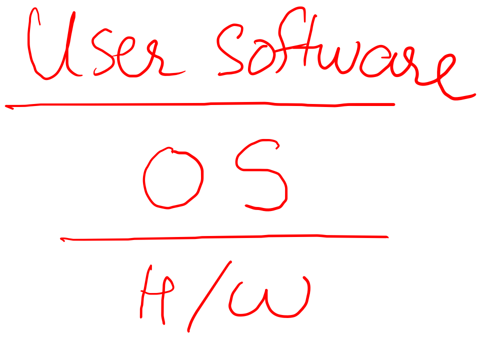
Design and Engineering of Computer Systems

Lecture 5: Introduction to Operating Systems

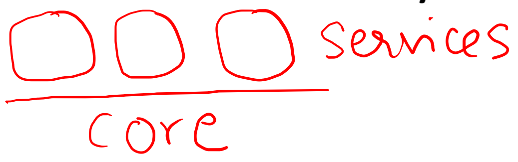
Mythili Vutukuru

IIT Bombay

What is an operating system?



- System software, to manage the computer system hardware
  - Distinct from user software (web browser, web server, gaming engine, ..)
- Special program (compiled OS executable) is stored on hard disk, starts running at boot up
  - Once OS starts, it starts other user programs
- Operating system has kernel + system programs
  - Kernel = the core part of the operating system
  - System programs are useful programs to manage system (e.g., program to list all files in a directory "ls")
- Most OS today (e.g., Linux) are monolithic, one big executable of the kernel
  - Install kernel modules at run time to add extra functionality (e.g., device drivers)
- Alternate architecture: microkernels, more modular but not popular
  - Small core kernel, most functionality as services running on microkernel



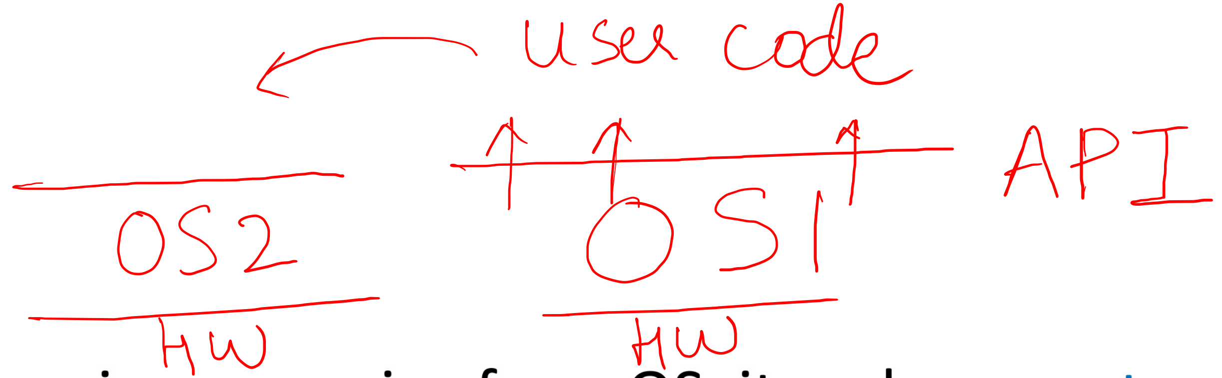
Why do we need operating systems?

- Convenience: makes life easy
  - Every user program need not worry about handling hardware on its own
- Isolation: makes life secure
  - Multiple programs on a computer system are protected from each other
- Better utilization of resources: makes life efficient
  - Easier to optimize usage of system resources via careful planning
- Operating systems started out as simple libraries for user programs
  - Later, support for isolation via CPU privilege levels
  - Later, support for multiprogramming

What OS does: process management

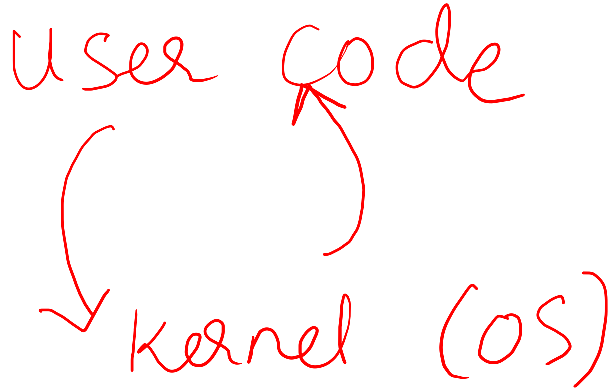
- OS runs multiple processes concurrently on underlying CPU
  - User programs, system programs, ..
- OS manages life cycle of processes: creation, execution, termination
- OS schedules processes on CPU as per scheduling policies
- OS switches context between processes on the same CPU core
- OS handles interrupts that occur during process execution
  - Process execution is paused, CPU switches to OS interrupt handling code
- OS handles program faults that occur during process execution
  - Example: segmentation fault

System calls



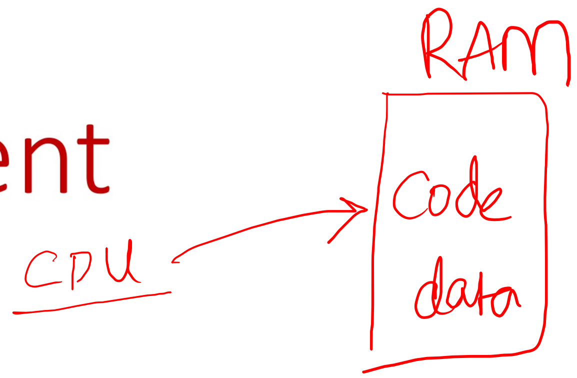
- When user program requires a service from OS, it makes a system call
  - Example: Process makes system call to read data from hard disk
  - Why? User process cannot run privileged instructions that access hardware
  - CPU jumps to OS code that implements system call, and returns back to user code
- System calls supported by an OS form the API to user programs
- POSIX API: standard set of system calls defined for portability
  - User program written on one POSIX-compliant OS will run without change on another POSIX-compliant OS
  - However, program may have to be recompiled if architectures are different
- Normally, user program does not call system call directly, but uses language library functions
  - Example: printf is a function in the C library, which in turn invokes the system call to write to screen

User mode and kernel mode

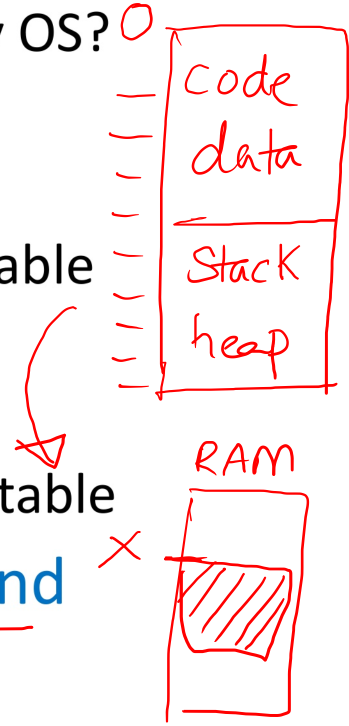


- Modern CPUs operate at multiple privilege levels
- User programs run in unprivileged user mode of CPU
- CPU shifts to privileged kernel mode for running OS code during:
    - Interrupts: external events
    - System calls: user request for OS services
    - Program faults: errors that need OS attention
- OS code executes in kernel mode, and returns back to user code
    - CPU switches back to low privilege level to run user code

What OS does: memory management

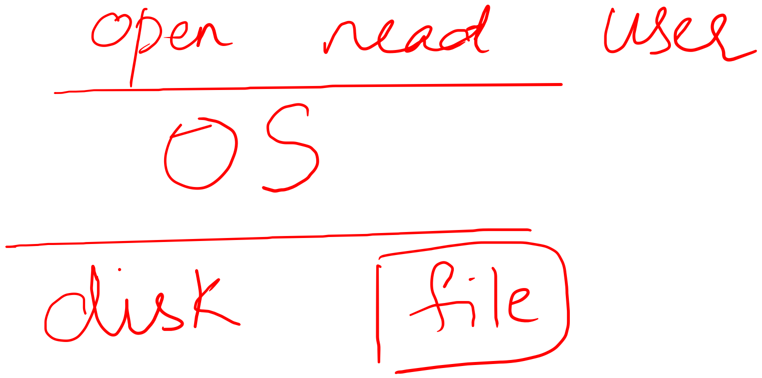


- OS allocates memory for the memory image of a process
  - Memory allocated in fixed granularity of pages
  - Upon process termination, memory is freed up and assigned to other processes
- How do we assign memory addresses to process code+data?
  - How does a compiler know which memory locations will be given to process by OS?
- Process code+data are assigned virtual addresses initially
  - Compiler assigns memory addresses starting from 0 in executable
  - Later parts of memory image (stack, heap..) continue at addresses after executable
- OS maintains mapping between virtual memory addresses and real (physical) addresses in page table
  - Virtual addresses translated to physical addresses during execution using page table
- OS ensures efficient usage of memory, by allocating memory on demand



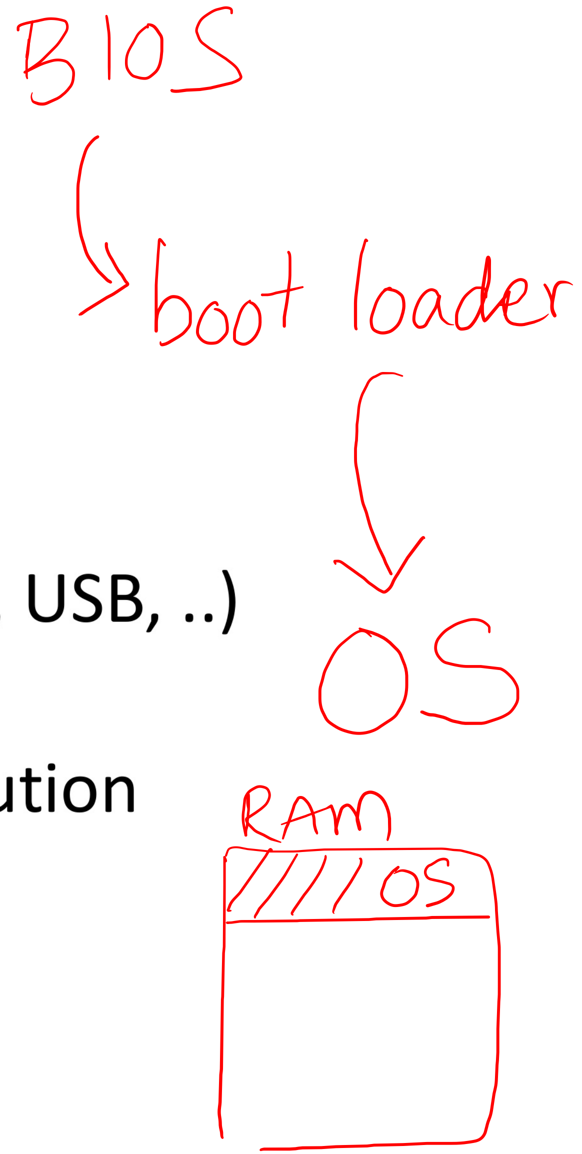
What OS does: I/O management

- OS has device drivers for all I/O devices connected to the system
  - Device drivers form a big part of the code of modern OS
  - Giving commands to I/O devices, interrupt handling, …
- File system: OS code that takes care of storing user file data persistently on the hard disk
  - Works with hard disk device driver to store/retrieve blocks from disk
- Network stack: OS manages communication with other machines over the network



Booting your system

- What happens when you boot up a computer system?
- Basic Input Output System (BIOS) starts to run
  - Resides in non-volatile memory, sets up all other hardware
- BIOS locates the boot loader in the boot disk (hard disk, USB, ..)
  - Simple program whose job is to locate and load the OS
- Boot loader loads OS in memory and sets it up for execution
- CPU starts executing OS code
- OS exposes a terminal / shell / other interfaces to user
- User runs programs, starts processes



Summary

- In this lecture:
  - Introduction to operating systems
  - Need for operating systems
  - Functions of OS: process management, memory management, I/O
- Next week: process management in operating systems