Course Wrapup

Computer Operating Systems, Summer 2025

Instructors: Joel Ramirez Travis McGaha

TAs: Ash Fujiyama Sid Sannapareddy Maya Huizar

Poll Everywhere

pollev.com/tqm

- What did you learn in this course? Is there anything you wish we talked about more? Anything you wish we talked about less? Any Feedback in general?

Administrivia: Final Exam & End of Semester

- Final Exam; July 31st from 7pm to 9pm
  - Final Exam Policies posted on course website
  - Old exams & exam questions

- Final Exam review in recitation tomorrow and some in lecture next week

- PennOS Peer Evaluation Survey: Due Sunday August 3rd
  - Only submit after your PennOS Demo. Each groupmate submits individually and privately
  - You get a little PennOS Extracredit for completing the survey

Administrivia

- Some notes:
  - DO NOT mmap the entire File System. Only mmap the Allocation Table, the rest of the file system needs to be handled with lseek/write.
    - Do not keep the contents of the file in memory, it should be stored in the file
    - If your PennFat is killed with kill -9, your file contents should still be saved in disk
  - Advice for using gdb to debug
    - **`handle SIGUSR1 noprint nostop`**
      Makes it so that gdb doesn't report every time SIGUSR1 goes and interrupts you
  - (more on next slide)

Administrivia

- Some notes:
  - Reminder, you instead of just doing:

```
lseek(FAT_FD, offset, SEEK_SET);
write(FAT_FD, contents, size);
```

  you may need to do:

```
lseek(FAT_FD, offset, SEEK_SET);
write(FAT_FD, contents, size);
lseek(FAT_FD, offset, SEEK_SET);
write(FAT_FD, contents, size);
```

  - With the description of `setitimer()`, it just says that sigalarm is delivered to the process, not necessarily the calling thread. To make sure siglaram goes to the scheduler, you may want to make it so that all threads (spthread or otherwise) that aren't the scheduler call something like: **`pthread_sigmask(SIG_BLOCK, SIGALARM)`**
    - Which will block SIGALARM in that thread.

Administrivia

- If you are having issues with the scheduler not running you can try running
  - **`strace -e 'trace=!all' ./bin/pennos`**
  - You may have to install strace: **`sudo apt install strace`**
  - This will print out every time a signal is sent to your pennos
  - (Usual fix is the pthread_sigmask thing on the previous slide)

Lecture Outline

- **Course Wrap-up**

What have we been up to for the last 10 weeks?

- Ideally, you would have "learned" everything in this course, but we'll use red stars ☆ today to highlight the ideas that we hope stick with you beyond this course

Operating Systems: The Why

- The programming skills, engineering discipline, and knowledge you need to build a system
  1) Understanding the “layer below” makes you a better programmer at the layer above
  2) Gain experience with working with and designing more complex “systems”
  3) Learning how to handle the unique challenges of low-level programming allows you to work directly with the countless “systems” that take advantage of it

So What is a System?

- "A **system** is a group of interacting or interrelated entities that form a unified whole. A system is delineated by its spatial and temporal boundaries, surrounded and influenced by its environment, described by its structure and purpose and expressed in its functioning."
  - https://en.wikipedia.org/wiki/System
  - Still vague, maybe still confusing

- How can we apply this to the things we have done in this class?

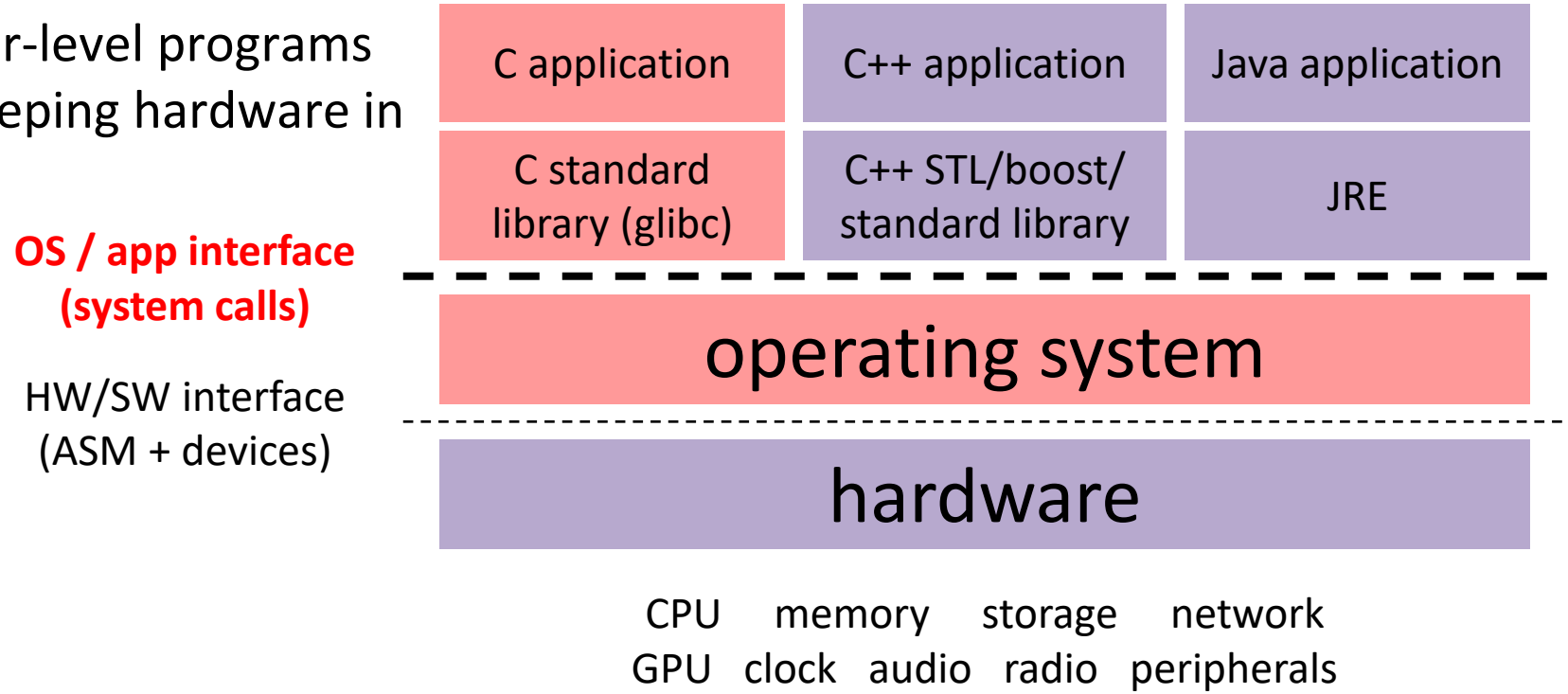
Software System

- Writing complex software systems is *difficult*!
  - Modularization and encapsulation of code
  - Resource management
  - Documentation and specification are critical
  - Robustness and error handling
  - Must be user-friendly and maintained (not write-once, read-never)

- **Discipline:** cultivate good habits, encourage clean code
  - Coding style conventions
  - Unit testing, code coverage testing, regression testing
  - Documentation (code comments, design docs)

The Computer as a System

- Modern computer systems are increasingly complex!
  - threads, processes, pipes, files, Buffered vs. unbuffered I/O, blocking calls, caches, virtual memory. And how all these topics affect and relate to one another
  - Support nice abstractions for user-level programs and keeping hardware in mind



Systems Programming: The What

- The programming skills, engineering discipline, and knowledge you need to build a system
  - **Programming:** C (& other languages)
  - **Discipline:** design, testing, debugging, performance analysis
  - **Knowledge:** long list of interesting topics
    - Concurrency, OS interfaces and semantics, techniques for consistent data management, …
    - Most important: a deeper understanding of the "layer below" and keeping this knowledge in mind to write better software.

Main Topics

- C
  - Low-level programming language
- Memory management & allocation
- System interfaces and services
- Concurrency basics – POSIX threads, synchronization
- Multi-processing Basics – Fork, Pipe, Exec
- Buffering, Caches, Locality
- Operating System Internals
  - File systems
  - Scheduling
  - Virtual Memory

Topic Theme: Abstraction

- C: `void*` to abstract away types for some functions (pthread_create, read, write, etc).
- abstract away details of interacting with system resources via system call interface (e.g. file descriptors and pids)
- The concept of processes and virtual memory to abstract away sharing hardware
- Read Write Locks and monitors abstract away their implementation of using a mutex & condition variable

- Nice abstractions minimize cognitive complexity and make it harder for users of the abstraction to fuck up.

Topic Theme: Data & Locality

- I/O to send and receive data from outside of your program (*e.g.*, disk/files, network, streams)
  - Linux/POSIX treats all I/O similarly
  - Takes a LONG time relative to other operations
  - Blocking vs. non-blocking (and the sin that is spinning)
- C: Memory model (Stack vs Heap)
- Buffers can be used to temporarily hold data
  - Buffering can be used to reduce costly I/O accesses, depending on access pattern
- Caching & Locality
  - Some memory is quicker to access than others
  - Hardware makes assumptions on your program's access patterns

Topic Theme: Allocating Resources

- It is often the tasks of a system to distribute/allocate a finite number of resources:
  - Scheduling algorithms allocate which threads can utilize the CPU
  - Memory allocation schemes (slab allocator, buddy algorithm)
  - Virtual Memory: allocating pages in physical memory
  - Caches: deciding what memory is in the cache.
  - File System: Allocating Blocks in file system

- These allocation schemes need to consider:
  - Efficient utilization of the resource that is being allocated
    - Fragmentation, fairness, minimize times we go to slower storage
  - Minimal overhead in the allocation scheme.
    - Time spent on the allocation is time not spent doing other things

Topic Theme: Concurrency

- Processes
  - Exec
  - Process Groups
    - Terminal Control
  - IPC
    - Pipe
    - Signals
- Threads
  - Synchronization
    - mutex
    - Condition variables
  - Deadlock
- Concurrency vs parallelism

MISSING Topic Theme: Society

- One flaw (among others) of this course is how we don't talk ENOUGH about how this relates to the rest of the world
  - These systems we build do not have to necessarily be "evil", but can often be used in those ways
  - We need to work and communicate with other people, even in CS.

- Actions:
  - Take Algorithmic Justice (CIS 7000) with Danaë Metaxa
  - Take Software Engineering (CIS 3500)
  - Join a community of people working on things that matter to you, (Unions or other organizations)
  - Join as a TA for 2400 or 54800 next year. We are trying to further integrate ethics.

Congratulations!

- Look how much we learned!

- Lots of effort and work, but lots of useful takeaways:
  - Debugging practice
  - Reading documentation
  - Tools (gdb, valgrind)
  - C familiarity
  - Concurrent Programing
  - Designing large systems
  - Working with others

- Go forth and build cool systems!

Future Courses

- Systems Courses
  - CIS 3990: Intermediate Computer Systems Programming
  - CIS 5050: Software Systems
  - CIS 5530: Networked Systems
  - CIS 5521: Compilers
  - CIS 5550: Internet and Web Systems
  - CIS 5500: Database and Information Systems
  - CIS 5470: Software Analysis
- Otherwise related courses
  - CIS 5600 Interactive Computer Graphics
  - CIS 5650 GPU Programming and Architecture
  - CIS 5510 Security
  - CIS 6010: GPGPU Programming and Architecture
- ESE Courses
  - ESE 3500:Embedded Systems
  - ESE 5190: Smart Devices
  - ESE 5490: Hardware/Software Co-Design for Machine Learning
  - ESE 6150: RoboRacer Autonomous Racing Cars

Thanks for a great semester!

- Special thanks to all the instructors before us (Both at UPenn and UW) who have influenced me to make the course what it is

- Huge thanks to the course TA's for helping with the course!

Thanks for a great semester!

- Special thanks to all the instructors before us (Both at UPenn and UW) who have influenced me to make the course what it is

- Huge thanks to the course TA's for helping with the course!

Thanks for a great semester!

- Thanks to you!
  - It has been another tough semester. Look at the state of Society ☺
  - Things are still a bit rough in the course as we change it.
    - Especially being adapted to the summer
  - You've made it through so far, be proud that you've made it and what you've accomplished!

- **Please take care of yourselves, your friends, and your community**

Ask Me Anything: PennOS or Otherwise