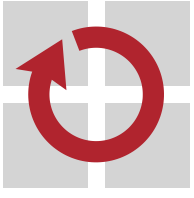
Übungen zu Systemnahe Programmierung in C

Abschnitt 10.4: Fehlerbehandlung

29.06.2020

Tim Rheinfels
Benedict Herzog
Bernhard Heinloth

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



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Fehlerursachen

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

Fehlerbehandlung

- Gute Software:
  - Erkennt Fehler
  - Führt angebrachte Behandlung durch
  - Gibt aussagekräftige Fehlermeldung aus

- Kann ein Programm trotz Fehler sinnvoll weiterlaufen?

  Beispiel 1: Ermittlung des Hostnamens zu einer IP-Adresse, um beides in eine Logdatei einzutragen

  ⇒ IP-Adresse ins 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 `libc` Funktionen auf
  - Erkennbar i.d.R. am Rückgabewert (Manpage)
  - Fehlerüberprüfung essentiell
- Fehlerursache steht meist in `errno` (globale Variable)
  - Einbinden durch `errno.h`
  - Fehlercodes sind `> 0`
  - Fehlercode für jeden möglichen Fehler (siehe `errno(3)`)
- `errno` nur interpretieren, wenn Fehler signalisiert wurde
  - Funktionen dürfen `errno` beliebig verändern

    $\Rightarrow$ `errno` kann auch im Erfolgsfall geändert worden sein

Fehler in Bibliotheksfunktionen

- Fehlercodes ausgeben:
  - `perror(3)`: Ausgabe auf `stderr`
  - `strerror(3)`: Umwandeln in Fehlermeldung (String)

Beispiel:

```
01 char *mem = malloc(...);
02
03 // Fehlerfall
04 if(NULL == mem) {
05   fprintf(stderr, "%s:%d: malloc failed with reason: %s\n",
06       __FILE__, __LINE__-5, strerror(errno));
07   //alternativ: perror("malloc");
08
09   exit(EXIT_FAILURE);
10 }
```

Erweiterte Fehlerbehandlung

- Signalisierung durch Rückgabewert nicht immer möglich
- Rückgabewert EOF: Fehlerfall **oder** `End-Of-File`

```
01 int c;
02 while ((c=getchar()) != EOF) { ... }
03 /* EOF oder Fehler? */
```

- Erkennung bei I/O Streams: `ferror(3)` bzw. `feof(3)`

```
01 int c;
02 while ((c=getchar()) != EOF) { ... }
03 /* EOF oder Fehler? */
04 if(ferror(stdin)) {
05   /* Fehler */
06   ...
07 }
```