Prof. Dr. Uwe Schmidt

ptlwedel
PHYSIKALISCH-TECHNISCHE LEHRANSTALT

25. Januar 2010

Aufgaben zur Klausur **Unix** im WS 2009/10 (IAT 351, IAW 351)

Zeit: 75 Minuten erlaubte Hilfsmittel: keine

Bitte tragen Sie Ihre Antworten und fertigen Lösungen ausschließlich an den gekennzeichneten Stellen in das Aufgabenblatt ein. Ist ihre Lösung wesentlich umfangreicher, so überprüfen Sie bitte nochmals Ihren Lösungsweg.

Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Viel Erfolg!

Diese Klausur besteht einschließlich dieses Deckblattes aus 8 Seiten.

Aufgabe 1:

Gegeben seien folgende sechs C–Quellcode–Dateien `Test.c, Mod1.c, Mod2.c, Mod1.h, Mod2.h` und `Types.h`. `Test.c` enthält das Hauptprogramm, dieses benutzt Routinen, die in `Mod1.h` deklariert sind und in `Mod1.c` implementiert sind. `Mod1.c` benutzt Routinen, die in `Mod2.h` deklariert sind und in `Mod2.c` implementiert sind. Alle `.c`–Dateien verwenden globale Datendefinitionen aus `Types.h`

Schreiben Sie einen `Makefile` zum Erzeugen eines Programms `Test` aus den oben beschriebenen Dateien. Beachten Sie dabei alle Abhängigkeiten zwischen den Dateien, entwickeln Sie den `Makefile` aber so, daß keine überflüssigen Aktionen gemacht werden.

Nutzen Sie keine im `make`–System vordefinierten oder eingebauten Regeln.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Aufgabe 2:

Gegeben sei folgendes bash-Skript:

```
#!/bin/bash

echo '#!/bin/bash'

for i in $*
do
    echo "echo $i 1>&2"
    echo "cat >$i <<'Ende von $i'"
    cat $i
    echo "Ende von $i"
done
```

Verfeinern Sie dieses Skript so, dass es auch mit Dateinamen korrekt arbeitet, die Leerraum und andere Sonderzeichen enthalten.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 3:

Beschreiben Sie, was das folgende Kommando berechnet:

find / −name '∗.html' −print | xargs wc −l

. . . . . .

. . . . . .

. . . . . .

Erweitern Sie das Kommando so, dass Fehlermeldungen ignoriert werden und nicht mehr in einer Konsole sichtbar werden.

. . . . . .

Entwickeln Sie ein Kommando, das zählt wie häufig das Wort *Unix* in allen HTML-Dateien im Dateisystem vorkommt.

. . . . . .

. . . . . .

Entwickeln Sie ein Kommando mit der gleichen Funktionalität wie bei dem oben gegebenen, das aber das xargs-Programm nicht verwendet.

. . . . . .

. . . . . .

Aufgabe 4:

Reguläre Ausdrücke sind ein gutes Werkzeug zur Beschreibung von Textmustern. Für den Aufbau von regulären Ausdrücken gibt es folgende Regeln:

1. jedes Zeichen aus dem Alphabeth (z.B. dem ASCII Zeichensatz) ist ein regulärer Ausdruck.
2. Sonderzeichen, die zum Aufbau der Ausdrücke verwendet werden,
   z.B. `\ [ ] . * + ( ) | ?`, müssen mit einem `\` maskiert werden.
   `\.` steht also für das Zeichen `.`, für nicht druckbare Zeichen gibt es Ersatzsequenzen, z.B. für Zeilenvorschub `\n`, für Tabulator `\t`.
3. `.` steht für ein beliebiges Zeichen
4. `[`$z_1 z_2 z_3$`]` steht für eine Menge von Zeichen z_1, z_2, z_3 , hier dürfen auch Intervalle angegeben werden: `[`z_1`-`z_n`]`.
5. `[^`$z_1 z_2 z_3$`]` steht für die Menge aller Zeichen außer z_1, z_2 und z_3.
6. Wenn r_1 und r_2 reguläre Ausdrücke sind, dann auch $r_1 r_2$ (Folge, Sequenz, r_1 gefolgt von r_2)
7. Wenn r_1 und r_2 reguläre Ausdrücke sind, dann auch r_1 `|` r_2 (Alternativen, Auswahl, r_1 oder r_2)
8. r_1`*` steht für die 0,1,2,...-fache Wiederholung von r_1
9. r_1`+` steht für die 1,2,...-fache Wiederholung von r_1
10. r_1`?` steht für r_1 oder die leere Zeichenfolge.
11. `(`r_1`)` steht für r_1, Klammern stehen also zum Zusammenfassen von regulären Ausdrücken.

Beispiele:

`[a-zA-Z0-9]`
: steht für die Menge der Buchstaben und Ziffern.

`[^&]`
: steht für die Menge aller Zeichen außer `&`.

`[^0-9]`
: alles außer Ziffern.

`ab*`
: Eine Zeichenfolge, die mit `a` als 1. Zeichen gefolgt von beliebig vielen `b`'s.

`ab+`
: Eine Zeichenfolge, die mit `a` als 1. Zeichen gefolgt von beliebig vielen `b`'s aber mindestens einem `b`.

`(ab)*`
: Eine Zeichenfolge, die abwechselnd aus `a`'s und `b`'s besteht.

`abc|def`
: entweder `abcef` oder `abdef`.

`(abc)|(def)`
: entweder `abc` oder `def`.

`a?`
: ein `a` oder nichts.

Aufgabe:

Mit einer über eine Pipe verbundenen Kommandosequenz der folgenden Gestalt

```
ls -l <datei...> | egrep <muster>
```

werden Texte des Formates erzeugt und gefiltert

```
-rw-r--r--   1 root root      15785 2002-10-04 22:08 a2ps.cfg
drwxr-xr-x   4 root root        104 2004-12-09 18:10 acpi
-rw-r--r--   1 root root       2579 2004-10-02 02:54 aliases
drwxr-xr-x   2 root root         48 2004-10-04 17:24 aliases.d
-rw-r--r--   1 root root      12288 2005-01-03 16:23 aliases.db
crw-------   1 theo audio   14,   4 2004-10-02 10:38 audio
brw-r-----   1 root disk    29,   0 2004-10-02 10:38 aztcd
brw-r-----   1 root disk    29,   0 2004-10-02 10:38 aztcd0
prw-------   1 root root          0 2005-01-30 10:53 initctl
brw-r-----   1 root disk    30,   0 2004-10-02 10:38 lmscd
srw-rw-rw-   1 root root          0 2005-01-31 10:00 log
crw-------   1 root root    10, 140 2004-10-02 10:38 relay8
br--------   1 root root    31,  24 2003-10-02 10:38 rflash0
...
```

Geben Sie für die Suchanfragen auf der nächsten Seite Textmuster für das `egrep`–Kommando an, mit denen aus einer wie oben strukturierten Liste (nicht nur genau aus dieser Liste) bestimmte Einträge aus dem Dateisystem selektiert werden können.

Alle Verzeichnisse:

. . . . . .

Alle Dateisystemeinträge, die `theo` als Besitzer haben:

. . . . . .

Alle HTML-Quelldateien (die üblicherweise die Dateiendung `.html` besitzen):

. . . . . .

Alle Einträge, die in 2004 oder später das letzte Mal beschrieben worden sind:

. . . . . .

Alle Dateien, die zwischen 10:00 und 10:59 beschrieben worden sind:

. . . . . .

Alle vollständig schreibgeschützten Einträge:

. . . . . .

Alle einfachen Dateien, die leer sind:

. . . . . .

Alle einfachen Dateien, auf die es mehr als einen Hardlink gibt:

. . . . . .

Alle Einträge, die ein `?` im Namen haben

. . . . . .

Aufgabe 5:

Was ist ein *link* im UNIX Filesystem?

. . . . . .

. . . . . .

Wie erzeugt man einen *link*?

. . . . . .

. . . . . .

Wie löscht man einen *link*?

. . . . . .

. . . . . .

Wozu braucht man *links*?

. . . . . .

. . . . . .

Was ist ein symbolischer *link*?

. . . . . .

. . . . . .

Wozu braucht man symbolische *links*?

. . . . . .

. . . . . .