Lecture 3 — Sunrise to Sunset

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

March 26, 2023

Sunrise, Sunset

Operating System Lifetime

We talked about the lifecycle of a process, but an OS has one too!

It is started, executes, and eventually shuts down.

Start Me Up

Let’s imagine the operating system is already installed.

System startup begins with the BIOS or UEFI.

Boot Loader

The BIOS or UEFI starts the boot loader.

The boot loader is a small piece of code to start the OS.

It can do some other things.

Start Here

The boot loader itself has at least some initialization code located in the first block of the hard drive that is the boot device.

Only if the first block of the drive contains this information can it be used for booting.

Kernel Start

Once the kernel is running, it can start the relevant operating system services.

Some services and utilities are always started when the OS begins running; others are configurable.

Although they are started by the OS without any direct user interaction, most processes are still user-level programs that operate like normal.

By Your Command

"By your command."

The main purpose of the operating system is not to run itself, but is there to make things work for the user-level programs that are supposed to run.

What Have the Romans Ever Done For Us?

Some examples of things we already asked the OS to do for us:

- Process and thread creation, termination
- Inter-process communication
- Concurrency control
- Memory allocation

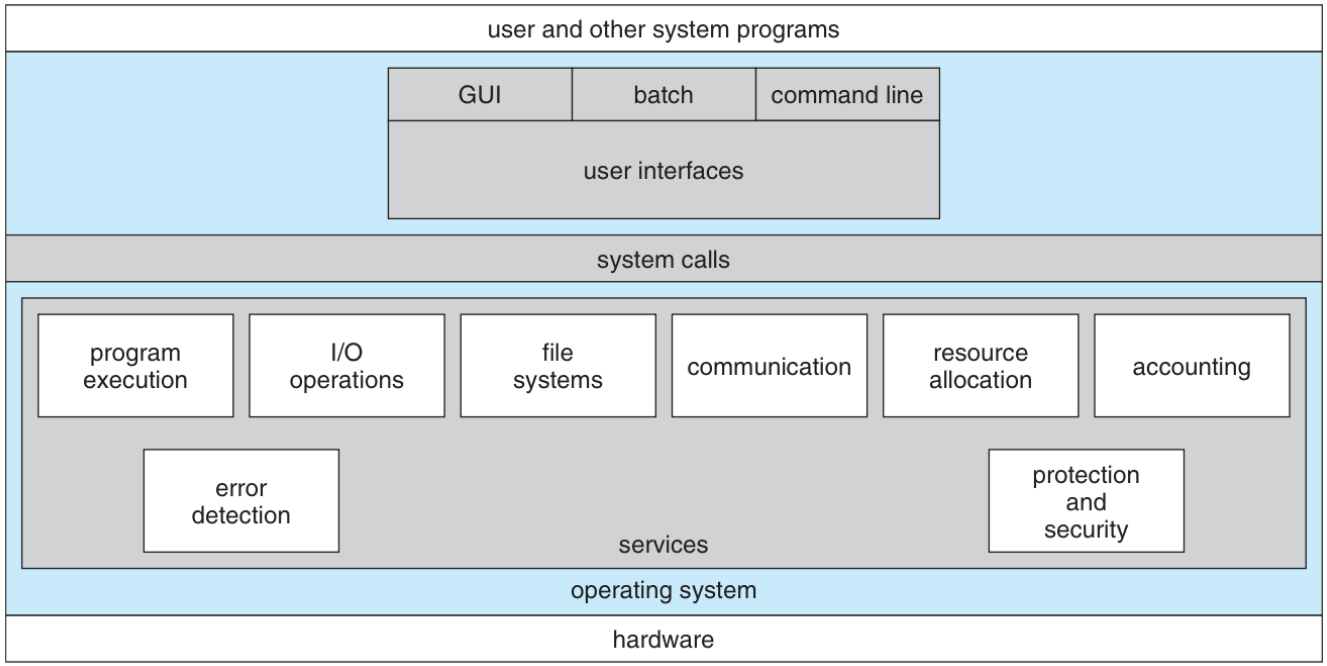
When You See It…

The operating system does a lot more than this, and the visibility of that functionality varies.

The OS must check permissions whenever you want to open a file, but you probably do not notice or think about it unless somehow permission is denied.

Scheduling?

OS Structure



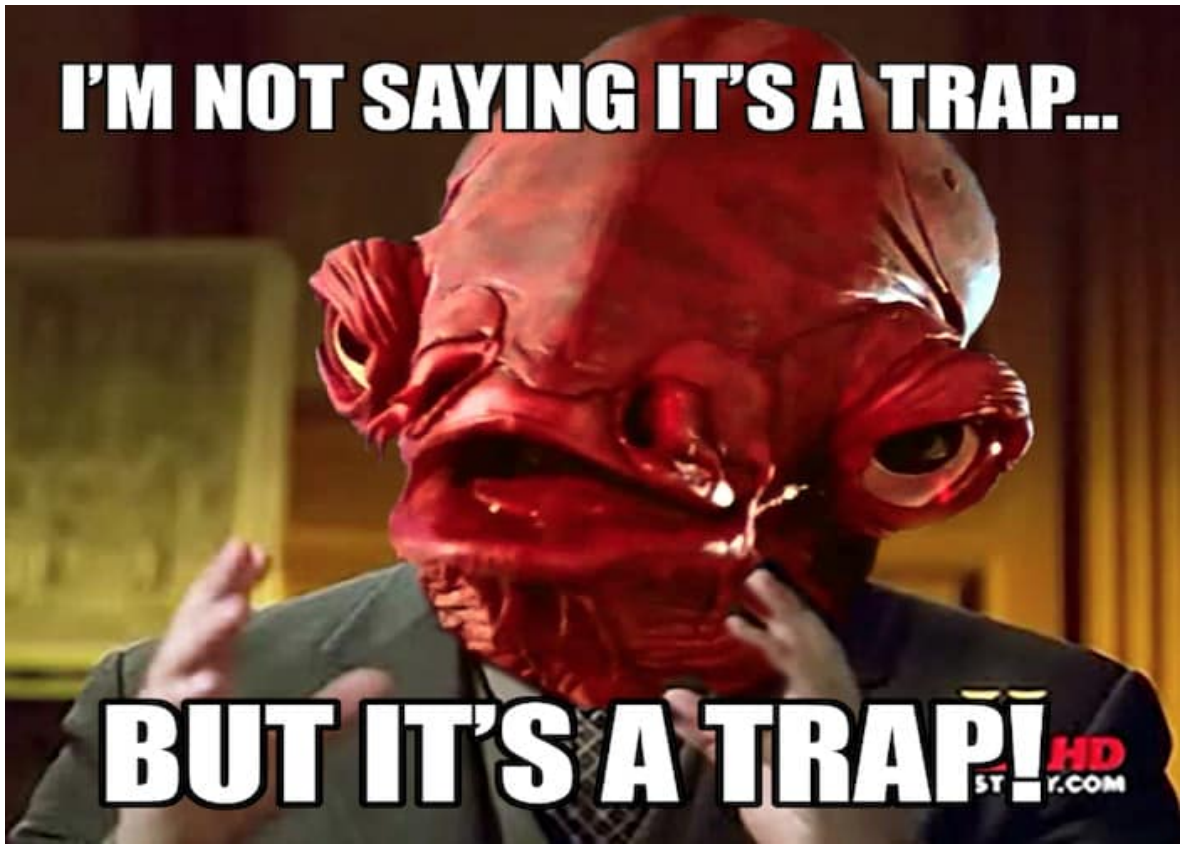
Steady-State and Trap

At steady-state, the operating system runs whatever background tasks it needs to do.

The interesting things happen as the result of a user-level program asking for the OS to do something.

And you know how that works…

Of Course You Do



We've covered this from the user-program point of view.

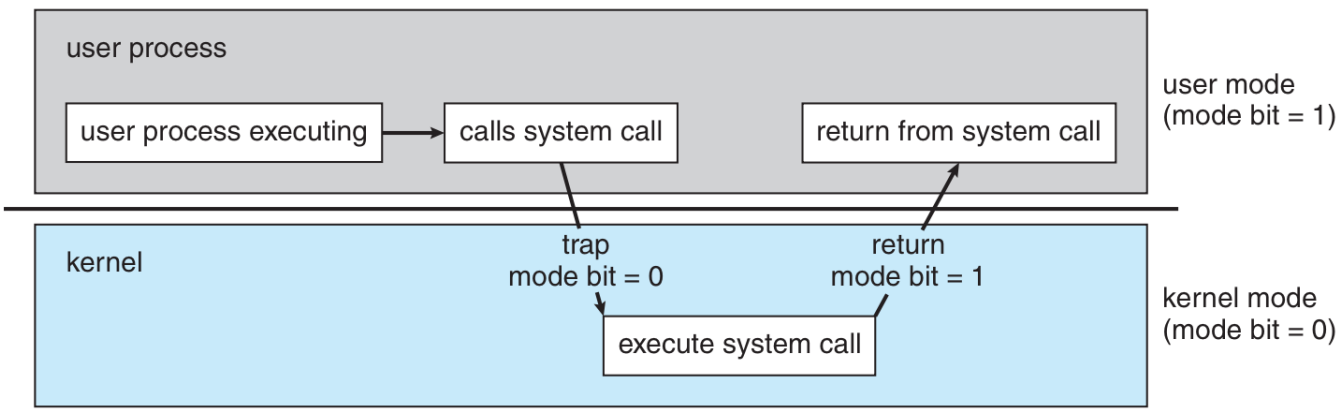
Mode Bits

We also talked about user mode & supervisor mode instructions…

Mode bits for the CPU…

Why the I/O operations are managed by the kernel…

Trap Operation



Transition from user to supervisor (kernel) mode [**?**].

User Mode and Kernel Mode: Motivation

Thanks Uncle Ben: “with great power comes great responsibility”.

Same as why we have user accounts and administrator accounts.

To protect the system & its integrity against errant and malicious users.

User Mode and Kernel Mode: Motivation

Multiple programs might be trying to use the same I/O device at once.

Program 1 tries to read from disk. This takes some time.

If Program 2 wants to read from the same disk, the operating system forces Program 2 to wait its turn.

Without the OS, it would be up to the author(s) of Program 2 to check and wait patiently for it to become available.

Works if everyone plays nicely.

Without enforcement of the rules, a program will do something nasty.

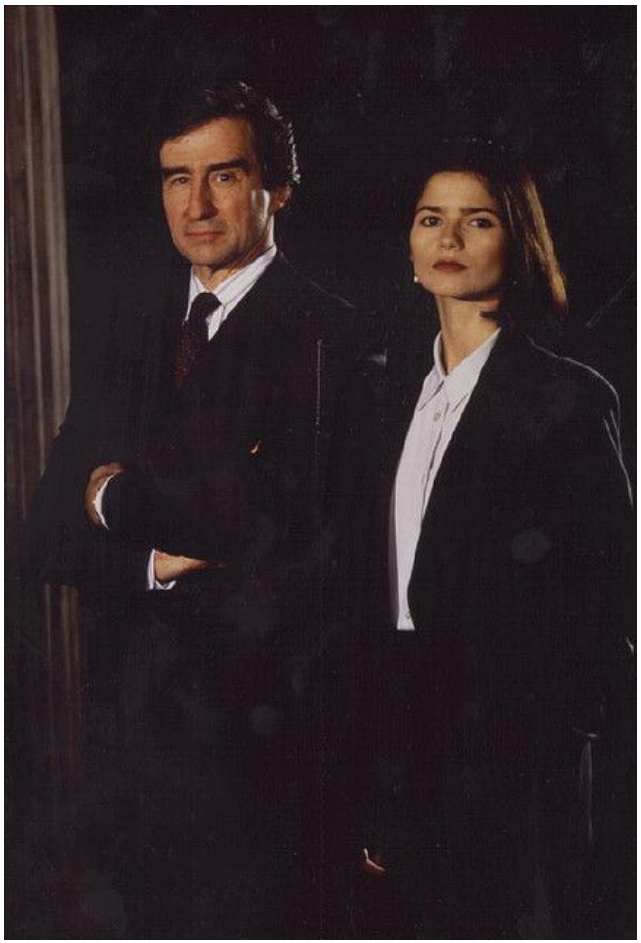
User Mode and Kernel Mode: Motivation

There is a definite performance trade-off.

Switching from user to kernel mode takes time.

The performance hit is worth it for the security.

Policy and Mechanism



Policy: What will be done.

Mechanism: How policy is carried out.

Policy

Some policies are configurable:

Yes: How much do priorities matter?

: No: Can I read from files where I lack the permission?

What is configurable is a question of OS design!

Acropalypse Now

Applying the policy involves both design and implementation.

OSes err on the side of fewer options.

Do OS authors know best?

Eat Your Vegetables

From the point of view of the user program, policy is something that we have to contend with but may not have any say in it.

Having to follow the rules may be less convenient, but it's not optional.

The operating system code itself has no such restrictions!

The operating system also has to be responsible for the mechanism.

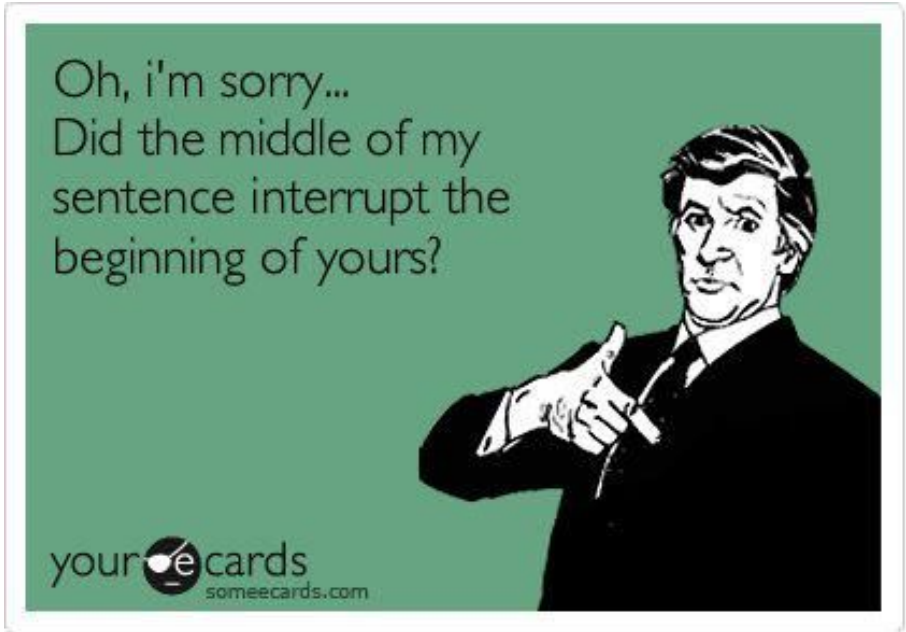
Switching Time

Previously: how a switch occurs. Now: when (why)?

From the point of view of the application program we say that the process switch could happen at any time.

The OS does get to choose but it’s not a trivial decision.

Interrupt



Interrupt always pauses a thread; may result in a thread switch.

A thread switch must occur when the currently-executing one gets blocked.

… or if it terminates.

OS Decision

After dealing with an interrupt we can choose a new thread.

We also want to prevent monopolization of the CPU…

- Timer interrupts perhaps?

Take Detailed Notes

The operating system will have to keep track of the various resources.

- Memory Tables
- I/O Tables
- File Tables

Memory Tables

Track the state of memory: what's free and what's not?

What is the OS memory?

Is any memory shared?

When do we update them?

I/O Tables

Keep track of the state of I/O devices.

Pay attention to what operations are in progress…

File Tables

Keep track of which files are open – process and system level.

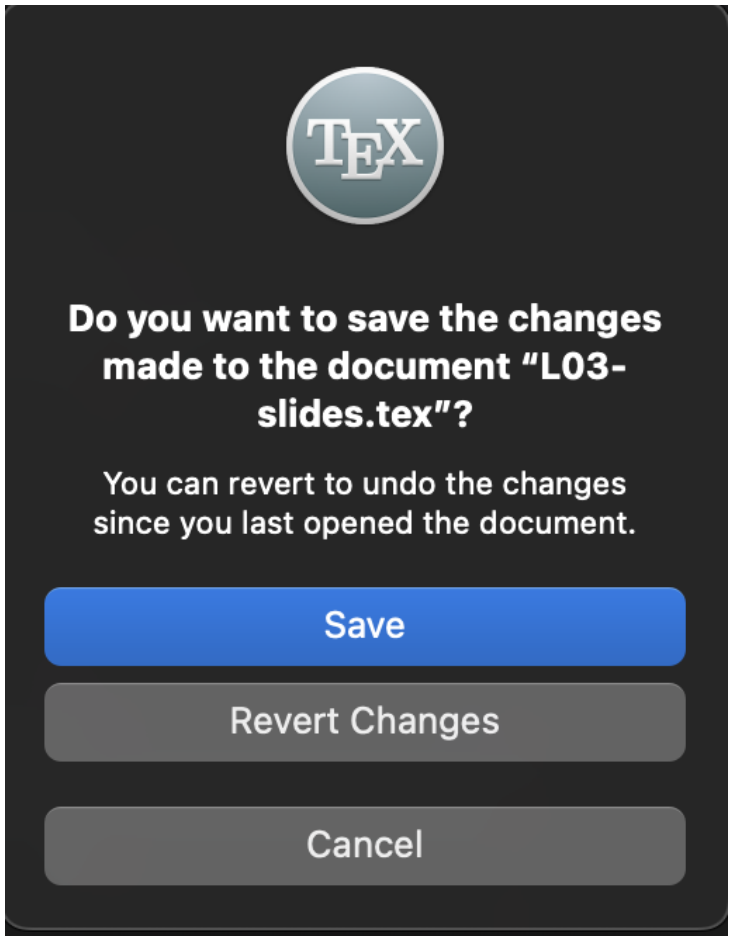
Remember – everything in UNIX is a file!

Power Down

Notify all the running processes they should exit.

But Will They?

Asking programs to shut down politely may not actually get them to terminate.



How long do we wait?

Can We Do This?

Can every user call a shutdown?

Once all processes are done, the OS can terminate its services.

And that's it, power off (or restart).