CS453 - Coding standards

Overview

This document applies to all projects and homework turned in during the semester. Each project has a specification document that describes the functionality and features that the program must have. In addition to what is described in the specification document your program must follow the guidelines as set forward below. Each section is clearly marked with what is expected of your code, point deductions will be assessed for any violations.

Most of you are graduating this year and will be required to adopt a coding standard at your new place of employment. This is a great way to get used to following a strict set of guidelines so you are better prepared for industry.

The starter code that we push out to you may violate some of the guidelines below. It is your responsibility to bring the code up to these standards. If your instructor does not intend for you to fix any pre-existing issues it will be clearly marked by a comment in the source. **Don't assume that all code pushed out by your instructors is perfect!** Learning how to read and fix other engineers code is a very import skill to learn. Building an entire solution from scratch alone is very rare in industry.

What is the purpose of this?

Every well managed software team will have a set of coding guidelines that each employee must follow. Some teams are more strict than others for various reasons. For example someone that is writing software to control a nuclear power plant, the space shuttle, or a subsonic jet will have extremely pedantic, sometimes painful, restrictions on how they write code. Other teams who may have looser restrictions may put out code faster at the expense of letting bugs slip through to the consumer. These bugs could be harmless or they may cause disastrous, sometimes fatal results.

Coding guidelines for C

We will use the Linux coding guide as a base set with a few additions for this class. The Linux coding style is based on the coding style used in the Kernighan & Ritchie's C Programming Language book (usually abbreviated as K&R style). If you ever want to (or need to) submit a patch to the Linux kernel you are required to follow these coding guidelines. Your patch will be rejected for trivial reasons such as indentation or naming conventions. Make sure and read through this document and thoroughly understand what is required. Lucky for us, the Linux code style guide is short and we will be focusing on

sections 1 - 8 during this class. Below are direct links to each section. You are responsible for reading and understanding all the information in these sections. If you have any questions please post it to the class discussion board.

1. Indentation - K&R style with one tab stop as indent (set to 8 spaces in the editor)
2. Breaking long lines and strings - 80 char limit unless it substantially hinders reading
3. Placing Braces and Spaces - K&R style
4. Naming - Pretty standard C naming conventions
5. Typedefs - You should never typedef in this class except in the library wrapper project.
6. Functions - Functions should be short and sweet with 1 defined role just like in any other language
7. Centralized exiting of functions - you can use GOTO statements for structured error handling. GOTO is used extensively in the kernel.
8. Comments - This is the hardest one for new developers. You must learn to comment about the WHY of your code not the HOW. This is something that will get easier with experience.
9. Luckily for you a lot of these rules can be handled automatically with your IDE or with an external tool so you don't have to even think of it. You just have to make sure and run the tool before you submit your code.

Secure programming in C

We will be using the following set of compiler flags this semester (you will set these in your Makefile if they are not already set for you). We will be using the gnu89 standard because that is what is required to build the Linux kernel. We will add in the -Wextra so we get more help from the compiler. It may be painful but will make your code more bullet proof (which will result in a higher grade) in the long run.

```
gcc -Wall -Wextra -Wpointer-arith -Wstrict-prototypes -std=gnu89  -O2
```

There are thousands of pages written about secure coding in C. While it would be awesome to know everything, it is simply not possible in a 16 week course. Read chapter 1 of the C programming language defensive coding document from the Fedora security team. The chapter is short and details some very important issues regarding secure programming in C such as absolutely banned interfaces, functions to avoid, good string processing functions, strncpy issues, strncat issues, and memory allocations.

In particular, we will follow the following five principles:

1. Proper use of global variables - must be declared static and the usage minimized
2. All string handling functions should be of the bounded type
3. All variables must be initialized at declaration time.
4. After a call to free() you must set the pointer to NULL
5. There should be no read/write errors when using Valgrind

Valgrind

Valgrind is your friend and will help you detect and eliminate a lot of errors in your program. Run Valgrind often! Don't wait until you are all done to run Valgrind or you will have a very difficult time fixing all the errors. **You should be running Valgrind as often as you run your tests or better, as often as you compile your code.** That way you will find issues early and often.

General programming guidelines

In addition to the issues specific to the C programming language as detailed in the sections above, don't forget about general programming standards that apply to ALL programs. All projects submitted should have some form of automatic testing. Automatic testing can be either unit tests or system tests. Manual testing is rarely sufficient to ensure your program is correct. Here are some highlights that I will be looking for this semester.

1. Does each function check to make sure the arguments are valid? For example, checking for NULL pointers.
2. Does each function have a documentation header that describes the purpose of said function.
3. Your main function should just process arguments, set any flags and hand off responsibility. No logic should be present in the main function.