CIS3110 Class Transcript - January 10, 2025

Course Information and Logistics

Welcome everybody to Class #2. We now have the course outline available.

We have a large number of GTAs available for consultation through office hours who will also be running the labs. Please email course-related discussions to CIS3110@[university email] rather than emailing me directly, as my personal inbox is becoming unmanageable (I received 147 email messages on Monday alone).

Lab Information

The first lab is designed to ensure everyone can successfully work with the Linux.soc server environment. This is crucial because last year, many students in CIS2520 submitted code that didn't work on Linux.soc despite claiming they had developed it there.

A common point of confusion arises with VS Code:

- Option 1 (Correct): Log in via SSH to the Linux.soc server, edit code on the server, and use the command line on the server to run your code
- Option 2 (Problematic): Opening the explorer, attaching to the server using SSH, which brings files locally - students edit assignments that end up on their local machine while thinking they're on the Linux cluster

If you see a path like "C:\Program Files" in your terminal, you're not on the Linux cluster. Lab 1 is designed to make sure everyone can tell which environment they're working in.

You must ensure your code compiles and runs on the Linux.soc server where it will be graded. Even if your code compiles at home on your own Linux machine, we will be grading on the official server.

We'll be using `make` for compilation across related C files in one project. Please use the provided makefiles.

Course Assessment Update

The marking scheme has changed slightly since the draft version. The three assignments that are strongly OS-related are now worth 10.5% each, and I've reduced the weight of the memory management assignments.

The memory management assignment will involve building a tool similar to what's behind malloc - essentially building your own memory allocator that manages distribution and reclamation of memory within a process. This assignment only requires basic pointer knowledge, so you can start working on it early in the term.

Later assignments will cover semaphores, and we'll wrap up with a networking assignment, which will give you exposure to networking tools that aren't covered in the core curriculum.

The midterm and final exams will be primarily, if not completely, multiple choice due to the large class size.

Textbook

The textbook is Silberschatz, Galvin, and Gagne. Any version from the 6th edition onward will support you well. The new editions cost around $90, but there are many used copies available, and there are copies in the library.

Code of Conduct

It's important that everything we do welcomes everybody and that everyone has an equal chance to participate. Harassment, discriminatory, or derogatory behavior cannot be tolerated. If asked to change behavior, we expect that to take effect immediately.

Academic Misconduct

The key idea in academic misconduct is that there is no misconduct provided you are authentically demonstrating your proficiency with the material. There are two common problems:

1. **Plagiarism**: If you use someone else's idea without acknowledgment, you've committed plagiarism. If you include their idea with proper citation, that's not plagiarism - you're acknowledging the source.
2. **Paraphrasing**: Even if you rewrite someone's idea in your own words, it's still their idea and needs citation.

When citing sources, especially online sources, don't just provide a URL as they can become invalid over time. Include the title, author, and year of publication or access date.

If circumstances are interfering with your ability to demonstrate proficiency in course material, talk to us before deadlines, not after grading. We're happy to discuss alternative ways for you to demonstrate proficiency, but we can't simply change grades after the fact.

Operating Systems Concepts

CPU and Memory

From the point of view of a running program, the CPU is where everything happens. Everything else is external to the CPU:

- RAM is a huge ordered list of data buckets we can access from the CPU
- Instructions running on the CPU can only deal immediately with registers or must perform load/store instructions to access data in memory

Physical memory organization:

- The bottom section (zero page) contains I/O ports mapped to memory locations
- The majority of RAM is simple storage - values stored there stay until power is turned off or they are overwritten

The sharing of data between multiple programs is possible because data persists in RAM when we swap the contents of the CPU from one program to another.

Process Management and CPU State

On the CPU, we have tools to keep track of a single running program (process):

- Program counter - tells us where to get the next instruction
- Stack pointer/frame pointer - keep track of the stack data structure
- Registers - store data being actively processed

When we switch from one process to another, we need to copy all this state data to memory so we can restore it exactly when we switch back.

Interrupt System

The interrupt vector is an array of function pointers indexed by hardware device IDs. When hardware needs attention, it triggers an interrupt, and the corresponding function in the vector is called.

The instruction cycle normally gets data from memory, increments the program counter, and executes instructions in sequence, but can be interrupted.

Processor status includes whether the CPU is in supervisor mode:

- In supervisor mode (kernel mode), the program can do anything
- In non-supervisor mode (user mode), programs have limited permissions

This protection is essential - in early systems like DOS and Windows 95, one program with a bad pointer could crash other programs. Modern systems prevent this by restricting user programs from accessing memory outside their allocated space.

Context Switching

When a process needs to do something that only the kernel can do, it causes a supervisor mode transition by calling a system trap. This is how we cross from user space into kernel space.

A context switch happens when one running process is swapped for a different process. This can happen between any two instructions in a running program - for example, when a keyboard sends a character.

The program has no control over when interrupts occur, so we have to write programs knowing that at any point after one instruction, we might get switched out and then switched back in before the next instruction.

When an interrupt occurs:

1. Save the program counter, processor mode, and all registers
2. Load the new program counter from the interrupt vector
3. Continue execution in the new state
4. When finished, restore everything that was saved and reset the CPU to the previous mode

Interrupt Priority

We can establish a priority order for interrupts by masking off less important interrupts. For example, if the disk needs attention (a fast device), we can tell slower devices like the mouse to wait.

When the disk interrupt is complete, we remove the mask and return to the normal model. Since the mouse operates at a much slower rate, it will barely notice the delay.

Device Drivers

Types of Devices

We have three categories of devices based on buffer size:

1. **Block Devices**: Use a fixed-size buffer for every communication (e.g., disks, CD-ROMs, video). Each update uses the same buffer size, making communication faster because each interaction contains multiple bytes.
2. **Character Devices**: Operate one byte at a time (e.g., mouse, serial ports, touchscreen, joystick). Good for devices where you don't know how many bytes you'll need to send.
3. **Network Devices**: A hybrid where the block size can change over time.

Device Driver Components

Device drivers include the following components:

- **Probe**: Called at boot time to determine if a specific device is attached to the computer and to configure it if present.
- **Open**: Called when a user process wants to start communicating with a device.
- **Close**: Shuts down communication with a device.
- **Read/Write**: Kernel-level routines to handle the action of reading or writing data to a device.
- **Interrupt Service Routine (ISR)**: Code that responds to hardware interrupts. The starting address of this function goes into the interrupt vector.

Device Access in Unix-like Systems

In Unix-based systems, every device has a file system handle in the `/dev` directory. You can access devices by opening these special files:

- `/dev/mouse` - access mouse data
- `/dev/disk0` - interface to a disk
- `/dev/null` - discards all data written to it (useful for suppressing output)
- `/dev/random` - produces random numbers

Data Exchange with Devices

When a device generates data (like a keypress), it:

1. Places the data at a predetermined memory address in zero page
2. Raises an interrupt to signal the kernel
3. The interrupt handler reads the data and stores it in a data structure

We typically use a circular queue (ring buffer) to handle the difference in speed between hardware events and user-space processing. This allows devices to provide multiple pieces of information before the first one is processed.

The circular queue operations:

- **Adding data**: Place data at the end position, increment the end position (wrapping around if necessary), and increment the count
- **Removing data**: Take data from the start position, increment the start position, and decrement the count

This design allows us to add and remove data without having to move existing elements in the buffer.