

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 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.

Prozess Lebenszyklus

1) Was ist der Output dieses Programms und wieso?

Punkte: ___ / 6

```
01: int main() {  
02:     while(fork() == 0) {  
03:         printf("%d\n", getpid());  
04:     }  
05:     return 0;  
06: }
```

Output und Begründung hier eintragen; Annahme: #includes sind vorhanden.

2) Welche dieser Aussagen sind korrekt?

Punkte: ___ / 6

Zutreffendes ankreuzen:

- | | |
|---|---|
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Rückgabewert von <i>fork()</i> ist immer eine Prozess ID. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Ein Child Prozess teilt den Speicher mit dem Parent. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Der Speicher des Parents wird beim <i>fork()</i> -en kopiert. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Ein Parent lebt immer länger als seine Child Prozesse. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Ein Parent Prozess kann zum Zombie Prozess werden. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Ein Child Prozess kann den <i>init</i> Prozess "adoptieren". |

Threads und Synchronisation

3) Was ist der Output dieses Programms, und wieso?

Punkte: ____ / 8

```

01: pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
02: pthread_cond_t c = PTHREAD_COND_INITIALIZER;
03:
04: void *start(void *arg) {
05:     pthread_mutex_lock(&m);
06:     printf("A\n");
07:     pthread_cond_signal(&c);
08:     pthread_mutex_unlock(&m);
09: }
10:
11: int main() {
12:     printf("B\n");
13:     pthread_t thread;
14:     pthread_mutex_lock(&m);
15:     pthread_create(&thread, NULL, start, NULL);
16:     printf("C\n");
17:     pthread_cond_wait(&c, &m);
18:     printf("D\n");
19:     pthread_mutex_unlock(&m);
20: }

```

Output und Begründung hier eintragen; Annahme: #includes sind vorhanden.

Output	Begründung

IPC mit Pipes

4) Schreiben Sie ein Programm, das ein per Command Line übergebenes Wort über eine Pipe vom Parent zum Child-Prozess sendet, und dort auf `STDOUT_FILENO` ausgibt. P.kte: __ / 12

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

<code>pid_t fork(void); // create a child process, returns 0 in child process</code>
<code>int pipe(int pipe_fd[2]); // create pipe, from pipe_fd[1] to pipe_fd[0]</code>
<code>int close(int fd); // close a file descriptor</code> <code>ssize_t read(int fd, void *buf, size_t count); // read from a file descr.</code> <code>ssize_t write(int fd, const void *buf, size_t count); // write to a file</code>
<code>size_t strlen(const char *s); // calculate the length of a string</code>

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

Sockets

5) Schreiben Sie ein Programm, das UNIX Domain Datagram Nachrichten unter der Adresse `/echo` empfängt, und den Inhalt der Nachricht an den Absender zurück schickt. P.kte: ___ / 14

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

```
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
```

A UNIX domain socket address is represented in the following structure:

```
struct sockaddr_un {
    sa_family_t sun_family; // AF_UNIX
    char sun_path[108]; // Pathname
};
```

```
int bind(int socket, struct sockaddr *addr, socklen_t addrlen); // bind
```

```
ssize_t recvfrom(int socket, void *buf, size_t len, int flags, // = 0
    struct sockaddr *address, socklen_t *address_len); // receive message
```

```
ssize_t sendto(int socket, void *message, size_t len, int flags, // = 0
    struct sockaddr *dest_addr, socklen_t dest_len); // send message
```

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

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

6) Vergleichen Sie UNIX Datagram Sockets mit Internet Datagram Sockets. Punkte: __ / 6

UNIX Domain Datagram Sockets	Internet Datagram Sockets (UDP)

POSIX IPC

7) Schreiben Sie ein Programm, das immer n Child-Prozesse hat, welche zufällig, nach `rand()` Sekunden terminieren. Ein Named Semaphor soll die Anzahl konstant halten. Und ein Signal Handler soll bei SIGCHLD jeweils `wait()` aufrufen, um Zombies zu verhindern. Punkte: __ / 16

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

<code>int atoi(const char *nptr); // convert a string to an integer</code>
<code>int rand(); // returns a random number [0, RAND_MAX]</code>
<code>void exit(int status); // cause normal process termination</code>
<code>pid_t fork(void); // create a child process; returns 0 in child process;</code>
<code>// the termination signal of the child is always SIGCHLD</code>
<code>pid_t wait(int *wstatus); // wait for process to change state</code>
<code>sem_t *sem_open(char *name, int oflag, mode_t mode, unsigned int value);</code>
<code>// initialize and open a named semaphore; O_CREAT, ...; S_IRUSR, S_IWUSR, ...</code>
<code>int sem_post(sem_t *s); // increment a semaphore</code>
<code>int sem_wait(sem_t *s); // decrement a semaphore, blocking if <= 0</code>
<code>typedef void (*sighandler_t)(int); // handler signature</code>
<code>sighandler_t signal(int signum, sighandler_t handler); // install handler</code>
<code>int sleep(int seconds); // calling thread sleeps for a number of seconds</code>

Fortsetzung auf der nächsten Seite.

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

9) Schreiben Sie ein Programm, welches auf Grund der aktuellen Lokalzeit das Datum des ersten Sonntags im nachfolgenden Monat berechnet und wie im Beispiel ausgibt. P.kte: _ / 14

```
$ ./sunday  
04.07.2021
```

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

```
int printf(const char *fmt, ...); // fmt int %d or e.g. %02d, string %s  
  
time_t time(time_t *t); // get local time in seconds since Epoch  
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  
  
struct tm {  
    int tm_sec, tm_min, tm_hour; // (0-60), (0-59), (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)  
};  
  
size_t strftime(char *s, size_t max, char *format, struct tm *tm);  
// format date and time, e.g. "%d.%m.%Y"
```

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

Zusatzblatt zu Aufgabe Nr. ____ von (Name) _____