Übungen zur Job- und Prozess-Verwaltung

1. Starten Sie Linux (in der virtuellen Maschine).
2. Melden Sie sich als normaler Anwender (nicht als *root*) mit Ihrem Benutzernamen (Standard: *fom*) und dem Passwort (auch *fom*) an.
3. Erzeugen Sie im Home-Verzeichnis mit dem Befehl `touch` drei Dateien `1.txt`, `2.txt` und `3.txt`. (Sie können also drei Dateinamen gleichzeitig als Argumente für `touch` vergeben.)
4. Öffnen Sie die erste der drei Dateien im Editor `vi`. Drücken Sie dann (im Kommandomodus von `vi`) [Strg-Z], um den Editor zu unterbrechen (aber nicht zu beenden). Prüfen Sie mit `jobs`, dass er nun als „stopped" in der Job-Liste auftaucht.
5. Wiederholen Sie Schritt 4 mit den Dateien `2.txt` und `3.txt`; nun gibt es also drei suspendierte Editoren (und entsprechend drei Jobs in der Job-Liste).
6. Das Kommando `jobs` können Sie auch mit der Option `-l` aufrufen: Dann zeigt es neben den Job-Nummern auch die Prozess-IDs an. Probieren Sie das aus.
7. Geben Sie `ps` ein und identifizieren Sie die drei Editor-Jobs in der (größeren) Prozessliste. Geben Sie `ps auxw` ein, um die Liste aller Prozesse in Langform zu sehen – finden Sie auch hier wieder die Prozesse. Nutzen Sie z. B. `grep`, um die Ausgabe von `ps` zu filtern.
8. In der Vorlesung haben Sie gesehen, dass man die Spalten in der `ps`-Ausgabe über die Option `-o` und geeignete Argumente anpassen kann (z. B. `ps -o user,pid,cmd` für die Ausgabe von Benutzername, Prozess-ID und Kommando). Suchen Sie in der Manpage zu `ps` (`man ps`) die richtigen Argumente für `-o`, um a) die Prozess-ID, b) die „parent process id" (Prozess-ID des Vaterprozesses), c) den Nice-Level, d) die Anzahl der Threads (in Linux-Notation: „light weight processes") und e) das Kommando anzuzeigen.
9. 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.
10. Starten Sie durch dreimalige Eingabe von `sleep 1000 &` im Hintergrund drei Jobs, die einfach 1000 Sekunden schlafen und sich dann beenden. Beenden Sie den ersten Job mit `kill`, wobei Sie die Job-ID dieses Jobs verwenden. Achten Sie dabei auf die richtige Syntax für die Angabe der Job-ID. Prüfen Sie mit `jobs`, dass der Prozess verschwunden ist. Beenden Sie danach den zweiten Prozess, diesmal aber über seine Prozess-ID. (In Aufgabe 6 haben Sie gesehen, wie Sie die Zuordnung Job-ID → Prozess-ID herausfinden.) Prüfen Sie erneut, dass nun nur noch der dritte `sleep`-Prozess übrig ist.
11. Die Editor-Prozesse werden Sie mit `kill` nicht so einfach los, weil sie suspendiert sind und das Signal erst nach dem „Aufwecken" (mit `fg`) verarbeiten würden. Sie können aber das KILL-Signal (Nr. 9, also `kill -9 …` oder `kill -KILL …`) an die Editor-Prozesse schicken, das auch suspendierte Prozesse erfolgreich beendet. Schießen Sie auf diese Weise die ersten beiden Editor-Prozesse ab (wahlweise über die Job- oder Prozess-IDs). Holen Sie dann den verbleibenden Editor-Prozess in den Vordergrund: Sie sollten dann wieder im Editor arbeiten können (in der Datei `3.txt`). Verlassen Sie den Editor auf „normale" Weise, z. B. über [Esc] und `:wq`.
12. Geben Sie den Befehl `top` ein, um eine sich regelmäßig aktualisierende Liste der laufenden Prozesse zu erhalten. Nach welcher Spalte ist die Ausgabe sortiert? Ändern Sie die Sortierspalte und beobachten Sie, wie dadurch andere Prozesse „nach oben" kommen. Tipp: Die Hilfe in `top` erreichen Sie über `?` oder `h`. Verlassen Sie das Programm (`q`).

    Übrigens: Eine hübsche Alternative zu `top`, die auch das Scrollen zu weiteren Prozessen (die bei `top` aus der Anzeige rausfallen) erlaubt, ist `htop`. Sie können es mit `sudo apt-get install htop` nachinstallieren.

Verständnisfragen

13. Was ist der Unterschied zwischen Jobs und Prozessen?
14. Sie können Jobs und Prozesse beenden, indem Sie ihnen eines der beiden Signale `TERM` und `KILL` schicken. Welches Signal verschickt das `kill`-Kommando, wenn Sie kein Signal explizit angeben? Wie unterscheidet sich die Prozessbeendung via `TERM` und `KILL`?
15. Welche Signale kennt Ihr Linux-System? (Tipp: Die Manpage von `kill` erklärt, wie Sie eine Liste aller bekannten Signale anzeigen lassen können.)
16. Warum ist es (bei Linux/Unix) eine gute Eselsbrücke, von Nice-Levels statt von Prioritäten zu sprechen? Welche Nice-Levels kennt ein Linux-System?
17. Was meinen Sie: Warum kann ein normaler Anwender (nicht *root*) Prozessen zwar mit `nice`/`renice` eine niedrigere als die Standardpriorität zuweisen, aber keine höhere?
18. Um einen Prozess auch über die Abmeldung vom System hinaus weiter arbeiten zu lassen, können Sie `nohup` oder `disown` verwenden. Erklären Sie den Unterschied.

Aufgaben zur Programmierung und zur Theorie

19. **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 ();`)?

20. **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 nedit 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?

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

22. **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?