Prof. Dr. Franz Schweiggert Institut für Angewandte Informationsverarbeitung 2. November 2008
Matthias Röhm Blatt 3

Allgemeine Informatik III / Systemnahe Software I (WS 2008)

Abgabetermin: 11.11.2008

1 Aufgabe 1 (5 Punkte)

Bit-Operationen sind für die hardwarenahe Programmierung sehr wichtig. Viele Eigenschaften und Einstellungen erreicht man nur durch gezielte Manipulation von Bits. So werden einige Eigenschaften einer Datei in 16 Bit kodiert. Abbildung 1 zeigt die Eigenschaften und die zugehörigen Bits. In der Philosphie von UNIX ist prinzipiell alles eine Datei, sodass die obersten vier Bits den Dateityp festlegen. Im folgenden wollen wir uns auf die ersten 9 Bits, welche die Berechtigungen definieren, beschränken. Für jede Datei (reguläre Datei, Verzeichnis, ...) lassen sich drei Berechtigungen setzen:

Dateityp							Zugriffsrechte								
Reguläre Datei, Socket, Sym. Link	Verzeichnis, Socket, Blockorientierte Gerätedatei	Zeichen-, Blockorientierte Gerätedatei, Sym. Link	FIFO (benannte Pipe)				Leserecht, Dateieigentümer	Schreibrecht, Dateieigentümer	Ausführungsrecht, Dateieigentümer	Leserecht, Eigentümer-Gruppe	Schreibrecht, Eigentümer-Gruppe	Ausführungsrecht, Eigentümer-Gruppe	Leserecht, Alle anderen Benutzer	Schreibrecht, Alle anderen Benutzer	Ausführungsrecht, Alle anderen Benutzer
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Abbildung 1: Einige Eigenschaften einer Datei unter UNIX.

1. Lesen
2. Schreiben
3. Ausführen

Diese Rechte können einzeln für den Eigentümer, die Eigentümer-Gruppe und alle Anderen gesetzt werden. Sind beispielsweise die Bits 9, 8, 6 und 3 gesetzt bedeutet dies, dass der Eigentümer die Datei lesen und beschreiben darf. Angehörige der Eigentümer-Gruppe und alle anderen dürfen die Datei nur lesen.

Aufgabe
Schreiben sie ein Programm, welches einen 16 Bit integer (*short int S*) von der Standardeingabe einliest. Geben sie nun die Berechtigungen aus, die durch die ersten 9 Bits dieser Zahl repräsentiert werden. Anschließend soll ihr Program überprüfen, ob der Eigentümer Lese- und Ausführungsrechte besitzt. Falls nicht, setzen sie die entsprechenden Berechtigungen.

Beispiel 1
Wurde $S = 489$ eingelesen sind die ersten 9 Bits wie folgt gesetzt: 111101001
Somit kann der Eigentümer lesen, schreiben und ausführen. Die Eigentümer-Gruppe kann lesen und ausführen. Alle anderen können die Datei nur ausführen.

Beispiel 2
Wurde $S = 201$ eingelesen sind die ersten 9 Bits wie folgt gesetzt: 011001001
Somit kann der Eigentümer nur schreiben und ausführen. Es muss also das 9. Bit gesetzt werden, sodass sich 111001001 ergibt.

2 Aufgabe 2 (5 Punkte)

Wollen wir wichtige Informationen versenden, können wir diese verschlüsseln. Doch wer vermag schon zu sagen, wie sicher ein Verschlüsselungsverfahren ist - nicht nur aktuell, sondern in aller Zukunft? Da bietet es sich an, eine (verschlüsselte) Botschaft so zu übertragen, dass ein potentieller Angreifer überhaupt nicht auf die Idee kommt, dass es da etwas zum Angreifen gäbe. Dies ist das Gebiet der Steganographie (etwa: „verborgenes Schreiben"). Zu den klassischen Beispielen zählen die Zitronentinte, die erst durch Erwärmen des Blattes wieder sichtbar wird. Glücklicherweise stehen uns heute einfachere Methoden zur Verfügung. Hier wollen wir nun einen Text in Zahlen umwandeln. Diese können dann als Tabellenkalkulation oder ähnliches getarnt versendet werden.

2.1 Tarnen

Schreiben sie ein Programm, welches einen ASCII-Text Zeichenweise von der Standardeingabe entgegennimmt und wie folgt tarnt. Immer vier Zeichen werden in einen *long int* gepackt. Diese werden dann, als Zahl, durch Leerzeichen getrennt ausgegeben.

Beispiel 1
Der Text „test" soll getarnt werden.

```
t (01110100) hinzufuegen:
00000000 00000000 00000000 01110100
e (01100101) hinzufuegen:
00000000 00000000 01110100 01100101
s (01110011) hinzufuegen:
00000000 01110100 01100101 01110011
```

```
t (01110100) hinzufuegen:
01110100 01100101 01110011 01110100

Ausgabe:
1952805748
```

Beispiel 2
Der Text „Ich bin“ soll getarnt werden.

```
I (01001001) hinzufuegen:
00000000 00000000 00000000 01001001
c (01100011) hinzufuegen:
00000000 00000000 01001001 01100011
h (01101000) hinzufuegen:
00000000 01001001 01100011 01101000
Leerzeichen (00100000) hinzufuegen:
01001001 01100011 01101000 00100000

Ausgabe:
1231251488

b (01100010) hinzufuegen:
00000000 00000000 00000000 01100010
i (01101001) hinzufuegen:
00000000 00000000 01100010 01101001
n (01101110) hinzufuegen:
00000000 01100010 01101001 01101110

Ausgabe:
1231251488 6449518
```

2.2 Enttarnen

Schreiben sie ein Programm, welches einen oder mehrere *long int* (getrennt durch Leerzeichen) von der Standardeingabe entgegennimmt und enttarnt.
Beispiel 3
Die Zahl 1952805748 soll enttarnt werden.

```
Aus 1952805748:
01110100 01100101 01110011 01110100

t (01110100) auslesen.
00000000 01110100 01100101 01110011

e (01100101) auslesen.
00000000 00000000 01110100 01100101

s (01110011) auslesen.
00000000 00000000 00000000 01110100

t (01110100) auslesen.
00000000 00000000 00000000 00000000

Ausgabe:
test
```

Hinweise
Der Inhalt von Dateien kann an die Standardeingabe mit dem „<“-Operator umgelenkt werden. Mit

```
./a.out < x.txt
```

erhält das Program den Inhalt (den zu tarnenden Text oder die zu enttarnenden Zahlen) von *x.txt* auf der Standardeingabe. Ebenso lässt sich die Standardausgabe in eine Datei mit dem „>“-Operator umleiten. Mit

```
./a.out > y.txt
```

schreibt das Program seine Standardausgabe in die Datei *y.txt*.

In den Übungen werden die Funktion *print_bin_short* und *print_bin_long* vorgestellt, welche die Ausgabe eines *short int* bzw. *long int* in Binärdarstellung ermöglichen.

```
print_bin_short(3);
Ausgabe:
00000000 00000011

print_bin_long(1952805748);
Ausgabe:
01110100 01100101 01110011 01110100
```

Viel Erfolg!