Technische Informatik 2
Tutor: Marc Hildebrandt

WS 2016/17

C/06

Niklas Koenen
Jan Klüver
Vincent Jankovic

Übungsblatt 2

1	2	3	4	5	$\sum$

Lösungsvorschlag
Abgabe: 14.11.2016

Aufgabe 1

In dieser Aufgabe sollten wir ein Shellscript **mp3-rename.sh** schreiben, welches MP3-Dateien übergeben bekommt und diese dann in die Form *Kuenstler-Album-Titel-NN.mp3* umbenennen soll. Die notwendigen Daten können aus den letzten 128 Bytes der Datei herausgefunden werden, der ID3v1-Tag. Dazu haben wir jeweils die Bereiche mit dem Befehl *tail -cx-y* herausgeschnitten, in denen nach der Definition des ID3v1-TAGs das Album, der Titel, der Künstler oder die Tracknummer steht. Das Script fängt damit an, dass wir sagen mit welcher *Shell* wir gearbeitet haben:

```
0   #! /usr/bin/env bash
```

Da das Programm nicht nur ein Element übergeben bekommen kann, startet da Programm mit einer *for*-Schleife, um alle übergebenen Parameter durchzugehen.

```
1
2   for datei in "$@";do
```

Nun finden wir zuerst die Tracknummer heraus, die sich im im 127. Byte als Oktalzahl befindet. Dazu greifen wir mit dem Befehl *cat ./$datei | tail -c2* auf die letzten beiden Bytes der Datei zu. Dann bekommen wir mit *cut -b1* nur den ersten Byte und formatieren diesen Wert mit dem Befehl *od -d* in eine Dezimalzahl um. Nun haben wir herausgefunden, dass der relevante Teil in dem mittleren Zahlenblock liegt. Diese Zahl ist (scheinbar) immer größer als 2560 (Da wir alle Möglichkeiten der Operatoren des Befehls *od* getestet haben und nur bei dem Dezimalenausdruck "Müll"herauskommt, haben wir keine Erklärung dafür, dass die mittlere Zahl minus 2560 genau die Tracknummer ergibt, was wir mit einer ASCII-Tabelle und einer *char* Ausgabe verglichen haben). Zunächst wird mit einer *if*-Anweisung überprüft, ob die Tracknummer existiert. Wenn sie nicht existiert, wird in der Variable *number* der Nullstring gespeichert. Ansonsten wird geschaut ob die Tracknummer ein- zwei- oder dreistellig ist, indem in einer *if*-Anweisung überprüft wird, ob die erste der drei Stellen *0* ist. Wenn dies nicht der Fall ist, werden nur die letzten beiden Zahlen in der Variable *number* gespeichert. Der Code dazu sieht wie folgt aus:

```
4   # Hier wird die Tracknummer bestimmt
5       if [ $( cat ./$datei | tail -c2 | cut -b1 | od -d | cut -b10-14  ) -eq 2560 ]
6       then
7           number= ""
8       else
9                   if [ $(expr $( cat ./$datei | tail -c2 | cut -b1 | od -d |
10                  cut -b10-14  ) - 2560) -lt 10 ]
11                  then
12                          number=-0$(expr $( cat ./$datei | tail -c2 | cut -b1 |
13                          od -d | cut -b10-14  ) - 2560)
```

```
14                  else
15                  number=-$(expr $( cat ./$datei | tail -c2 | cut -b1 | od -d |
16                  cut -b10-14  ) - 2560)
17                  fi
18      fi
```

Als nächstes haben wir den Titel des Liedes gesucht, der sich in den Bytes 4 bis 33 des ID3v1-TAGs befindet. Dazu haben wir ähnlich wie bei der Tracknummer mit *cat ./$datei | tail -c125* die letzten 125 Bytes der Datei bekommen und davon die ersten 30 mit dem Befehl *cut -c1-30* herausgeschnitten. Aufgrund der Normierung dieses TAGs wissen wir, dass wir nun schon den Titel haben. Aber die Ausgabe soll ohne Leerzeichen sein, die man mit dem Befehl *tr -d [:blank:]* raus löscht. Nun wird mit der *if*-Anweisung und dem Operator *-z* getestet, ob der Inhalt leer ist oder nicht. Denn wenn er leer ist, wird der Unterstrich *"_"* in der Variable *titel* gespeichert. Insgesamt sieht dieser Teil also wie folgt aus:

```
17           # Hier wird der der Titel bestimmt
18          if [ -z "$(cat ./$datei | tail -c125 | cut -c1-30 | tr -d [:blank:])" ]
19          then
20                  titel="_"
21          else
22                  titel=$(cat ./$datei | tail -c125 | cut -c1-30 | tr -d [:blank:])
23          fi
```

Nun wird äquivalent wie beim Titel der Künstler bestimmt. Dabei werden zunächst die letzten 95 Bytes der Datei mit *tail -c95* genommen und davon die ersten 30 betrachtet (*cut -c1-30*). Alle Leerzeichen werden wieder gelöscht und mithilfe einer *if-else*-Anweisung die Variable *kuenstler* "_"gesetzt, falls die Ausgabe leer ist, ansonsten wird der Variable der ganze Wert zugeordnet:

```
23           #Hier wird der Kuenstler bestimmt
24          if [ -z "$(cat ./$datei | tail -c95  | cut -c1-30 | tr -d [:blank:])" ]
25          then
26                  kuenstler="_"
27          else
28                  kuenstler=$(cat ./$datei | tail -c95  | cut -c1-30 | tr -d [:blank:])
29          fi
```

Abschließend wird noch das Album herausgefiltert. Dies erfolgt analog zu den vorherigen. In einer *if-else*-Anweisung wird geschaut, ob der im TAG definierte Bereich für das Album ohne Leerzeichen leer ist und dann gegebenenfalls die Variable *album* gleich dem Unterstich "_"gesetzt. Ansonsten wird der ganze Wert in der Variable gespeichert. Der Quellcode dazu sieht wie folgt aus:

```
29          #Hier wird das Album bestimmt
30      if [ -z $(cat ./$datei | tail -c65  | cut -c1-30 | tr -d [:blank:]) ]
31      then
32          album="_"
33      else
34          album=$(cat ./$datei | tail -c65  | cut -c1-30 | tr -d [:blank:])
35      fi
```

Nun kann man aus allen berechneten Variablen die Datei den Auflagen entsprechend umbenennen. Dazu verwenden wir die Methode *mv*. Der neue Name der Datei setzt sich aus den Variablen in der Reihenfolge *Künstler-Album-Titel-NN* zusammen. Am Ende muss man noch den Typ der Datei hinzufügen, nämlich *.mp3*. Insgesamt sehen die letzten Zeilen des Programms so aus:

```
35          #Nun wird die Datei der Form entsprechend umbenannt
36      mv "$datei" "$( echo "$kuenstler"-"$album"-"$titel""$number".mp3 )"
37  done
```

Der gesamte Quellcode sieht am Stück wie folgt aus:

```
0   #! /usr/bin/env bash
1
2   for datei in "$@";do
3
4       # Hier wird die Tracknummer bestimmt
5       if [ $( cat ./$datei | tail -c2 | cut -b1 | od -d | cut -b10-14  ) -eq 2560 ]
6       then
7           number= ""
8       else
9                   if [ $(expr $( cat ./$datei | tail -c2 | cut -b1 | od -d |
10                  cut -b10-14  ) - 2560) -lt 10 ]
11                  then
12                          number=-0$(expr $( cat ./$datei | tail -c2 | cut -b1 |
13                          od -d | cut -b10-14  ) - 2560)
14                  else
15                  number=-$(expr $( cat ./$datei | tail -c2 | cut -b1 | od -d |
16                  cut -b10-14  ) - 2560)
17                  fi
18      fi
19
20      # Hier wird der der Titel bestimmt
21      if [ -z "$(cat ./$datei | tail -c125 | cut -c1-30 | tr -d [:blank:])" ]
22      then
23          titel="_"
24      else
25          titel=$(cat ./$datei | tail -c125 | cut -c1-30 | tr -d [:blank:])
26      fi
27
28      #Hier wird der Kuenstler bestimmt
29      if [ -z "$(cat ./$datei | tail -c95  | cut -c1-30 | tr -d [:blank:])" ]
30      then
31          kuenstler="_"
32      else
33          kuenstler=$(cat ./$datei | tail -c95  | cut -c1-30 | tr -d [:blank:])
34      fi
35
36      #Hier wird das Album bestimmt
37      if [ -z $(cat ./$datei | tail -c65  | cut -c1-30 | tr -d [:blank:]) ]
38      then
39          album="_"
40      else
41          album=$(cat ./$datei | tail -c65  | cut -c1-30 | tr -d [:blank:])
42      fi
43
44      #Nun wird die Datei der Form entsprechend umbenannt
```

```
45        mv "$datei" "$( echo "$kuenstler"-"$album"-"$titel""$number".mp3 )"
46    done
47
```

Wir haben das Programm mit allen Dateien als Parameter ausgeführt. Dazu muss man zunächst dem Programm das Recht geben, ausgeführt werden zu können. Also war die Eingabe:

```
0    ./mp3-rename.sh bugs.mp3 mt.unsafe.mp3 recursion.mp3 track1.mp3 track2.mp3
1     track3.mp3 track4.mp3 track5.mp3 track6.mp3 track7.mp3 unknown
```

Die Ergebnisse haben wir mit den Daten in den Eigenschaften der Dateien verglichen und stimmten überein. Leider konnten wir nicht testen, was passiert, wenn die Tracknummer nicht existiert, weil wir in eine MP3-Datei nicht an dem 127. Byte des ID3v1-TAGs den Oktalenwert *"000"* als ein Byte schreiben können.

Aufgabe 2

a)

Im folgenden sind drei *a.out*-Format Tabellen für *main.s*, *show.s* und *fib.s* angegeben. Diese wurden wie in der Vorlesung erstellt. Um zu erkennen, welche Änderungen wir an den Texttabellen vorgenommen haben, sind die Elemente in den geschweiften Klammern nicht mehr in der 'echten' Tabelle vorhanden. Byte-Sprüenge sind farbig makiert und in Klammern. Bei *call* Aufrufen wird mit der orangenen Null auf die Reloactiontabelle verwiesen, die auf die Symboltabelle und diese wiederum auf die Stringtabelle verweist. Die Texttabelle wurde über den Assemblercode erstellt, jede Instruktion ist 4 Byte lang.

Textsegment	show.s	
{show__Fm:}	pushq %rbp	0
	movq %rsp, %rbp	4
	subq $16, %rsp	8
	movq %rdi, -8(%rbp)	12
	movq -8(%rbp), %rax	16
	movq %rax, %rsi	20
	leaq cout(%rip), %rdi	24
	call {__ls__7ostreamm}0	28
	movl $32, %esi	32
	movq %rax, %rdi	36
	call {__ls__7ostreamc}0	40
	ret	44
Datensegment		Leer
Relocation Tabelle		
0	28 1	
1	40 2	
Symbol-Tabelle		
0	4 Text 0	
1	13 UNDEFINED	
2	29 UNDEFINED	
Stringtabelle		
0	Laenge (4 Bytes)	
4	show	
8	__Fm	
12	/0__l	
16	s__7	
20	ostr	
24	eamm	
28	/0__l	
32	s__7	
36	ostr	
40	eamc	

	main.s	Bytes
{main:}	pushq %rbp	0
	movq %rsp , %rbp	4
	subq $32 , %rsp	8
	movq %rsi , −32(%rbp)	12
	cmpl $1 , −20(%rbp)	16
	jle {.L2}(+96)	20
	movq −32(%rbp) , %rax	24
	addq $8 , %rax	28
	movq (%rax) , %rax	32
	movl $10 , %edx	36
	movl $0 , %esi	40
	movq %rax , %rdi	44
	call {strtol}0	48
	movq %rax , −16(%rbp)	52
	movq $0 , −8(%rbp)	56
{.L4:}	movq −8(%rbp) , %rax	60
	cmpq −16(%rbp) , %rax	64
	ja {.L3 }(+32)	68
	movq −8(%rbp) , %rax	72
	movq %rax , %rdi	76
	call {fib__Fm}0	80
	movq %rax , %rdi	84
	call {show__Fm}0	88
	addq $1 , −8(%rbp)	92
	jmp {.L4}(-36)	96
{.L3:}	movq endl__F7ostream(%rip)	100
	movq %rax , %rsi	104
	leaq cout(%rip) , %rdi	108
	call {__ls__7ostreamm}0	112
{.L2:}	movl $0 , %eax	116
	ret	120

Datensegment		Leer
Relocation Tabelle		
0	48 1	
1	80 2	
2	88 3	
3	112 4	
Symbol–Tabelle		
0	4 Text 0	
1	9 UNDEFINED	
2	16 UNDEFINED	
3	24 UNDEFINED	
4	33 UNDEFINED	
Stringtabelle		
0	Laenge (4 Bytes)	
4	main	
8	\0str	
12	tol\0	
16	fib_	
20	_Fm\0	
24	show	
28	__Fm	
32	\0__l	
36	s__7	
40	ostr	
44	eamm	

	fib.s	Bytes
{fib__Fm:}	pushq %rbp	0
	movq %rsp, %rbp	4
	pushq %rbx	8
	subq $40, %rsp	12
	movq %rdi, −40(%rbp)	16
	movq $1, −24(%rbp)	20
	cmpq $1, −40(%rbp)	24
	jbe {.L2}(+52)	28
	movq −40(%rbp), %rax	32
	subq $2, %rax	36
	movq %rax, %rdi	40
	call {fib__Fm}(-44)	44
	movq %rax, %rbx	48
	movq −40(%rbp), %rax	52
	subq $1, %rax	56
	movq %rax, %rdi	60
	movq %rax, %rdi	64
	call {fib__Fm}(-68)	68
	addq %rbx, %rax	72
	movq %rax, −24(%rbp)	76
{.L2}	movq −24(%rbp), %rax	80
	addq $40, %rsp	84
	popq %rbx	88
	popq %rbp	92
	ret	96
Datensegment		Leer
Relocation Tabelle		Leer
Symbol−Tabelle		
0	4 Text 0	
Stringtabelle		
0	Laenge (4 Bytes)	
4	fib_	
8	_Fm\0	

b)

Der Linker würde die Tabellen in der Reihenfolge *main.s*, *fib.s* und *show.s* 'aufeinander stapeln'. Z.B. wird die Texttabelle von *fib.s* an die von *main.s* gebunden, usw. Wenn *main.s* dann *fib.s* aufruft, lässt sich dies über einen pc-relativen Sprung ausdrücken. Es sind nur zwei Offsets zu berechen. Einmal der *call* in der Zeile von Byte 80 der *main.s*-Texttabelle und einmal der *call* 8 Byte weiter. Die Berechnung des ersten Offsets: $120 - 80 = +40$ Bytes. Also ein Sprung von 40 Byte. Der zweite Sprung berechnet sich folgendermassen: $(120 - 88) + 96 = +128$ Byte. Wenn in *main.s* also *fib.s* aufgerufen wird, springt der Compiler 128 Byte im Code.

c)

Die in b) berechnete Datei wäre noch nicht ausführbar, da die Bibliotheken für die Operatoren (siehe z.B. die Zeile von Byte 40 in der Texttabelle von *show.s*)fehlen. Die Adressen köennen nicht aufgelöst werden und die entsprechenden Bibliotheken müssten noch hinzugefügt werden.

Aufgabe 3

Zum Lösen der Aufgabe haben wir zuerst das Programm *geheim* mit dem GDB aufgerufen (*gdb geheim*). Danach haben wir das Kommando *disassemble main* verwendet, da dies zunächst der einzige Anhaltspunkt war. Darauf wurde unter anderem die Funktion *check_password* aufgelistet. Da dort wohl vermutlich das eingegebene Passwort überprüft wird, haben wir *disassemble check_password* aufgerufen. Dann haben wir bei dem letzten Vergleich einen Breakpoint gesetzt mit *break *0x000000000400f79* und das Programm mit *run* ausgeführt. Zuletzt haben wir uns mit *info locals* die Variablen zeigen lassen. Darunter war auch:

pass = "TeI2-ist_einfach"

Dies ist das gesuchte Passwort.

Das ganze ist möglich, da der GDB die Symboltabelle lesen kann. Aus dieser zieht er die Rückschlüsse auf die Funktionsweise des Programms.