Lecture 1 — Introduction and Our C Toolkit

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

September 3, 2024

Course Syllabus

As our first order of business, let's go over the course syllabus.

Collaborative Course

The source material for the MTE 241 notes and slides is open-sourced via Github.

If you find an error in the notes/slides, or have an improvement, go to `https://github.com/jzarnett/mte241` and open an issue.

If you know how to use `git` and LaTeX, then you can go to the URL and submit a pull request (changes) for me to look at and incorporate!

Computer Structures and Real-Time Systems

There's two main areas of discussion in the course.

Computer structures: what is a computer made of?

Real-Time (Operating) Systems: what manages the computer?

To execute a program we need:

1. **Main Memory**
2. **System Bus**
3. **Processor**

Of course, this is the minimal set.

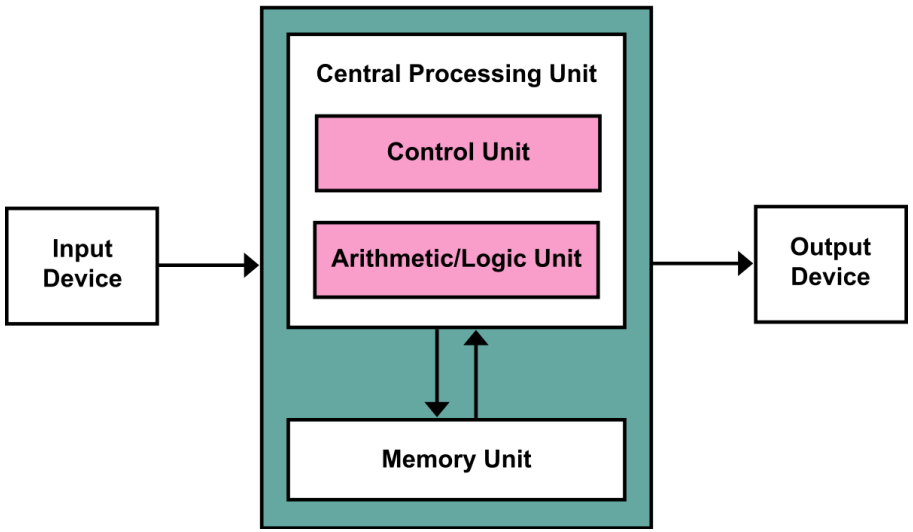


Image credit: Wikipedia user Kapooht

Real-Time Systems

Real-time systems are those with deadlines.

They are meant to monitor, interact with, control, or respond to the physical environment.

How are they different?

- Safety
- Performance
- Fault-Tolerance
- Robustness
- Scalability
- Security

Examples of Real-Time Systems

- Traffic control of aircraft
- Communications
- Nuclear power plant control
- Patient monitoring
- Smart homes

Embedded Systems

Elicia White's definition of an embedded system is:

a computerized system that is purpose-built for its application.

Embedded systems usually have limitations or constraints.

Embedded Systems Constraints

- Cost
- Correctness requirements
- Low memory
- Code size restrictions
- Processor speed
- Power consumption
- Available hardware

We Need a System

If the system, embedded or otherwise, is going to do more than one thing, we need some sort of management.

Solution: the Operating System!

Introduction to Operating Systems

Operating systems are those programs that interface the machine with the applications programs.

The main function of these systems is to dynamically allocate the shared system resources to the executing programs.

- What Can Be Automated?: The Computer Science and Engineering Research Study, MIT Press, 1980

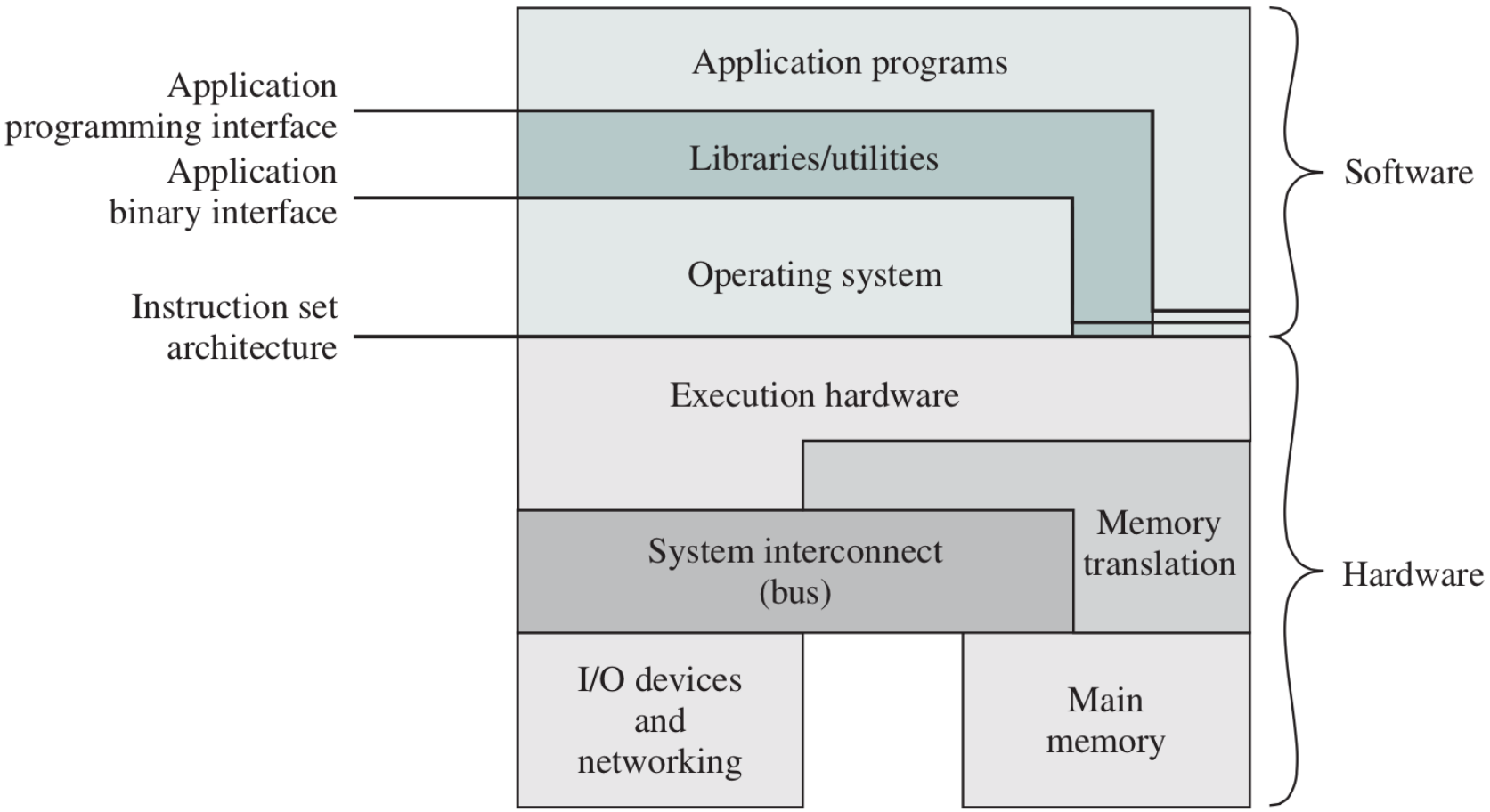

How likely are you to recommend Windows 10 to a friend or colleague?
1
2
3
4
5
Not at all likely
Extremely likely
Please explain why you gave this score.
I need you to understand that people don't have conversations where they randomly recommend operating systems to one another
THIS IS NOT TRUE
have you ever seen a linux user

An operating system (OS) sits between the hardware and programs.

It has many goals, that often conflict with one another.

Its job is to make it so other programs can run efficiently.

Structural Diagram of a Modern Computer



OS: Resource Manager

The OS is responsible for resource management and allocation.

Resources like CPU time or memory space are limited.

The OS must decide how to allocate & to keep track of system resources.

In the event of conflicting requests, choose the winner.

OS: Environment Provider

The OS enables useful programs like Photoshop or Microsoft Word to run.

The OS is responsible for abstracting away the details of hardware.

This is so program authors do not have to worry about the specifics.

Imagine Hello World had to be written differently for different hardware.

Multiple programs means some resources are shared.

$\rightarrow$ A source of conflicts!

OS creates and enforces the rules so all can get along.

Sometimes processes want to co-operate and not compete.

The OS can help them to do so.

OS: Efficiency

Another goal may be to use the computer hardware efficiently.

Image Credit: Argonne National Laboratory

Any moment when the supercomputer is not doing useful work is a waste.

OS: What is it, really?

Operating systems tend to be large and do a lot of things.

We expect now that an OS comes with a web browser, an e-mail client, some method for editing text, et cetera.

The part of the operating system we will study is the Kernel.

The kernel is the “core”; the portion of the OS that is always present in main memory and the central part that makes it all work.

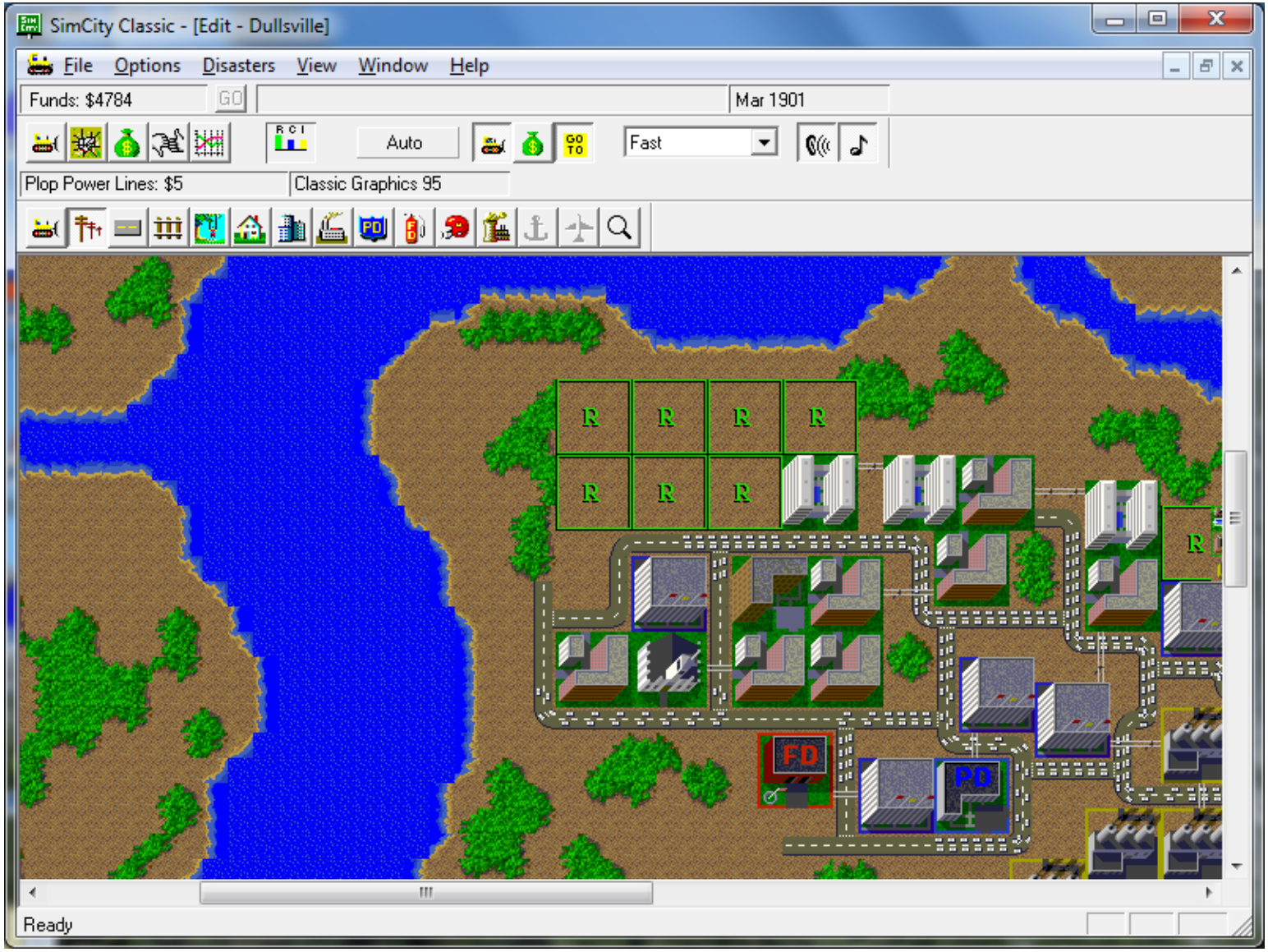
OS: Evolution

Operating systems will evolve over time.

There will be new hardware released, new types of hardware, new services added, and bug fixes.

Evolution is constrained: a need to maintain compatibility for programs.

Example: SimCity and Windows 95



Time, What is Time?

People assume that time is a strict progression from cause to effect, but actually from a non-linear, non-subjective viewpoint, it's more like a big ball of wibbly-wobbly, timey-wimey stuff.

Real-Time vs. Non Real-Time

Real-time systems are the ones where wall-clock deadlines matter.

Examples: aviation, industrial machinery, video conferencing, satellites…

There are deadlines, and there are consequences for missing deadlines.

Fast is not as important as predictable.

Hard and Soft Real-Time

Hard real-time: it has a deadline that must be met to prevent an error, prevent some damage to the system, or for the answer to make sense.

If a task is attempting to calculate the position of an incoming missile, a late answer is no good.

A soft real-time task has a deadline that is not, strictly speaking, mandatory; missing the deadline degrades the quality of the response, but it is not useless.

Concurrency

A program is said to be concurrent if it can support two or more actions in progress at the same time.

It is parallel if it can have two or more actions executing simultaneously.

Soon enough we will spend a great deal of time examining the differences between parallelism and concurrency in the program.

Concurrency

It is already the case that many programs you use are to a greater or smaller degree concurrent.

Depending on your level of programming experience, you may have already written a concurrent program, intentionally or without knowing it.

We will learn about how to take a program and make it concurrent, as well as how to write it with concurrency in mind from the ground up.

Concurrency Problems

Consider a program that performs a simple calculation given some input.

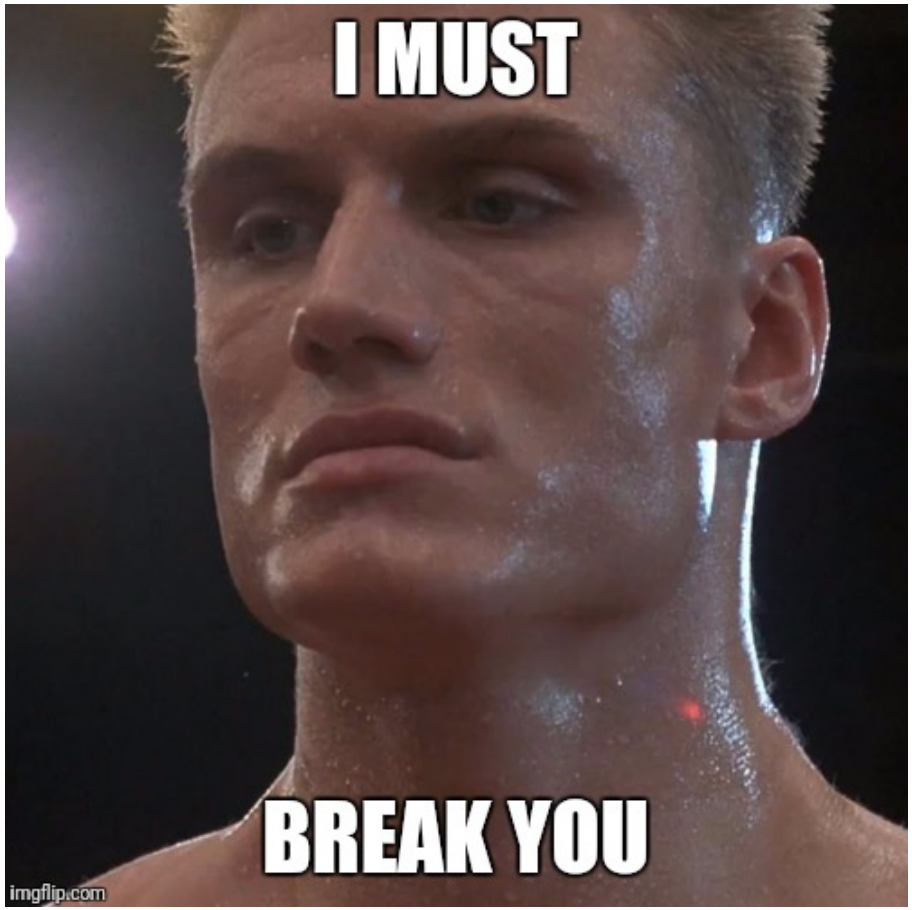
If the program has a concurrency problem, then the answer could be:

1. Consistently the wrong answer every single time
2. Different on consecutive runs with the same input, or
3. Correct some of the time but incorrect some of the time.

As you can imagine, none of these options are acceptable.

Not Gonna Lie...

This course is going to be hard.



If your programming skills need work, better to start trying to catch up now.

The first 6 lecture topics will move pretty fast across varied topics.
This is to help you get ready for the labs.

Moreover...

We have to fit a LOT of things into this course:

1. Computer Structures
2. Operating Systems
3. Some Systems Programming
4. Real-Time Systems / Reliability
5. Concurrency & Synchronization

Each of those has some amount of theory and hands-on part.

Our C Toolkit

We will need some introduction to the conventions and tools of C:

- Functions
- Header files
- Comments
- Structures
- Type Names
- Memory Allocation, Deallocation, and Pointers
- Dereferencing, Address-Of, The Arrow
- Arrays
- Strings
- Calling Conventions, `errno`
- Printing
- Constants, Global Variables, `extern`
- `main` and its Arguments
- `void*` and Function Pointers