Fachhochschule Nordwestschweiz
Hochschule für Technik

System-Programmierung (syspr) 02. Juni 2020 thomas.amberg@fhnw.ch

Assessment

Vorname / Name: *(via GitHub)* Punkte: ___ / 90, Note: ___

Klasse: 4ibb1 *Frei lassen für Korrektur.*

Hilfsmittel:

- Aufgaben werden allein zu Hause am Computer gelöst.
- Alle Unterlagen (Slides, Bücher) sind erlaubt, im Sinn von *open book*.
- Plus online Zugriff auf die Linux man pages unter http://man7.org/linux/man-pages

Nicht erlaubt:

- Jegliche Kommunikation mit anderen Personen.

Bewertung:

- Offene Fragen: Bewertet wird Korrektheit, Vollständigkeit und Kürze der Antwort.
- Programme: Bewertet wird die Skizze/Idee 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.

Bearbeiten der Aufgaben:

- Lesen Sie die Aufgabenstellungen in dieser PDF Datei.
- Bearbeiten Sie die bestehenden *C* und *TXT* Dateien.
- Compilieren Sie die bestehenden *C* Dateien mit

```
$ make FILENAME_OHNE_C
```

z.B. um die Datei hello.c zu compilieren

```
$ make hello
```

Ein Makefile ist bereits vorhanden.

Abgabe via GitHub:

- Committen Sie alle Änderungen an bestehenden *C* und *TXT* Dateien mit

```
$ git commit -m "update" *.c
$ git commit -m "update" *.txt
```

- Übermitteln Sie alle lokalen Commits an GitHub mit

```
$ git push
```

- Es zählt der Stand *auf GitHub* beim letzten Commit vor dem / am Ende der Prüfung.

0: name.txt

Fügen Sie Ihren *Vornamen* und *Namen* in die Datei *name.txt* ein. Verbindlich

1: args.c

Ergänzen Sie das Programm *args.c* so, dass es die Häufigkeit jedes Kleinbuchstabens von *a - z* in den übergebenen Argumenten zählt und wie in diesem Beispiel ausgibt. Punkte: ___ / 12

```
$ ./args the quick brown fox jumps over the lazy dog
a: 1
b: 1
c: 1
...
```

Verwenden Sie dazu die folgenden System Calls, Fehlerbehandlung können Sie weglassen:

```
memset(), printf(), strlen().
```

Fügen Sie ihre Idee (kurz) und ihren Code in die Datei args.c ein.

2: functions.c

Ergänzen Sie das Programm *functions.c* so, dass jedes Argument und jede Funktion genau einmal benutzt werden, und das Resultat = *42* ist, mit möglichst wenig Code. Punkte: ___ / 8

```
$ ./functions
42
```

Fügen Sie Ihren Code in die Datei *functions.c* ein.

3: malloc.c

Ergänzen Sie das Programm *malloc.c* so, dass es so viel Speicher an einem Stück alloziert, wie möglich, in maximal 1 Sekunde. Geben Sie die allozierte Anzahl Bytes aus. Punkte: ___ / 12

```
$ ./malloc
268435456
```

Verwenden Sie dazu die folgenden System Calls. Fehlerbehandlung nur soweit wie nötig:

```
clock(), free(), malloc(), printf().
```

Fügen Sie ihre Idee (kurz) und ihren Code in die Datei malloc.c ein.

4: segfault.txt

Wieso führt das untenstehende Programm zu einem *Segmentation Fault*? Punkte: ___ / 4

```
 1: #include <stdio.h>
 2:
 3: char *f() {
 4:     char a[] = "hello";
 5:     return a;
 6: }
 7:
 8: int main() {
 9:     char *s = f();
10:     printf("%s\n", s);
11: }
```

Antworten Sie in der Datei *segfault.txt*.

5: bytes.c

Ergänzen Sie das Programm *bytes.c* so, dass es die letzten *n* Bytes der Datei *file* ausgibt. Die Parameter *n* und *file* werden als Command Line Argumente übergeben. Punkte: ___ / 12

```
$ printf "abcdefghijklmnopqrstuvwxyz" > abc.txt
$ ./bytes 3 abc.txt
xyz
```

Verwenden Sie dazu die folgenden System Calls, Fehlerbehandlung können Sie weglassen:

```
atoi(), lseek(), open(), read(), write().
```

Fügen Sie ihre Idee (kurz) und ihren Code in die Datei bytes.c ein.

6: signal.c

Ergänzen Sie das Programm *signal.c* so, dass sich der Parent Prozess und ein Child Prozess das Signal SIGUSR1 *n* mal hin und her senden. Der Parent soll dabei jeweils "ping" ausgeben, das Child "pong". Der sendende Prozess soll vor dem Senden des Signals jeweils 1 Sekunde warten. Der Parameter *n* wird als Command Line Argument übergeben. Punkte: ___ / 18

```
$ ./signal 2
ping
pong
ping
pong
```

Verwenden Sie dazu die folgenden System Calls, Fehlerbehandlung können Sie weglassen:

```
atoi(), getpid(), getppid(), fork(), kill(), pause(), printf(), signal(),
sleep(), wait().
```

Hinweis: $ make signal compiliert mit -std=gnu99, d.h. Signal Handler bleibt registriert.

Fügen Sie ihre Idee (kurz) und ihren Code in die Datei signal.c ein.

7: clock.c

Ergänzen Sie das Programm *clock.c* so, dass es zuerst *n* Prozesse und dann *n* Threads erstellt und jeweils die gesamte CPU Zeit misst. Der Parameter *n* wird als Command Line Argument übergeben. Messen Sie möglichst nur die Erstellung, keine weitere Aktivität. Punkte: ___ / 16

```
$ ./clock 300
Created 300 children in 0.117629 s
Created 300 threads in 0.074987 s
```

Verwenden Sie dazu die folgenden System Calls, Fehlerbehandlung können Sie weglassen:

```
atoi(), clock(), exit(), fork(), printf(), pthread_create(),
pthread_detach(), pthread_exit(), pthread_join(), wait().
```

Fügen Sie ihre Idee (kurz) und ihren Code in die Datei clock.c ein.

8: mutex.c

Ergänzen Sie den Code in *mutex.c* so, dass Operationen auf Bankkonten Thread-safe werden, d.h. fehlerfrei aus mehreren Threads gleichzeitig benutzt werden können. Punkte: ___ / 8

Verwenden Sie dazu die folgenden System Calls, Fehlerbehandlung können Sie weglassen:

```
pthread_mutex_init(), pthread_mutex_lock(), pthread_mutex_unlock().
```

Fügen Sie ihren Code in die Datei mutex.c ein (ausnahmsweise hat's keine //... als Hinweis).