Prof. Dr. M. Jäger
FB MNI

15. April 2014

Übung 1 - Betriebssysteme

Im UNIX-Umfeld ist eine „Shell“ ein Kommando-Interpretierer. Wir verwenden die „bash“-Shell. Die Shell kennt einige interne Kommandos z.B. „pwd“ (Print Working Directory) oder „cd“ (Change Directory). Ansonsten sind die Kommandos in der Regel Programmaufrufe. Dabei können den Programmen Parameter übergeben werden. Beispiel:

```
ls -l /home
```

Das aufzurufende Programm heißt „ls“, die dem Programm zu übergebenen Parameter sind „ls“, „-l“, und „/home“. Das heißt, dem Programm „ls“ wird als 1. Parameter der eigene Programmname übergeben.

Aufgabe 1 Shell

Lesen Sie zuhause eine „Bedienungsanleitung“ für eine POSIX-konforme Shell, z.B. http://homepages.thm.de/~hg52/lv/bs1/skripten/bash/pdf/bash.pdf.

Üben Sie den Umgang mit der UNIX-Shell anhand folgender Probleme

Aufgabe 2 Umlenkung der Standardausgabe und der Standardfehlerausgabe

Wir legen im Heimatverzeichnis ein neues Verzeichnis „aufgabe1“ an, wechseln in das neue Verzeichnis, erzeugen dort eine neue leere Datei namens „d“ und lassen uns deren Dateiattribute anzeigen.

```
cd               # gehe in das Heimatverzeichnis
mkdir aufgabe1   # erzeuge neues Verzeichnis
cd aufgabe1      # gehe in das neue Verzeichnis
touch d          # erzeuge leere Datei "d"
ls -l d          # Dateiattribute von "d" anzeigen
ls -l            # Dateiattribute aller Dateien im Verzeichnis anzeigen
```

Die spezielle Datei „/dev/null“ ist bei der Ausgabe eine Art „schwarzes Loch“, in der alle Ausgabedaten verschwinden.

Welches Ausgaben erwarten Sie am Bildschirm in der oben beschriebenen Situation bei den unten angegebenen Kommandos? Beachten Sie dabei: „ls -l d e“ soll alle Attribute der dateien „d“ und „e“ anzeigen, eine Datei „e“ existiert aber nicht!

```
ls -l e
ls -l d e
ls -l d e > /dev/null
ls -l d e 2> /dev/null
ls -l d e > /dev/null  2> /dev/null
ls -l d e >/dev/null 2>&1
```

Aufgabe 3

Ein Kommando liefert einen ganzzahligen Statuscode zurück: 0 heißt immer „OK“, andere Werte stehen für bestimmte Fehler. Den Statuscode des letzten Kommandos zeigt man an mit

```
echo $?
```

Betrachten Sie noch einmal das Beispiel von oben: Welche Statucodes liefern die nachfolgenden Kommandos und welche Ausgaben sieht man am Bildschirm?

```
ls -l e; echo $?
ls -l d; echo $?
ls -l d e; echo $?
ls -l d e 2> /dev/null; echo $?
ls -l d e > /dev/null  2> /dev/null; echo $?
ls -l d e > /dev/null  2> /dev/null; echo $? 2> /dev/null
ls -l d e > /dev/null  2> /dev/null; echo $? > /dev/null
```

Aufgabe 4

a) Starten Sie ein Konsolenprogramm, z.B. das Programm „xterm“. Verwenden Sie den „man“-Befehl (Online-Manual) um die Beschreibung des Befehls „ps“ (Prozesstatus) zu lesen.

b) Verwenden Sie den „ps“-Befehl, um herauszufinden, welche Prozesse unter ihrer Benutzeridentifikation existieren und welche Prozessnummern diese Prozesse haben.

c) Verwenden Sie den „kill“-Befehl, um einen ihrer Prozesse zu terminieren. Welche Parameter kann man für „kill“ verwenden?

Aufgabe 5 Programme schreiben und ausprobieren

a) Lesen Sie im Onlinemanual, wie die Systemaufrufe *getpid* und *getppid* benutzt werden.

b) Erstellen Sie ein C-Programm in der Datei „a1.c“, das seine eigene Prozessnummer und die seines Elternprozesses ausgibt. Übersetzen Sie das Programm in ein Objektmodul in der Datei „a1.o“. Binden Sie das Objektmodul zu einem Programm „a1“. Rufen Sie „a1“ mit dem Kommandointerpretierer auf.

c) Erweitern Sie Ihr Programm wie folgt:

   Nach der Ausgabe erfolgt ein Aufruf der Systemfunktion *fork*, ohne Parameter. Nach dem fork()-Aufruf steht die gleiche Ausgabeanweisung noch einmal, das Resultat des fork-Aufrufs wird außerdem ausgegeben.

```
printf("Meine PID ist %d,mein Elternprozess ist %d\n", getpid(), getppid());
int i=fork();
printf("Meine PID ist %d,mein Elternprozess ist %d\n", getpid(), getppid());
printf("fork()-Resultat: %d\n", i);
```

   Rufen Sie das Programm auf und erklären Sie die Ausgabe.