Fachhochschule Nordwestschweiz
Hochschule für Technik

System-Programmierung (syspr) 04. November 2025 thomas.amberg@fhnw.ch

Assessment I

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.

Erste Schritte in C

1) Welche dieser Deklarationen von *main()* sind korrekt in C? Punkte: _ / 4

Zutreffendes ankreuzen

☐ Ja | ☐ Nein `int main(char **argv);`

☐ Ja | ☐ Nein `int main(int argc, char *argv[]);`

☐ Ja | ☐ Nein `int main(int argc, string argv[]);`

☐ Ja | ☐ Nein `int main(char *argv[], int argc);`

2) Gegeben die folgenden Structs, welche Statements sind korrekt? Punkte: _ / 4

```
struct r { long a; long b; long c; };
struct s { long n; struct r *p; };
```

Zutreffendes ankreuzen

☐ Ja | ☐ Nein `sizeof(struct r) == 3 * sizeof(long)`

☐ Ja | ☐ Nein `sizeof(struct s) > sizeof(struct r)`

☐ Ja | ☐ Nein `sizeof(struct s) == sizeof(long) + sizeof(struct r)`

☐ Ja | ☐ Nein `sizeof(struct r *) < 2 * sizeof(long)`

Funktionen in C

3) Wie funktioniert ein System-Call? Punkte: _ / 4

Erklärung hier eintragen, vier wesentliche Schritte im Ablauf, je ein kurzer Satz:

0) Programm-Code im User Mode ruft glibc-Wrapper auf.
1)
2)
3)
4)
5) Resultat wird als return-Wert an Code zurückgegeben.

4) Schreiben Sie ein Programm *shuffle*, welches $n > 0$ übergebene Command Line Argumente in zufälliger Reihenfolge auf die Konsole ausgibt, wie hier im Beispiel. Punkte: _ / 12

```
$ ./shuffle one two three
three one two
$ ./shuffle a b c d e f
b a d e f c
```

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

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

```
long random(void); // returns a value between 0 and (2^31) - 1
```

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

File In-/Output

5) Schreiben Sie ein Programm *ncopy*, das eine Quelldatei in eine variable Anzahl Zieldateien kopiert. Die Dateinamen übergibt man per Command Line, wie im Beispiel. Punkte: _ / 14

```
$ echo "hello" > my.txt
$ ./ncopy my.txt your.txt their.txt
$ ls *.txt
my.txt   their.txt    your.txt
```

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

```
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_TRUNC, O_RDONLY, O_WRONLY. Modes, which are used together with O_CREAT, include S_IRUSR and S_IWUSR.
```

```
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 C Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Frage-Nr.:

6) Welche der folgenden Aussagen zu File In-/Output sind korrekt? Punkte: _ / 4

Zutreffendes ankreuzen:

☐ Ja | ☐ Nein Ein File Deskriptor identifiziert ein File über Funktionsaufrufe hinweg.

☐ Ja | ☐ Nein *STDOUT_FILENO* ist ein File Deskriptor, auf den man schreiben kann.

☐ Ja | ☐ Nein Der Return-Wert von *read(fd, buf, n)* ist genau *n*, oder -1 bei Fehlern.

☐ Ja | ☐ Nein Die Einträge in der Open Files Tabelle zeigen auf File Deskriptoren.

Prozesse und Signale

7) Welche der folgenden Aussagen zum Stack sind im Allgemeinen korrekt? Punkte: _ / 4

Zutreffendes ankreuzen:

☐ Ja | ☐ Nein Der Stack hat den virtuellen Speicher des Prozesses für sich allein.

☐ Ja | ☐ Nein Auf dem Stack allozierte Variablen überdauern Funktionsaufrufe.

☐ Ja | ☐ Nein Argumente und globale Variablen werden auf dem Stack alloziert.

☐ Ja | ☐ Nein Zwischen Stack und Heap befindet sich nicht-allozierter Speicher.

|

|

|

|

|

|

|

|

|

(Aufgabe 8 auf der nächsten Seite)

8) Schreiben Sie ein Programm *count*, das ausgibt, wie weit ein Prozess in 1 Sekunde zählen kann. Nutzen Sie dazu die *alarm()* Funktion, die das SIGALRM Signal auslöst. Punkte: _ / 12

```
$ ./count
counted to 75654776
```

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

```
int alarm(int sec); // arranges for a SIGALRM signal to be delivered to
the calling process in sec seconds; returns remaining sec of prev. alarm
```

```
int pause(void); // sleep until a signal causes invocation of a handler
```

```
typedef void (*sighandler_t)(int); e.g. SIGINT = 2, ^C; SIGTSTP = 20, ^Z
sighandler_t signal(int sig, sighandler_t handler); // set SIG_IGN,
SIG_DFL, or a programmer-defined function to handle the signal sig
```

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

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

Prozess Lebenszyklus

9) Schreiben Sie ein Programm *domino*, das eine Reihe von *n* Prozessen "aufstellt", die dann in umgekehrter Reihenfolge wieder "umfallen" bzw. terminieren. Der Parameter *n* wird per Command Line übergeben, der Output soll wie im Beispiel aussehen, mit PIDs. Punkte: _ / 12

```
$ ./domino 2
1001 set up
1002 set up
oops...
1002 fallen
1001 fallen
```

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

```
int atoi(const char *nptr); // convert a string to an integer
noreturn void exit(int status); // cause normal process termination
pid_t fork(void); // create a child process, returns 0 in child process
pid_t getpid(void); // returns the process ID of the calling process
int printf(const char *format, …); // format string %s, char %c, int %d
pid_t wait(int *wstatus); // wait for child process to terminate
```

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

10) Welche dieser Aussagen zu Prozessen sind korrekt? Punkte: _ / 4

Zutreffendes ankreuzen:

☐ Ja | ☐ Nein Der Parent bekommt von *fork()* die Child Prozess ID.

☐ Ja | ☐ Nein Die Prozess ID eines Childs ist nach dem *fork()* == 0.

☐ Ja | ☐ Nein Der Child Prozess terminiert immer vor dem Parent.

☐ Ja | ☐ Nein Die Funktion *execve()* führt ein neues Programm aus.

Threads und Synchronisation

11) Gegeben den folgenden Code, der zwei Schalter simuliert, die insgesamt genau *n* Konzert-Tickets verkaufen, implementieren Sie die *sell()* Funktion. Falls keine Tickets mehr verfügbar sind (*n* == *0*), gibt der jeweilige Thread seine ID plus " done." aus, und stoppt. Punkte: _ / 12

```
#include ... // ignore

int n;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;

void *sell(void *arg); // TODO

int main(int argc, char *argv[]) {
    n = atoi(argv[1]);
    pthread_t t1, t2;
    pthread_create(&t1, NULL, sell, NULL);
    pthread_create(&t2, NULL, sell, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    return 0;
}
```

|

|

|

Fortsetzung auf der nächsten Seite

(11) Verwenden Sie folgende Calls (soweit sinnvoll), ohne #includes und Fehlerbehandlung:

`int printf(const char *format, …); // format string %s, char %c, int %d`
`int pthread_mutex_lock(pthread_mutex_t *mutex); // lock a mutex, returns 0 on success; If the mutex object is already locked by another thread, the calling thread shall block until the mutex becomes available.`
`int pthread_mutex_trylock(pthread_mutex_t *mutex); // lock a mutex, non-blocking, returns 0 on success, EBUSY if the mutex could not be acquired because it was already locked.`
`int pthread_mutex_unlock(pthread_mutex_t *mutex); // unlock a mutex, returns 0 on success; If there are threads blocked on the mutex object, scheduling policy shall determine which thread shall acquire the mutex.`
`pthread_t pthread_self(void); // obtain ID of the calling thread`

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

12) Was sind zwei wesentliche Merkmale einer *Critical Section*? Punkte: _ / 4

Merkmale hier eintragen, jeweils einen ganzen Satz ausformulieren:

Zusatzblatt zu Aufgabe Nr. ____ von (Name) _____