Beispiel `04_processes`

Dr. Günter Kolousek

1. Oktober 2018

1 Allgemeines

- Im ersten Beispiel gibt es genaue Anweisungen zum Aufbau und der Durchführung eines Beispiels. Bei Bedarf nochmals durchlesen!
- Verwende das bereitgestellte Archiv `template.tar.gz` zum Erstellen eines Meson-Projektes.
- Die Anpassungen der Datei `.hgignore` sollten schon erledigt sein, sodass das Verzeichnis `build` nicht versioniert wird.
- In diesem Sinne ist jetzt ein neues Verzeichnis `04_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. 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?

2. 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.

3. 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?

4. 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”…

5. 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.

6. 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.

7. 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!

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

9. 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 (vielleicht nach 3 Sekunden?) und kümmert sich darum, dass keine Zombies entstehen.

10. Letztendlich 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.

4 Übungszweck dieses Beispiels

- Verständnis für Prozesse vertiefen
- Starten von Prozessen mittels `fork`
- Erstes Kennenlernen von `sleep`, und `sleep_for`.
- Versenden von Signalen, Beenden eines anderen Prozesses samt warten auf diesen.
- Verwenden von `execl`.
- Zugriff auf Umgebungsvariable