Aufgaben zur Vorlesung „Betriebssysteme“ – Unix

A. Praktische Aufgaben I

Hinweis. Vergessen Sie nicht, sich nach Ihren Sitzungen im FRZ korrekt abzumelden!

1. Diese Aufgabe soll Ihnen die Benutzung von Unix-Manpages mit Hilfe des Kommandos `man` vertraut machen. Veranschaulichen Sie sich bei c) bis e) Ihre Antworten durch eigene Beispiele!
   a) Wie kann man allgemein die Manpage zu einem Shell-Kommando aufrufen? Wie lautet speziell der Aufruf, um sich über Aufbau und Inhalt von Manpages zu informieren? Durch welche Eingabe wird die Anzeige einer Manpage beendet (und damit Rückkehr zur Shell möglich)?
   b) Die Unix-Manpages sind in Kapitel eingeteilt. Welche Kapitel gibt es und wie greift man darauf zu?
   c) Welche `man`-Option zeigt Kurzbeschreibungen von Manpages auf der Kommandozeile an?
   d) Welche Option von `man` durchsucht alle im System verfügbaren Manpages nach einem bestimmten Begriff?
   e) Benutzen Sie die Manpages der entsprechenden Kommandos, um die nachfolgenden Fragen zu beantworten.
   - Mit dem Kommando `ls` kann man sich den Inhalt eines Verzeichnisses anzeigen lassen.
     – `ls` zeigt standardmäßig versteckte Dateien (deren Namen mit einem Punkt beginnen) nicht an. Wie kann man dieses Verhalten ändern? Wie kann man noch zusätzlich alle Dateien im ausführlichen Format (d. h. mit Rechten, Größe usw.) anzeigen?
     – Wie kann man sich die Dateien der Größe nach geordnet anzeigen lassen?
     – Auf welche Weise kann man die Dateigröße in einem leichter lesbaren Format (mit Einheiten für Byte, KByte und MByte) anzeigen?
   - Welche Header-Dateien muß man einbinden, um den `open()`-Systemaufruf verwenden zu können? Was sind die gültigen Rückgabewerte von `open()`, welche Fehler können beim Aufruf von `open()` auftreten?
   - Mit dem Befehl `cd` kann man in der Shell das Verzeichnis wechseln. Welche zusätzlichen Optionen versteht `cd`?

2. Zum besseren Verständnis der nachfolgenden Aktivitäten sollten Sie sich den im Laufe der Aufgabe entstehenden Verzeichnisbaum aufzeichnen und sich vergegenwärtigen, in welchem Verzeichnis Sie sich jeweils befinden.
   a) Legen Sie in Ihrem home-Verzeichnis ein Verzeichnis `mydir` an. Wechseln Sie in dieses Verzeichnis und erzeugen Sie mittels `touch myfile` eine (leere) Datei `myfile`. Zeigen Sie die ausführlichen Informationen über diese Datei an.
   b) Kehren Sie in Ihr home-Verzeichnis zurück. Legen Sie ein weiteres Verzeichnis `yourdir` an und informieren Sie sich über den ausführlichen, ***vollständigen*** Inhalt Ihres home-Verzeichnisses.
   c) Erzeugen Sie in dem Verzeichnis `yourdir` einen Hardlink `yourfile`, der auf `myfile` zeigt. Zeigen Sie den ausführlichen Inhalt (ohne versteckte Dateien) der Verzeichnisse `mydir` und `yourdir` an. Worin unterscheiden sich diese Inhalte? Worin unterscheidet sich der jetzige Zustand von `myfile` gegenüber dem urspünglichen Zustand nach a)? Erkären Sie die Ergebnisse!
   d) Mit dem Kommando `cat` und den beiden Formen der Ausgabeumlenkung `>` und `>>` ist es möglich, einzelne Wörter direkt in eine Datei zu schreiben; die Wörter werden jeweils durch Return (Enter) getrennt, die gesamte Eingabe wird mit Strg+d beendet. Untersuchen Sie

die Wirkung der beiden Eingabeumlenkungen, indem Sie eine Datei `test` erzeugen und in diese Datei in der beschriebenen Weise mehrmals schreiben. Informieren Sie sich jedesmal über den Inhalt der Datei. Löschen Sie am Ende die Datei `test`.

e) Fügen Sie abwechselnd folgende Wörter in die Datei `myfile` und in die Datei `yourfile` ein: Thread, Process, Task, Memory, TLB. Was beobachten Sie beim Betrachten der Dateiinhalte?

f) Vergewissern Sie sich, in welchem Verzeichnis Sie sich befinden; wechseln Sie ggf. in das Verzeichnis `mydir`. Suchen Sie in der Datei `myfile` nach allen Zeilen, die mit dem Buchstaben T beginnen; sortieren Sie das Ergebnis absteigend und geben Sie die erste Zeile aus.

Hinweis. Das Kommando `grep` durchsucht Inhalte von Dateien, `sort` sortiert Zeilen, und das Kommando `head` beschränkt eine Liste auf eine Anzahl von Zeilen. Nutzen Sie Pipes zum Verknüpfen von Befehlen.

g) Welche beiden Möglichkeiten haben Sie, in einem Schritt (also durch ***ein***malige Anwendung von `cd`) in das Verzeichnis `yourdir` zu wechseln? Wählen Sie eine davon aus und überzeugen Sie sich vom Erfolg. Ändern Sie Ihr eigenes Zugriffsrecht zur Datei `yourfile` auf „nur-lesend-zugreifbar“. Versuchen Sie nun, in die Datei `myfile` zu schreiben. Erklären Sie den Effekt.

h) Legen Sie ein Verzeichnis `somedir` an und wechseln Sie in dieses Verzeichnis. Kopieren Sie die Datei `myfile` in dieses Verzeichnis. Überprüfen Sie den Inhalt der Datei `yourfile`.

i) Zeigen Sie den ausführlichen Inhalt des aktuellen Verzeichnisses an. Wieso hat der Link-Zähler der Datei `myfile` den Wert 1, obwohl es diese Datei in Ihrem Verzeichnisbaum zweimal gibt?

j) Löschen Sie die Datei `myfile` zunächst in `somedir` und anschließend in `mydir`. Untersuchen Sie jeweils die Eigenschaften von `yourfile` und erklären Sie die Veränderungen.

Zusatzaufgabe. Führen Sie die oben beschriebenen Schritte durch, aber verwenden Sie einen symbolischen Link (der gleichfalls durch `ln` erzeugt werden kann) statt eines Hardlink.

3. Vollziehen Sie die in der Vorlesung vorgeführten Beispiele zur Programmentwicklung und zu Grundlagen von Unix (Folien 8 bis 15) nach! Versuchen Sie, nach Möglichkeit alle damit verbundenen Vorgehensweisen, Informationen und Systemaktivitäten zu erklären!

   Hinweis. Die C-Programme finden Sie unter http://os.inf.tu-dresden.de/Studium/Bs/hello1.c usw. Die Dateien können Sie mit folgenden Befehlen in Ihrem aktuellen Verzeichnis ablegen:

```
$> wget http://os.inf.tu-dresden.de/Studium/Bs/hello1.c
$> wget http://os.inf.tu-dresden.de/Studium/Bs/hello2.c
$> wget http://os.inf.tu-dresden.de/Studium/Bs/hello3.c
```

4. Informieren Sie sich anhand der Manpages über die folgenden Systemaufrufe:
   `fork, execve, exit, wait, waitpid`.
   Notieren Sie sich im Hinblick auf die theoretischen Übungen (Aufg. 7, 8, 9) Format, Wirkungsweise und wichtige Eigenschaften dieser Systemaufrufe!

B. Theoretische Aufgaben

1. Erläutern Sie die einzelnen Schritte des Übergangs von einem (gedanklich oder schriftlich vorliegenden) Algorithmus zu einem lauffähigen Programm! Geben Sie dazu jeweils folgende Informationen an:

– Schritt bzw. ausführendes Programm
– Programmname in Unix
– Ergebnis (Bezeichnung und Art [Textdatei,...] der erzeugten Datei, Beschreibung ihres Inhalts)
– Dateisuffix unter Unix.

2. Ordnen Sie folgende Funktionen bzw. Programme in die in der Vorlesung eingeführte Unix-Übersicht ein (Mehrfachzuordnungen möglich)! Begründen Sie Ihre Zuordnung!

   a) `open()`: Öffnen eines Files
   b) `sort`: Sortierte Ausgabe von eingegebenen Daten
   c) `memcpy()`: Kopieren von Daten
   d) `copyin()`: Kopieren von Daten aus dem Nutzeradreßraum in den Kernadreßraum
   e) `qsort()`: Sortieren eines Feldes
   f) `set_pte()`: Eintragen einer Seite in die Pagetable und damit Eintragen einer Seite in einen Adreßraum (Adreßräume sind eines der wesentlichen Schutzkonzepte des Betriebssystems)
   g) `gcc`: Compilieren eines Programms

   Innerhalb welcher Bereiche können die folgenden Intel-Maschinenbefehle auftreten?

   g) `movl $3, %eax`: Laden des general purpose register `eax` mit 3
   h) `movl %eax, %cr3`: Laden des Registers `cr3` und damit Umschalten auf einen anderen Adreßraum.

	Kernel	Standard Library	Utility Program
`open()`			
`sort`			
`memcpy()`			
`copyin()`			
`qsort()`			
`set_pte()`			
`gcc`			
`movl $3, %eax`			
`movl %eax, %cr3`			

3. In der Vorlesung wurde die Adreßraumstruktur von Unix eingeführt. Ordnen Sie für das links stehende Beispiel die in der Tabelle enthaltenen Symbole demgemäß ein (Begründungen!)!

```
#include <stdlib.h>
int a[20];
int x=1;
void foo(void) {
   int b[20];
   void *p = malloc(100);
   free(p);
}
```

	Text	Data	BSS	Stack
`a`				
`x`				
`foo`				
`b`				
`p`				
`*p`				

4. In Unix hat jedes Programm standardmäßig drei Dateien offen: stdin, stdout und stderr. Wozu braucht man einen separaten Kanal für Fehlermeldungen (stderr), wenn man bereits stdout hat?

5. Welche Voraussetzungen muß ein Programm erfüllen, das in einer Kommandosequenz wie `ls|grep ps|sort` verwendet werden soll, oder anders gefragt, warum kann das Beispiel `latex foo.tex|dvips|lpr` so nicht funktionieren? Informieren Sie sich zuvor in der man-page über die Funktionalität der auftretenden Kommandos!

6. In der Vorlesung wurde der Systemruf `unlink()` zum Löschen einer Datei erwähnt. Was geschieht beim Ausführen der folgenden Codesequenz?

```
int fd = creat("/tmp/test", S_IRWXU);
unlink("/tmp/test");
write(fd, "Hello world\n", 12);
close(fd);
```

   Wann wird das File gelöscht (und ist damit für andere nicht mehr sichtbar)? Kann `write()` erfolgreich abgeschlossen werden, obwohl die Datei gelöscht wurde? Was geschieht bei `close()`?

7. Unix-Programme werden direkt oder indirekt mit `exit(result)` beendet, wobei ein Resultat übergeben werden kann. Was geschieht mit diesem Resultat und wie kann man es abfragen?

8. a) Erklären Sie, warum nach `fork()` *parent* und *child* zum einen an der gleichen Stelle des Programms fortgesetzt werden, zum anderen dann aber unterschiedliche Wege gehen können!

   b) Welche Ressourcen werden von *parent* und *child* geteilt?

   c) Kann das folgenden Programmstück zu verschiedenen Ergebnissen führen? Begründen Sie Ihre Antwort und nennen Sie das Ergebnis bzw. einige Ergebnisse. Dabei bewirkt `puts` die unformatierte Ausgabe der als Parameter angegebenen Zeichenkette.

```
1.       pid = fork();
2.       if (pid < 0) {
3.          perror("Error during fork()");
4.          exit(1);
5.       }
6.       if (pid) {
7.          for (i=0; i<4; i++)
8.             puts("parent ");
9.       }
10.      else {
11.         for (i=0; i<4; i++)
12.            puts("child ");
13.      }
```

   d) Wieso kann `i` in beiden Prozessen benutzt werden, ohne daß sich die beiden Prozesse beeinflussen?

9. Entwerfen Sie ein Bild, das den Ablauf beim Starten eines Programms darstellt und die Rolle von `fork()`, `execve()`, `wait()`, `exit()` veranschaulicht! Erläutern Sie dabei auch den Prozeßstatus *zombie*, in dem der Prozeß nur noch einige hundert Bytes an Ressourcen belegt und ansonsten nichts mehr tut. Wie kommt ein Prozeß in diesen Status und was kann man tun, um ihn aus diesem Status herauszuholen?

10. Welche Reaktionsmöglichkeiten auf Signale gibt es in Unix? Was muß man tun, um von der Standardreaktion auf eine andere Reaktion zu wechseln? Gibt es Signale, für die keine Abweichung von der Standardreaktion möglich ist? Wenn ja, welche?

C. Praktische Aufgaben II

1. Das unten angehängte `tar`-File enthält den Rahmen für eine Shell, die eine recht einfache Kommandosyntax der folgenden Form hat:

   ```
   command [arg1 ... arg9] [< input_redirect] [> output_redirect]
   [| command2 ... ] [&].
   ```

   Die Shell liest Eingaben und zerlegt sie in ihre Bestandteile. Am Ende der Eingabe ruft sie eine Funktion `execute_command_line()` auf, die die Ausführung der eingegebenen Kommandos veranlassen soll.

   a) Schreiben Sie eine einfache Version von `execute_command_line()`, die lediglich das erste Kommando ausführt und dabei das in Aufgabe 10 entworfene Ablaufschema mit `fork()`, `execve()` und `wait()` umsetzt!

   b*) Ergänzen Sie die Funktion derart, daß eine eingegebene Umleitung der Ein- oder Ausgabe durchgesetzt wird. Sehen Sie sich dazu die Beschreibung der Funktionen `open()`, `dup2()` und `close()` an!

   c*) Ergänzen Sie die Funktion so, daß eine Pipe-Sequenz der Art `ls|more` korrekt ausgeführt wird. Studieren Sie dazu den `pipe()`-Systemruf an und überlegen Sie, wie mit seiner Hilfe die Ausgabe eines Prozesses zur Eingabe eines anderen Prozesses werden kann!

   Die mit `*` versehenen Aufgaben haben einen höheren Schwierigkeitsgrad.

2. Das Unix-Programm `find` sucht Dateien, die bestimmte Bedingungen erfüllen. Implementieren Sie ein kleines `find`-Programm, das folgende Syntax akzeptiert:

   ```
   find -name <name> -type <f|d>.
   ```

   Sind die Bedingungen erfüllt, wird der Name des Files ausgedruckt. Sehen Sie sich dazu die Funktionen der Familie `readdir()` und `stat()` an.

Material

Software:

`tar-File` für den Rahmen der Shell, entpacken mit `tar xzf seminar.tgz`, übersetzen mit `make` auf einer Linux Maschine oder einer der FRZ Unix-Maschinen.

Literatur:

- W.R. Stevens: Advanced Programming in the UNIX Environment
- A. Tanenbaum: Modern Operating Systems