1. Prozessbaum

a) Beim letzten Vorlesungstermin haben wir besprochen, wie ein Prozess mit `fork()` verdoppelt wird. Nach dem Aufruf mit

```
pid = fork();
```

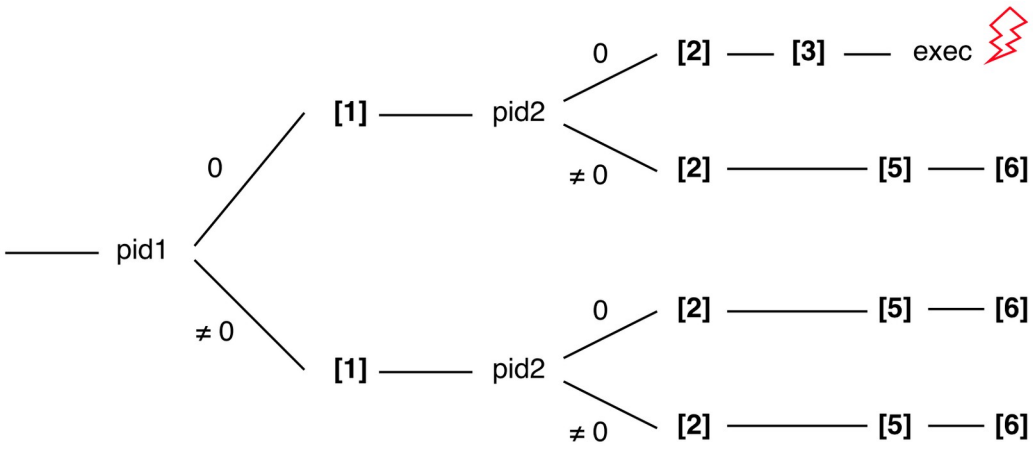
gibt es eine fast identische Kopie des ursprünglichen Prozesses, und beide setzen ihre Programmausführung unmittelbar hinter diesem `fork()`-Aufruf fort. Meist wird dann mit einer Fallunterscheidung zwischen Vater- und Kindprozess unterschieden:

```
if (pid == 0) {
  // im Kindprozess
  ...
} else
  // im Vaterprozess; pid ist die Prozess-ID des neu erzeugten Kindprozess
  ...
}
```

Durch die Fallunterscheidung können Vater- und Kindprozesse dann unterschiedliche Aufgaben erledigen; wir haben eine Gabelung (engl. *fork*). Gibt es im Programm mehrere `fork()`-Aufrufe, entsteht ein ganzer Baum. Betrachten Sie als Beispiel den folgenden Ausschnitt eines C-Programms:

```
int pid1 = fork();
printf ("%s\n", "[1] Ein Fork ist durch, einer muss noch.");
int pid2 = fork();
printf ("%s\n", "[2] Zeit für eine Fallunterscheidung.");
if ( (pid1==0) && (pid2==0) ) {
  printf ("%s\n", "[3] Ich starte jetzt emacs.");
  execl ("/bin/emacs", "emacs", "/etc/fstab", (char *)NULL);
  int pid3 = fork();
  printf ("%s\n", "[4] Nach dem dritten Fork.");
} else {
  printf ("%s\n", "[5] Ich gucke nur zu.");
};
printf ("%s\n", "[6] Nach der if-Abfrage endet das Programm.");
```

Es gibt sich der folgende Prozessbaum:



Die Zahlen in eckigen Klammern stehen für die nummerierten Ausgaben, die mit `printf()` erzeugt werden. Schauen Sie mit

```
man 3 printf
```

in die Manpage von `printf()`. Sie beenden den Dateibetrachter mit `q` (Taste [Q] drücken).

Bei jedem `fork()` entsteht eine Gabelung bzw. zwei Teilbäume. Beachten Sie, dass bei einem Aufruf von `execl()` ein anderes Programm (hier: `/bin/emacs`) in den laufenden Prozess geladen wird, so dass dieser Prozess nicht länger das ursprüngliche Programm ausführt!

b) Ein Programm führt nach dem Start die folgenden Befehle aus:

```
pid1 = fork();
pid2 = fork();
pid3 = fork();
pid4 = fork();
```

Wie viele Prozesse laufen nach dem letzten `fork()`-Aufruf? Zeichnen Sie den Baum, wie oben in Aufgabe **a)**.

Können Sie die Antwort verallgemeinern? Wenn (ohne Fallunterscheidungen) *k*-mal hintereinander `fork()` aufgerufen wird, aus wie vielen Prozessen besteht dann der so erzeugte (Teil-) Prozessbaum?

c) Betrachten Sie das folgende Programm `forktest.c` (Listing 1) und erstellen Sie wie in Aufgabe **a)** einen Prozessbaum, aus dem Sie ablesen können, wie oft die Ausgaben [1] bis [7] von den Prozessen erzeugt werden.

```
 1    // forktest.c
 2    // Betriebssysteme 1, v0.1, 2022-04-06
 3
 4    #include <stdlib.h>
 5    #include <unistd.h>
 6    #include <stdio.h>
 7    int main () {
 8      int pid1,pid2,pid3,pid4;
 9
10      printf ("[1] Start\n");
11      pid1 = fork ();
12      if (pid1 == 0) {
13        printf ("[2] vor dem exec\n");
14        execl ("/bin/true", "true", (char *)NULL);
15        printf ("[3] nach dem exec\n");
16        pid2 = fork ();
17        printf ("[4]\n");
18      }
19      else {
20        printf ("[5] zweiter Zweig\n");
21        pid3 = fork ();
22        pid4 = fork ();
23        if (pid3+pid4 == 0) {
24          printf ("[6] pid-Summe ist 0\n");
25          exit (0);
26        }
27        printf ("[7] Ende\n");
28      }
29    }
```

Hinweis zur Fallunterscheidung `if (pid3+pid4 == 0)` in Zeile 23: Prozess-IDs sind positive (!) Integer-Zahlen. Wann kann also `pid3+pid4` den Wert `0` annehmen?

Weiterer Hinweis: Mit `exit(0)` wird der aufrufende Prozess beendet. Der Effekt (für die hier gesuchte Baumdarstellung) ist also ähnlich wie beim Aufruf von `execl()`: Dieser Prozess führt keinen weiteren Code aus dem obigen Programm aus.

d) Testen Sie `forktest.c` in Docker:

(i) Wechseln Sie in der Shell in den Ordner, in dem Sie Übungsdateien zu Betriebssysteme 1 ablegen.

(ii) Starten Sie Docker wie in Übung 1 mit

```
docker run -it --rm -v ${PWD}:/realworld hgesser/ubuntu-dev
```
(Windows Powershell, Linux, macOS)

bzw.

```
docker run -it --rm -v %cd%:/realworld hgesser/ubuntu-dev
```
(Windows `CMD.EXE`)

(iii) Sie sehen jetzt die Linux-Shell des Docker-Containers. Wechseln Sie mit

```
cd /realworld
```

in das Host-Arbeitsverzeichnis, das Sie mit `-v` in den Container gemountet haben.

(iv) Laden Sie mit

```
wget swf.hgesser.de/bs-b1/prakt/bs1-ue02.zip
```

ein Paket mit Quelldateien herunter und entpacken Sie es mit

```
unzip bs1-ue02.zip
```

Wechseln Sie dann mit

```
cd ue02
```

in den Ordner `ue02` und geben Sie dort

```
gcc -o forktest forktest.c
./forktest ; sleep 5
```

ein. Vergleichen Sie die Häufigkeiten der Ausgaben mit den von Ihnen in Aufgabe **c)** ermittelten Werten.

(v) Verlassen (und beenden) Sie den Ubuntu-Container durch Eingabe von `exit`.

e) (Zusatzaufgabe)

Erstellen Sie ein neues Programm, das ebenfalls mit `fork()` eine Reihe von Kindprozessen erzeugt. Es soll folgende Ausgaben mit den angegebenen Häufigkeiten erzeugen:

[1] 1x [2] 4x [3] 7x [4] 14x

Gehen Sie dazu umgekehrt wie in Aufgabe **c)** und **d)** vor: Erstellen Sie zunächst einen Prozessbaum, der (bis zu) 16 Prozesse enthält, und planen Sie, an welchen Stellen Ausgaben erfolgen und wo ggf. Prozesse mit `exit()` zu beenden sind oder Ausgaben durch eine Bedingung nur in einigen Prozessen erfolgen.

Prüfen Sie dann (durch Kompilieren und Starten), dass Sie das richtig geplant haben.