CS29206: Systems Programming Lab

Autumn 2025

Prof. Pralay Mitra

Prof. Arobinda Gupta

Prof. Rajat Subhra Chakraborty

Course Information

- Attendance is mandatory.
  - Only BCRTH medical certificate is accepted for routine medical issues.
  - Please feel free to contact the teachers beforehand if you have any other big reasons for missing a lab.
- 40%-50% marks on two lab tests (written)
- Remaining marks on assignments
  - 7-8 small in-class assignments (1-2 marks each)
  - 2-3 larger in-class assignments (6-8 marks each)

What you have done so far

- Learnt the syntax of C programming language
- Learnt how to write a C file, compile it, and execute it in Linux
- Learnt a few basic commands in Linux in the process
- Your program development experience so far
  - Small assignments, tiny codes
  - You try to write correct code, debug using printf's
  - If still stuck, you blame the compiler or the machine ☺
  - Your forget about your code once the course is over
  - Main motivation is to get good grades for a course ☺

Characteristics of programs you have written

- Written for your assignments or other small problems
- Small in size (a few hundred lines is also very very small)
- Entire code written in one file, must have a main() function
- Debugging with putting printf's, recompiling, running again
- Written only by you (hopefully ☺)
- No one else in the world uses your program (again, for assignments, hopefully ☺)
- Even you re-using parts of your program for some other program needs cut-and-paste of source code
- Goal is to write correct programs, sixe or running time is not an issue
- Your programs disappear when course is over, or you lose interest in them

The real world of software development

- Large projects with groups of people working together
- Overall software has many many features/services
- Complex software with hundreds of thousands to many millions of lines of codes spread over hundreds/thousands of files
- You write only part of it
- There are some common parts that are written by many people
- Other people use what you write (ex. calls functions in your code)
- You use what other people write
- Some codes are used by a lot of people (code for some common tasks)
- Your code has to run on customer's machines, not just yours
- Your code stays when you leave the project/company

Some of the problems/issues that arise

- How do you organize the large number of files?
  - What are some of the rules/guidelines for breaking up your program into the different files?
  - How do you organize the files to make sense of what is where?
- How do you compile the files?
  - Would you just do "cc ….." and specify all the .c files as you have done so far? How do you even keep track of all the files?
  - What if you changed only a few files? Would you have to compile all of them again?
  - How to handle dependencies between different parts?
- How do you debug when your program does not run correctly?
  - Debugging with printfs is very inefficient, requires you to recompile your program every time
  - What information can be helpful in debugging?

- How can codes be reused?
  - Everyone should not have to write them again
  - Cut-and-paste is not a good option, creates too many copies that are hard to keep consistent
- How can we make it easy for others to use your code and you to use theirs?
- You and I write to a common file. How do I ensure you will not overwrite something I wrote (and vice-versa)? Do we need to talk always before changing the file?
- Your code was working 10 days back. You added some more code, not working now. How do you know exactly what changes you did in these 10 days to isolate the problem faster? Do you need to keep a version with you every day say?

- Someone else may look at your code and will need to change/maintain it. How can I make my code more understandable?
  - This is a critical requirement (not that others are not important). In college, you write code that you understand (maybe also your TA/teacher). In the real world, you have to write code that others understand.
- Your code runs correctly, but is too slow (compared to what you need). How do you know exactly what part(s) is/are slowing it down?

What this course aims to do

- Introduce you to some tools that help you deal with some of these issues
- Important to remember what issue you are addressing and why you need to address it, rather than just learning the tools
  - We will only do some features of the tools, the full feature set for some of them are quite expensive
  - You will encounter other tools in future that may be better/easier to use (many of them are just GUI-based wrappers around these tools only). Use them by all means then.
  - But important to instill some good habits early on and practice it for code you write as you get into more serious programming.
- Some of these issues will also become more clear as you take other courses, this lab wants to start you off

What we will cover

- Introduce you to some basic Linux commands
- Basics of the GNU C compiler gcc
  - Compiling multiple files into a single executable file
  - Creating/Using static and dynamic libraries for reusing code
  - Options for better warning/error reporting, macros for conditional compilation
- Basics of makefiles
  - Creating/using makefiles to organize file dependencies for compiling large projects
- Basics of the GNU debugger gdb
  - Basic debugging without having to recompile programs
- Basics of the profiling tool gprof
  - Debugging performance issues

- Basics of valgrind
  - Debugging memory leaks, buffer overflow errors
- Introduction to regular expressions and pattern matching using grep
- Introduction to text processing using awk
- Shell programming
  - More like learning a programming language
  - Helps you to use basic Linux commands to create new commands, automate repetitive tasks etc.
  - Very useful to configure and manage your Linux system. Integral to learn for linux system administration