Beispiel `01_processes`

Dr. Günter Kolousek

29. August 2020

1 Allgemeines

- Drucke dieses Dokument **nicht** aus!
- Halte **unbedingt** die Coding Conventions ein! Zu finden am ifssh!
- **Alle** Beispiele müssen innerhalb eines **Mercurial** oder **Git**-Repositories sein. Dieses Repository hat auf Anforderung abgegeben zu werden! Es soll `<lastname>` benannt werden wobei **nur** Kleinbuchstaben zu verwenden sind. Solch ein Repository kann z.B. mittels `hg init mustermann` oder `git init mustermann` angelegt werden.

  Stelle zuerst sicher, dass sich eine richtig gefüllte `.hgrc` Datei im Homeverzeichnis befindet bzw. konfiguriere mittels `git config...`!
- Es ist `oft` zu "commiten". Zumindest **fünf** Mal pro Beispiel. Je mehr desto besser!
- regelmäßiges Backup deines Repos nicht vergessen (oder github…)
- Es sind nur Sourcedateien, Konfigurationsdateien und ähnliches ins Repository hinzuzufügen. Explizit nicht hinzuzufügen sind ausführbare Dateien, Softwarearchive, Backupdateien und ähnliches. Dazu ist die `.hgignore` bzw. die `.gitignore` sinnvoll zu warten!!!
- Ist das Repository nicht in einem entsprechenden Zustand, führt das zu einer **negativen** Beurteilung!
- An Betriebssystem sollte Linux verwendet werden, da die Angaben zwar weitgehend generisch, im Einzelfall jedoch auf Linux zugeschnitten, sind!!!
- Als Entwicklungsumgebung kannst du verwenden was du willst. Bei den praktischen Arbeiten sind nur einige Entwicklungsumgebungen im Testsystem vorhanden. Überprüfe dies im Vorhinein!
- **Jedes** Beispiel ist in einem Unterverzeichnis `<beispielname>` zu speichern! Für das aktuelle Beispiel heißt dieses Verzeichnis also `01_processes`. In diesem Sinne ist jetzt ein neues Verzeichnis `01_processes` anzulegen.

2 Aufgabenstellung

Es geht darum 2 Programme zu schreiben:

- Ein Programm gibt auf `stdout` in einer Endlosschleife im Sekundentakt das Zeichen aus, das auf der Kommandozeile mitgegeben wird.
- Ein zweites Programm startet zwei Prozesse des ersten Programmes: eines mit dem Zeichen "A" und eines mit dem Zeichen "B".

Die genaue Anleitung folgt im nächsten Abschnitt und setzt ein POSIX konformes System vorraus.

3 Anleitung

1. Lege ein erstes Meson Projekt gemäß der bereitgestellten `meson.build` an.
2. Schreibe zuerst ein Programm `aba`, das lediglich aus der Datei `aba.cpp` besteht.

   Dieses Programm soll sich forken und der Elternprozess gibt in einer Endlosschleife das Zeichen "B" aus und der Kindprozess tut das gleiche mit dem Zeichen "A" wobei jeder Prozess zwischen den Ausgaben eine Sekunde schläft.

   Ein Prozess kann mittels der Funktion `sleep(int)` aus der Headerdatei `unistd.h` schlafen gelegt werden.

   Die Ausgabe sollte in entwa folgendermaßen aussehen:

   ```
   BABABABABABABA...
   ```

   Von was hängt die genaue Ausgabe ab? Muss immer "B" am Anfang stehen?
3. Ändere die das Programm so ab, dass jeder Prozess sich nur eine halbe Sekunde schläft.

   Teste und wenn alles funktioniert, dann weiter zum nächsten Punkt.
4. Ok, funktioniert das so wie du dir das vorgestellt hast? Ja? Nein? Das hängt natürlich davon ab, was du programmiert hast. Aber ich gehe einmal davon aus, dass du so etwas wie `sleep(0.5)` verwendet hast, nicht wahr? Ok, das kann nicht funktionieren, da `sleep` eine ganze Zahl erwartet. Aber der Compiler hat es ja anstandslos akzeptiert…

   Warum?
5. So, jetzt mal richtig. Dazu wechseln wir von dem C-API zu dem `C++`-API:

   ```
   std::chrono::milliseconds sleeptime(500);
   std::this_thread::sleep_for(sleeptime);
   ```

Dazu musst du die Headers `chrono` und `thread` inkludieren. Wir sehen, dass es hier zwei Unternamensräume gibt.

Übersetze und schaue dir das Ergebnis wieder an.

Es könnte durchaus sein, dass du auch so etwas zu Gesicht bekommst:

```
BABABABABABAABABA...
```

Beachte die beiden aufeinanderfolgenden "A"!

6. Ändere das Programm jetzt folgendermaßen ab, dass der Elternprozess nach 3 Sekunden den Kindprozess abbricht (also nach 6 "B"s) und sich danach selbst beendet.

   Zum Beenden schicke dem Kindprozess ein entsprechendes Signal, das dieser nicht ablehnen kann…

   Die Ausgabe sollte dann in etwa folgendendermaßen aussehen:

   ```
   BABABABABABA
   ```

   Was ist passiert? Hast du einen Zombie? Wenn ja, warum? Wenn nein, warum nicht?

   Wie lässt sich ein Zombie erzeugen und wie lässt sich dieser auf der Konsole zeigen? Gib die pid aus, lege den Prozess für 10s schlafen und zeige dir den Prozess auf der Konsole mittels `ps <pid>` an.

7. Verändere das Programm jetzt so, dass kein Zombie entsteht (und sich sonst genau gleich verhält).

   D.h. bis jetzt liegt ein Miniprogramm vor, das nur "A"s bzw. "B"s in zwei Prozessen ausgibt, wobei der Elternprozess den Kindprozess nach insgesamt 3s beendet und danach auf diesen wartet.

8. Schreibe jetzt vorerst ein Programm `charout`, das das als Kommandozeilenargument übergebene Zeichen in einer Endlosschleife auf `stdout` ausgibt, wobei wiederum die 500ms Schlafenszeit einzuhalten ist. Ändere dazu die Datei `meson.build` entsprechend ab, dass ein weiteres Executable erzeugt wird.

   Teste!

9. Schreibe jetzt das Programm `aba` so um, dass es sich im Kindprozess durch das Programm `charout` ersetzt.

   Achte darauf, dass es das Programm `charout` unter Umständen nicht gibt. Diese Fehlersituation sollte erkannt werden und die richtige Fehlermeldung ausgegeben werden. Also in Abhängigkeit von `errno`…

   Außerdem soll sich der Prozess mit dem Exit-Code 1 beenden.

10. Baue nun das Programm `aba` so um, dass sich dieses zwei Mal forkt und jeweils durch `charout` ersetzt, wobei einmal "A" und einmal "B" als Kommandozeilenargument genommen wird.

    Das Hauptprogramm übernimmt das Beenden beider Programme (nach 3 Sekunden) und kümmert sich darum, dass keine Zombies entstehen!

11. Nun wollen wir noch auf die Umgebungsvariablen zugreifen. Dazu soll das Programm `aba` anstatt der Zeichen "A" und "B" die Zeichen der Umgebungsvariablen `ABA_LETTER_A` und `ABA_LETTER_B` übernehmen, falls diese Umgebungsvariablen existieren.

12. Letztendlich baue die Ausgabe so, dass diese in der folgenden Art und Weise erscheinen wird (also bzgl. der Formatierung):

```
$ aba
waiting for 3 seconds
ABABABABABAB
killing both subprocesses with pids 8120 and 8121
waiting for both subprocesses to be dead
subprocess 8120 exited with 0
subprocess 8121 exited with 0
```

    Existiert das Executable `charout` nicht, dann sollte die Ausgabe folgendermaßen aussehen:

```
$ aba
waiting for 3 seconds
starting charout failed: No such file or directory
starting charout failed: No such file or directory

killing both subprocesses with pids 8949 and 8950
waiting for both subprocesses to be dead
subprocess 11661 exited with 1
subprocess 11662 exited with 1
```

4 Übungszweck dieses Beispiels

- Meson kennenlernen
- Verständnis für Prozesse vertiefen
- Starten von Prozessen mittels `fork`
- Exit-Codes einsetzen

- Erstes Kennenlernen von `sleep`, und `sleep_for`
- Versenden von Signalen, beenden eines anderen Prozesses, warten auf einen beendeten Prozess
- Verwenden von `execl`, `errno` und `strerror`
- Zugriff auf Umgebungsvariable