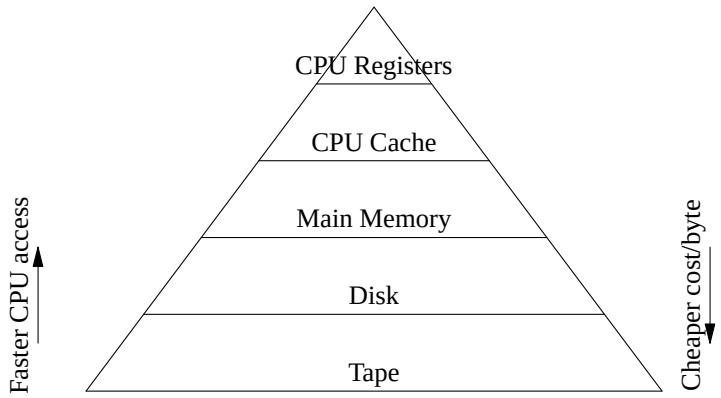
Lecture 2: Operating Systems Basics and Structure

Memory

Memory stores programs and data while there's power. It's a linear array that maps addresses to words of data. Attributes of memory include the size of the smallest and largest addressible units of data.

Memory is part of the storage heirarchy.



Peripherals and Busses

Peripherals are the fun parts of the system that interact with humans: monitors, disk drives, tape drives, CDROM readers/writers, network interfaces. They are frequently have associated controllers.

Peripherals are connected to the CPU/memory by busses (usually). Busses provide an interconnection network with a certain bandwidth (number of bytes/sec that can be transferred) and latency (time to transfer one byte).

Resources

Collectively, these are the resources that an operating sytem manages. It must allocate space and time on these devices, including the CPU and memory, for the computer to do useful things for people.

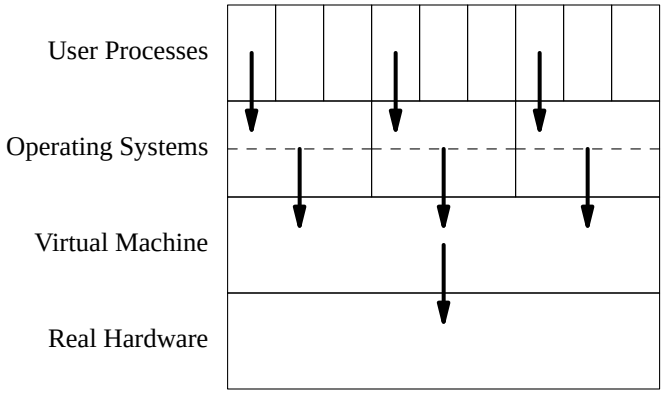
The layout of the course will largely follow these resources from the CPU out to the peripherals.

Virtual Machines

Virtual machines insulate many users from each other or one user from many types of hardware. All simultaneous users see the same interface to the resources of the machine as if they were the only user. The interface presented is simpler (abstracted from) the real hardware interface.

Some famous Virtual Machine Interfaces

- IBM's Virtual Machines - the first popular VM interface
- Java Virtual Machine - hardware hiding
- VMWare - multiple OS on a single machine IBM VM under Linux



Each layer adds a level of abstraction which may help mulitplexing and simplify code. Each layer may affect performance as may interactions between layers.

Software Structure

An OS is conceptuallly divided into several parts. The first structure is the *kernel*, the set of trusted software function that must be run by the processor at a higher privlege level (supervisor mode). Everything else is run at a lower privlege level (user mode). Code executing kernel routines is said to be running in *kernel space;* other code is running in *user space.* Moving between the two spaces is called a *boundary crossing*, an expensive operation that usually involves a software trap. The scheme generalizes to more levels.

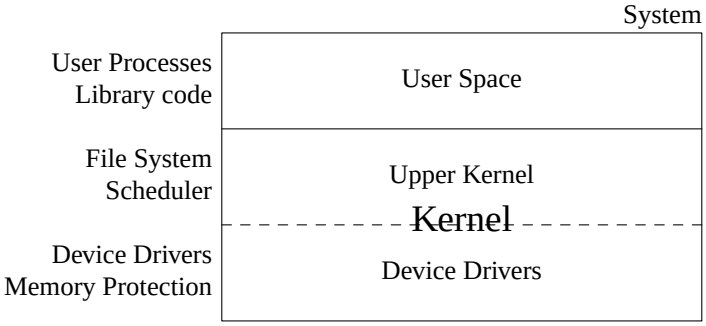
This is an abstraction (and generalization) of CPU modes.

The rationale is that trust is placed in the kernel which should be auditable and well-understood. Kernels should be small, but often systems bloat. A kernel stripped down to its bare essentials is called a *microkernel*, more on them in cs555.

Top and Bottom halves of the kernel

Even within the kernel there are distinctions. Code that deals with high-level abstractions, for example files, is said to exist in the top half of the kernel. Hardware specific code, such as code to manipulate a specific device (called a *device driver*) is in the bottom half of the kernel. The top half is machine-independent code, and vice versa. Porting an operating system to a new platform should only consist of porting the bottom half.

The provision of OS services (VM/RA) has elements in all three spaces. User-level libraries help provide interfaces, top half kernel code provides high level resource allocation and a clean interface, and bottom-level code multiplexes physical hardware and translates to the higher-level interfaces.



Accessing OS services - System Calls

Operating system services (reading or writing files for example) can only be accessed when the CPU is in supervisor mode, but user programs must run in user mode. The connection can be made through software interrupts. Specifically, the interrupt table for software interrupts is initialized by the OS to point to code that changes to supervisor mode and calls appropriate OS routines. (The correct routine either be directly determined by the number of the SW interrupt or by one of the parameters to the system call.) Such a table and the ISRs must be protected by the OS.

A system call is made by:

- Arranging the parameters to the system call in memory or registers
- Exectuting a trap instruction to cause a software interrupt
- The ISR for that trap determines the right OS routine to call and does so in supervisor mode
- The ISR returns to the user program.

System calls and interrupts are expensive in processor time.