System-Level Programming

23 Supplements: Error Handling

Peter Wägemann

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)

Summer Term 2025

http://sys.cs.fau.de/lehre/ss25

Error Handling

- Nearly every system call or library call can fail.

  $\Rightarrow$ **Error handling is inevitable!**

- Goal:

  No program should crash without an error message!

Error Handling

- Approach
  - Always check the return value of system/library calls
  - In the case of an error (usually indicated by -1 or `NULL`): error code is stored in the global variable `errno`
- Error message can be printed to `stderr` with the function `perror`:

```
#include <errno.h>
void perror(const char *s);
```

- Check temporary results for plausibility

```
#include <assert.h>
void assert(int condition);
```

If `condition` is not “true”, the program is aborted with an error message.

Error Handling (continued)

- Error handling has to be adapted to the context:
  - Errors due to user errors
    (e. g., user inputs wrong file name or wrong URL)
    - notify the user about the error
    - allow a new input
    - **repeat** the failed part of the program
  - Errors due to lack of resources
    (e. g., memory or disk is full)
    - notify the user about the error
    - allow the user to “clean up”
    - **repeat** the failed part of the program
  - Programming error
    (e. g., computed result is wrong)
    - output an error message
    - **abort** program
  - ...

Error Handling – Example

```
...

assert(argv[1] != NULL);

/* Open file for writing. */
FILE *fp = fopen(argv[1], "w");
if (fp == NULL) {
    perror(argv[1]);
    exit(EXIT_FAILURE);
}

/* Write to file. */
...

/* Close file. */
int ret = fclose(fp);
if (ret == EOF) {
    perror("fclose");
    exit(EXIT_FAILURE);
}

...
```

```
~> ./test
test.c:9: main: Assertion
          'argv[1] != NULL' failed.
~>
```

```
~> ./test /etc/shadow
/etc/shadow: Permission denied
~>
```

```
~> ./test hallo.txt
fclose: Quota exceeded
~>
```