CSCI 8530
Advanced Operating Systems

Instructor: Dr. Pei-Chi Huang
Department of Computer Science
The University of Nebraska at Omaha

Who am I?

- Education
  - Ph.D. in CS from UT Austin
- Teaching and Tutoring
  - Computer Systems, Computer Architecture, Cyber-physical Systems, Elements of Software Design, Python Language
- Service
  - CS Faculty
- Research
  - Cyber-Physical Systems with Machine Learning
  - Real-time Computing and Scheduling Algorithms
  - Wireless Communication/Networking Systems

https://unocps.github.io

Acknowledgments and Copyright

- In the preparation of this course, I used materials from Dr. Stanley Wileman, Dr. Alison N. Norman, Mary Eberlein, Mike Scott, Kathryn McKinley, Shyamal Mitra, Calvin Lin, Charlie Garrod, Jennifer Brown, Al Mok, Mike Dahlin, Emmett Witchel, Lorenzo Alvisi, Maria Jump, Michael Walfish, Jerry Breecher, Andy Wang, John Bell, and Zero Assurance Recovery.
- Special thanks to Dr. Wileman's guidance and assistance.

Introductions and Expectations

Confession

- The phrase “Introductions and Expectations” was stolen – specifically from the ACM SIGOPS (Special Interest Group on Operating Systems) series on “How To Write An Operating System”.
- However, the notes made there on what is expected of a student are appropriate as well to students in this course.
- So – to some extent – take the following notes with a “grain of salt.” But keep in mind that the observations were made by folks presenting a course on operating systems to a “much larger” audience.

The Goal (Part 1)

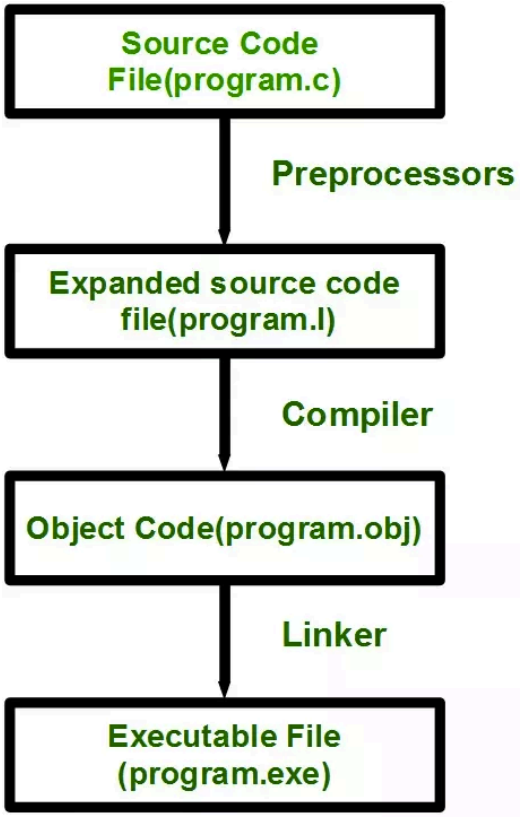
- “Writing an operating system is a bit harder than writing a conventional program, by its nature: you're actually writing the code that works the machine and runs other people’s code.”
- ”The lack of debuggers, libraries, and standard code makes it the black art that it has become. You can’t debug using traditional methods because you're writing at a lower level than the debugger itself (even the debugger calls the operating system).”
- “Libraries are essentially non-existent because each operating system treats the machine in a different manner, and it would be extremely difficult to write the libraries for general-purpose use.”

The Goal (Part 2)

- “In fact, the only thing we can really do is provide a structure for development and some example working code.”
- Goal: We’ll be studying a real operating system at the code level to gain an understanding of what’s really involved in an operating system.

Prerequisites (Part 1)

- **"Programming experience**, preferably C or C++ – "
  - " Since systems programming is harder than conventional programming, it's assumed that if you want to write an operating system, you have at least some programming experience. "
- " You should know pointers, structs and the basic data types, control loops, and the preprocessor well, but don't worry about the standard libraries, ANSI or otherwise, since those are essentially irrelevant in systems programming (you'll most likely have to write these yourself in the later stages)."



Prerequisites (Part 2)

- **“Knowledge of the architecture** - Knowing the assembly language and the architecture of the machine you’re developing for is a definite bonus.’’
- If you don’t know these already, you can pick them up along the way, but it makes development that much harder.
  - x86 Assembly Guide

Prerequisites (Part 3)

- **“A Machine** - You need at least one machine that can be rebooted frequently for testing; preferably, you have another machine for editing and/or compiling. …”
- We will be using Intel Galileo (Gen 2) boards to test the operating system and your work, and a Linux system for development. Usage details are still being determined.

Prerequisites (Part 4)

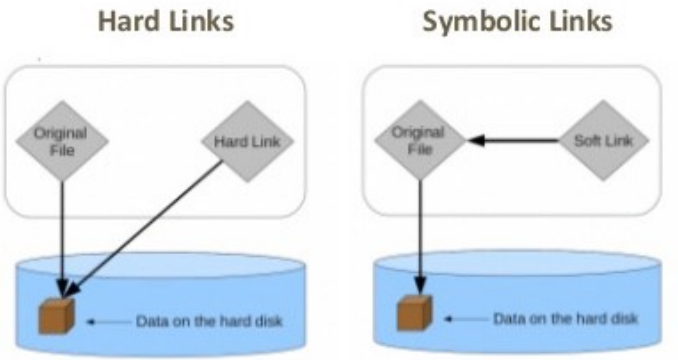
- **“Persistence and Mental Stability** - all the documentation on the processor that I found was either lacking this critical piece of information or was very misleading, close enough to being actually wrong (and this includes the manufacturer’s official documentation).
  - Be prepared to spend a good portion of your time beating your head against a wall.”
- But you have options – the instructor will hopefully safeguard your mental stability. Use email as necessary!

Prerequisites (Part 5)

- “That said, operating system development is some of the most frustrating and the most rewarding programming you may ever do. To know that your code is the only code running on the machine is a kind of power. It is at once sobering and comforting, knowing that any minor glitch could send it hurtling into a rebooting inferno, but that despite this the machine is still humming blithely along.”
- It’s difficult to add to that observation about the significance of OS in the entire spectrum of software used on computing systems. Simply put, the OS is the preeminent required software on a computer.

Prerequisites (Part 6)

- Concepts from a previous OS course:
  - Concurrent programming: you should have written a program that uses *fork* or the equivalent
  - Understanding of deadlock and race conditions
  - I/O: you should know the difference between standard library functions (e.g., *fopen*, *putc*, *getc*, *fread*, *fwrite*) and system calls (e.g., *open*, *close*, *read*, *write*)
  - File systems and hierarchical directories
  - Symbolic and hard links
  - File modes and protection



Prerequisites (Part 7)

- Understanding of runtime storage components
  - Segments (text, data, and bss) and their layout
  - Runtime stack, activation records, and argument passing
  - Basic heap storage management (free list)
- C programming
  - At least one nontrivial program
  - Comfortable with low-level constructs (e.g., bit manipulation and pointers)

Prerequisites (Part 8)

- Working knowledge of basic UNIX tools
  - Text editor (e.g., vi)
  - Compiler / linker / loader
  - Tar archives
  - Make and Makefiles
- Desire to learn

Course Scope

- This is a course about the design and structure of computer operating systems. It covers the concepts, principles, functionality, tradeoffs, and implementation of systems that support concurrent processing.

What We Will Cover

- Operating system design
- Functionality an operating system offers
- Major system components
- Interdependencies and system structure
- The key relationships between operating system abstractions and the underlying hardware (especially processes and interrupts)
- Implementation details

What You Will Learn

- Fundamental
  - Principles
  - Design options
  - Tradeoffs
- How to modify and test operating system code
- How to design and build an operating system

What We Will NOT Cover

- Comparison of large commercial and open source operating systems
- Description of features or instructions on how to use a particular operating system
- Survey of research systems and alternative approaches that have been studied
- Set of techniques for building operating systems on unusual hardware

Questions

1. What is the pwd command?
2. How do you insert comments in the command line prompt?
3. Write a command that will display all .txt files.
4. How can you append one file to another in Linux?
5. Explain how you can find a file using Terminal?
6. Explain how you can view the text file using Terminal?
7. How do you change permissions under Linux?
8. How to create symbolic link/hard link