Technische Informatik 2
Tutor: Marc/Bingbin

WS 2016/17

C05

Tabea Eggers
Jan Fiedler
Florian Pflüger
Jonas Schmutte

Übungsblatt 2

1	2	3	4	5	$\sum$

Lösungsvorschlag
Abgabe: 14.11.2016

Aufgabe 1

Wir haben eine while-Schleife die Argument für Argument durchgeht bis kein weiteres mehr vorhanden ist. Hier wird falls es sich um einen ID3v1-Tag handelt, Titel, Interpret und Album ausgelesen und formatiert. Der Track wird ebenfalls angehängt. Leider ist uns nicht klar wie wir testen ob es einen Track gibt, deshalb hängen wir überall einen an. Anschließend wird die Datei dementsprechend umbenannt. Falls es sich nicht um einen ID3v1-Tag handelt wird eine entsprechende Meldung ausgegeben. Für die Ausgaben auf der Konsole verzichten wir auf Pfadangaben, da wir dies schöner finden.

```
1  #!/usr/bin/env bash
2
3  echo "Die Dateien: $* werden umbenannt."
4
5  #Geht Eingabe fuer Eingabe durch
6  while [ "$1" != '' ]
7  do
8      DATA=$(basename $1) #Nur Dateiname ohne Pfad, sieht schoener aus
9      echo "$DATA wird umbenannt..."
10
11     # ID3v1-Tag ?
12     TAG=$(tail -c 128 $1)
13     if [[ ${TAG:0:3} == TAG ]]; then
14
15         # Titel bestimmen
16         TITEL=$(tail -c 128 $1|cut -b4-33|tr -d "\000")
17         TITEL=${TITEL//[ -]/_} # ' ' und - durch _ ersetzen
18         if [[ ${TITEL} == "" ]]; then
19             TITEL+="_"
20         fi
21
22         # Interpret bestimmen
23         INTERPRET=$(tail -c 128 $1|cut -b34-63|tr -d "\000")
24         INTERPRET=${INTERPRET//[ -]/_}
25         if [[ ${INTERPRET} == "" ]]; then
26             INTERPRET+="_"
27         fi
28
29         # Album bestimmen
30         ALBUM=$(tail -c 128 $1|cut -b64-93|tr -d "\000")
```

```
31          ALBUM=${ALBUM//[ -]/_}
32          if [[ ${ALBUM} == "" ]]; then
33              ALBUM+="_"
34          fi
35
36          # Titel bauen
37          NAME=${INTERPRET}-${ALBUM}-${TITEL}
38
39          #Titel anhaengen falls vefuegbar(If bedingung ist uns unklar)
40          #IFTRACK=$(tail -c 128 $1| cut -b126)
41          #if [[ ${IFTRACK} == "\000"  ]]; then
42          if [[ true ]]; then
43              TRACK=$(tail -c 128 $1|cut -b127|tr -d "\000"|od|cut -b13
                    -14)
44              NAME+=-${TRACK}
45          fi
46
47          NAME+=".mp3"
48
49          mv -f $1 ${NAME} #umbenennen
50
51          echo "$DATA wurde umbenannt in $NAME"
52      else
53          echo "$1 besitzt keinen ID3v1-Tag..."
54          echo "$1 konnte nicht umbenannt werden..."
55      fi
56
57      shift # $x nach links shiften
58  done
59  echo "Fertig :)"
```

Zum Testen haben wir wieder ein kleines Testscript geschrieben:
Befehl: `./test.sh`

```
1  #!/usr/bin/env bash
2
3  echo "--- Starte Test:"
4  echo ""
5  echo "--- Verzeichnisse:"
6  ls
7  echo "--- Unterverzeichnis:"
8  cd the\ programmer
9  ls
10 cd ..
11 echo ""
12 echo "--- umbenennen:"
13 ./mp3-rename.sh track1.mp3 track2.mp3 track3.mp3 track4.mp3 track5.
       mp3 track6.mp3 track7.mp3 unknown
14 echo ""
15 echo "--- umbenennen2:"
16 cd the\ programmer
```

```
./../mp3-rename.sh bugs.mp3 mt.unsafe.mp3 recursion.mp3
cd ..
echo ""
echo "--- Verzeichnisse danach:"
ls
echo "--- Unterverzeichnis:"
cd the\ programmer
ls
cd ..
echo ""
echo "--- fertig"
```

hier folgen die Ausgaben:
Befehl: `./test.sh >test_result.txt`

```
--- Starte Test:

--- Verzeichnisse:
mp3-rename.sh
test_result.txt
test.sh
the programmer
track1.mp3
track2.mp3
track3.mp3
track4.mp3
track5.mp3
track6.mp3
track7.mp3
unknown
--- Unterverzeichnis:
bugs.mp3
mt.unsafe.mp3
recursion.mp3

--- umbenennen:
Die Dateien: track1.mp3 track2.mp3 track3.mp3 track4.mp3 track5.mp3 track6.mp3 track7.mp3 unknown werden umbenannt.
track1.mp3 wird umbenannt...
track1.mp3 wurde umbenannt in J._Postel_&_The_Packet_Drops-Source_Quench-Fragments_of_IP-01.mp3
track2.mp3 wird umbenannt...
track2.mp3 wurde umbenannt in J._Postel_&_The_Packet_Drops-Source_Quench-Traffic_Class_Assignment-02.mp3
track3.mp3 wird umbenannt...
track3.mp3 wurde umbenannt in J._Postel_&_The_Packet_Drops-Source_Quench-No_Route_to_Host-03.mp3
track4.mp3 wird umbenannt...
track4.mp3 wurde umbenannt in The_OpenSSH-Roaming_Around-Triple_7-04.mp3
track5.mp3 wird umbenannt...
```

```
track5.mp3 wurde umbenannt in The_OpenSSH-Roaming_Around-B.E.A.S.T.-05.mp3
track6.mp3 wird umbenannt...
track6.mp3 wurde umbenannt in The_OpenSSH-Roaming_Around-Heartbleed-06.mp3
track7.mp3 wird umbenannt...
track7.mp3 wurde umbenannt in BGP-Distance_Vectors-RIP-07.mp3
unknown wird umbenannt...
unknown wurde umbenannt in The_Drammer_Boys-_-_-44.mp3
Fertig :)

--- umbenennen2:
Die Dateien: bugs.mp3 mt.unsafe.mp3 recursion.mp3 werden umbenannt.
bugs.mp3 wird umbenannt...
bugs.mp3 wurde umbenannt in _-Bugs_galore-Three_little_bugs-60.mp3
mt.unsafe.mp3 wird umbenannt...
mt.unsafe.mp3 wurde umbenannt in _-Bugs_galore-Race_Condition-27.mp3
recursion.mp3 wird umbenannt...
recursion.mp3 wurde umbenannt in _-Bugs_galore-Endless_Recursion-01.mp3
Fertig :)

--- Verzeichnisse danach:
BGP-Distance_Vectors-RIP-07.mp3
J._Postel_&_The_Packet_Drops-Source_Quench-Fragments_of_IP-01.mp3
J._Postel_&_The_Packet_Drops-Source_Quench-No_Route_to_Host-03.mp3
J._Postel_&_The_Packet_Drops-Source_Quench-Traffic_Class_Assignment-02.mp3
mp3-rename.sh
test_result.txt
test.sh
The_Drammer_Boys-_-_-44.mp3
The_OpenSSH-Roaming_Around-B.E.A.S.T.-05.mp3
The_OpenSSH-Roaming_Around-Heartbleed-06.mp3
The_OpenSSH-Roaming_Around-Triple_7-04.mp3
the programmer
--- Unterverzeichnis:
_-Bugs_galore-Endless_Recursion-01.mp3
_-Bugs_galore-Race_Condition-27.mp3
_-Bugs_galore-Three_little_bugs-60.mp3

--- fertig
```

Uns ist bewusst, dass `./mp3-rename.sh ./the programmer/bugs.mp3` nicht funktioniert, wir hatten aber keine Idee, wie wir dieses Problem beheben könnten. Wie man an den Ausgaben sieht funktioniert unser Skript aber mit diesen beiden Einschränkungen.

Aufgabe 2

Tabelle 1: Ursprünglicher Code

main.s				fib.s				show.s			
0	main:	pushq	%rbp	0	fib_Fm:	pushq	%rbp	0	show_Fm:	pushq	%rbp
4		movq	%rsp, %rbp	4		movq	%rsp, %rbp	4		movq	%rsp, %rbp
8		subq	$32, %rsp	8		pushq	%rsx	8		subq	$16, %rsp
12		movl	%edi, -20(%rbp)	12		subq	$40, %rsp	12		movq	%rdi, -8(%rbp)
16		movq	%rsi, -32(%rbp)	16		movq	%rdi, -40(%rbp)	16		movq	-8(%rbp), %rax
20		cmpl	$1, -20(%rbp)	20		movq	$1, -24(%rbp)	20		movq	%rax, %rsi
24		jle	.L2	24		cmpq	$1, -40(%rbp)	24		leaq	cout(%rip), %rdi
28		movq	-32(%rbp), %rax	28		jbe	.L2	28		call	_ls_7ostreamm
32		addq	$8, %ray	32		movq	-40(%rbp), %rax	32		movl	$32, %esi
36		movq	(%rax), %rax	36		subq	$2, %rax	36		movq	%rax, %rdi
40		movl	$10, %edx	40		movq	%rax, %rdi	40		call	_ls_7ostreamc
44		movl	$0, %esi	44		call	fib_Fm	44		ret	
48		movq	%rax, %rdi	48		movq	%rax, %rbx				
52		call	strtol	52		movq	-40(%rbp), %rax				
56		movq	$0, -8(%rbp)	56		subq	$1, %rax				
60		movq	%rax, -16(%rbp)	60		movq	%rax, %rdi				
64	.L4:	movq	-8(%rbp), %rax	64		call	fib_Fm				
68		cmpq	-16(%rbp), %rax	68		addq	%rbx, %rax				
72		ja	.L3	72		movq	%rax, -24(%rbp)				
76		movq	-8(%rbp), %rax	76	.L2:	movq	-24(%rbp), %rax				
80		movq	%rax, %rdi	80		addq	$40, %rsp				
84		call	fib_Fm	84		popq	%rbx				
88		movq	%ray, %rdi	88		popq	%rbp				
92		call	show_Fm	92		ret					
96		addq	$1,-8(%rbp)								
100		jmp	.L4								
104	.L3:	movq	endl_F7ostream(%rip), %rax								
108		movq	%rax, %rsi								
112		leaq	cout(%rip), %rdi								
116		call	_ls_7ostreamm								
120	.L2:	movl	$0, %eax								
124		ret									

Tabelle 2: Interne Sorünge

main.s				fib.s				show.s			
0	main:	pushq	%rbp	0	fib_Fm:	pushq	%rbp	0	show_Fm:	pushq	%rbp
4		movq	%rsp, %rbp	4		movq	%rsp, %rbp	4		movq	%rsp, %rbp
8		subq	$32, %rsp	8		pushq	%rsx	8		subq	$16, %rsp
12		movl	%edi, -20(%rbp)	12		subq	$40, %rsp	12		movq	%rdi, -8(%rbp)
16		movq	%rsi, -32(%rbp)	16		movq	%rdi, -40(%rbp)	16		movq	-8(%rbp), %rax
20		cmpl	$1, -20(%rbp)	20		movq	$1, -24(%rbp)	20		movq	%rax, %rsi
24		jle	96	24		cmpq	$1, -40(%rbp)	24		leaq	cout(%rip), %rdi
28		movq	-32(%rbp), %rax	28		jbe	48	28		call	_ls_7ostreamm
32		addq	$8, %ray	32		movq	-40(%rbp), %rax	32		movl	$32, %esi
36		movq	(%rax), %rax	36		subq	$2, %rax	36		movq	%rax, %rdi
40		movl	$10, %edx	40		movq	%rax, %rdi	40		call	_ls_7ostreamc
44		movl	$0, %esi	44		call	fib_Fm	44		ret	
48		movq	%rax, %rdi	48		movq	%rax, %rbx				
52		call	strtol	52		movq	-40(%rbp), %rax				
56		movq	$0, -8(%rbp)	56		subq	$1, %rax				
60		movq	%rax, -16(%rbp)	60		movq	%rax, %rdi				
64	.L4:	movq	-8(%rbp), %rax	64		call	fib_Fm				
68		cmpq	-16(%rbp), %rax	68		addq	%rbx, %rax				
72		ja	32	72		movq	%rax, -24(%rbp)				
76		movq	-8(%rbp), %rax	76	.L2:	movq	-24(%rbp), %rax				
80		movq	%rax, %rdi	80		addq	$40, %rsp				
84		call	fib_Fm	84		popq	%rbx				
88		movq	%ray, %rdi	88		popq	%rbp				
92		call	show_Fm	92		ret					
96		addq	$1,-8(%rbp)								
100		jmp	-36								
104	.L3:	movq	endl_F7ostream(%rip), %rax								
108		movq	%rax, %rsi								
112		leaq	cout(%rip), %rdi								
116		call	_ls_7ostreamm								
120	.L2:	movl	$0, %eax								
124		ret									

Tabelle 3: Globale Symbole und Relocation

main.s				fib.s					show.s				
(Datensegment ist leer)				(Datensegment ist leer)					(Datensegment ist leer)				
	52	1								24	1		
	84	2								28	2		
	92	3								40	3		
	104	4											
	116	5											
0	0	T	0	0	0	T	0		0	0	T	0	
1	5	U							1	9	U		
2	12	U							2	14	U		
3	20	U							3	30	U		
4	29	U											
5	45	U											
0	main			0	fib_				0	show			
4	\0str			4	_Fm\0				4	_Fm			
8	tol\0								8	\0cou			
12	fib_								12	t\0__			
16	_Fm\0								16	ls__			
20	show								20	7ost			
24	_Fm								24	ream			
28	\0end								28	m\0__			
32	l_F								32	ls__			
36	7ost								36	7ost			
40	ream								40	ream			
44	\0_l								44	c\0			
48	s__7												
52	ostr												
56	eamm												
60	\0												
64													

Tabelle 4: Linken

a.out			
(Datensegment ist leer)			
	52	1	
	84	2	WDISP 44
	92	3	WDISP 132
	104	4	
	116	5	
	248	6	
	252	7	
	264	8	
0	0	T	0
1	5	U	
2	12	T	128
3	20	T	224
4	29	U	
5	45	U	
6	61	U	
7	66	U	
8	83	U	
0	main		
4	\0str		
8	tol\0		
12	fib_		
16	_Fm\0		
20	show		
24	__Fm		
28	\0end		
32	l__F		
36	7ost		
40	ream		
44	\0__l		
48	s__7		
52	ostr		
56	eamm		
60	\0cou		
64	t\0__		
68	ls__		
72	7ost		
76	ream		
80	m\0__		
84	ls__		
88	7ost		
92	ream		
96	c\0		

Wir gehen davon aus das die unterschiedlichen Segmente direkt hintereinander liegen, die leeren Zeilen haben wir nur hinzugefügt, damit wir die gleichen Segmente nebeneinander liegen haben.

b) Beim linken können zwei Sprünge berechnet werden, den für "fib__Fmünd ßhow__Fm", mit

dem Offsets von 44 und 132. Wir gehen davon aus das die Textsegmente jeweils hinten an das vorherige Textsegment angehängt werden und die String- und Symboltabelle sowie die Text/Data-Relocation-Table ergänzt werden.

c) Die so erzeugt Datei kann nicht ausgeführt werden, weil noch nicht alle Symbole aufgelöst wurden, es fehlen alle Systemsymbole für die Ausgabe und Umwandlung.

Aufgabe 3

Zunächst einmal gehen wir in das Programm mit `gdb geheim`.
Dann bekommen wir eine Übersicht über das Programm mit Hilfe von `disassemble main`.
Da wir das Passwort heraus finden sollen, ist uns direkt folgende Zeile aufgefallen:
`0x0000000000400c23 <+163>: callq 0x400f10 <check_password(char const*)>`
Um uns `check_password` genauer anzusehen, sind wir mit `disassemble check_password` in diesen Programmabschnitt gegangen.
Dort fiel uns dann folgende Zeile auf:
`0x0000000000400f7d <+109>: jne 0x400f68 <check_password(char const*)+88>`
Diese wollten wir nun näher untersuchen und haben dafür zuerst einen Breakpoint mit
`break *0x0000000000400f7d` gesetzt.
Anschließend muss das Programm mit Hilfe von `run` ausgeführt werden, um an die Stelle des Breakpoints zu kommen.
Jetzt konnten wir die Programmstelle mit dem Befehl `x\s $rsi` näher untersuchen:
`0x7fffde169a90: "TeI2-ist_einfach"`
Dies ist schon das Passwort. Um aber sicher zu gehen, haben wir dies noch weiter angesehen mit `info locals`:

```
i = 1
pass = "TeI2-ist_einfach", '\000' <repeats 83 times>
ok = false
```

Nun könnte man sich noch mit `p pass` (`$1 = "TeI2-ist_einfach", '\000' <repeats 83 times>`) und `x $1` (`0x7fffde169a90: "TeI2-ist_einfach"`) das ganze noch genauer ansehen, aber dies ist um das Passwort heraus zu finden nicht weiter nötig.
gdb weiß so viel über das Programm, da gdb ein Debugger ist. Außerdem ist alles intern auf dem Rechner gespeichert, also kann alles ausgeführt und dadurch auch von gdb verarbeitet werden. gdb kann somit zur Laufzeit des Programms an einer Stelle das Programm anhalten und die aktuellen Variablen ansehen. Dies hat zur Folge, dass man z.B. ein Passwort herausfinden kann, indem man die Stelle des unbekannten Programms sucht, wo die Eingabe überprüft wird und dort dann schaut, welche Eingabe korrekt ist.