

Operating Systems: General Theory

Operating System:

a **program** that acts as an **interface** between a user of a computer and the computer hardware

- ◆ make the computer system convenient to use
- ◆ use the computer hardware in an efficient manner

program = a set of instructions to control the computer hardware; most modern operating systems are written in C, C++ or assembly language

interface = a boundary between two layers – typically, how a "lower" layer appears to a "higher" layer. Interfaces exist all throughout computers. A "higher" layer means a higher level of **abstraction** – a "lower" layer often represents "nuts & bolts".

examples: RS-232 interface (serial port)
USB interface
Application Programmer Interface (API)

An interface exists so that a computer user does not need to know the underlying details – how the hardware "really" works.

(Interestingly enough, the field of computer engineering is all about understanding **both** the higher and lower layers...)

Non-computer analogy:

An automobile is a complicated machine – most drivers do not actually understand how a car *really* works. This is because, as a driver, you do not need to know what happens. You have a standard **interface**.

The interface for the automobile is the steering wheel, accelerator pedal, brake pedal, gearshift, gauges (speedometer, fuel gauge, possible tachometer), turn signal, (optional) clutch pedal ...

The bottom line: you can drive any car, from a lowly Yugo to the latest Formula One racer, because the interfaces on every car are similar enough!

In this course we are going to study four popular operating systems: Microsoft Windows, Apple MacOS X, Sun Solaris (UNIX), and Red Hat Linux. Each presents an **interface** to the computer hardware to the user.

All Microsoft Windows operating systems, from Windows 95 to Vista, have a similar interface.

All UNIX operating systems, from Solaris to AIX to HP-UX to the BSDs, have a similar interface.

All Linux operating systems, from Red Hat to Slackware to Suse to Debian, have a similar interface.

To simplify things even further, Linux and UNIX are similar (almost identical).

Furthermore, MacOS X has BSD UNIX under the hood. (Not to mention the most intuitive user interface.)

In fact, you can run Windows, MacOS X, Solaris, and Linux on the exact same hardware! (*see Mike Boldin's MacBook Pro*)

All operating systems, have the same two goals (as listed on the previous page):

- ◆ make the computer system convenient to use
- ◆ use the computer hardware in an efficient manner

Task:

an independently-scheduled thread of execution. a.k.a. **process**, thread
(*a running program*)

Kernel:

the code that is directly responsible for the **tasking model** – i.e., how the CPU's time is allocated.

Preemptive Multitasking:

The capability of an operating system to equitably share CPU time between several well-defined tasks currently scheduled to run on the system.

Meeting the two goals:

- ◆ be responsive to the user when running multiple tasks (allow fairly quick switching between tasks)
- ◆ keep the CPU (and the rest of the hardware) busy at all times

Resource Management

Another way of looking at operating systems:

An operating system manages resources for applications and services. (This is its only real purpose in life).

- ◆ CPU time sharing (*context switching*)
- ◆ Memory management (*virtual memory*)
- ◆ Hardware (timers, DMA, interrupt controllers, etc.)
- ◆ Interprocess communications (IPCs)

Both *applications* and *services* are just fancy names for programs that run on a computer (in the form of tasks/processes/threads).

In pursuing the goal of managing resources, the operating system designer walks the line between convenience and efficiency.

Resource #1: **CPU time**

The CPU is rated at a certain speed – way back in the day, kHz; recently, MHz; now, GHz; in the future THz ...

CPU designers try to make CPU's as fast as possible (see Intel vs. AMD) using a variety of techniques: *pipelining*, *superscaling*, *superpipelining*, *Hyperthreading*, etc.

The faster the CPU, the more instructions can be executed per second.

- 1 kHz = 1,000 instructions per second (ips)
- 1 MHz = 1,000,000 ips
- 1 GHz = 1,000,000,000 ips

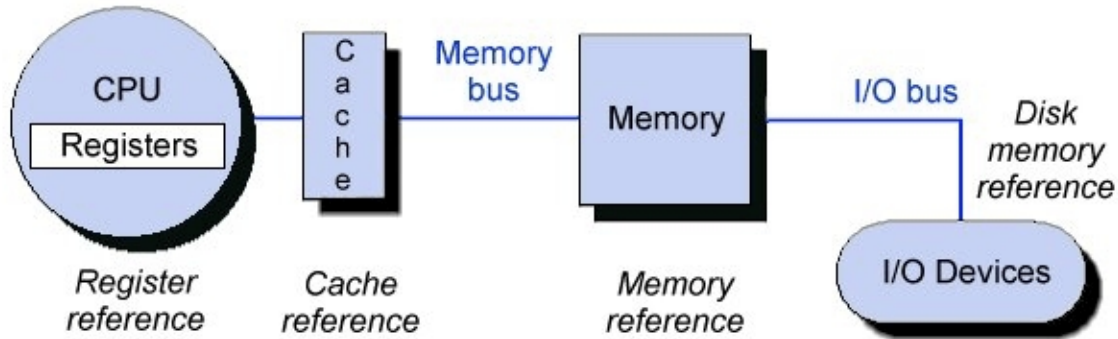
- ◆ Instructions are what make up programs.
- ◆ Programs run on a computer in the form of tasks/processes/threads.
- ◆ A faster CPU means programs run faster. (Although, not necessarily... lots of other factors!)
- ◆ An operating system is a program, made up of instructions. To get *its* work done, those instructions run on the CPU.
- ◆ Some operating systems try to use as little CPU time as possible, so that more of the CPU time is available for running programs.
- ◆ How efficient an operating system/CPU combination is can be measured in *context switch time*, typically in μs .
- ◆ The lower the context switch time, the more efficient an operating system is at managing the CPU's time.

Walking the Line Between Convenience and Efficiency:

Efficiency is important for **real-time** operating systems – that are used in applications like *process control*. It is also important in server operating systems that need to deal with high network traffic. Desktop operating systems place less importance on efficiency and more importance on convenience.

Resource #2: **Memory**

There are multiple levels of memory in a computer system:



- ◆ The operating system controls **physical memory** (RAM) and **virtual memory** (paging file or swap space – on disk).
- ◆ RAM is always at least 1,000 times faster than the fastest hard disk – therefore, adding more RAM will (almost always) make the operating system better able to operate efficiently and also provide a more convenient user experience.
- ◆ For the same reason, **file system caching** is also used on all modern operating systems, to minimize the amount of hard disk access.

Resource #3: **Hardware**

- ◆ All devices that make up a computer system are connected to the CPU
- ◆ Programs control the CPU only; the CPU controls the hardware
- ◆ The CPU can only access instructions and data that is in RAM
- ◆ As important (more important for some applications!) as CPU clock speed and the amount of RAM is **I/O throughput** – how much data can be moved between RAM and the other devices – disk drives, network interfaces, etc.
- ◆ Related to throughput is **bandwidth** – how fast data can be moved.
- ◆ Bandwidth is a rate, and is measured in bits per second (bps) or bytes per second (Bps), also:
 - kilobits per second (kbps) / kilobytes per second (kBps)
 - megabits per second (Mbps) / megabytes per second (MBps)
 - gigabits per second (Gbps) / gigabytes per second (GBps)
- ◆ The higher the bandwidth, the higher the throughput.
- ◆ I/O operations are typically done via **interrupts** – an I/O device will signal (interrupt) the CPU when it needs attention
- ◆ The operating system responds by running a program, called an **interrupt service routine (ISR)**, made up of instructions to control the hardware device – the instructions are executed (processed) on the CPU itself.
- ◆ An ISR has three responsibilities:
 1. Control the hardware device and read its status.
 2. Move data from the hardware device to RAM (input).
 3. Move data from RAM to the hardware device (output).
- ◆ Another measure: **interrupt latency** – how fast the operating system/CPU combination can respond to interrupts – typically measured in μ s or even ms.
- ◆ The lower the interrupt latency, the more interrupts can be processed per second. And the more data can be moved – i.e., higher throughput.

Resource #4: **Inter-Process Communications (IPCs)**

- ◆ This means co-operation between tasks/processes/threads (a.k.a., running programs).

- ◆ Examples:
 - ▶ *pipelines*
 - ▶ "drag and drop"
 - ▶ printing
 - ▶ helper applications/file associations
 - ▶ CGI applications
 - ▶ database-driven web sites

- ◆ The lower the IPC overhead (latency), the faster the communications (higher bandwidth and throughput).