Übungsblatt 3

Abgabe: 20. Mai 2019

Aufgabe 3.1: Prozesse unter Linux (2 + 1 + 1 + 1 + 3 = 8 Punkte)

Der zentrale Begriff in Betriebssystemen ist der Prozess. Das Prozesssystem unter Linux gleicht in gewissem Sinne dem Dateisystem: Prozesse bilden eine hierarchische Baumstruktur, an deren Wurzel ein `root`-Prozess steht, der zusammen mit dem Linux-System gestartet wird. Jeder Prozess kann mehrere Kindprozesse erzeugen, jeder Kindprozess hat hingegen genau einen Vaterprozess.

Die Identifikation der Prozesse erfolgt über eine Nummer, die Prozess-ID (`PID`). Welche Prozesse gerade laufen und welche PID sie haben, können Sie mit dem Kommando `ps` (process status) ermitteln.

Die Erzeugung eines neuen Prozesses geschieht unter C mit dem Kommando `fork()`. Durch diesen Aufruf wird aus einem Prozess heraus ein zweiter Prozess (Kindprozess) gestartet. Dieser erhält eine Kopie der Systemumgebung des Vaterprozesses, d.h. er arbeitet auch auf demselben Programmcode. Die Unterscheidung zwischen Vater- und Kindprozess erfolgt über den Rückgabewert von `fork()`: der Vaterprozess erhält die PID des Kindprozesses, der Kindprozess erhält als Rückgabewert stets 0. Im Falle eines Fehlers ist der Rückgabewert < 0. Wenn also Vater- und Kindprozess unterschiedliche Aufgaben ausführen sollen, kann im Quelltext anhand des Rückgabewertes verzweigt werden (`ifelse`). Benötigt ein Prozess seine eigene PID, kann er sie mittels `getpid()` erfragen.

Mit der Funktion `wait()` kann der Vaterprozess so lange schlafen gelegt werden, bis der Kindprozess terminiert. Ebenfalls in Zusammenhang mit `fork()` wird häufig eine Variante von `exec()` verwendet, mit dem der Kindprozess eine neue Programmdatei lädt und ausführt.

ACHTUNG: In dieser Aufgabe benutzen wir Endlosschleifen. Die Programmausführung kann mit `CTRL+C` abgebrochen werden.

a) Als Anhang zu diesem Übungsblatt finden Sie im L2P das Programm `prozesse.c`. Compilieren und starten Sie es. Bitte warten Sie nicht darauf, dass das Programm terminiert; es beinhaltet eine Endlosschleife.

   Modifizieren Sie das Programm derart, dass der Kindprozess wiederum einen Kindprozess startet. Der "Enkel"prozess soll eine eigene Funktion erhalten, die (analog zu den bestehenden) das Zeichen 'C' ausgibt.

b) Ihnen ist sicher aufgefallen, dass die Zeichen auf zwei verschiedene Arten auf die Standardausgabe geschrieben werden (welches man im Allgemeinen nicht tun sollte). Der Kindprozess nutzt `fprintf`, welches zur Standard-C-Bibliothek (libc) gehört. Der Vater-Prozess verwendet `write`, welches auf den gleichnamigen Syscall zurückfällt. Was bewirkt das Kommando `fflush` beim `fprintf` und wieso wird es beim `write` nicht verwendet? Welche der beiden Varianten halten Sie ganz allgemein (sprich: wenn Sie mehr als nur einen Character schreiben) für effizienter?

c) Nehmen Sie nun an, dass einer der beiden Kindprozesse nur eine begrenzte Zahl an Schleifendurchläufen durchführen würde. Wie Sie in diesem Fall z.B. durch `top` feststellen können, existiert dieser Prozess auch nach seiner Beendigung noch als 'Zombie-Prozess' weiter. Was ist ein Zombie-Prozess und wofür ist dieser Zustand nötig?

d) Wird ein Kindprozess erzeugt, wird in seinem PCB die PID des Vaterprozesses abgespeichert. Geben Sie einen Grund an, warum dies gemacht wird.

e) Eine interessante Frage ist nun noch: werden Vater- und Kindprozess gleich behandelt? Geben alle Prozesse aus Aufgabenteil a) ungefähr gleich häufig ihr Zeichen aus? Um dies zu ermitteln, schreiben Sie

ein Programm, welches Zeichen von der Standardeingabe (`stdin`) liest und in regelmäßigen Abständen die Häufigkeiten der einzelnen Buchstaben auf der Standardausgabe ausgibt. Wenden Sie es auf den von Ihnen modifizierten Buchstabengenerator (aus Aufgabenteil a)) an und überprüfen Sie, ob alle Prozesse etwa dieselbe Prozessorzeit erhalten. Geben Sie als Lösung ihren Programmcode ab.

Tipp: Erinnern Sie sich bitte, wie Sie im Übungsblatt 2 Piping (`|`) verwenden konnten, um den Standard Output eines Programmes mit dem Standard Input eines anderen Programmes zu verbinden.

Aufgabe 3.2: Fork (3,5 + 1 + 2 = 6,5 Punkte)

a) Wir betrachten folgendes C-Programm:

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    if (fork() == 0)
    {
        if (fork() == 0)
            printf("B \n");
        else
            printf("u \n");
        printf("S \n");
    }
    else
    {
        fork();
        fork();
        printf("20 \n");
        if (fork() == 0)
            printf("Zwan \n");
        else
            printf("Zig \n");
    }
    return 0;
}
```

Zeichnen Sie einen Baum, der die Prozessstruktur darstellt. Die *Knoten* des Baums sollen jeweils die *`fork`-Verzweigstellen des Programms* repräsentieren. Verzweigen Sie den Baum an diesen Stellen, indem Sie mit dem *Kindprozess im linken* und mit dem *Elternprozess im rechten Teilbaum* fortfahren. Die Blätter des Baums beschreiben so jeweils genau einen Prozess im Lebenszyklus des Programms. Schreiben Sie in die *Blätter* des Baums alle *Ausgaben, die `write` für diesen Prozess erzeugt hat*.

b) Betrachten Sie die Ausgabe von 'B' und 'u'. Welches der beiden Zeichen wird eher ausgegeben? Begründen Sie Ihre Antwort.

c) Im obigen Code kann es vorkommen, dass der Root-Prozess terminiert, bevor alle Kindprozesse terminieren. Wie kann man sicherstellen, dass alle Kinder bereits terminiert sind, bevor man den Root-Prozess beendet? Erweitern Sie den Code dahingehend und erklären Sie welchen Effekt ihre Codezeilen für Root- und Kindprozesse haben. **Tipp:** Lesen Sie sich die Dokumentation von `wait(NULL)` durch, insbesondere der Rückgabewerte.

Aufgabe 3.3: IPC und Threads (2+1,5+1+1 = 5,5 Punkte)

a) Pipes und Shared Memory sind zwei Konzepte zum Austausch von Informationen zwischen Prozessen. Stellen Sie die Vor- und Nachteile der beiden Konzepte gegenüber.

b) In dieser Aufgabe betrachen wir Named Pipes, Message Passing and Shared Memory:

Eine Firefox-Instanz läuft und man gibt auf der Konsole folgendes ein:

```
$ firefox www.gnu.org
```

dann wird der Link im bereits bestehenden Fenster geöffnet. Anschließend beendet sich der gerade auf der Konsole aufgerufene `firefox`-Prozess sofort. Die IPC findet hier zwischen dem Prozess statt, der das geöffnete Fenster verwaltet, und dem Prozess, der auf der Konsole gestartet wird.

Welche Methode für IPC hat jeweils welche Vor- und Nachteile? Beachte auch die Vor- und Nachteile bei der Implementierung.

c) Was sind die wesentlichen Unterschiede zwischen Prozessen und Threads?

d) Warum ist es im Allgemeinen nicht sinnvoll, zu viele Threads innerhalb eines Prozesses zu verwenden?