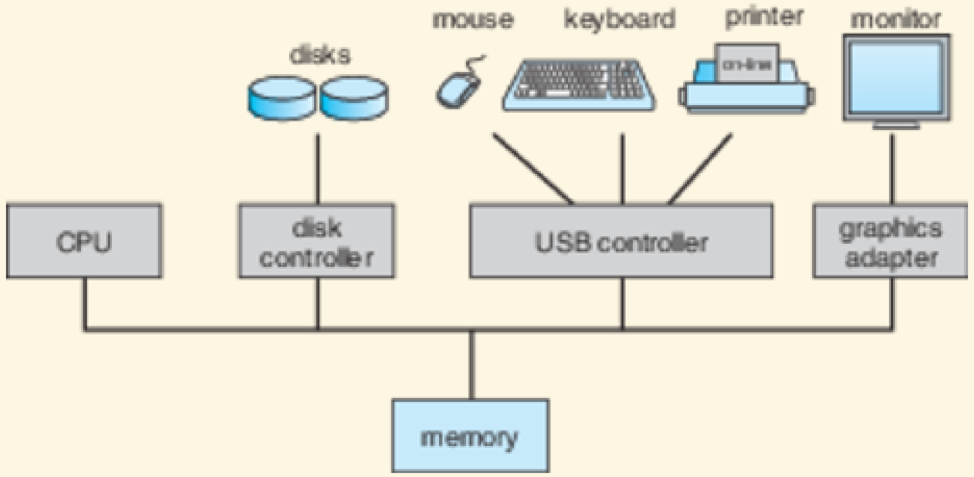
CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

What is an operating system?

- An operating system (OS) is a **resource manager**
- An OS **provides an environment** where computing can be done conveniently and efficiently


mouse
keyboard
printer
monitor
disks
on-line
CPU
disk controller
USB controller
graphics adapter
memory

Resource management:

- Processor (CPU)
- Memory (RAM)
- Storage
- I/O devices

Why do resources need managing?

Recall your computer architecture course (don't worry, you'll be okay). Simple computer had just one program running, using entire memory and hogging CPU.

This is fine if you have one just user who wants to run one program. How can different users, or different programs run by one user, share the system? This is where the OS comes in

A few things the OS handles

- Switching between running programs (processes)
- Assigning chunks of memory
- Takes flat array of disk storage and introduces the **file** concept
- Keeps users from snooping, or even accidentally crashing other programs
- All reading and writing to peripherals

Note that some of this requires hardware support.

Don't worry -- we'll go into *much* more detail later on

To manage all of this, OS is *always* running.

Other programs come and go, but the OS is first to start and does not stop until system shuts down.

Reminder:

- An operating system (OS) is a **resource manager**
- An OS **provides an environment** where computing can be done conveniently and efficiently

Back to architecture again -- did you ever use `syscall`?

Many uses, including

- allocating memory
- printing to screen
- accepting user input
- terminating program

Without an OS, programmer would need all their own code for managing memory, accessing peripherals, implementing files, etc.

What a nightmare!

Even on our unlikely single-user single-program machine, OS still has value.

What we've been describing so far would often be called the **kernel** -- the always-running program that manages resources

I will often use "kernel" and "OS" interchangeably in this course

It would also be fair to consider the **system software** as included in the definition of OS (though we won't)

- compilers
- shells/CLIs
- daemons

It is less common to consider user applications, such as a web browser or an email client, to be a part of the OS

Different distributions of Linux are generally considered different OSes, but they all come with same (possibly patched) kernel

Many things that look completely different (Ubuntu with Unity, Gentoo with i3, or a Fedora server with no window system) would be considered the same for our purposes

How is an OS implemented?

You might assume that an OS would need to be written in assembly for performance. However, that is generally not the case.

In reality, most modern OSes are written in C (or C++)

Code written in C is portable (you just need a compiler for the target architecture)

With modern compilers, you likely are not taking much of a performance hit. Optimizing compilers can sometimes produce code that is *faster* than hand-programmed assembly

Certain small, performance-intensive sections may still be assembly

Also, modern OSes are **huge**. The Linux kernel has about 27.8 million lines of code as of 2020. Would you want to write an assembly program that big?

How do we study something that big?

The CS way! By breaking down OSes and focusing on their parts

We will focus one-at-a-time on:

- Processing
- Memory
- Filesystems