1. Das Kommando ps

Rufen Sie unter Linux mit `ps auxw` eine Liste aller in der virtuellen Maschine laufenden Programme (Prozesse) auf. Schlagen Sie in der Manpage zu `ps` (die Sie mit `man ps` aufrufen) die Bedeutung der verschiedenen Spalten nach. Besonders interessant sind hier: `USER`, `PID`, `%CPU`, `STAT`, `START` und `TIME` -- je nach `ps`-Version können die Spalten auch andere Namen (oder deren deutsche Übersetzungen) als Titel haben.

a) Finden Sie heraus, wie Sie die Ausgabe mit der Option `-o xxx,yyy,zzz...` so anpassen, dass Sie nur die folgenden Informationen erhalten:

- Prozess-ID
- Prozess-ID des sog. Vaterprozesses (Parent Process ID)
- User-ID (nicht User-Name) des Prozessbesitzers (also: von Ihnen)
- Kommando, das ausgeführt wird

Geben Sie das vollständige Kommando an (das zu jedem Prozess alle vier Werte anzeigt).

b) Mit welcher Option für `ps` können Sie die Ausgabe auf Prozesse beschränken, die einen bestimmten Namen haben? Geben Sie probeweise eine Liste aller Shell-Prozesse (Name: `bash`) aus.

2. fork() im C-Programm

Betrachten Sie das kleine C-Programm `forktest.c`, das die Systemfunktion `fork()` aufruft, um einen neuen Prozess zu erzeugen:

```
#include <stdio.h>
main() {
  printf ("vor dem Fork-Aufruf \n");
  int pid = fork ();
  printf ("nach dem Fork-Aufruf \n");
}
```

Das Programm können Sie mit `gcc -o forktest forktest.c` kompilieren und dann mit `./forktest` (wichtig: mit führendem `./`) aufrufen.

a) Welche Zeilen gibt das Programm aus? Erklären Sie die Ausgabe und nennen Sie die Anzahl der Prozesse, die liefen.

b) Wie viele Prozesse laufen insgesamt, wenn Sie einen zweiten `fork()`-Aufruf unmittelbar nach dem ersten einbauen (`int pid2 = fork ();`)?

3. Prozesse und Signale

Öffnen Sie (in der grafischen Oberfläche; Start mit `startx`) ein Kommandozeilenfenster (Konsole, `xterm` etc.) und rufen Sie darin einen Editor auf, z. B. nedit:

```
nedit &
```

(Mit `&` starten Sie den Editor im Hintergrund und können in der Shell weiterarbeiten. Wenn gedit nicht installiert ist, müssen Sie ein anderes Programm starten, z. B. xeyes.)

Suchen Sie nun mit `ps auxw | grep nedit` nach dem Prozess und finden Sie seine Prozess-ID heraus. Mit dem Befehl `kill` können Sie dem Prozess Signale senden, z. B.

- `SIGSTOP` zum Anhalten des Prozesses (`kill -STOP` *ID*)
- `SIGCONT` zum Fortsetzen des Prozesses (`kill -CONT` *ID*)
- `SIGTERM` zum Beenden des Prozesses (`kill -TERM` *ID* oder `kill` *ID*)

Probieren Sie zunächst die ersten beiden Signale aus: Schicken Sie dem Editor das `STOP`-Signal und

versuchen Sie dann, im Editor-Fenster Text einzugeben. Wechseln Sie danach zurück in die Konsole und schicken Sie dem Editor das `CONT`-Signal. Sehen Sie nun den Text, den Sie im gestoppten Zustand eingegeben haben?

Beenden Sie schließlich den Prozess, indem Sie ihm das `TERM`-Signal schicken.

a) Suchen Sie nach einem Prozess, der Ihnen nicht gehört. Was passiert, wenn Sie diesem ein Signal (z. B. SIGSTOP) schicken?

b) Lesen Sie die Manpage zu `killall` durch. Was würde passieren (nicht ausprobieren...), wenn Sie den Befehl `killall tcsh` bzw. `killall bash` (je nach Standardshell auf Ihrem Linux-System) eingeben?

4. Prozesszustände

Warum gibt es die Zustandsübergänge **a)** blockiert → laufend und **b)** bereit → blockiert nicht?

5. Prozesse und Speicher

Bei der Definition des Prozesses haben wir als wichtig erkannt, dass zu einem Prozess immer ein separater Speicherbereich gehört, die Schnittmenge der Speicherbereiche von zwei Prozessen ist immer leer. Wie genau eine Aufteilung des physikalischen Hauptspeichers erfolgt, haben wir noch nicht besprochen, und das ist für diese Aufgabe auch irrelevant.

Gehen Sie mal von folgender alternativen „Definition“ aus, die nur im Rahmen dieser Übungsaufgabe gültig sein soll:

> Ein Betriebssystem unterteilt den Hauptspeicher in mehrere paarweise disjunkte (sich nicht überschneidende) Speicherbereiche. Ferner gibt es mehrere „Aktivitätsstränge“ (ausführbarer Code), und es gilt: Das Betriebssystem ordnet jedem Speicherbereich keinen, einen oder mehrere dieser Aktivitätsstränge zu.
>
> Der Scheduler eines solchen Betriebssystems wählt (nach einem nicht näher zu bestimmenden Verfahren) einen Speicherbereich aus und aktiviert dann einen der Aktivitätsstränge, die diesem Speicherbereich zugeordnet sind. Wir sprechen nicht von „Prozesswechsel“, sondern von „Speicherbereichswechsel“.

a) Wenn einem Speicherbereich 0 Aktivitätsstränge zugeordnet sind, was bedeutet das?

b) Einem Speicherbereich ist ein Aktivitätsstrang zugeordnet. Haben wir es hier mit einem Thread oder einem Prozess (bei klassischer Betrachtung) zu tun?

c) Einem Speicherbereich sind drei Aktivitätsstränge zugeordnet. Handelt es sich dabei (bei klassischer Betrachtung) um Threads oder Prozesse?

6. Zombie-Prozess erzeugen

Übersetzen Sie das Programm `fork-zombie.c` (die Datei finden Sie auf dem Webserver, http://fom.hgesser.de/) mit folgendem Befehl: `gcc -o fork-zombie fork-zombie.c` und führen Sie es anschließend aus: `./fork-zombie`

Betrachten Sie anschließend in einem zweiten Terminalfenster mit

```
ps -e -o pid,ppid,cmd,state -C fork-zombie
```

einen Teil der Prozessliste. Über die Process ID (PID) und die Parent Process ID (PPID) können Sie herausfinden, welcher der beiden Prozesse der Vater und welcher der Sohn ist.

a) Welcher Prozess (Vater oder Sohn) ist der Zombie?

b) Wie kommt es hier dazu, dass ein Zombie-Prozess entsteht?