TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Professur für Betriebssysteme

Betriebssysteme und Sicherheit, WS 2018/19

1. Aufgabenblatt – Unix

Geplante Bearbeitungszeit: drei Wochen

Teil A Grundlegende praktische Aufgaben

Der praktische Teil der Übung soll allen Studenten, die bisher noch keine Erfahrungen mit Unix-artigen Systemen – wie dem im Rechenzentrum bereitgestellten „Ubuntu" – gesammelt haben, den Umgang mit einem solchen näher bringen. Die Beantwortung der konkreten Fragen steht dabei nicht im Vordergrund, sie sind vielmehr als Anregung zu verstehen sich mit den typischen Arbeitsweisen in einem Unix-System vertraut zu machen. Insofern ist es nicht zielführend die Fragen allein mittels Recherche im Internet zu beantworten.

Aufgabe 1.1 Diese Aufgabe macht Sie mit der Benutzung von Unix-Hilfeseiten („Manpage", kurz für „manual page") mit Hilfe des Kommandos `man` vertraut. Veranschaulichen Sie sich bei c) bis e) Ihre Antworten durch eigene Beispiele. Zum Nachvollziehen der Kommandozeilen-Aufrufe können Sie u. a. auch `http://explainshell.com` nutzen.

(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 die Rückkehr zur Shell möglich)?

(b) Die Unix-Manpages sind in Abschnitte eingeteilt. Welche Abschnitte 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 die Kurzbeschreibungen aller 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 lassen sich 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 müssen eingebunden werden, um den `open()`-Systemaufruf verwenden zu können? Was sind mögliche Rückgabewerte von `open()` und welche Fehler können beim Aufruf von `open()` auftreten?
- Mit dem Befehl `cd` kann man das aktuelle Arbeitsverzeichnis wechseln. Warum muss `cd` durch die Shell selbst bereitgestellt werden und kann nicht als separates Programm implementiert sein? Welche zusätzlichen Optionen unterstützt `cd`?

Aufgabe 1.2 Zum besseren Verständnis der nachfolgenden Aktivitäten können 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 namens `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.

Hinweis: Das Kommando `mkdir` erzeugt ein neues Verzeichnis, `cd` ermöglicht den Wechsel des Arbeitsverzeichnisses und die Kommandos `ls` und `stat` zeigen Datei-/Verzeichnis-Informationen an.

(b) Kehren Sie in Ihr Home-Verzeichnis zurück. Legen Sie ein weiteres Verzeichnis mit dem Namen `yourdir` an und informieren Sie sich über den ausführlichen, *vollständigen* Inhalt Ihres Home-Verzeichnisses.

(c) Erzeugen Sie in dem Verzeichnis `yourdir` mit Hilfe des Kommandos `ln` einen Hardlink `yourfile`, der auf `myfile` zeigt. Zeigen Sie den ausführlichen Inhalt der Verzeichnisse `mydir` und `yourdir` an. Worin unterscheiden sich diese Inhalte? Worin unterscheidet sich der jetzige Zustand von `myfile` gegenüber dem ursprünglichen Zustand nach a)? Erklären 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 einen Zeilenumbruch getrennt, die gesamte Eingabe wird mit Strg+d beendet. Untersuchen Sie die Wirkung der beiden Eingabeumlenkungen, indem Sie im Verzeichnis `mydir` eine Datei `test` erzeugen und in diese Datei in der beschriebenen Weise mehrmals schreiben. Informieren Sie sich jedes Mal über den Inhalt der Datei (z. B. mit dem Kommando `cat`). Löschen Sie am Ende die Datei `test` (Kommando: `rm`).

(e) Fügen Sie folgende Wörter abwechselnd in die Dateien `myfile` und `yourfile` ein: thread, process, task, memory, TLB. Was beobachten Sie beim Betrachten der Inhalte beider Dateien?

(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 den Buchstaben `t` enthalten; sortieren Sie das Ergebnis alphabetisch 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 *einmalige* 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 für die Datei `yourfile` auf „nur-lesend-zugreifbar“. Versuchen Sie nun, in die Datei `myfile` zu schreiben. Erklären Sie den Effekt.

Hinweis: Zur Änderung von Zugriffsrechten kann das Kommando `chmod` verwendet werden.

(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 Hardlinks.

Aufgabe 1.3 Vollziehen Sie die in der Vorlesung vorgeführten Beispiele zur Programmentwicklung und zu Grundlagen von Unix nach. Versuchen Sie nach Möglichkeit alle damit verbundenen Vorgehensweisen, Informationen und Systemaktivitäten zu erklären.

Hinweis: Die C-Programme stehen im Internet bereit und können mit den folgenden Befehlen in Ihr aktuelles Verzeichnis geladen werden:

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

Aufgabe 1.4 Informieren Sie sich anhand der Manpages über die folgenden Systemaufrufe:

`fork()`, `execve()`, `exit()`, `wait()`, `waitpid()`

Machen Sie sich im Hinblick auf die theoretischen Übungen (Aufg. 11, 12, 13) mit Format, Wirkungsweise und wichtigen Eigenschaften dieser Systemaufrufe vertraut.

TEIL B THEORETISCHE AUFGABEN

Aufgabe 1.5 Erläutern Sie die einzelnen Schritte des Übergangs von einem (gedanklich oder schriftlich vorliegenden) Algorithmus über die Implementierung in C bis hin zu einem lauffähigen Programm. Geben Sie dazu jeweils folgende Informationen an:

- Schritt bzw. ausführendes Programm
- Programmname in Unix
- Ergebnis (Bezeichnung und Art der erzeugten Datei, Beschreibung ihres Inhalts)
- Dateiendung unter Unix

Aufgabe 1.6 In der Vorlesung wurde die Adressraumstruktur von Unix eingeführt. Ordnen Sie für das links stehende Beispiel die in der Tabelle enthaltenen Symbole demgemäß ein.

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

	Text	Data	BSS	Heap	Stack
a	☐	☐	☐	☐	☐
x	☐	☐	☐	☐	☐
foo	☐	☐	☐	☐	☐
b	☐	☐	☐	☐	☐
p	☐	☐	☐	☐	☐
*p	☐	☐	☐	☐	☐

Aufgabe 1.7 Welche der folgenden Funktionen müssen im Kern, welche können im Nutzermodus implementiert werden (Mehrfachzuordnungen möglich). Begründen Sie Ihre Entscheidung.

	(a)	(b)	(c)	(d)	(e)	(f)	(g)
Kern	☐	☐	☐	☐	☐	☐	☐
Nutzer	☐	☐	☐	☐	☐	☐	☐

(a) `qsort()` – Sortieren eines Feldes

(b) `memcpy()` – Kopieren von Daten

(c) `open()` – Öffnen einer Datei

(d) `copyin()` – Kopieren von Daten aus dem Nutzeradressraum in den Kernadressraum

(e) `set_pte()` – Eintragen einer Seite in die Seitentabelle und damit Einblenden von Speicher in einen Adressraum (Adressräume sind eines der wesentlichen Schutzkonzepte des Betriebssystems)

Innerhalb welcher Bereiche treten die folgenden Intel-Maschinenbefehle normalerweise auf?

(f) `movl $3, %eax` – Laden des frei verwendbaren Registers („general purpose register“) `eax` mit 3

(g) `movl %eax, %cr3` – Laden des Spezialregisters `cr3` und damit Umschalten auf einen anderen Adressraum

Aufgabe 1.8 In Unix hat jedes Programm standardmäßig drei Dateien geöffnet: `stdin`, `stdout` und `stderr`. Weshalb ist ein separater Kanal für Fehlermeldungen (`stderr`) sinnvoll, wenn doch bereits `stdout` existiert?

Aufgabe 1.9 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 mit Hilfe der Manpages über die Funktionalität der auftretenden Kommandos.

Aufgabe 1.10 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 die Datei `test` gelöscht und ist damit für andere nicht mehr sichtbar? Kann `write()` erfolgreich abgeschlossen werden? Was geschieht beim Aufruf von `close()`?

Aufgabe 1.11 C-Programme werden unter Unix – direkt oder indirekt – mit `exit(result)` beendet, wobei ein Resultat übergeben werden kann. Was geschieht mit diesem Resultat und wie kann man es abfragen?

Aufgabe 1.12 Entwerfen Sie ein Bild, das den Ablauf beim Starten eines Programms durch die Shell 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?

Aufgabe 1.13

(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* gemeinsam genutzt?

(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 zeilenweise Ausgabe der als Parameter angegebenen Zeichenkette.

```
int i, pid = fork();
if (pid < 0) {
  perror("Error␣during␣fork()");
  exit(1);
}
if (pid) {
  for (i = 0; i < 4; i++)
    puts("parent");
} else {
  for (i = 0; i < 4; i++)
    puts("child");
}
```

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

Aufgabe 1.14 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?

Teil C Vertiefende praktische Aufgaben

Dieser Teil der Übungsaufgaben zur individuellen Bearbeitung durch fortgeschrittene Studenten gedacht und wird nicht in den Übungsgruppen behandelt. Die Übungsleiter beantworten aber natürlich gern Fragen zu den Aufgaben.

Aufgabe 1.15 Das bereitgestellte tar-Archiv[1] (entpacken mit `tar xzf seminar.tar.gz`, übersetzen mit `make`) 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 13 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.

Aufgabe 1.16 Das Unix-Programm `find` sucht Dateien, die bestimmte Bedingungen erfüllen. Implementieren Sie ein kleines `find`-Programm, das folgende Syntax akzeptiert:

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

Sind die Bedingungen erfüllt, wird der Name der Datei bzw. des Verzeichnisses ausgegeben. Sehen Sie sich dazu die Funktionen der Familie `readdir()` und `stat()` an.

[1] `https://os.inf.tu-dresden.de/Studium/Bs/uebung/seminar.tar.gz`

Klausuraufgabe I

In einem Unix-ähnlichen System existieren im Verzeichnis `/tmp` die ausführbaren Dateien `Alpha` und `Omega`. Den beiden Programmen liegt folgender C-Code zugrunde:

Alpha

```
1  int main() {
2     int pid;
3     pid = fork();
4
5     if (pid < 0) { printf("F"); }
6     else {
7        if (pid == 0) { exec("/tmp/Omega"); }
8        else          { unlink("/tmp/Omega"); }
9     }
10    printf("A");
11    return 0;
12 }
```

Omega

```
1  int main() {
2     printf("M");
3     return 0;
4  }
```

a) Nennen Sie zwei mögliche Ursachen dafür, dass die Funktion `exec` fehlschlägt.

b) Das Programm `Alpha` wird nun ausgeführt. Der aktuelle Benutzer verfügt dabei über alle erforderlichen Rechte an den beteiligten Dateien und Verzeichnissen. Skizzieren Sie kurz *alle* möglichen Programmabläufe und geben Sie jeweils an, welche Ausgaben dabei auf der Konsole erscheinen.
HINWEIS: Beachten Sie, dass Systemaufrufe fehlschlagen können.

c) Welche Auswirkungen hat es, wenn innerhalb des Quelltextes von `Alpha` in den Zeilen 7 und 8 „Omega“ durch „Alpha“ ersetzt wird?

Klausuraufgabe II

Betrachtet werde das folgende Programm:

```
1  int fd1, fd2, pid1, pid2;
2
3  fd1  = creat("foo.txt");
4  pid1 = fork();
5
6  if (pid1 == 0) {
7     fd2  = creat("bar.txt");
8     write(fd1, "Lorem", 5);
9
10    pid2 = fork();
11    if (pid2 == 0) {
12       write(fd2, "Ipsum", 5);
13       wait();
14    }
15    exit();
16 }
17
18 printf("X");
19 wait();
20 exit();
```

HINWEIS: Die Funktion…

`creat` legt die angegebene Datei an und öffnet sie. Sollte die Datei bereits existieren, wird ihr Inhalt beim Öffnen vollständig gelöscht.

`wait` blockiert den aufrufenden Prozess so lange bis sich ein beliebiger Kindprozess beendet. Existiert kein solcher, so kehrt die Funktion sofort erfolgreich zurück.

`exit` beendet den aufrufenden Prozess sofort.

Ein Nutzer führt das Programm nun aus. Alle auftretenden Funktionsaufrufe sind dabei erfolgreich.

a) Was wurde auf der Konsole ausgegeben unmittelbar nachdem sich der Hauptprozess beendet hat? Skizzieren Sie kurz wie es zu dieser Ausgabe kommt. Sind auch andere Ausgaben möglich? Falls ja, geben Sie die Ursache dafür an und nennen Sie eine zweite mögliche Ausgabe!

b) Welchen Inhalt haben die beiden Dateien `foo.txt` und `bar.txt` unmittelbar nachdem sich der Hauptprozess beendet hat? Gibt es mehrere Möglichkeiten, so geben Sie die Ursache dafür an und nennen Sie einen zweiten möglichen Dateiinhalt!

c) Durch das Entfernen einer einzelnen Zeile des obigen Codes lässt sich erreichen, dass beim Beenden des Hauptprozesses *immer genau drei* `X` ausgegeben wurden. Um welche Zeile (Angabe der Zeilennummer genügt) handelt es sich? Erläutern Sie kurz wieso damit das beschriebene Ergebnis erreicht wird!

d) Welches Ergebnis hätte ein Aufruf von `read(fd1, buf, 10)` – also das Einlesen von bis zu 10 Byte aus `fd1` in den (zuvor angelegten) Puffer `buf` – unmittelbar vor Zeile 20?