

Übungen zu Systemnahe Programmierung in C (SPiC)

Peter Wägemann, Sebastian Maier, Heiko Janker
(Lehrstuhl Informatik 4)

Übung 7



Sommersemester 2015



Inhalt

Linux

Terminal
Arbeiten unter Linux
Arbeitsumgebung
Manual Pages

Fehlerbehandlung

Aufgabe: cworld

Anhang



Inhalt

Linux

Fehlerbehandlung

Aufgabe: cworld

Anhang



Terminal - historisches (etwas vereinfacht)

- Als die Computer noch größer waren:



- Als das Internet noch langsam war:



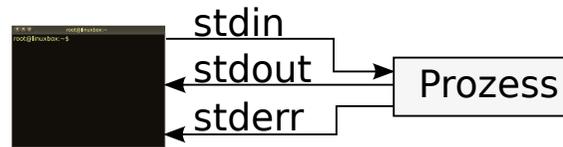
- Farben, Positionssprünge, etc werden durch spezielle Zeichenfolgen ermöglicht



¹Televideo 925

Terminal - Funktionsweise

- Drei Kanäle:



- stdin Eingaben
- stdout Ausgaben
- stderr Fehlermeldungen



Terminal - stdout und stderr

- Beispiel stdout und stderr

- Ausgabe in eine Datei schreiben

```
1 find . > ordner.txt
```

- Vor allem unter Linux wird stdout häufig direkt mit stdin anderer Programme verbunden

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

- Vorteil von stderr:

⇒ Fehlermeldungen werden weiterhin am Terminal ausgegeben



Die Shell

- Wichtige Kommandos

- cd (change directory) Wechseln in ein Verzeichnis

```
1 cd /proj/i4spic/<login>/aufgabeX
```

- ls (list directory) Verzeichnisinhalt auflisten

```
1 ls
```

- cp (copy) Datei kopieren

```
1 cp /proj/i4spic/pub/aufgabeX/vorgabe.h /proj/i4spic/<login>/\n  ↳ aufgabeX
```

```
2 # oder
```

```
3 cd /proj/i4spic/<login>/aufgabeX
```

```
4 cp /proj/i4spic/pub/aufgabeX/vorgabe.h .
```

- rm (remove) Löschen

```
1 rm test1.c
```

```
2 # Ordner mit allen Dateien löschen
```

```
3 rm -r aufgabe1
```



Programme beenden

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



Arbeitsumgebung

- Unter Linux:
 - Kate, gedit, Eclipse cdt, Vim, Emacs,
- Zugriff aus der Windows-Umgebung über SSH (nur Terminalfenster):
 - Editor unter Linux via SSH:
 - mcedit, nano, emacs, vim
 - Editor unter Windows:
 - AVR-Studio ohne Projekt
 - Notepad++
 - Dateizugriff über das Netzwerk
 - Übersetzen und Test unter Linux (z.B. via Putty)



Übersetzen

- Wir Testen die Abgaben mit:

```
1 gcc -pedantic -Wall -Werror -std=c99 -D_XOPEN_SOURCE=500 -o ↵  
   ↳ printdir printdir.c
```

- spezielle Aufrufoptionen des Compilers
 - -pedantic liefert Warnungen in allen Fällen, die nicht 100% dem verwendeten C-Standard entsprechen
 - -Wall Warnt vor möglichen Fehlern (z.B.: `if(x = 7)`)
- diese Optionen führen zwar oft zu nervenden Warnungen, helfen aber auch dabei, Fehler schnell zu erkennen.
- -std=c99 Setzt verwendeten Standard auf C99
- -Werror wandelt Warnungen in Fehler um
- -D_XOPEN_SOURCE=500 Fügt unter anderem die POSIX Erweiterungen hinzu die in C99 nicht enthalten sind
- -o print Die Ausgabe wird in die Datei print geschrieben.
Standardwert: a.out



Manual Pages

- 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)`

```
1 # man [section] Begriff  
2 man 3 printf
```

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



Inhalt

Linux

Fehlerbehandlung

Bibliotheksfunktionen

Kommandozeilenparameter

Aufgabe: cworld

Anhang



Fehlerursachen

- Fehler können aus unterschiedlichsten Gründen im Programm auftreten
 - Systemressourcen erschöpft
 - ⇒ `malloc(3)` schlägt fehl
 - Fehlerhafte Benutzereingaben (z.B. nicht existierende Datei)
 - ⇒ `fopen(3)` schlägt fehl
 - Transiente Fehler (z.B. nicht erreichbarer Server)
 - ⇒ `connect(2)` schlägt fehl



Fehlerbehandlung

- Gute Software erkennt Fehler, führt eine angebrachte Behandlung durch und gibt eine aussagekräftige Fehlermeldung aus
- Kann das Programm trotz des Fehlers sinnvoll weiterlaufen?
- Beispiel 1: Ermittlung des Hostnamens zu einer IP-Adresse für Log
 - ⇒ Fehlerbehandlung: IP-Adresse im Log eintragen, Programm läuft weiter
- Beispiel 2: Öffnen einer zu kopierenden Datei schlägt fehl
 - ⇒ Fehlerbehandlung: Kopieren nicht möglich, Programm beenden
 - ⇒ Oder den Kopiervorgang bei der nächsten Datei fortsetzen
 - ⇒ Entscheidung liegt beim Softwareentwickler



Fehler in Bibliotheksfunktionen

- Fehler treten häufig in Funktionen der C-Bibliothek auf
 - erkennbar i.d.R. am Rückgabewert (Manpage!)
- Die Fehlerursache wird meist über die globale Variable `errno` übermittelt
 - Bekanntmachung im Programm durch Einbinden von `errno.h`
 - Bibliotheksfunktionen setzen `errno` nur im Fehlerfall
 - Fehlercodes sind immer >0
 - Fehlercode für jeden möglichen Fehler (siehe `errno(3)`)
- Fehlercodes können mit `perror(3)` und `strerror(3)` ausgegeben bzw. in lesbare Strings umgewandelt werden

```
1 char *mem = malloc(...); /* malloc gibt im Fehlerfall */
2 if(NULL == mem) {        /* NULL zurück */
3     fprintf(stderr, "%s:%d: malloc failed with reason: %s\n",
4         __FILE__, __LINE__-3, strerror(errno));
5     perror("malloc"); /* Alternative zu strerror + fprintf */
6     exit(EXIT_FAILURE); /* Programm mit Fehlercode beenden */
7 }
```



Erweiterte Fehlerbehandlung

- Signalisierung von Fehlern normalerweise durch Rückgabewert
- Nicht bei allen Funktionen möglich, z.B. `getchar(3)`
- Rückgabewert `EOF` sowohl im Fehlerfall als auch bei End-of-File
- Erkennung im Fall von I/O-Streams mit `ferror(3)` und `feof(3)`

```
1 int c;
2 while ((c=getchar()) != EOF) { ... }
3 /* EOF oder Fehler? */
```

```
1 int c;
2 while ((c=getchar()) != EOF) { ... }
3 /* EOF oder Fehler? */
4 if(ferror(stdin)) {
5     /* Fehler */
6     ...
7 }
```



Kommandozeilenparameter

```
1 ...
2 int main(int argc, char *argv){
3     strcmp(argv[argc - 1], ... )
4     ...
5     return EXIT_SUCCESS;
6 }
```

■ Übergabeparameter:

- main() bekommt vom Betriebssystem Argumente
- argc: Anzahl der Argumente
- argv: Vektor aus Strings der Argumente (Indices von 0 bis argc-1)

■ Rückgabeparameter:

- Rückgabe eines Wertes an das Betriebssystem
- Zum Beispiel Fehler des Programms: return EXIT_FAILURE;



Inhalt

Linux

Fehlerbehandlung

Aufgabe: cworld

Aufgabenstellung
Allgemeine Hinweise
Hinweise zu malloc

Anhang



Aufgabe: cworld

■ Ausgabe von "Hello World" unter Verwendung von malloc und strcpy

- Allokieren von Speicher für Zeichenfolge (malloc(3), strlen(3))
- Kopieren der Zeichenfolge in den allokierten Speicher (strcpy(3))
- Ausgabe (printf(3)):
Der Inhalt von pCH lautet: "<Inhalt von pCH>". pH zeigt auf <Adresse von pH>; pCH zeigt auf <Adresse von pCH>;
- Freigabe des allokierten Speichers (free(3))

■ Behandlung von Fehlern (Ausgabe über stderr)



Allgemeine Hinweise

■ Nur soviel Speicher anlegen wie notwendig

■ Fehlerüberprüfung von malloc

```
1 char* s = (char *) malloc(strlen(...) + 1);
2 if(s == NULL){
3     perror("malloc");
4     exit(EXIT_FAILURE);
5 }
```

■ Fehlerüberprüfung von printf (ab jetzt) nicht mehr notwendig

■ Formatierungsstrings: %s, %d, %c, %p, ...

■ Dereferenzierungsoperator: *

■ Addressoperator: &



Hinweise zu malloc

■ malloc()

```
1 void *malloc(size_t size);  
2 void free(void *ptr);
```

- malloc() reserviert mindestens size Byte Speicher
- Der Speicher muss mit free wieder freigegeben werden

■ Was ist ein Segfault

⇒ Zugriff auf Speicher der dem Prozess nicht zugeordnet ist
≠ Speicher der reserviert ist



Inhalt

Linux

Fehlerbehandlung

Aufgabe: cworld

Anhang

Arbeiten im Terminal
Debuggen



Arbeiten im Terminal

■ Navigieren/Kopieren:

```
1 cp /proj/i4spic/pub/aufgabeX/vorgabe.h /proj/i4spic/<login>/↵  
   ↳ aufgabeX  
2 # oder  
3 cd /proj/i4spic/<login>/aufgabeX  
4 cp /proj/i4spic/pub/aufgabeX/vorgabe.h .
```

■ Kompilieren:

```
1 gcc -pedantic -Wall -Werror -std=c99 -D_XOPEN_SOURCE=500 -o ↵  
   ↳ printdir printdir.c
```

■ Bereits eingegebene Befehle: Pfeiltaste nach oben



Debuggen

```
1 gcc -g -pedantic -Wall -Werror -O0 -std=c99 -D_XOPEN_SOURCE=500
```

- -g: aktiviert das Einfügen von Debug-Symbolen
- gdb: Standard-Debugger
 - gdb ./a.out
- cgdb: „schönerer“ Debugger
 - cgdb --args ./a.out arg0 arg1 ...
- Kommandos
 - b(reak): Breakpoint setzen
 - r(un): Programm bei main() starten
 - n(ext): nächste Anweisung (nicht in Unterprogramme springen)
 - s(tep): nächste Anweisung (in Unterprogramme springen)
 - p(rint) <var>: Wert der Variablen var ausgeben

⇒ **Debuggen ist effizienter als Trial-and-Error!**

