

U4 4. Übung

- Benutzerumgebung
- Standard Ein-/Ausgabe
- Fehlerbehandlung
- Aufgabe 6

U4-1 UNIX/Linux Benutzerumgebung

1 Manual Pages

- 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)`

```
man [section] Begriff

z.B. man 3 printf
```
- Suche nach Sections: `man -f Begriff`
Suche von man-Pages zu einem Stichwort: `man -k Stichwort`

2 Übersetzen und Ausführen

■ spezielle Aufrufoptionen des Compilers

`gcc -pedantic` liefert Warnungen in allen Fällen, die nicht 100% dem ANSI-C-Standard entsprechen

`... -Wall` liefert in vielen evtl. zweifelhaften Situationen (die zwar korrekt sein könnten, aber häufig nicht sind) Warnungen

diese Optionen führen zwar oft zu nervenden Warnungen, helfen aber auch dabei, Fehler schnell zu erkennen:

```
gcc -pedantic -Wall -ansi -o trac trac.c
```

■ Beispiel zur Verwendung von `printf()` siehe Vorlesungsfolien

■ Um `printf` verwenden zu können, muss `stdio.h` eingebunden werden

```
#include <stdio.h>
```

■ Ausführen: `./trac ab ba`

U4-2 Standard Ein-/Ausgabe

■ Einzelnes Zeichen von Standardeingabe lesen: **getchar(3)**

```
#include <stdio.h>
int getchar();
```

- ◆ liefert das von der Standardeingabe **stdin** gelesene Zeichen
- ◆ bei Eingabeende den speziellen **int**-Wert **EOF**

■ Einzelnes Zeichen auf Standardausgabe ausgeben: **putchar(3)**

```
#include <stdio.h>
int putchar(int c);
```

- ◆ gibt das Zeichen **c** auf den Standardausgabekanal **stdout** aus

■ Die `stdio`-Bibliothek arbeitet zeilengepuffert

- ◆ gelesene Zeichen werden erst nach Abschluss der kompletten Zeile geliefert
- ◆ ausgegebene Zeichen werden erst zum nächsten Newline ausgegeben
- ◆ Ausnahme: Programmende bzw. End-of-File

1 Erkennen des Dateiendes

■ Eingabe von der Tastatur

- ◆ `Ctrl-D` bzw. `Strg-D` am Zeilenanfang
- ◆ in der Zeilenmitte durch zwei mal `Ctrl-D`
- ◆ Zeichen `Ctrl-D` hat ASCII-Code `0x04`
EOF ist aber nicht `0x04`
- ◆ `Ctrl-D` wird vom Tastaturtreiber des Betriebssystems erkannt
 - Betriebssystem schließt daraufhin den Eingabekanal für das Programm
 - beim nächsten Lesebefehl an das Betriebssystem (innerhalb von `getchar()`) meldet das BS, dass nichts mehr kommt
 - `getchar()` liefert daraufhin den Wert EOF (= -1)

1 Erkennen des Dateiendes

■ Lesen aus einer Datei (von der Platte)

- ◆ Eingabekanal z. B. mit `<Eingabedatei` umgeleitet
- ◆ Betriebssystem erkennt wenn das letzte Zeichen aus der Datei gelesen wurde
 - Betriebssystem schließt daraufhin den Eingabekanal für das Programm
 - beim nächsten Lesebefehl an das Betriebssystem (innerhalb von `getchar()`) meldet das BS, dass nichts mehr kommt
 - `getchar()` liefert daraufhin den Wert EOF (= -1)

U4-3 Fehlerbehandlung

- Fehler können aus unterschiedlichsten Gründen im Programm auftreten
 - Systemressourcen erschöpft
 - ➡ **malloc(3)** schlägt fehl
 - Fehlerhafte Benutzereingaben (z.B. nicht existierende Datei)
 - ➡ **open(2)** schlägt fehl
 - Transiente Fehler (z.B. nicht erreichbarer Server)
 - ➡ **connect(2)** schlägt fehl
 - ...
- 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 Logeintrag
 - ➡ Fehlerbehandlung: IP-Adresse im Log eintragen, Programm läuft weiter
 - ◆ Beispiel 2: Öffnen einer zu kopierenden Datei schlägt fehl
 - ➡ Fehlerbehandlung: Kopieren nicht möglich, Programm beenden

1 Fehler in Bibliotheksfunktionen

- Fehler treten häufig in Funktionen der C-Bibliothek auf
 - erkennbar i.d.R. am Rückgabewert (Manpage!)
- Die Fehlerursache wird über die globale Variable **errno** übermittelt
 - Fehlercode für jeden möglichen Fehler (siehe **errno(3)**)
 - Der Wert **errno=0** bedeutet Erfolg, alles andere ist ein Fehlercode
 - Bibliotheksfunktionen setzen **errno** im Fehlerfall
 - Bekanntmachung im Programm durch Einbinden von **errno.h**
- Fehlercodes können mit den Funktionen **perror(3)** und **strerror(3)** ausgegeben bzw. in lesbare Strings umgewandelt werden

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

2 Fehlerbehandlung Reloaded

- Signalisierung von Fehlern normalerweise durch Rückgabewert
- Nicht bei allen Funktionen möglich, z.B. **getchar(3)**

```
int c;
while ((c=getchar()) != EOF) { ... }

/* EOF oder Fehler? */
```

- Rückgabewert `EOF` sowohl im Fehlerfall als auch bei End-of-File

U4-3 Fehlerbehandlung Reloaded

- Erkennung im Fall von I/O-Streams mit **ferror(3)** und **feof(3)**

```
int c;
while ((c=getchar()) != EOF) { ... }

/* EOF oder Fehler? */
if(ferror(stdin)) {
    /* Fehler */
    ...
}
```

U4-3 Fehlerbehandlung Reloaded

- Nicht in allen Fällen existieren solche Spezialfunktionen
- Allgemeiner Ansatz durch Setzen und Prüfen von `errno`

```
#include <errno.h>
int c;
while (errno=0, (c=getchar()) != EOF) {
    ... /* keine break-Statements in der Schleife */
}
/* EOF oder Fehler? */
if(errno != 0) {
    /* Fehler */
    ...
}
```

- `errno=0` *unmittelbar* vor Aufruf der problematischen Funktion
 ➔ `errno` wird nur im Fehlerfall gesetzt und bleibt sonst evtl. unverändert
- Abfrage der `errno` *unmittelbar* nach Rückgabe des pot. Fehlerwerts
 ➔ `errno` könnte sonst durch andere Funktion verändert werden

U4-4 Aufgabe 4

- Translate-Programm ähnlich UNIX `tr(1)`
- Liest einzelne Zeichen von der Standardeingabe
- Ersetzt diese ggf. durch andere Zeichen
- Gibt die Zeichen auf der Standardausgabe aus
- Aufrufbeispiel:

```
mike@milk[~] ./trac ab ba
Abba Hallo
Aaab Hbllo
spicboard
spicaobrd
```

1 Notwendiges Vorwissen

- Siehe Vorlesungsskript aus dem SS2008, Kapitel F
- C-Strings
- Argumenten-Vektor *argv*
- Arrays/Felder

2 Lernziele

- Umgang mit Zeigern und Feldern
- Verwertung von Kommandozeilenparametern
- Einfache Verwendung der Standard E/A-Kanäle
- Fehlerbehandlung

3 Programmaufbau

- Abbildungsfeld mit einem Eintrag pro möglichem Zeichen
- ASCII-Code eines Zeichens dient zur Indizierung des Arrays
- Der entsprechende Feldeintrag enthält das Zielzeichen
☞ wird ein Zeichen nicht verändert, so steht dort das Zeichen selbst
- Beispiel: (Aufruf: `./trac ab ba`)
 - ◆ `charmap[(int) 'a'] = 'b';`
 - ◆ `charmap[(int) 'c'] = 'c';`