

Übung 6

Benedict Herzog
Bernhard Heinloth
Tim Rheinfels

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme
und Betriebssysteme



Linux

Vorstellung Aufgabe 3

Terminal - Historisches (etwas vereinfacht)



- Als die Computer noch größer waren:



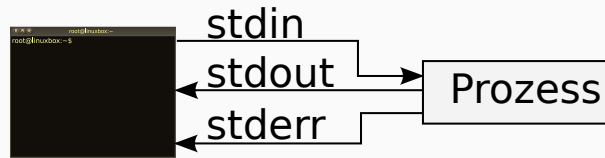
- Als das Internet noch langsam war:



- Farben, Positionssprünge, etc. werden durch spezielle Zeichenfolgen ermöglicht



- Drei Standardkanäle für Ein- und Ausgaben



stdin Eingaben

stdout Ausgaben

stderr Fehlermeldungen

- Standardverhalten
 - Eingaben kommen von der Tastatur
 - Ausgaben & Fehlermeldungen erscheinen auf dem Bildschirm

2



- stdout in Datei schreiben

```
01 find . > ordner.txt
```

- stdout als stdin für anderer Programme

```
01 cat ordner.txt | grep tmp | wc -l
```

- Vorteil von **stderr**
 - ⇒ Fehlermeldungen werden weiterhin am Terminal ausgegeben
- Übersicht
 - > Standardausgabe **stdout** in Datei schreiben
 - >> Standardausgabe **stdout** an exist. Datei anhängen
 - 2> Fehlerausgabe **stderr** in Datei schreiben
 - < Standardeingabe **stdin** aus Datei einlesen
 - | Ausgabe eines Befehls als Eingabe für anderen Befehl

3



- Wechseln in ein Verzeichnis mit **cd** (change directory)

```
01 cd /proj/i4spic/<login>/aufgabeX/ # absolut in Ordner
02 cd aufgabe5/                    # relativ zum aktuellen Ordner
03 cd ~                            # Benutzerverzeichnis (Home)
04 cd ..                           # übergeordnete Verzeichnis
```

- Verzeichnisinhalt auflisten mit **ls** (list directory)

```
01 ls                               # zeige Dateien im akt. Ordner
02 ls -A                           # zeige auch versteckte Dateien
03 ls -lh                          # zeige mehr Metadaten
```

4



- Datei oder Ordner kopieren mit **cp** (copy)

```
01 # Kopiere Datei ampel.c aus dem Home in Projektverzeichnis
02 cp ~/ampel.c /proj/i4spic/xy42abcd/aufgabe5/ampel.c
03
04 # Kopiere Ordner aufgabe5/ aus dem Home in Projektverzeichnis
05 cp -r ~/aufgabe5/ /proj/i4spic/xy42abcd/
```

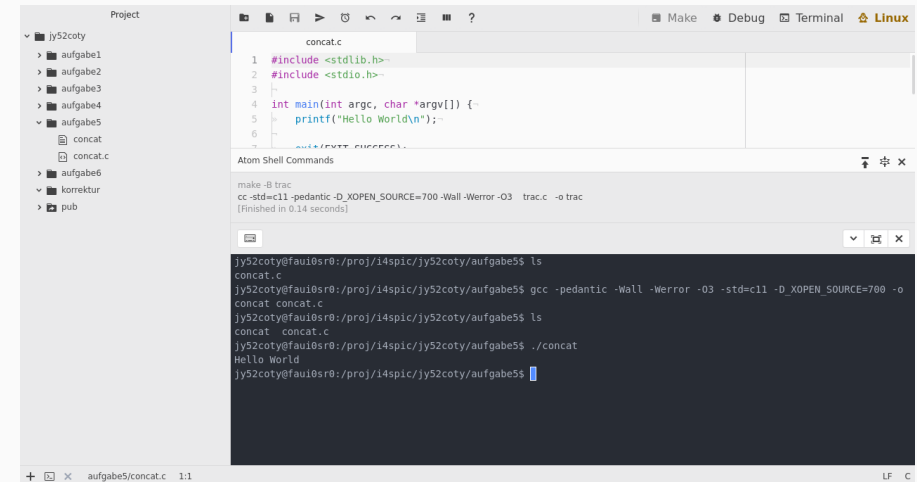
- Datei oder Ordner unwiederbringlich löschen mit **rm** (remove)

```
01 # Lösche Datei test1.c im aktuellen Ordner
02 rm test1.c
03
04 # Lösche Unterordner aufgabe1/ mit allen Dateien
05 rm -r aufgabe1
```

4



- Per Signal: CTRL-C (kann von Programmen ignoriert werden)
- Von einer anderen Konsole aus: `killall concat` beendet alle Programme mit dem Namen "concat"
- Von der selben Konsole aus:
 - CTRL-Z hält den aktuell laufenden Prozess an
 - `killall concat` beendet alle Programme namens concat
 - ⇒ Programme anderer Benutzer dürfen nicht beendet werden
 - `fg` setzt den angehaltenen Prozess fort
- Wenn nichts mehr hilft: `killall -9 concat`



- **Terminal:** öffnet ein Terminal und startet eine Shell
 - effiziente Interaktion mit dem System
 - optional Vollbild
- **Debug:** startet den Debugmodus
- **Make:** siehe übernächste Woche

5

6

Übersetzen & Ausführen



Übersetzen & Ausführen



- Programm mit dem GCC übersetzen

```
01 gcc -pedantic -Wall -Werror -O3 -std=c11 -D_XOPEN_SOURCE=700 -o
   ↪ concat concat.c
```

gcc ruft den Compiler auf (GNU Compiler Collection)

- pedantic** aktiviert Warnungen (Abweichungen vom C-Standards)
- Wall** aktiviert Warnungen (typische Fehler, z.B.: `if(x = 7)`)
- Werror** wandelt Warnungen in Fehler um
- O3** aktiviert Optimierungen (Level 3)
- std=c11** setzt verwendeten Standard auf C11
- D_XOPEN_SOURCE=700**
 - fügt POSIX Erweiterungen hinzu
- o concat** legt Namen der Ausgabedatei fest (Standard: `a.out`)
- concat.c ...** zu kompilierende Datei(en)

- Ausführen des Programms mit `./concat`
- Abgaben werden von uns mit diesen Optionen getestet

- Programm mit dem GCC übersetzen
(inklusive Debugsymbole und ohne Optimierungen)

```
01 gcc -pedantic -Wall -Werror -O0 -std=c11 -D_XOPEN_SOURCE=700 -g -
   ↪ o concat concat.c
```

- O0** verhindert Optimierungen des Programms
- g** belässt Debugsymbole in der ausführbaren Datei

⇒ ermöglicht dem Debugger Verweise zur Quelldatei herzustellen

- Hinweis: Pfeiltaste `↑` iteriert durch frühere Befehle
- ⇒ GCC Aufruf nur einmal tippen

7

8

- Informationen über:
 - Speicherlecks (malloc()/free())
 - Zugriffe auf nicht gültigen Speicher
- Ideal zum Lokalisieren von Segmentation Faults (SIGSEGV)
- Aufrufe:
 - `valgrind ./concat`
 - `valgrind --leak-check=full --show-reachable=yes`
 ↳ `--track-origins=yes ./concat`
- Die Ausgabe ist deutlich hilfreicher, wenn das analysierte Binary mit Debugsymbolen gebaut wird

9

- Das Linux-Hilfesystem
- aufgeteilt nach verschiedenen Sections
 - 1 Kommandos
 - 2 Systemaufrufe
 - 3 Bibliotheksfunktionen
 - 5 Dateiformate (spezielle Datenstrukturen, etc.)
 - 7 verschiedenes (z.B. Terminaltreiber, IP, ...)
- man-Pages werden normalerweise mit der Section zitiert:
`printf(3)`

```
01 # man [section] Begriff
02 man 3 printf
```

- Suche nach Sections: `man -f Begriff`
- Suche von man-Pages zu einem Stichwort: `man -k Stichwort`

10

- Abgespeckte (hübschere) Version der Manpages
- Bieten nur eine Übersicht, keine vollständige Spezifikation
- Aus der SPiC-IDE abrufbar (Hilfe-Button wenn im Linux-Modus)
- Auf der Webseite zu finden
https://www4.cs.fau.de/Lehre/WS19/V_SPIC/Linux/libc-api/
- Unsere Übersicht ersetzen die Manpages nicht
- In der Klausur: ausgedruckte Manpages!

11

Vertiefung C Strings



- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: strlen(s) + 1

```
01 char *s = "World\n";
02 char c = s[0];
03 char c = s[4];
04 char *s2 = s + 2;
05 char c = s2[1];
```

⋮	Stack ↓
...	0x0911
	0x0910
	0x090f
	0x090e
	0x090d
	0x090c
	0x090b
	0x090a
	0x0909
	0x0908
	0x0907
	0x0906
⋮	

12

- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: strlen(s) + 1

```
01 char *s = "World\n";
02 char c = s[0];
03 char c = s[4];
04 char *s2 = s + 2;
05 char c = s2[1];
```

⋮	Stack ↓
...	0x0911
s[6]	'\0'
s[5]	'\n'
s[4]	'd'
s[3]	'l'
s[2]	'r'
s[1]	'o'
s[0]	'W'
	0x0909
	0x0908
	0x0907
	0x0906
⋮	

12



- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: strlen(s) + 1

```
01 char *s = "World\n";
02 char c = s[0];
03 char c = s[4];
04 char *s2 = s + 2;
05 char c = s2[1];
```

⋮	Stack ↓
...	0x0911
s[6]	'\0'
s[5]	'\n'
s[4]	'd'
s[3]	'l'
s[2]	'r'
s[1]	'o'
s[0]	'W'
C	'W'
	0x0909
	0x0908
	0x0907
	0x0906
⋮	

12

- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: strlen(s) + 1

```
01 char *s = "World\n";
02 char c = s[0];
03 char c = s[4];
04 char *s2 = s + 2;
05 char c = s2[1];
```

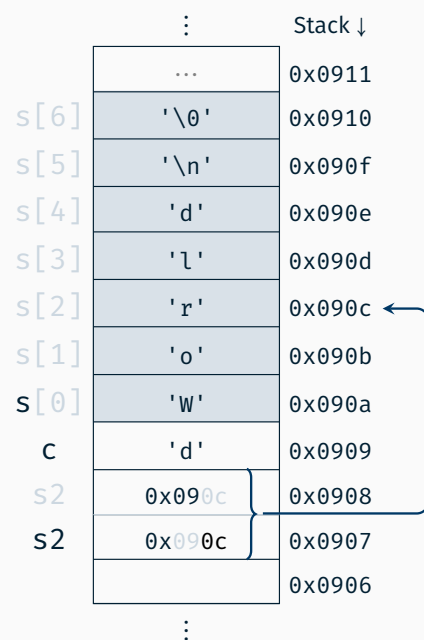
⋮	Stack ↓
...	0x0911
s[6]	'\0'
s[5]	'\n'
s[4]	'd'
s[3]	'l'
s[2]	'r'
s[1]	'o'
s[0]	'W'
C	'd'
	0x0909
	0x0908
	0x0907
	0x0906
⋮	

12



- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: strlen(s) + 1

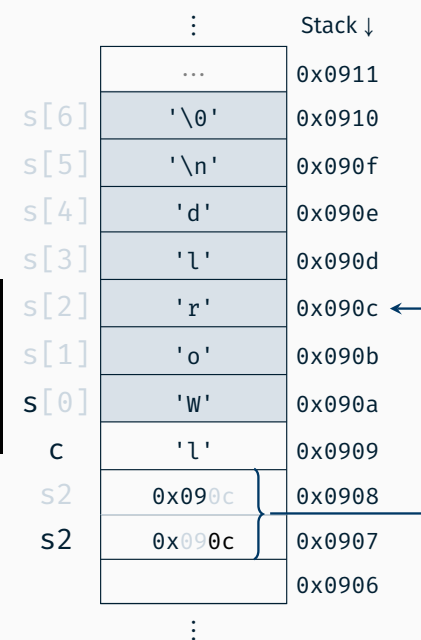
```
01 char *s = "World\n";
02 char c = s[0];
03 char c = s[4];
04 char *s2 = s + 2;
05 char c = s2[1];
```



12

- char: Einzelnes Zeichen (z.B. 'a')
- String: Array von chars (z.B. "Hello")
- In C: Letztes Zeichen eines Strings: '\0'
⇒ Speicherbedarf: strlen(s) + 1

```
01 char *s = "World\n";
02 char c = s[0];
03 char c = s[4];
04 char *s2 = s + 2;
05 char c = s2[1];
```



12

Umgang mit Strings



- `size_t strlen(const char *s)`
 - Bestimmung der Länge einer Zeichenkette s (ohne abschließendes Null-Zeichen)
- `char *strcpy(char *dest, const char *src)`
 - Kopieren einer Zeichenkette src in einen Puffer dest (inkl. Null-Zeichen)
 - Gefahr: Buffer Overflow (⇒ `strncpy(3)`)
- `char *strcat(char *dest, const char *src)`
 - Anhängen einer Zeichenkette src an eine existierende Zeichenkette im Puffer dest (inkl. Null-Zeichen)
 - Gefahr: Buffer Overflow (⇒ `strncat(3)`)
- Dokumentation: `strlen(3)`, `strcpy(3)`, `strcat(3)`

Hands-on: Buffer Overflow



■ Passwortgeschütztes Programm

```
01 # Usage: ./print_exam <password>
02 ./print_exam spic
03 Correct Password
04 Printing exam...
```

■ Passwortgeschütztes Programm

```
01 # Usage: ./print_exam <password>
02 ./print_exam spic
03 Correct Password
04 Printing exam...
```

■ Ungeprüfte Benutzereingaben ⇒ Buffer Overflow

```
01 long check_password(const char *password) {
02     char buff[8];
03     long pass = 0;
04
05     strcpy(buff, password);
06     if(strcmp(buff, "spic") == 0) {
07         pass = 1;
08     }
09     return pass;
10 }
```

14

14



■ Passwortgeschütztes Programm

```
01 # Usage: ./print_exam <password>
02 ./print_exam spic
03 Correct Password
04 Printing exam...
```

■ Ungeprüfte Benutzereingaben ⇒ Buffer Overflow

```
01 long check_password(const char *password) {
02     char buff[8];
03     long pass = 0;
04
05     strcpy(buff, password);
06     if(strcmp(buff, "spic") == 0) {
07         pass = 1;
08     }
09     return pass;
10 }
```

```
01 long check_password(const char *password) {
02     char buff[8];
03     long pass = 0;
04
05     strcpy(buff, password);
06     if(strcmp(buff, "spic") == 0) {
07         pass = 1;
08     }
09     return pass;
10 }
```

■ Mögliche Lösungen

- Prüfen der Benutzereingabe
- Dynamische Allokation des Puffers
- Sichere Bibliotheksfunktionen verwenden ⇒ z.B. strncpy(3)

14

15