string, local time, default 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 normalised
time_t time(time_t *t); // returns time in seconds since 01.01.1970, UTC

struct tm {
    int tm_sec; // Seconds (0-60)
    int tm_min; // Minutes (0-59)
    int tm_hour; // Hours (0-23)
    int tm_mday; // Day of the
                // month (1-31)
    int tm_mon; // 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)
    int tm_isdst; // Daylight saving time
};
```

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

10) Welche der folgenden Aussagen zu Zeitmessung treffen im Allgemeinen zu? Punkte: \_ / 4

*Zutreffendes ankreuzen:*

- ☐ Ja | ☐ Nein Die Linux Epoche ist die Zeit seit dem Aufstarten.
- ☐ Ja | ☐ Nein User CPU Zeit ist immer grösser als System CPU Zeit.
- ☐ Ja | ☐ Nein CPU Zeit wird relativ gemessen, an mehr als einem Punkt.
- ☐ Ja | ☐ Nein Reale Zeit ist die Summe von User und System CPU Zeit.

## Terminals

11) Welche der folgenden Aussagen zu Terminals treffen im Allgemeinen zu? Punkte: \_ / 4

*Zutreffendes ankreuzen:*

- ☐ Ja | ☐ Nein Echo führt dazu, dass der User-Prozess jedes Zeichen zweimal sieht.
- ☐ Ja | ☐ Nein Im Canonical Mode können User Tippfehler in der Shell korrigieren.
- ☐ Ja | ☐ Nein Text-Editoren wie *nano* lesen Terminal Input jeweils Zeile für Zeile.
- ☐ Ja | ☐ Nein Eine Änderungen der Terminal Fenstergrösse löst ein Signal aus.

|  
|  
|  
|  
|  
|  
|  
|

*Zusatzblatt auf der nächsten Seite ...*



*Zusatzblatt zu Aufgabe Nr. \_\_\_\_ von (Name) \_\_\_\_\_*