Universität Bielefeld

Fakultät für Biologie
Lehrstuhl für Genetik
Rechnerbetrieb

Unix – ganz schmerzlos

Lösungen zum Übungszettel 8

Aufgabe 23: Die Fakultät einer Zahl läßt sich z.B. mit dem folgenden Skript berechnen:

```
#!/bin/sh

n=$1
fak=$n

while [ $n -gt 1 ]; do
    n=‘expr $n - 1‘
    fak=‘expr $fak \* $n‘
done

echo $1! = $fak
```

Die `while`-Schleife wird n-mal durchlaufen. Dabei wird jedesmal der Zähler `n` um eins erniedrigt und anschließend mit der bisher berechneten Fakultät multipliziert. Sobald die Schleife alle Faktoren abgearbeitet hat, wird sie beendet, und das Ergebnis wird ausgegeben.

Das Programm `expr` besitzt starke Einschränkungen. Abgesehen davon, daß es nur mit ganzen Zahlen rechnen kann, dürfen die Zahlen nicht zu groß werden. Wenn man z.B. 17! berechnen möchte, erhält man als Ergebnis -288522240, was offensichtlich falsch ist. Wie bereits in der Vorlesung erwähnt, sollte man in Skripten nur das nötigste Rechnen.

Aufgabe 24: Das folgende Skript ermöglicht die Konversion von mehreren GIF-, JPEG- oder TIFF-Bildern in PNG-Dateien:

```
#!/bin/sh

for file in $*; do
    case $file in
        *.gif)
            name=‘basename $file .gif‘
            prog="giftopnm"
            ;;
        *.jpg)
            name=‘basename $file .jpg‘
            prog="djpeg"
            ;;
        *.tiff)
            name=‘basename $file .tiff‘
            prog="tifftopnm"
            ;;
        *)
            echo $0: unknown format in $file, skipping
            continue
    esac
    if [ -f "$name.png" ]; then
        echo $0: $name.png already exists, skipping
    else
        $prog $file | pnmtopng > $name.png
    fi
done
```

Das Skript enthält zwei Dinge, die nicht in der Vorlesung erwähnt wurden:

- Der „catch-all-Fall“ der `case`-Anweisung (`*)`) enthält das Schlüsselwort `continue`. Dies bewirkt, daß der augenblickliche Durchlauf der `for`-Schleife abgebrochen wird, und sofort mit dem nächsten Wert weitergemacht wird. Dadurch wird der nachfolgende Teil (der Test von `$name.png` und der Aufruf des Konverters) nicht durchlaufen, da dies in der dann vorliegendenen Situation sowieso keinen Sinn machen würde.
- Das Skript kann zwei Fehlerzustände abfangen. In beiden Fällen wird neben der Fehlermeldung die Variable `$0` ausgegeben. Diese Variable enthält den Namen des Skripts. Dadurch ist die Fehlermeldung leichter zuzuordnen. Dieses Verfahren wird als Guter Stil™ angesehen und sollte auch in eigenen Skripten verwendet werden.

Aufgabe 25: Das Shell-Skript benötigt sehr viel länger zur Berechnung der 100 Quadratzahlen, als die beiden anderen Programme. Sowohl das Perl-Programm als auch das aus den C-Quellen erzeugte Programm laufen deutlich schneller. Beim Aufruf des Perl-Programms bemerkt man eventuell eine minimale Verzögerung, bevor die Ausgabe startet. In dieser Zeit analysiert der Perl-Interpreter das Programm und erzeugt eine leichter zu verarbeitende interne Darstellung des Skripts. Diese Analyse ist bei dem C-Programm nicht mehr nötig. Die reine Abarbeitsungzeit ist bei beiden Programmen fast identisch. Bei anderen Tests ließe sich aber eine prinzipiell höhere Geschwindigkeit von C erkennen.

Vor allem die Aufrufe von externen Programmen machen Shell-Skripten so langsam. Im vorliegenden Fall wird das Programm `expr` 200mal ausgeführt, und dies kostet sehr viel Zeit. Weil Perl und C alle Berechnungen „intern“ durchführen (und wegen diverser anderer Gründe), laufen in diesen Sprachen geschriebene Programme deutlich schneller ab.