



Systeme I - WS 07/08 Übungsblatt 5 - Praxis

Aufgabe 1

Der Befehl *find* bietet die Möglichkeit, Dateien zu suchen und entsprechende Aufrufe mit den gefundenen Dateien zu verknüpfen. Zum Beispiel

```
find -name "*.pdf" -exec evince {} \;
```

zeigt nacheinander sämtliche PDF-Dateien (Endung *.pdf*) im aktuellen Verzeichnis und allen Unterverzeichnissen an. Dabei steht „{}“ für den aktuellen Datenamen und „\;“ beendet den für jedes gefundenes File ausgeführten Befehl (in diesem Fall *evince*).

Anderer Optionen beschränken die Suche nach Alter, nach Größe, nach Dateirechten etc. (siehe „*man find*“ und Abschnitt „2.2.9 Finden von Dateien“ des Linux-Skript auf der Wiki-Seite.

Entwerfen Sie *find*-Befehlszeilen, die folgendes leisten:

- Anzeige aller Dateien in Ihrem Home-Verzeichnis
- Anzeige aller Dateien im Verzeichnis */etc/*, deren Name mit einem „s“ beginnt
- Ausgehend vom aktuellen Verzeichnis: Entfernen aller leeren Dateien (Größe 0 Byte) mit Abfrage vor dem Löschvorgang.
 (Hinweis: Wechseln Sie in ein temporäres Verzeichnis, damit Sie nicht versehentlich Dateien löschen. Mit „*touch datei*“ können Sie eine neue, leere Datei erzeugen.)
- Was bewirkt folgende Befehlszeile:

```
find -type f -name ?*Bob*Dylan*.mp3? -exec xms -e {} \;
```

Aufgabe 2

Viele Befehle, Programme (oder allgemeiner Prozesse) lesen Eingaben, verarbeiten diese und erzeugen Ausgabedaten ohne weitere Interaktion mit dem Benutzer. Derartige Programme nennt man **Filter**. Durch **Pipes** (synchronisierende Zwischendateien) kann man die Eingabe eines Filters mit der Ausgabe eines anderen Filters verknüpfen und dadurch mit einfachen Mitteln relativ komplexe Anweisungen geben. Das Symbol für eine Pipe in der Shell ist ein senkrechter Strich „|“.

Unix/Linux stellt zu diesem Zweck vordefinierte Kanäle - das sind Dateizugriffsmechanismen - zur Verfügung: **stdin** Standardeingabe, **stdout** Standardausgabe, **stderr** Standardfehlerausgabe. Schematische Darstellung:

```
{ Eingabe -> } befehl1 | befehl 2 { -> Ausgabe }
```

(Die Inhalte der geschweiften Klammern { ... } dienen nur zur Verdeutlichung und sind keine korrekte Unix/Linux-Syntax)

Mehr zum Thema siehe Abschnitt „5.2 Kommandos, Pipes und Dateien“ des Linux-Skript auf der Wiki-Seite.

- Der Befehl **find -name "*.tex"** durchsucht das aktuelle Verzeichnis nach Dateien mit der Endung **.tex*; der Befehl **wc -l** zählt die Zeilen seiner Eingabe. Verknüpfen Sie diese beiden Befehle, um die Anzahl aller Dateien mit der Endung *.tex* zu erhalten.
- Erstellen Sie einen Ordner in Ihrem Verzeichnis mit dem Name „*Ausarbeitung*“. Suchen und kopieren Sie alle Dateien mit den Endungen **.pdf* und **.tex* in den vorherigen erstellten Ordner. Aber nur diejenigen, die in den letzten beiden Tagen verändert wurden.
- Erstellen Sie ein Tar-Archiv (siehe *man tar*) aus dem Ordner „*Ausarbeitung*“ und komprimieren Sie es mit dem Komprimierungstool *gzip*.
- (Optional)

Der Befehl *xargs* liest die Standard-Eingabe, erstellt daraus eine durch Leerzeichen getrennte Dateiliste und führt damit einen Befehl aus. Wenn kein Befehl angegeben wird, wird der Befehl *echo* ausgeführt. An den auszuführenden Befehl können wiederum Optionen übergeben werden (siehe *man xargs*).

Erzeugen Sie in einem Testordner verschiedene leere Dateien, sodass einige auch Leerzeichen im Dateinamen haben (z.B. (*touch 'datei name'*)). Führen Sie folgende Befehlszeile aus:

```
find ./ -size 0 -print | xargs rm
```

Was bewirkt diese Befehlszeile? Interpretieren Sie die Ausgabe und erweitern Sie den Befehl, sodass alle Dateien (auch diejenigen, die Leerzeichen im Dateinamen enthalten) gelöscht werden.

Abgabe: Vorführung nächste Woche in den Übungsgruppen.



Freiburg, 17. Januar 2008

Systeme I - WS 07/08 Übungsblatt 5 - Theorie

Aufgabe 1

Vier Prozesse (p_1, p_2, p_3, p_4) greifen auf fünf Ressourcenklassen (A, B, C, D, E) zu. Der Vektor verfügbarer Ressourcen ist $V = (6, 9, 9, 7, 9)$ und die Maximalanforderungsmatrix ist:

$$M = \begin{pmatrix} 3 & 7 & 5 & 2 & 4 \\ 2 & 1 & 8 & 5 & 7 \\ 4 & 2 & 4 & 1 & 3 \\ 3 & 6 & 2 & 4 & 3 \end{pmatrix}$$

Geben Sie für die folgenden Zustände mittels des Bankieralgorithmus an, ob sie sicher oder unsicher sind. Begründen Sie Ihre Aussage. Geben Sie für die nicht sicheren Zustände zusätzlich an, welche Prozesse an dem potentiellen Deadlock beteiligt sind.

a) Der Zustand Z_1 mit der folgenden Belegungsmatrix:

$$E = \begin{pmatrix} 1 & 3 & 2 & 2 & 0 \\ 1 & 0 & 1 & 0 & 4 \\ 2 & 2 & 1 & 0 & 1 \\ 0 & 2 & 2 & 4 & 2 \end{pmatrix}$$

b) Der Zustand Z_2 , der sich aus Zustand Z_1 ergibt, wenn p_1 eine weitere Ressource aus der Klasse B zugeteilt bekommt.

c) Der Zustand Z_3 , der sich aus Zustand Z_1 ergibt, wenn p_4 eine weitere Ressource aus der Klasse B zugeteilt bekommt.

Aufgabe 2

a) In der Vorlesung wurde das Buddy-System besprochen. Es sei nun $2^U = 16\text{MB}$ und $2^L = 1\text{KB}$. Was geschieht bei folgendem Ablauf:

1. Anforderung A: 512 KB
2. Anforderung B: 2 MB
3. Anforderung C: 512 KB
4. Freigabe A
5. Freigabe C
6. Anforderung D: 2 MB
7. Freigabe B

8. Freigabe D

- b) Kann es vorkommen, dass zwei benachbarte freie Speicherbereiche die gleiche Größe haben, aber nicht verschmolzen werden? Begründen Sie Ihre Aussage.
 - c) Angenommen der verfügbare Hauptspeicher habe die Größe 2^U und die Größe des kleinsten zuteilbaren Blocks sei 2^L mit $L < U$.
1. Geben Sie eine worst-case-Situation für die interne Fragmentierung an, bei der der meiste Verschnitt entsteht.
 2. Angenommen, es sind insgesamt 2^{U-1} Speicherzellen belegt (evtl. in verschiedenen Blöcken). Geben Sie unter dieser Voraussetzung eine Situation an, bei der die Größe des größten verfügbaren Blocks minimal ist.

Aufgabe 3

Bei der dynamischen Partitionierung haben Sie die Speicherzuteilungsalgorithmen *Best Fit*, *First Fit* und *Next Fit* kennengelernt.

Seien im Speicher folgende Blöcke frei (beachten Sie die Reihenfolge):

11 KB, 4 KB, 20 KB, 18 KB, 7 KB, 9 KB, 13 KB, 15 KB

Es gibt folgende Speicherauforderungen:

1. 12 KB
2. 10 KB
3. 9 KB

a) Welche der freien Blöcke werden durch die einzelnen Algorithmen ausgewählt? Geben Sie jeweils auch eine Liste der verbleibenden/neuen freien Blöcke an.

b) Bei der Strategie *Best Fit* haben Sie gelernt, dass diese zu sehr kleinen Speicherlöchern führt, die kaum noch verwendet werden können. Um diesen Effekt zu vermeiden kann man auch die Strategie *Worst Fit* verwenden. Sie wählt immer den größten freien Speicherblock. Wenden Sie *Worst Fit* auf das obige Szenario an. Geben Sie wieder die neue Liste der freien Blöcke an.

c) Welche Nachteile hat die Strategie *Worst Fit*?

Aufgabe 4

Überprüfen ob die folgenden Algorithmen der Klasse der Keller-Algorithmen angehören:

- Second Chance
- NFU
- Aging
- Working Set

Begründen Sie ihre Antwort ausführlich!

Abgabe: Freitag, 01.02.2008 vor der Vorlesung.