Prof. Dr. Th. Letschert
FB MNI
22. März 2002

Übung 2 - Betriebssysteme I

Aufgabe 1

1. Informieren Sie sich mit Hilfe der Manualseiten, der Online–Unterlagen von Dr. Jäger oder anderer Unterlagen über Unix–Kommandos und die Bash.
2. Erläutern Sie die Aufgabe der Dateien `.profile` und `.bashrc`.
3. Schreiben Sie ein C++–Programm, das seine Kommandozeilenparameter ausgibt. Übersetzen Sie dieses Programm und schreiben Sie eine Kommandoprozedur mit Interpreteraufruf. Ihr übersetzes Programm ist der Interpreter. Erläutern Sie die Wirkung der Kommandoprozedur!
4. Was ist die “bash” bzw. eine *Shell* ganz allgemein: Teil des Betriebssystem(kern)s, ein Programm, ein Prozess, ...?

Aufgabe 2, Bonusaufgabe 2, Termin 21. April 2001

1. Was ist eine *Shell*, warum nennt man eine *Shell* “Shell”? Erklären Sie die Wahl der Bezeichnung.
2. Erläutern Sie die Wirkung der folgenden Kommandos (soweit sie korrekt sind):

```
chmod u+x f
chown hugo f
chgrp hugo f

grep hugo *
grep -v hugo *.cc

echo ls
echo ‘ls‘
echo $(ls)
echo $PATH
echo $(PATH)
echo $($PATH)

if ! test -d Verz; then mkdir Verz; else echo Schon Da; fi
for f in $(ls); do if test -f $f; then  cat $f; fi; done

find /usr/include -name iostream -print
find /home/hg4711 -size +10000000c -print
find /tmp -type f -user hg4711 -ls
find /tmp -user hg4711 -atime +100 -exec rm {} \;
find /tmp -user hg4711 -ok -exec rm {} \;
```

3. Nehmen Sie an das Folgende sei jeweils der Inhalt einer Datei. Wie und mit welcher Wirkung können die Dateien ausgeführt werden:

```
-------------------------------------------------
for p in $*
do
  echo $p
done
-------------------------------------------------
for f in *.c
do
    mv $f ${f%.c}.cc
done
-------------------------------------------------
a=$1
case $a in
  J*) echo JA ;;
  N*) echo NEIN ;;
  *)  echo SONSTWAS ;;
esac
-------------------------------------------------
```

4. Schreiben Sie eine Kommandoprozedur `reinige` die in ein als Parameter übergebenes Verzeichnis wechselt und dort alle Dateien mit der Endung `.log`, `.aux`, `.tmp` und `.o` löscht.

5. Schreiben Sie einen Interpreter, der Kommandoprozeduren verarbeitet, die aus Kommandozeilen mit folgender Syntax bestehen:

```
<Kommando>  ::= <AusgabeK>   // Ausdruck auswerten und ausgeben
              | <SpeicherK>  // Ausdruck auswerten und speichern
<AusgabeK>  ::= DRUCK <Var> "=" <Ausdruck>
<SpeicherK> ::= SPEICHER <Var> "=" <Ausdruck> IN <Datei>
<Ausdruck>  ::= <Int-Zahl>
              | "(" <Ausdruck> <Op> <Ausdruck> ")"
<Op>        ::= "+" | "-" | "*" | "/"
<Datei>     ::= <String>
<Var>       ::= <String>
```

   Leerzeilen und und Zeilen die mit # beginnen werden ignoriert.

   Das Ausgabekommando berechnet den Wert des Ausdrucks und gibt ihn aus. Das Speicherkommando berechnet den Wert des Ausdrucks und speichert ihn in der Datei (ohne deren bisherigen Inhalt zu löschen). Beispiel: Das Programm

```
#!/Pfad/zu/Ihrem/Interpreter/Programm

SPEICHER x = ((2+3)*2) IN hugo.txt
SPEICHER y = 7 IN hugo.txt
DRUCK z = ((2+3)*2)
```

   führt dazu, dass in der Datei `hugo.txt`

```
x = 10
y = 7
```

gespeichert und `z = 7` ausgegeben wird.

6. Legen Sie eine eigene Homepage auf dem Server der FH an. (Minimalausstattung reicht!) Platzieren Sie dort eine Kommandoprozedur, die, als CGI–Skript aktiviert, Datum und Uhrzeit ausgibt.

Aufgabe 3

1. Ein moderner Prozessor kann einen Maschinenbefehl in einer Nano–Sekunde (10^{-9} Sekunden) ausführen. Ein Plattenzugriff dauert etwa eine Milli–Sekunde (10^{-3} Sekunden). Um diese Größen auf menschliche Maßstäbe zu übertragen nehmen wir an, dass ein Maschinenbefehl in einer Sekunde ausgeführt wird. Welche Zeitspanne entspricht dann einem Plattenzugriff. Welcher Zeitraum entspricht einem Kontextwechsel, wenn wir weiter annehmen, dass dieser tausend Maschinenbefehle benötigt?
2. Ein Prozess wird "quasi–parallel" und "virtuell ununterbrochen" ausgeführt. Was bedeutet das? Erläutern Sie die beiden Begriffe.
3. Erläutern Sie den Unterschied zwischen kooperativem und eingreifendem Multitasking.
4. Was ist ein *Time Sharing System* und welche Form des Multitasking benötigt es?
5. Was ist ein Batchsystem? Ist es sinnvoll ein Batchsystem mit Multitasking zu betreiben? Wenn ja welche Variante bietet sich an?
6. Was ist ein Kontextwechsel. Beschreiben Sie informal, wie ein Kontextwechsel ausgeführt wird.
7. Was bedeutet es für einen Prozess "im Hintergrund" ausgeführt zu werden.
8. In welcher Beziehung stehen Multitasking–Systeme und Multiprozessor–Systeme? Setzt das eine das andere voraus, oder folgt das eine aus dem anderen?
9. Bei welchem Systemaufruf *muss* auch ein System mit kooperativem Multitasking den laufenden Prozess wechseln?
10. Was bedeutet es für einen Prozess "blockiert" zu sein, geben Sie ein Beispiel für einen blockierten Prozess an.
11. Kann ein Prozess in einem *Single–Tasking System* , einem *Multi–Tasking System mit* und einem *Multi–Tasking System ohne Eingriff* jeweils blockiert sein?
12. Kann ein Prozess gleichzeitig bereit und blockiert sein?
13. Nehmen Sie an, Sie starten in zwei Kommandofenstern jeweils ein Programm das in einer Endlosschleife einen Wert von der Tastatur einliest, in einer Schleife mit 10000 Durchläufen die Wurzel berechnet und diese dann ausgibt. Erläutern Sie welche Zustände die Prozesse durchlaufen, die das Betriebssystem zur Abwicklung der beiden Programme erzeugt. Erläutern Sie auch die Unterschiede, die sich ergeben, wenn es sich um ein System mit kooperativem bzw. unterbrechendem Multitasking handelt.
14. Was versteht man unter einer "nebenläufigen Anwendung"?
15. Was ist ein SMP–System und wann ist ein solches symetrisch?

Aufgabe 4

1. Schreiben Sie mit Hilfe von `fork` einen Prozess, der vier Subprozesse (“Kinder”) erzeugt. Jeder Prozess soll seine PID und die seines Erzeugers ausgeben.

2. Schreiben Sie mit Hilfe von `fork` einen Prozess der ein Kind–, dieses einen Enkel– und der Enkelprozess einen Urenkelprozess erzeugt. Jeder Prozess soll seine PID ausgeben.

3. Schreiben Sie ein Programm `myExec`, das einen Prozess erzeugt und diesen dann einen als Programmargument übergebenen Befehl ausführen lässt. Z.B.:

   ```
   myExec ls -l
   ```

   Der Elternprozess soll auf das Ende des Subprozesses warten und im Fehlerfall eine eigene Fehlermeldung ausgeben. Beispiel:

   ```
   myExec xxls -l
   Exec fehlgeschlagen: Datei oder Verzeichnis nicht gefunden
   Programm xxls beendet mit Status 65280
   ```

   Benutzen Sie die Systemaufrufe `execvp`, `waitpid` und `perror`.

Aufgabe 5

Unix–Prozesse können im Vordergrund und im Hintergrund abgearbeitet werden. Ein Programm wird typischerweise dadurch im Hintergrund gestartet, dass man es mit dem &–Zeichen aktiviert. Ein Prozess im Hintergrund unterscheidet sich von einem im Vordergrund unter anderem dadurch, dass er nicht mit `CRTL-C` abgebrochen werden kann.

Das Programm `Hallo`

```
#include <iostream>
#include <unistd.h>

using namespace std;

int main () {
  for (;;) {
    cout << "Hallo, ich lebe noch immer\n";
    sleep (3);
  }
}
```

kann beispielsweise nur dann mit `CRTL-C` abgebrochen werden, wenn es im Vordergrund gestartet wird.

1. Übersetzen Sie das Beispielprogramm `Hallo` und starten es einmal als Vordergrund– und einmal als Hintergrund–Prozess. Bringen Sie beide Prozesse dann dazu zu stoppen.

2. Modifizieren Sie das Beispiel derart, dass mit `fork` ein Subprozess erzeugt wird, der die Endlosschleife ausführt. Der Elternprozess soll nach Erzeugen des Subprozesses in einer Endlosschleife mit 3 Sekunden Pause “Hallo Leute” ausgeben. Lässt sich Ihr modifiziertes Programm mit `CRTL-C` abbrechen?

3. Modifizieren Sie das Beispiel derart, dass mit `fork` ein Subprozess erzeugt wird, der die Endlosschleife ausführt. Der Elternprozess soll sich nach Erzeugen des Subprozesses sofort beenden. Lässt sich Ihr modifiziertes Programm mit `CRTL-C` abbrechen?

4. Schreiben Sie einen “Dämon”, der alle 15 Sekunden die Nachricht ausgibt “Aetsch – ich lebe noch!”.