

# Assessment II

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

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

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

*Frei lassen für Korrektur.*

Klasse: 3ia

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.

## IPC mit Pipes

1) Schreiben Sie ein Programm, welches einen Child Prozess erzeugt, und von dort aus, über eine Pipe, die Nachrichten "0%\n", "50%\n" und "100%\n" im Abstand von je einer Sekunde zum Parent Prozess sendet, wo sie auf `STDOUT_FILENO` ausgegeben werden. Punkte: \_ / 18

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

```
void exit(int status); // cause process termination; does not return
pid_t fork(void); // create a child process, returns 0 in child process
unsigned int sleep(unsigned int sec); // sleep for a number of seconds

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
```

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

## Sockets

2) Gegeben den folgenden Code, implementieren Sie in *main()* einen HTTP Server, der Web Requests < *MAX\_REQ\_LEN* auf *STDOUT\_FILENO* ausgibt und beantwortet. Punkte: \_ / 18

```
#define MAX_REQ_LEN 4096

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";

struct sockaddr_in sv_addr;

void init_sv_addr(void) {
    char *host = "0.0.0.0"; // any
    struct in_addr ip_addr;
    inet_pton(AF_INET, host, &ip_addr);
    size_t addr_size = sizeof(struct sockaddr_in);
    sv_addr.sin_family = AF_INET;
    sv_addr.sin_port = htons(8080);
    sv_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

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

(Fortsetzung siehe nächste Seite)

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

3) 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

4) Schreiben Sie ein Programm *mq\_move*, das eine existierende, via Command Line gegebene POSIX Message Queue *src* ausliest und jede Message in eine neu kreierte Message Queue *dst* verschiebt. Reihenfolge und Prioritäten der Messages sollen erhalten bleiben. Punkte:    / 14

```
$ ls -la /dev/mqueue/
-rw----- ... q1          # /q1 contains multiple messages, varying prio.
$ ./mq_move /q1 /q2
$ ls -la /dev/mqueue/
-rw----- ... q2          # /q2 contains copies of original /q1 messages
```

*Nutzen Sie die folgenden Aufrufe, #includes und Fehlerbehandlung können Sie weglassen:*

```
mqd_t mq_open(char *name, int flags);
mqd_t mq_open(char *name, int flags, mode_t mode, struct mq_attr *attr);
// open a message queue or create it; returns a message queue descriptor;
// flags include O_RDONLY, O_WRONLY, O_CREAT; mode incl. S_IRUSR, S_IWUSR;

int mq_getattr(mqd_t mqd, struct mq_attr *attr); // read attributes
struct mq_attr { ... long mq_maxmsg; long mq_msgsize; long mq_curmsgs; };
// max # of messages; max message size; current number of messages in mq

int mq_send(mqd_t mqd, char *msg, size_t len, unsigned int prio);
// send a message to a message queue; returns 0 on success
ssize_t mq_receive(mqd_t mqd, char *msg, size_t len, unsigned int *prio);
// receive a message from a message queue; returns # of bytes in message

int mq_close(mqd_t mqd); // close a message queue descriptor
int mq_unlink(const char *name); // remove a message queue
```

*Idee (kurz) und Source Code hier und auf Folgeseite, oder Blatt mit Namen und Frage-Nr.:*

*(4) Fortsetzung:*

5) Schreiben Sie ein Programm, das  $n$  Enten simuliert, die gleichzeitig versuchen, eins von  $m$  Stücken Brot zu ergattern. Nutzen Sie Threads und Semaphore so, dass exakt  $m$  Enten je ein Stück Brot bekommen ( $m < n$ ). Diese geben "gulp" aus, alle anderen "quack". Punkte: \_ / 16

*Hier die Doku, #includes und nicht-essentielle 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
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_trywait(sem_t *sem); // returns 0 on success, -1 if sem is locked
int sem_post(sem_t *s); // increment a semaphore
```

*Idee (kurz) und Source Code auf Folgeseite, oder auf Zusatzblatt mit Namen und Frage-Nr.*

(5) Fortsetzung:

## Zeitmessung

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

*Zutreffendes ankreuzen:*

- |   |  |
|---|--|
| <input type="checkbox"/> Ja   <input type="checkbox"/> Nein | Die Linux Epoche endet mit dem Überlauf von 32-bit Zeit im Jahr 2038.      |
| <input type="checkbox"/> Ja   <input type="checkbox"/> Nein | Mittels Locale wird bestimmt, welche Zeitzone für localtime() gesetzt ist. |
| <input type="checkbox"/> Ja   <input type="checkbox"/> Nein | Die reale Zeit ist $\geq$ die Summe von User CPU Zeit und System CPU Zeit. |
| <input type="checkbox"/> Ja   <input type="checkbox"/> Nein | Child CPU Zeit ist immer kleiner als die CPU Zeit des Parent Prozesses.    |

7) Schreiben Sie ein Programm *count*, das eine mit 0 initialisierte *long* Variable eine Sekunde lang — in User CPU Zeit — inkrementiert, und dann den aktuellen Wert ausgibt. P.kte: \_ / 12

```
$ time ./count
1870499
# output of time:
real    0m2.555s
user    0m1.000s
sys     0m1.555s
```

Nutzen Sie die folgenden Aufrufe, #includes und Fehlerbehandlung können Sie weglassen:

```
clock_t times(struct tms *t); // returns the number of clock ticks that
have elapsed since an arbitrary point in the past; stores the current
process times in the struct tms that t points to.
```

```
struct tms {
    clock_t tms_utime; // user time
    clock_t tms_stime; // system time
    clock_t tms_cutime; // user time of children
    clock_t tms_cstime; // system time of children
};
```

The number of clock ticks per second is:  
**sysconf(\_SC\_CLK\_TCK)**

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

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



## Terminals

8) Welche der folgenden Aussagen zu Pseudoterminals treffen im Allgemeinen zu? P.kte: \_ /4

*Zutreffendes ankreuzen:*

- ☐ Ja | ☐ Nein      Verbindet Keyboard-Hardware direkt mit User Space Prozess.
- ☐ Ja | ☐ Nein      Erlaubt einem Prozess im User Space, Zeichen "einzutippen".
- ☐ Ja | ☐ Nein      Besteht aus einem Master und mehreren Subordinate Devices.
- ☐ Ja | ☐ Nein      Ermöglicht Nutzung Terminal-orientierter Programme via SSH.

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