

CS 3204 Operating Systems

Lecture 2
Godmar Back



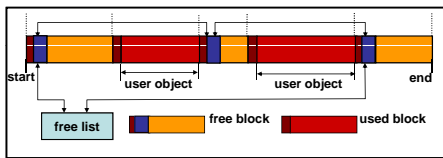
Announcements

- Please return your prerequisite forms if you haven't done so
- My office hours (MCB 637):
 - Tu 8-9:15am
 - Th 1:30pm-3:30pm (2:30pm if no one comes)
- TA office hours announced later today, check website
- Start thinking about groups
 - (but *do not* collaborate on Project 0)
- **Project 0 due on Sep 7**
 - Other due dates posted later today



Project 0

- Implement User-level Memory Allocator
 - Use address-ordered first-fit



Outline for today

- Motivation for teaching OS
- Brief history
- A survey of core issues OS address
- What you should get out of this class



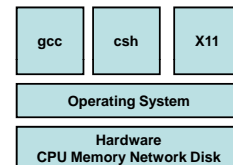
Why are OS interesting?

- OS are "magic"
 - Most people don't understand them – including sysadmins and computer scientists!
- OS are incredibly complex systems
 - "Hello, World" – program really 1 million lines of code
- 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

- Can take a wider view or a narrower definition what an OS is
- Wide view: Windows, Linux, Mac OSX are operating systems
 - Includes system programs, system libraries, servers, shells, GUI etc.
- Narrow definition:
 - OS often equated with the *kernel*.
 - The Linux kernel; the Windows executive – the special piece of software that runs with special privileges and actually controls the machine.
- In this class, usually mean the narrow definition.
- In real life, always take the wider view. (Why?)

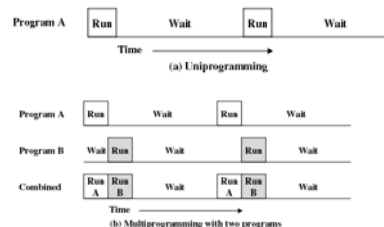
Evolution of OS

- OSs as a library
 - Abstracts away hardware, provide neat interfaces
 - Makes software portable; allows software evolution
 - Single user, single program computers
 - No need for protection: no malicious users, no interactions between programs
 - Disadvantages of uniprogramming model
 - Expensive
 - Poor utilization

Evolution of OS (II)

- 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, and OS from applications
- This requirement is **absolute**
 - In Pintos also: if one application crashes, kernel should not! Bulletproof.
- Three techniques
 - Preemption
 - Interposition
 - Privilege

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 4ms in Linux
 - Solves infinite loop problem!
- Q.: 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: hit
 - Carefully control transitions between user & kernel mode

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

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 bursty, typically
- Increases utilization

Resource Management (2)

- Multiplexing increases complexity
- Car Analogy (by Rosenblum):
 - Dedicated road per car would be incredibly inefficient, so cars share freeway. Must manage this.
 - (abstraction) different lanes per direction
 - (synchronization) traffic lights
 - (increase capacity) build more roads
- 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 jams

Resource Management (3)

- 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