Unix Prozesse

Aufgabe 1: **mach2s** und **machXs**

Entwickeln Sie zunächst die folgenden zwei Programme:

mach2s:

- den Namen des eigenen Programms ausgeben (den Namen bitte nicht „hard-codieren“)
- zwanzigmal alle 100 Millisekunden den string "`2 `" ausgeben
- einen Zeilenumbruch ausgeben

So sieht es dann aus:

```
mach2s:
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Hinweis: aus dem vollständigen Dateinamen können Sie mit `basename(...)` den kurzen Namen extrahieren (`basename` ist in `libgen.h` deklariert).

machXs:

- den Parameter x als Kommandozeilenparameter einlesen
- x in `int` wandeln (Funktion `atoi(...)oder strtol(...)`), evtl. Fehlermeldung/ Abbruch
- den Namen des eigenen Programms ausgeben
- x-Sekunden lang alle 100 Millisekunden den string "`x `" ausgeben
- maximal 20 "`x `" pro Zeile
- einen Zeilenumbruch ausgeben

Für x=5 sieht es so aus:

```
machXs:
x x x x x x x x x x x x x x x x x x x x
x x x x x x x x x x x x x x x x x x x x
x x x x x x x x x x
```

Üben Sie ein wenig den Umgang mit Prozessen aus dem Kommandofenster. Kopieren Sie dazu die executables Ihrer beiden Programme in ein gemeinsames Verzeichnis und öffnen Sie dort ein Terminal:

```
$ ./mach2s
$ ./machXs 5                    // 5 Sekunden
$ ./machXs 40 &                 // 40 Sekunden, background-process
$ ./machXs 40 > out.txt &       // 40 Sekunden, background-process, Ausgabeumleitung
```

Nun verbleibt hinreichend Zeit, den Prozessstatus zu beobachten:

```
$ ps -ef                        // process status, extended full listing
$ cat out.txt                   // Datei an stdout senden
$ more out.txt                  // Datei seitenweise ausgeben
$ ./machXs 100 &                // 100 Sekunden, background-process
```

In einem zweiten Terminalfenster:

```
$ ps -ef | grep mach            // Ausgabe filtern: nur Zeilen, die mach enthalten
```

Diesen Zeilen die PID xxxx entnehmen und damit den machXs-Prozess stoppen:

```
$ kill -9 xxxx                  // entspricht kill -SIGKILL xxxx
```

Alternativ kann die PID auch so gefunden werden:

```
$ pidof machXs
```

Aufgabe 2: systemStart

Schreiben Sie ein Programm, das:

- den eigenen Kurznamen ausgibt
- andere Prozesse startet (* unten)
- den Text "`fertig\n`" ausgibt

*: starten Sie mit dem blockierenden Aufruf **`system(...)`** folgende Prozesse
(immer nur alternativ die Fälle A), oder B) oder C) ... , den Rest dann auskommentieren):

A) `ls -l`

B) `mach2s`

C) `machXs 10` und
`mach2s`

D) `machXs 10&` und
`mach2s`

E) `machXs 20 > out1.txt&` und
`machXs 30 > out2.txt&` und
`mach2s`

Hinweis: Entwicklungsumgebungen ändern gern die current-directory - bei Code::Blocks ist es z.B. das Projektverzeichnis, nicht Release oder Debug. Alternativ können Sie daher:

- alle executables in die current-directory (etwa das Projektverzeichnis) kopieren
- alle executables in eines Ihrer Verzeichnisse kopieren und von dort das Programm per Kommando starten
- absolute Pfade zu den Projekt ... Release Verzeichnissen verwenden. Dies hat den Vorteil, dass Änderungen an den machxxx-Programmen direkt hier wirksam werden.

Aufgabe 3: fork01

Schreiben Sie folgendes Programm:

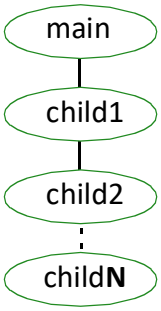
- den eigenen Kurznamen ausgeben
- mit **`fork()`** einen child-Prozess erzeugen. Der soll:
  - PID und PPID ausgeben
  - im 50ms Takt die Kleinbuchstaben a bis z ausgeben
  - den string "`fertig\n`" ausgeben und dann terminieren
- der parent-Prozess soll:
  - PID und PPID ausgeben
  - auf die Terminierung des childs warten und dann
  - im 50ms Takt die Großbuchstaben A bis Z ausgeben
  - den string "`fertig\n`" ausgeben und terminieren

Testen Sie auch was passiert, wenn der parent nicht wartet.

Aufgabe 4: fork02

Schreiben Sie folgendes Programm:

- einen Kommandozeilenparameter **N** (Anzahl childs) einlesen/verarbeiten
- den eigenen Kurznamen, PID, PPID und den Text `"Es werden jetzt..."` ausgeben
- mit **`fork()` N** childs erzeugen, sodass die Vererbungsstruktur eine ***Kette*** ergibt:



- die childs geben ihre child-Nr. i, PID und PPID aus und warten dann als parent auf die Terminierung Ihres childs

Für N=5 wird etwa dies ausgegeben:

```
fork02:
PID, PPID: 31042 5456
Es werden jetzt 5 child-Prozesse erzeugt:

 1: PID, PPID: 31043 31042
 2: PID, PPID: 31044 31043
 3: PID, PPID: 31045 31044
 4: PID, PPID: 31046 31045
 5: PID, PPID: 31047 31046
```

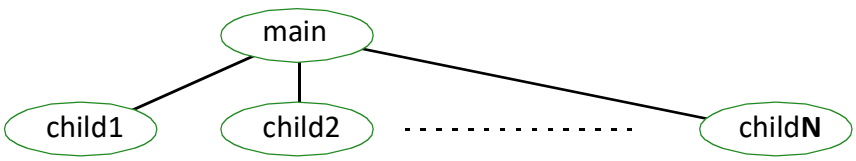
Anhand der PIDs und PPIDs können Sie die Vererbungsstruktur verifizieren.

Hinweis: In einer „Erzeugerschleife“ wird für alle parents `wait(...)` aufgerufen und anschließend die Schleife verlassen.

Aufgabe 5: fork03

Dieses Programm soll aus **fork02** erstellt und wie folgt abgeändert werden:

Die Vererbungsstruktur soll nun ein ***Fächer*** sein:



Der parent soll hier nicht auf die childs warten und terminiert dann vor zumindest einigen seiner childs (versuchen Sie z.B. N=10).

Stellen Sie fest, wer dann parent der allein gelassenen Kinder (orphans) ist.

Aufgabe 6: exec01

Schreiben Sie folgendes Programm:

- den eigenen Kurznamen ausgeben
- mit **`fork()`** einen child-Prozess erzeugen. Der soll:
  - Variante A): mit **`execv`** das **`mach2s`**-Programm ausführen
  - Variante B): mit **`execl`** das **`mach2s`**-Programm ausführen
- der parent soll auf die Terminierung des childs warten, dann "`fertig\n`" ausgeben

Hinweis: Die `execX`-Funktionen kehren nur bei Misserfolg zurück. Im Normalfall muss man daher im child-Zweig nach einem `execX`-Aufruf nicht über die weitere Ausführung nachdenken.

Aufgabe 7: exec02

Dieses Programm soll:

- den eigenen Kurznamen ausgeben
- mit **`fork()`** zwei direkte childs erzeugen:
  - child1 führt mit **`execl`** aus: `machXs 5`
    Ausserdem soll die Ausgabe in `out.txt` geleitet werden.
  - child2 führt mit **`execl`** aus: `mach2s`
- auf die Terminierung beider childs warten, dann "`fertig\n`" ausgeben

Die IO-Umleitung kann hier nicht mit | erfolgen, da keine shell ausgeführt wird (anders als beim `system()`-Aufruf).

In child1 muss daher vor dem `execl`-Aufruf die IO-Umleitung von `stdout` auf `out.txt` erfolgen:

- Datei `out.txt` öffnen
- Filedeskriptor für stdout schließen/freimachen
- den Deskriptor der Datei mit `dup()` an den jetzt freien Deskriptor binden

Als Ergebnis werden beide childs quasi parallel gestartet.
`mach2s` terminiert zuerst, `machXs` läuft drei Sekunden länger und schreibt seinen Namen sowie die fünfzig x in die Datei `out.txt`.