Common programming mistakes in C

Systems Programming

Michael Sonntag

Institute of Networks and Security

Not using all available tools

- GCC compiler options
  - **`-Wall`** show all warnings
  - **`-ansi -pedantic`** strict ANSI C conformance
- Dynamic analysis tools: at run-time
  - Uninitialized variables, memory violations…
  - **`valgrind [options] program [arguments]`**
    - **`--track-fds=yes`** show open files when program ends
    - **`--leak-check=full`** show allocated memory when program ends
    - **`--malloc-fill=0xA`** initialize allocated memory with chosen value
    - **`--free-fill=0xB`** fill memory with chosen value on free
  - Memory guards in OS
- Static analysis tools: source code
  - Type safety, reachability, unused results, coding style
  - Splint: http://www.splint.org

Leaving variables uninitialized (1)

- Results
  - □ At best: program crashes
  - □ Mostly: unusual behavior, hard to debug
- Not initialized automatically
  - □ Local variables
  - □ Memory from heap (exception: calloc)
- Manually initializing with safe values
  - □ Numerical types: **`0`** or **`0.0`**
  - □ Pointers: **`NULL`**
  - □ Arrays: e.g. **`int arr1[5] = {0}, arr2[] = {10, 20, 30};`**
  - □ Strings
    - ● **`char str1[] = "Hi", str2[10] = "", str3[5] = {0};`**
    - ● **`char *str4 = "Hi";`**
  - □ Structures, unions
    - ● **`struct s { int a; char b; float c; };`**
    - ● **`struct s instance1 = {0, 0, 0.0}, instance2 = {0}`**

Leaving variables uninitialized (2)

- Initializing allocated memory
  - Automatically on allocation
    - **`calloc`**: sets all bits to 0
    - **`int *mem = calloc(20, sizeof(int));`**
  - Later, whenever required
    - **`memset`**: set region of memory to chosen value
    - **`memset(mem, 5, 20 * sizeof(int));`**

- Beware
  - Binary 0 works as 0, 0.0 and NULL…

    … but may have other meaning for other data types!

Assuming the program is bug-free, if it runs at all somehow

- Problems can go unnoticed
  - Off-by-one memory violations
  - Un-terminated strings
  - Accessing stack at wrong position
    - Missing parameters for **`printf`**
    - Accessing **`argv`** without checking **`argc`**
- May not crash program
- May lead to strange behaviour or crash later
- Possible remedies
  - Defensive programming
    - Checking every parameter
    - Checking index ranges
    - Checking numeric ranges before calculations
    - …
  - Checker tools

Using feof() for detecting EOF

- Symptoms
  - Last character / word / line appears to be read twice
  - Invalid data (return value EOF) processed
- Reason
  - I/O functions usually reach end-of-file and return normal data
  - Set end-of-file flag only after next call
- Always check return value of I/O functions
- Use feof() only to check whether it’s really end-of-file
- Wrong

```
int c;
while (!feof(file)) {
   c = getc(file);
   ...
}
```

- Correct

```
int c;
   while ((c = getc(file)) != EOF) {
      ...
   }
   if (feof(file)) { ... }
   else if (ferror(file)) { ... }
```

Not checking return values of library functions

- Remember: No exceptions like in Java
- Only chance: Check return value every time immediately
- What to check for:
  - □ -1: **`mktime`**, **`system`**…
  - □ NULL: **`malloc`**, **`fopen`**, **`strdup`**, **`localtime`**, **`bsearch`**…
  - □ Less than the requested amount of data was processed: **`scanf`**, **`fread`**, **`fwrite`**…
  - □ Special constants: **`EOF`**
  - □ In general: Read the man page to find out!
- Only then
  - □ Look at **`errno`** (e.g., via **`perror`** or **`strerror`**)
  - □ Use special functions like **`feof`**

More common mistakes

- comp.lang.c FAQ
  - Clarifies many misconceptions
  - Solutions to common mistakes
  - http://www.c-faq.com

JOHANNES KEPLER
UNIVERSITÄT LINZ

INSTITUTE
OF NETWORKS
AND SECURITY

https://www.ins.jku.at

THANK YOU FOR YOUR ATTENTION!

Michael Sonntag
michael.sonntag@ins.jku.at
+43 (732) 2468 - 4137
S3 235 (Science park 3, 2nd floor)