CS29206 Systems Programming Laboratory
Spring 2024

Introduction

Abhijit Das
Pralay Mitra

Where you stand now after PDS and Algo Lab

- You know the syntax of the programming language C.
- You know how to write a C file, compile it, and execute it.
- Your development process
  - Your codes are tiny (a few hundred lines at most).
  - If you are stuck, you blame the compiler or the machine.
  - Your teachers and TAs typically help you by using the printf debugger.
  - You stop at the point when you are satisfied with the correctness of the code.
  - Later, you blame that the TA's compiler or machine is buggy.
- The purpose of development
  - A basic motivation for writing your codes is getting a good grade.
  - Your codes are meant only for you and your teachers and TAs.
  - Your codes are (hopefully) not used by anybody else in the world.
  - You also do not use other people's codes (hopefully again).
  - Nobody else (not even you) is interested in the codes after the course is over.

The reality is quite different

- A software project typically involves millions of lines of codes.
- You do not develop the code yourself alone.
- You do not write everything in a single C file.
- Your codes are to be used by:
  - The rest of the development team
  - Other development teams
  - The end users
  - Not really by you yourself
- Your codes justify your salary and reputation, so you need to make your codes:
  - as correct as you can
  - as bug-free as you can
  - as efficient as you can
- You do not build everything from the scratch, so you need to use other people's codes.

C, the C compiler, and the compilation process

- Learning only the C syntax at the PDS level is not enough.
- You have developed better familiarity in Algo Lab, still it is not complete.
- There is more to learn.
- You need to know how the compiler works.
  - How to compile codes distributed across multiple files.
  - How to use other people's code in your code.
  - Copy-paste from other people's code (if you have the source at all) is a bad idea.
- You need to know how the compilation process works.
- You do not always write codes with a main function.
- If you make a small change in a part of your code, you do not need to recompile everything.

Debugging

- Users are the victims of buggy codes.
- Developers are the culprits.
- The printf debugger is clumsy and very difficult to handle in large codes.
- You need to know the art of debugging.
- If you suspect your foo() function to be faulty, inspect it line by line.
- C does not support range checking on arrays.
  - Memory corruption may lead to strange results
  - Segmentation faults are some of your deadliest enemies
  - Buffer overflow may lead to security problems
- You need to learn the sources of the problem, so that you can repair them.
- You have no option of blaming the user's machine or compiler.

Efficiency

- A good algorithm is a first necessity.
- Sloppy implementations may make a good algorithm terribly slow.
- You need to identify the bottlenecks in your codes.
  - Which functions take unnecessarily large times.
  - Which functions are called more often than needed.
- A profiling of your codes is needed to figure these out.

Gradual development process

- You do not develop a million-line code overnight.
- You may have thousands of versions of the code, developed by multiple users.
- The same pieces of codes may be handled by multiple developers.
- Would you retain a copy of the entire project after every single change?
- How do you navigate through the history of changes?
- You need to know how version controlling can be done methodically.

Using the specialists

- Text processing
- Database processing
- There are excellent utilities, each perfected to handle specific jobs.
- You need to know some of these utilities.
- Integrating these utilities in your C codes is oftentimes not straightforward.
- Writing C codes for these is usually a meaningless waste of resources.

The master wrapper

- This is the shell.
- In the simple form, it is just a command interpreter.
- But you can program it.
- You can instruct a shell how to exploit:
  - Your C executables
  - The specialists
  - Itself
- All from the same platform.
- The shell specializes in several tasks like handling your file system.
- The shell is also useful for system administration.