OS & Syscall Basics 1: Operating Systems

1. **Operating Systems**
   - **os overview**
   - **layered design**
   - **kernel vs. user mode**
2. System Calls
3. Filesystems and Memory
4. Processes, IPC, Signals

Operating Systems

Requiring or even allowing users and application software direct access to a computer's hardware raises many problems:

- modern computer hardware typically requires long, complex sequences of instructions to accomplish the kinds of tasks computer users and applications want accomplished (e.g., storing a file on a harddrive)
- different types of hardware can require different instructions to accomplish basically the same task (e.g., reading a file from a harddrive vs. from a CDROM)
- direct access to the hardware would lead to reliability and security issues
- multiuser systems could have issues with resource contention among users

Operating Systems (contd.)

These issues make it impractical and inappropriate for users and application software to interact directly with computer hardware.

All but the most primitive modern computer systems have a key *layer of software* that sits between user application software and the hardware that makes up the computer system.

This software is called the **operating system** (OS).

One key function of an OS is providing an *abstract computer* that is much easier to use than the actual (hardware) computer.

This abstract computer will also provide a *uniform interface* across different hardware (e.g., Linux on PC vs. on mainframe).

Operating Systems (contd.)

So the first major function of an OS is providing a simplified and uniform interface to a computer's hardware.

The *second major function* of an OS is to *manage a computer's hardware resources* so programs can run *securely and efficiently*.

This function should be invisible to users if the OS is doing a good job.

When we talk about the hardware resources of a computer we are talking about: CPU(s), memory, hard drive(s), CDROM drives, network cards, printers, etc.

Operating Systems (contd.)

An OS provides a large number of services to users/programs:

- filesystem(s) for storing data
- security policies to limit access to the computer and its files
- allocating and sharing memory to allow multiple programs to run simultaneously
- scheduling CPU usage by programs to allow fair and efficient execution of multiple programs
- preventing programs from reading/writing in each others' memory
- controling access to devices like hard drives, and scheduling access to be efficient
- implementing network protocols for network communication

Operating Systems (contd.)

The core functionality of an OS is referred to as the **kernel**.

Modern OSs are large, complex pieces of software, on the order of 5 million lines of code!

On so called **monolithic kernel** OSs like Linux, the kernel is basically the entire OS.

On **microkernel** OSs, the kernel will not be the entire OS; instead some parts of the OS will not run in kernel mode (see below).

"Linux" is an OS kernel along with a large set of **device drivers**.

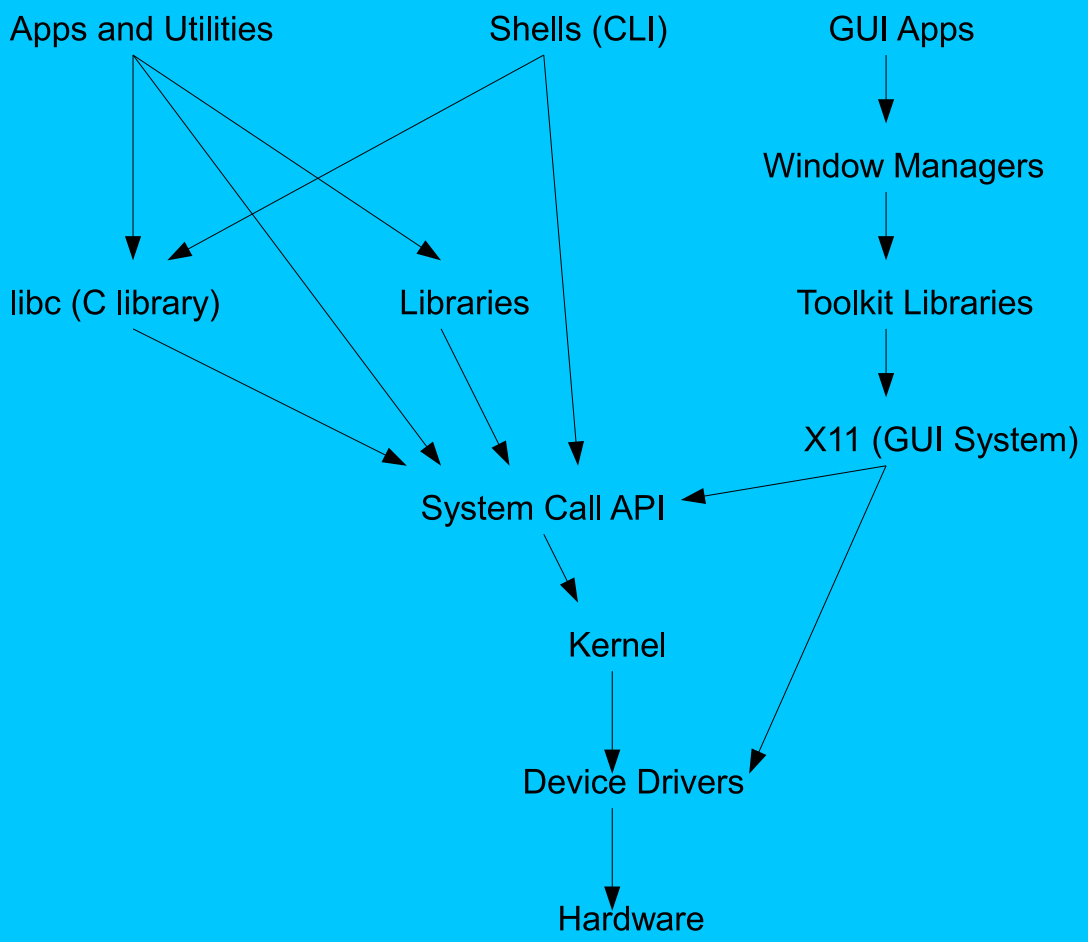
Layered Design of OS's

What most people think of as the "operating system" is actually the OS plus more:

- **device drivers**: interface between kernel and hardware
- a **kernel**: the true/basic OS
- **libraries**: higher level programming interfaces
- **shells**: **command line** interface to OS
- **GUIs/window managers**: graphical interface to OS
- **application software**: allows users to get things done

These components are arranged in *layers* between the hardware and users of the computer system.

Layered Design of OS's (contd.)



Linux Distributions

A Linux **distribution** is a packaging of:

- Linux kernel(s) + device drivers
- shells (bash, csh, etc.)
- GUI (X11, Xfree, X.org)
- window managers (KDE, Gnome, etc.)
- boot managers (LILO, GRUB)
- application software
- installation and maintenance tools

Because much of this additional software comes from the GNU project, some people refer to Linux as "GNU/Linux."

Kernel Mode vs. User Mode

Virtually all modern processors have multiple **modes**.

The instructions that can be carried out by a process running in a certain mode may be limited.

At the very least, processors will have two modes:

- **kernel mode** (also called supervisor, privileged, or master): the *unrestricted mode*, where any processor instruction is allowed to be executed
- **user mode**:
  a *restricted mode*, where instructions to access I/O devices may not be allowed, access to certain areas of memory may not be allowed, and so forth

Some CPUs support more than two modes, resulting in what are known as **protection rings** or **privilege rings**.

Kernel Mode vs. User Mode (contd.)

An OS *kernel* needs to have unlimited access to devices and memory, so it will need to run in *kernel mode*.

Applications cannot be allowed unlimited access, so they will need to run in a user mode.

The term **kernel space** refers to code requiring the processor be in kernel mode, while **user space** refers to code that can be run in a user mode.

A consequence of the use of modes is that every time the OS must do something for an application, the processor mode must be *switched*, and this involves some *time cost*.