htw saar

Studiengang Kommunikationsinformatik
Prof. Dr.–Ing. Damian Weber
Dipl.-Inf. Marion Bohr
Thorsten Jakobs, M.Sc.

Systemmanagement und Sicherheit

3. Übung

Aufgabe 1 (File open/read/close)

a) Auf der Systemmanagement-Website finden Sie einen Link zu einer Bilddatei. Laden Sie die Datei herunter.

b) Stellen Sie mit dem `strings` Kommando fest, mit welcher Art von Kamera zu welchem Zeitpunkt das Bild aufgenommen wurde.

   Lassen Sie sich die Offsets der Strings im Hexadezimalsystem ausgeben.

   Überprüfen Sie die Offsets mit dem Kommando `hexdump` (mit Option –C).

c) Schreiben Sie ein *C*–Programm, das den Namen der Bilddatei in der Kommandozeile erwartet und diese Offsets benutzt, um mit Hilfe von

   - open()
   - lseek()
   - read()
   - close()

   das Aufnahmedatum und Infos über die Kamera auszugeben.

   Sie können in Ihrem Programm hexadezimale Offsets in der Schreibweise `0x...` angeben.

d) Jeder Returncode dieser Systemcalls muss auf einen Fehlerfall abgefragt werden. Fügen Sie für den Fehlerfall auch eine Ausgabe mit `perror()` ein.

   Provozieren Sie beim Testen für die Systemcalls `open()` und `lseek()` einen möglichen Fehlerfall.

   Ihr Programm braucht nur für dieses Bild zu funktionieren.

e) Funktioniert Ihr Programm auch für andere Bilder? Testen Sie Ihr Programm anhand einiger weiterer Bilder aus dem Internet. Sie können Ihre Ergebnisse mit dem Kommando `jhead` überprüfen.

f) Starten Sie Ihr Programm auch mittels `ktrace` und finden Sie im `kdump`–Output Ihre programmierten Systemcalls.

Aufgabe 2 (Pipes in der Kommandozeile)

Wir untersuchen Filterprogramme, diese sind geeignet für Pipe–Operationen. Ein Programm ist ein Filterprogramm, wenn es eine Eingabe von Standardeingabe liest und die

Ausgabe auf Standardausgabe schreibt. In dieser Aufgabe werden verschiedene Filter eingeübt.

Ein Stream–Editor kann Dateien automatisiert editieren. Er liest einen Datenstrom von Standardeingabe, verändert ihn und schreibt den veränderten Datenstrom auf Standardausgabe.

Wir experimentieren hierzu mit einer Datei zur Fußball-Bundesliga Tabelle aus Wikipedia.

Diese laden Sie über die URL

```
https://de.wikipedia.org/wiki/Fu%C3%9Fball-Bundesliga_2017/18
```

und speichern sie als `fussball-tabelle.html`

Stream–Editor: sed

um alle `<th>` Tags in `<td>` Tags zu verwandeln geben Sie ein (eine Zeile)

```
sed -e "s:<th\>:<td\>:g" -e "s:</th>:</td>:g"
       <fussball-tabelle.html >fussball-tabelle2.html
```

Differenzen von Dateien: diff

Überprüfen können Sie die Ersetzungen mit dem `diff` Kommando.

Geben Sie folgendes ein:

```
diff -u fussball-tabelle.html fussball-tabelle2.html
```

Die Differenz wird als entfernte(–)/hinzugefügte(+) Zeile angezeigt.

Anzahl Zeilen, Dateianfang, Dateiende: wc, head, tail

Finden Sie mit `grep -n` heraus, in welcher Zeile die Tabelle anfängt. Nutzen Sie `head`, `tail` und ggfs `wc`, um den Tabelleninhalt der Bundesligatabelle in eine eigene Datei umzuleiten.

Suchen von Mustern: grep

Mit `fgrep` können Sie die Zeilen einer Eingabe ausgeben, die ein bestimmtes Muster enthalten.

Geben Sie beispielsweise

```
grep "a href=" fussball-tabelle.html >links
```

ein und schauen sich die Datei `links` an.

Jetzt koppeln wir zwei Filterprogramme, nämlich `grep` und `sed`. Versuchen Sie durch Verwendung des Pipe–Symbols `|` die Ausgabe des vorigen `grep` mit der Eingabe von `sed`, sodass nur noch die Namen der Fußballvereine und ihre Platzierung sichtbar sind. Das Suchmuster für `sed` ist ein regulärer Ausdruck. Beispielsweise können Sie

```
.* für eine beliebige Zeichenkette
[abc]* für eine beliebige aus a b c bestehende Zeichenkette
[a-z]* für eine beliebige aus Kleinbuchstaben bestehende Zeichenkette
... (weiteres z.B.unter
https://www.gnu.org/software/sed/manual/html_node/Regular-Expressions.html)
```

Download–Tool: curl

Laden Sie mit

```
curl https://en.wikipedia.org/wiki/List_of_sovereign_states >countries
```

eine Liste der Staaten der Erde als HTML–Code herunter.

Erhalten Sie mit Hilfe von `grep` und `sed` eine Liste der Staaten der Erde aus diesem HTML–Code.

Aufgabe 3 (Pipes in C)

Der Autor des folgenden Programms, das Pipes implementiert, hat die Kommentare vergessen. Testen Sie dieses Programm und ergänzen Sie in den Zeilen startend mit `pipe(fd)` fehlende Kommentare. Mit Hilfe Ihrer Kommentare sollte ein Student des 2. Semesters verstehen, was in diesem Programm vor sich geht.

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h> /* for printf */
#include <string.h> /* for strlen */

int main(int argc, char **argv)
{
        int n;
        int fd[2];
        char buf[1025];
        char *data = "hello... this is sample data";

        pipe(fd);
        write(fd[1], data, strlen(data));
        if ((n = read(fd[0], buf, 1024)) >= 0)
        {
                buf[n] = 0; /* terminate the string */
                printf("read %d bytes from the pipe: \"%s\"\n", n, buf);
        }
        else
                perror("read");
        exit(0);
}
```