Graduate Operating Systems
CSE 60641
(Introduction & Overview)

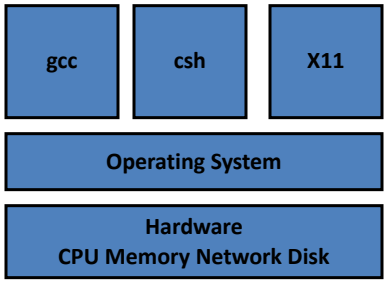
Fall 2020

Operating Systems

- Most operating systems are large & complex systems
  - Most people don't understand every aspect of them – including sysadmins and computer scientists!
  - Simple programs like "Hello, World" can be millions of lines of code
  - Many research projects study operating systems behavior
- Studying OS is learning how to deal with complexity
  - Abstractions (+interfaces)
  - Modularity (+structure)
  - Iteration (+learning from experience)

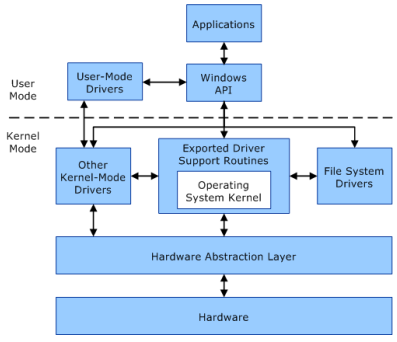
What does an OS do?

- Software layer that sits between applications and hardware
- Performs services
  - Abstracts hardware
  - Provides protection
  - Manages resources



OS vs Kernel

- Windows, Linux, Mac OS are operating systems
  - Includes system programs, system libraries, servers, shells, GUI, etc.
- Linux kernel, Windows executive, etc. – the special piece of software that runs with special privileges and actually controls the machine
- OS often equated with the kernel



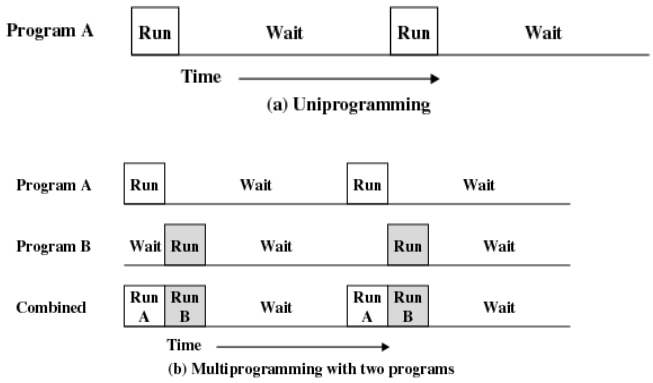
Evolution of OS

- OS as a library
  - Abstracts away hardware, provides neat interfaces
    - Makes software portable; allows software evolution
  - Single user, single program computers
    - No need for protection: no malicious users, no interactions between programs
    - No resource sharing
  - Disadvantages of uniprogramming model
    - Expensive
    - Poor resource utilization
    - Doesn't support complex/large applications

Evolution of OS

- Invent multiprogramming
  - First multi-programmed batch systems, then time-sharing systems
- Idea:
  - Load multiple programs in memory
  - Do something else while one program is waiting, don't sit idle (see next slide)
- Complexity increases:
  - What if programs interfere with each other (wild writes)
  - What if programs don't relinquish control (infinite loop)

Single Program vs Multiprogramming



Protection

- Multiprogramming requires isolation
- OS must protect/isolate applications from each other, but also OS from applications
  - Applications should not crash OS or other applications!
- Three techniques
  - Preemption
  - Interposition/mediation
  - Privileged mode

Protection #1: Preemption

- Resource can be given to program and access can be revoked
  - Example: CPU, Memory, Printer, "abstract" resources: files, sockets
- CPU preemption using interrupts
  - Hardware timer interrupt invokes OS, OS checks if current program should be preempted, done every few milliseconds in Linux
  - Solves infinite loop problem!
- Does it work with all resources equally?

Protection #2: Interposition

- OS hides the hardware
- Application have to go through OS to access resources
- OS can interpose checks:
  - Validity (Address Translation)
  - Permission (Security Policy)
  - Resource Constraints (Quotas)

Protection #3: Privilege

- Two fundamental modes:
  - "kernel mode" – privileged
    - aka system, supervisor, or monitor mode
    - Intel calls its PL0, Privilege Level 0 on x86
  - "user mode" – non-privileged
    - PL3 on x86
- Bit in CPU – controls operation of CPU
  - Protection operations can only be performed in kernel mode. Example: hlt
  - Carefully control transitions between user & kernel mode

```
int main()
{
        asm("hlt");
}
```

OS as a Resource Manager

- OS provides "illusions"; examples:
  - Every program is run on its own CPU
  - Every program has all the memory of the machine (and more)
  - Every program has its own I/O terminal
- "Stretches" resources
  - Possible because resource usage is typically "bursty"
- Increases utilization

Resource Management

- Multiplexing increases complexity
- Car analogy:
  - Dedicated road inefficient, so sharing is needed
  - Abstraction: different lanes per direction
  - Synchronization: traffic lights
  - Capacity: build more roads/lanes
- More utilization creates contention
  - Decrease demand: slow down
  - Backoff/retry: use highway during off-peak hours
  - Refuse service, quotas: force people into public transportation
  - System collapse: traffic jam

Resource Management

- OS must decide who gets to use what resource
- Approach 1: have admin (boss) tell it
- Approach 2: have user tell it
  - What if user lies? What if user doesn't know?
- Approach 3: figure it out through feedback
  - Problem: how to tell power users from resource hogs?

Goals for Resource Management

- Fairness
  - Assign resources equitably
- Differential Responsiveness
  - Cater to individual applications' needs
- Efficiency
  - Maximize throughput, minimize response time, support as many apps as you can
- These goals are often conflicting
  - All about trade-offs

Summary: Core OS Functions

- Hardware abstraction through interfaces
- Protection:
  - Preemption
  - Interposition
  - Privilege (user/kernel mode)
- Resource Management
  - Virtualizing of resources
  - Scheduling of resources

“Entrance Exam”

- What is a **multi-threaded** process?
- What is the purpose of **mutual exclusion**?
- What does it mean to say an operation is **atomic**?
- Use a brief example to describe what a **deadlock** is or how it can be caused.
- What is the difference between **deadlock** and **starvation**?

“Entrance Exam”

- What is the purpose of an **interrupt**?
- What is **priority inversion**?
- What does a **page table** do?
- What does **thrashing** mean?
- What is a **symbolic link**?
- What is a **parity bit**?
- What is an **i-node** (or **file control block**)?

“Entrance Exam”

- What does it mean to **fork** a process?
- What is the danger of **caching a write**?
- What is a **page fault**?
- What is the difference between **kernel space** and **user space**?
- What is **disk fragmentation**?
- What is a **critical section**?

“Entrance Exam”

- What is a **runqueue** (or **ready queue**)?
- What is a **binary semaphore**?
- What is the difference between a **direct pointer** and an **indirect pointer** in a file system such as EXT?
- Can you name and very briefly describe a **scheduling algorithm** that would be **fair** to all tasks awaiting execution?

"Entrance Exam"

- Can you name and very briefly describe a **scheduling algorithm** that might be a good choice in a **real-time system**?
- What is a **system call**?
- What does it mean for a system call to **block**?

Next Week

- Next week:
  - **OS History and Architecture**
    - **[1]** P. Brinch Hansen, "The Nucleus of a Multiprogramming System", Communications of the ACM, 238-242, April 1970.
    - **[2]** Dennis M. Ritchie and Ken Thompson, "The UNIX Time-Sharing System", Communications of the ACM, volume 17, number 7, July 1974.
    - **[3]** Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr., "Exokernel: An Operating System Architecture for Application-Level Resource Management", Proc. of the 15th Symposium on Operating Systems Principles, December 1996.