2023 Fall ECE 344: Operating Systems

Lecture 1 1.0.0

Why Operating Systems?

Jon Eyolfson

2023 Fall

Why Operating Systems?

Understanding the operating system will make you a better programmer

You will either write software that:

- Interacts with the operating system
- Is the operating system

I’m Jon, Your Instructor

Eyolfson

I’m Jon, Your Instructor

Elf

I’m Jon, Your Instructor

Elf son

I’m Jon, Your Instructor

Elfson

I’m Jon, Your Instructor

Eyolfson

Important URLs for Course Resources

Public: https://eyolfson.com/

Private: https://q.utoronto.ca/ (Quercus)

Labs on GitLab, Discussion on Discord, Streams on YouTube

Sign in: https://eyolfson.com/discord/

Lecture Attendance is Still Important

It’s much faster to get feedback from you and clarify if anything is unclear

We’ll have live coding, I’ll be able to explain any happy accidents

If there’s anything else I can do to make attending a better experience let me know!

Evaluation for this Course

Assessment	Weight	Due Date
Lab 0	1%	September 13
Lab 1	4%	September 20
Lab 2	4%	October 4
Lab 3	4%	October 18
Lab 4	4%	November 1
Midterm Exam	25%	November 15 (tentative)
Lab 5	4%	November 22
Lab 6	4%	December 6
Final Exam	50%	December 8 to December 20

Academic Honesty Policy

You can study together, discuss concepts on Discord

Don’t post lab code on Discord, any other code is okay

Any cheating is not tolerated, and will only hurt you

The Recommended Books Complement Lectures

“Operating Systems: Three Easy Pieces”
by Remzi Arpaci-Dusseau and Andrea Arpaci-Dusseau

“The C Programming Language”
by Brian Kernighan and Dennis Ritchie

Skills You Should Practice Again If Needed

C programming and debugging

Being able to convert between binary, hex, and decimal

Little-endian and big-endian

Memory being byte-addressable, memory addresses (pointers)

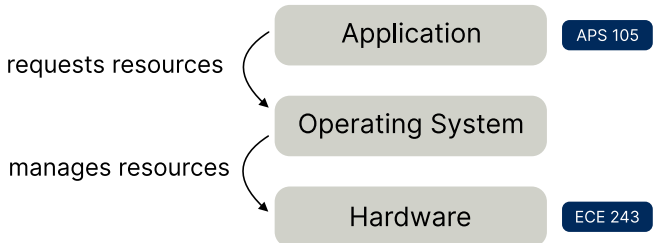
Please Provide Feedback!

This course is challenging, please let me know if anything is unclear

You can ask interesting questions, all programs interact with the OS

By the end of the course you'll be a better programmer

An Operating System Manages Resources



There’s 3 Core Operating System Concepts

Virtualization: share one resource by mimicking multiple independent copies

Concurrency: handle multiple things happening at the same time

Persistence: retain data consistency even without power

“All problems in computer science can be solved by another level of indirection”

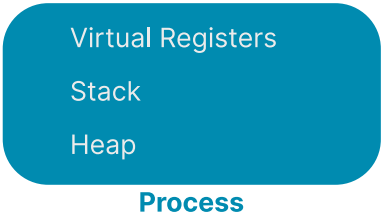
- David Wheeler

Our First Abstraction is a Process

Program: a file containing all the instructions and data required to run

Process: an instance of running a program

The Basic Requirements for a Process



My First Question to You

How are you able to run two different programs at the same time?

For example, a “hello world” program and another that counts up one every second

Does the OS Allocate Different Stacks For Each Process?

The stacks for each process need to be in physical memory

One option is the operating system just allocates any unused memory for the stack

Would there be any issues with this?

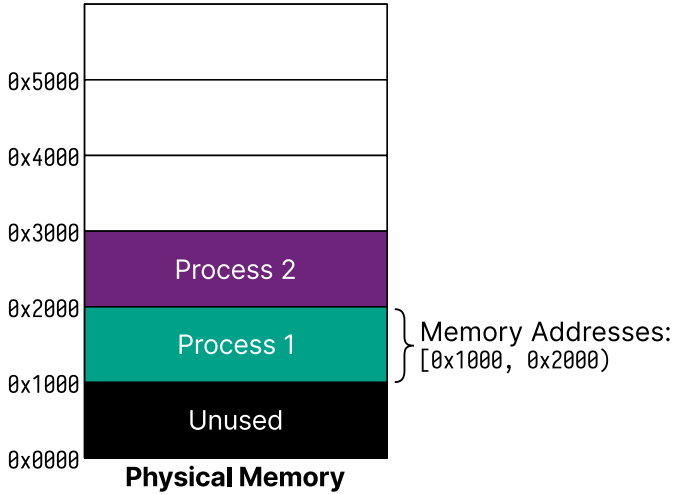
What About Global Variables?

The compiler needs to pick an address for each variable when you compile

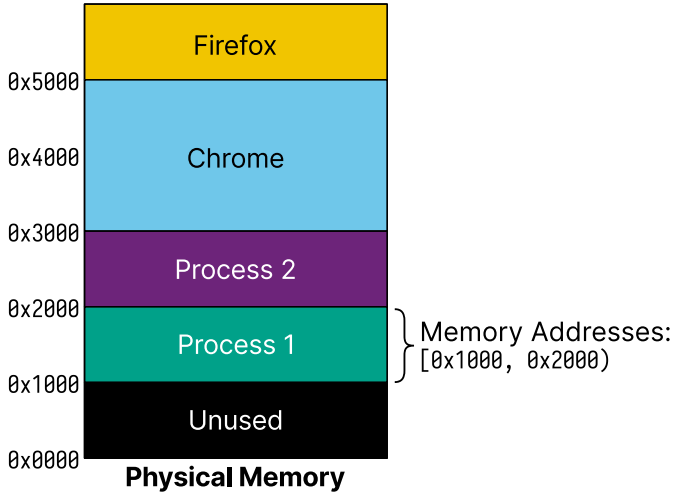
What if we had a global registry of addresses?

Would there be any issues with this?

Potential Memory Layout for Multiple Processes



Potential Memory Layout for Multiple Processes



What Happens If Two Processes Run the Same Program?

```c
#include <stdio.h>
#include <unistd.h>

static int global = 0;

int main(void) {
  int local = 0;
  while (1) {
    ++local;
    ++global;
    printf("local = %d, global = %d\n", local, global);
    sleep(1);
  }
  return 0;
}
```

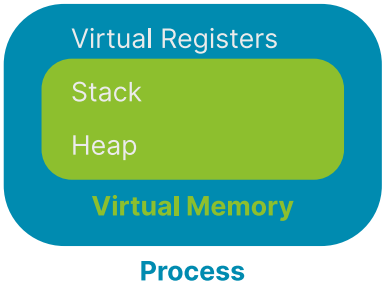
What Did We Find?

Was the address of `local` the same between the two processes?

Was the address of `global` the same between the two processes?

What else may be needed for a process?

A Process Has Its Own Virtual Memory



Example Code from This Class

All code will be in the “materials” repository located:
https://laforge.eecg.utoronto.ca/ece344/2023-fall/student/materials/

Compile the code:

```
cd lectures/01-why-operating-systems
meson setup build
meson compile -C build
```

Execute the code:

```
build/read-four-bytes <FILE>
```

Source: materials/lectures/01-why-operating-systems/read-four-bytes.c

Believe It or Not, This Is “Hello world”

```
0x7F 0x45 0x4C 0x46 0x02 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x02 0x00 0xB7 0x00 0x01 0x00 0x00 0x00 0x78 0x00 0x01 0x00 0x00 0x00 0x00 0x00
0x40 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x40 0x00 0x38 0x00 0x01 0x00 0x40 0x00 0x00 0x00 0x00 0x00
0x01 0x00 0x00 0x00 0x05 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00
0xA8 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xA8 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x10 0x00 0x00 0x00 0x00 0x00 0x00 0x08 0x08 0x80 0xD2 0x20 0x00 0x80 0xD2
0x81 0x13 0x80 0xD2 0x21 0x00 0xA0 0xF2 0x82 0x01 0x80 0xD2 0x01 0x00 0x00 0xD4
0xC8 0x0B 0x80 0xD2 0x00 0x00 0x80 0xD2 0x01 0x00 0x00 0xD4 0x48 0x65 0x6C 0x6C
0x6F 0x20 0x77 0x6F 0x72 0x6C 0x64 0x0A
```