EECS 678: Introduction to Operating Systems

Instructor: Heechul Yun
About Me

• Heechul Yun, Assistant Prof., Dept. of EECS
  – Office: 3040 Eaton, 236 Nichols
  – Email: heechul.yun@ku.edu

• Research Areas
  – Operating systems and architecture support for embedded/real-time systems
    • To improve time predictability, energy efficiency, and throughput
    • Multicore, memory systems

• Previously
  – Systems software engineer, Samsung Electronics, Nvidia
    • mainly worked on Linux kernel

• More Information
  – http://ittc.ku.edu/~heechul
About This Class

• Textbook: Operating System Concepts
• Objectives: Learn OS basics and practical system programming skills
  – Understand *how it works!*
• Audience: Senior and Junior undergraduate (grad students)
• Course website: [http://ittc.ku.edu/~heechul/courses/eecs678/](http://ittc.ku.edu/~heechul/courses/eecs678/)
About This Class

• Course structure
  – Lecture
    • Office hour: MF 11:00 – 11:50 @ 3040 Eaton
    • Discuss OS concepts and the design of major OS components
  – Lab
    • Hands-on system programming experiences.
    • Each lab includes lab discussion and an assignment
    • TA will help you better understand the concepts learned during the lecture.
  – Programming projects
    • Design and implement some parts of OS (e.g., shell, scheduler)
    • 3 projects are expected, each will be given 2~3 weeks to finish
    • To do in groups of two persons. Solo project needs permission
About This Class

• Grading
  – Exam: 50% (mid:20%, final:30%)
  – Quiz: 5%
  – Lab: 15%
  – Programming projects: 30%
About This Class

• Late submissions
  – 20% off / day

• Cheating
  – You can discuss about code and help find bugs of your peers. However, copying another’s code or writing code for someone else is cheating and, if identified, the involved students will be notified to the department chair
Operating Systems Are Everywhere

- Computers
- Smart phones
- Cars
- Airplanes
- ...  
- Almost everything
What is an Operating System?
What is an Operating System?

- A program that acts as an intermediary between users and the computer hardware.
What is an Operating System?

• An easy to use virtual machine – *User’s view*
  – Hide complex details for you.
    • What CPU am I using? Intel or AMD?
    • How much memory do I have?
    • Where and how to store my data on the disk?
  – Provide APIs and services
    • read(...), write(..)
    • Virtual memory, filesystems, ...
What is an Operating System?

• A resource manager – *System’s view*
  – Make everybody get a fair share of resources
    • Time and space multiplexing hardware resources
  – Monitor/prevent error or improper use

![Windows Explorer dialog box](image-url)
What is an Operating System?

• Is an internet browser part of an OS?
  – Everything that shipped by the OS vendor?
  – What about ‘solitaire’?

• The program that always runs
  – Typically in kernel mode (we will learn it later)
Why Needed?

• Programmability
  – You don’t need to know hardware details to do stuffs

• Portability
  – You can run the same program on different hardware configurations

• Safety
  – The OS protects your program from faults in other programs

• Efficiency
  – Multiple programs/users can share the same hardware efficiently
What to Study?

• Not “how to use”
  – I’m sure you know better than me about how to use the iOS in your iPhone.

• But “how it works!”
  – We will study the underlying concepts, standard OS components and their designs
OS Design Issues

• Structure
  – How to organize the OS?
• Communication
  – How to exchange data among different programs?
• Performance
  – How to maximize/guarantee performance and fairness?
• Naming
  – How to name/access resources?
• Protection
  – How to protect with each other?
• Security
  – How to prevent unauthorized access?
• Reliability
  – How to prevent system crash?
Why Study?

• I’m a user
  – Have you ever wondered how it works?
  – You can better tune the OS to improve performance (or save energy)

• I’m a system programmer
  – You can write more efficient programs by knowing how the OS works.

• I’m a hacker
  – You need to know the enemy (the OS) to beat it
Brief History of Computers

• Early computing machines
  – Babbage’s analytical engine
  – First programmer: Ada Lovelace

• Vacuum tube machines
  – 1940s ~ 1950s
  – Used to break code in WWII
  – No OS, No PL
Brief History of Computers

• Vacuum tubes $\rightarrow$ Transistors $\rightarrow$ IC $\rightarrow$ VLSI
  – Smaller, faster, and more reliable
  – Enable smaller computers
• 1960s Mainframes
• 1970s Minicomputers
• 1980s Microprocessor, Apple, IBM PC
• 1990s PC, Internet
• 2000s Cloud computing
• 2010s Mobile, Internet-of-things (IoT)
Evolution of Operating Systems

• Batch systems
  – Each user submits her job on punch cards
  – Collect a batch of jobs, read the batch before start processing
  – The ‘OS’ processes each job at a time
  – Problems
    • No interactivity
    • CPU is underutilize to wait I/O operations

IBM 029 card punch

http://www.catb.org/esr/writings/taouu/html/ch02s01.html
Evolution of Operating Systems

- Multiprogramming
  - Multiple runnable jobs at a time
  - I/O and compute can overlap
  - OS goal: maximize system throughput
  - IBM OS/360
Evolution of Operating Systems

• Timesharing
  – Multiple interactive users sharing a machine
  – Each user accesses the machine via a terminal
  – Provide each user an illusion of using the entire machine
  – OS goal: optimize response time
  – UNIX
Evolution of Operating Systems

• Parallel computing
  – Use multiple CPUs/cores to speed up performance
  – OS goal: fast synchronization, max utilization

• Distributed computing
  – Physically separate networked computers

• Virtualization
  – Multiple OSes on a single machine
Challenges for Future OS

• New kinds of hardware are keep coming
  – Heterogeneous multicore processors (e.g., ARM big.LITTLE)
  – Storage Class Memory (SCM): non-volatile DRAM-like memories

• New computing paradigms
  – Cloud computing
  – Internet-of-Things (IoT)
Summary

• In this class, you will learn
  – Major OS components
  – Their structure, interface, mechanisms, policies, and algorithms

• This class will (hopefully) help you
  – Understand the foundation of computing systems
  – Understand various engineering trade-offs in designing complex systems you would build in future
Computer Architecture and OS
Recap

• What is an OS?
  – An intermediary between users and hardware
  – A program that is always running
  – A resource manager
    • Manage resources efficiently and fairly
  – A easy to use virtual machine
    • providing APIs and services
Agenda

• Computer architecture and OS
  – CPU, memory, disk
  – Architecture trends and their impact to OS
  – Architectural support for OS
Computer Architecture and OS

• OS talks to hardware
  – OS needs to know the hardware features
  – OS drives new hardware features
Simplified Computer Architecture

A von Neumann architecture
A Computer System

– Essentials: CPU, Memory, Disk
– Others: graphic, USB, keyboard, mouse, ...
Central Processing Unit (CPU)

• The brain of a computer
  – Fetch instruction from memory
  – Decode and execute
  – Store results on memory/registers

• Moore’s law
  – Transistors double every 1~2yr
  – 5.56 billion in a 18-core Intel Xeon Haswell-E5
Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)

Single-core CPU

- Time sharing
  - When to schedule which task?
Multicore CPU

- Parallel processing
  - Which tasks to which cores?
    - May have performance implication due to cache contention → contention-aware scheduling
• Non-uniform memory access (NUMA) architecture
  – Memory access cost varies significantly: local vs. remote
  – Which tasks to which processors?
Memory Hierarchy

• Main memory
  – DRAM
  – Fast, volatile, expensive
  – CPU has direct access

• Disk
  – Hard disks, solid-state disks
  – Slow, non-volatile, inexpensive
  – CPU doesn’t have direct access.
Memory Hierarchy

Fast, Expensive

Slow, Inexpensive
Storage Performance

• Performance of various levels of storage depends on
  - distance from the CPU, size, and process technology used
• Movement between levels of storage hierarchy can be explicit or implicit

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>registers</td>
<td>cache</td>
<td>main memory</td>
<td>solid state disk</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Typical size</td>
<td>&lt; 1 KB</td>
<td>&lt; 16MB</td>
<td>&lt; 64GB</td>
<td>&lt; 1 TB</td>
<td>&lt; 10 TB</td>
</tr>
<tr>
<td>Implementation technology</td>
<td>custom memory with multiple ports CMOS</td>
<td>on-chip or off-chip CMOS SRAM</td>
<td>CMOS SRAM</td>
<td>flash memory</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Access time (ns)</td>
<td>0.25 - 0.5</td>
<td>0.5 - 25</td>
<td>80 - 250</td>
<td>25,000 - 50,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>Bandwidth (MB/sec)</td>
<td>20,000 - 100,000</td>
<td>5,000 - 10,000</td>
<td>1,000 - 5,000</td>
<td>500</td>
<td>20 - 150</td>
</tr>
<tr>
<td>Managed by</td>
<td>compiler</td>
<td>hardware</td>
<td>operating system</td>
<td>operating system</td>
<td>operating system</td>
</tr>
<tr>
<td>Backed by</td>
<td>cache</td>
<td>main memory</td>
<td>disk</td>
<td>disk</td>
<td>disk or tape</td>
</tr>
</tbody>
</table>
Caching

• A very important principle applied in all layers of hardware, OS, and software
  – Put frequently accessed data in a small amount of faster memory
  – Fast, most of the time (hit)
  – Copy from slower memory to the cache (miss)
Architectural Support for OS

- Interrupts and exceptions
- Protected modes (kernel/user modes)
- Memory protection and virtual memory
- Synchronization instructions
Interrupt

• What is an interrupt?
  – A signal to the processor telling “do something now!”

• Hardware interrupts
  – Devices (timer, disk, keyboard, ...) to CPU

• Software interrupts (exceptions)
  – Divide by zero, special instructions (e.g., int 0x80)
Interrupt Handling

- save CPU states (registers)
- execute the associated interrupt service routine (ISR)
- restore the CPU states
- return to the interrupted program
Timesharing

• Multiple tasks share the CPU at the same time
  – But there is only one CPU (assume single-core)
  – Want to schedule different task at a regular interval of 10 ms, for example.

• Timer and OS scheduler tick
  – The OS programs a timer to generate an interrupt at every 10 ms.
Dual (User/Kernel) Mode

• Some operations must be restricted to the OS
  – accessing registers in the disk controller
  – updating memory management unit states
  – ...

• User/Kernel mode
  – Hardware support to distinguish app/kernel
  – Privileged instructions are only for kernel mode
  – Applications can enter into kernel mode only via pre-defined system calls
User/Kernel Mode Transition

- System calls
  - Programs ask OS services (privileged) via system calls
  - Software interrupt. “int <num>” in Intel x86
Memory Protection

• How to protect memory among apps/kernel?
  – Applications shouldn’t be allowed to access kernel’s memory
  – An app shouldn’t be able to access another app’s memory
Virtual Memory

• How to overcome memory space limitation?
  – Multiple apps must share limited memory space
  – But they want to use memory as if each has dedicated and big memory space
  – E.g.,) 1GB physical memory and 10 programs, each of which wants to have a linear 4GB address space
Virtual Memory

Process A

Process B

Process C

MMU

Physical Memory
MMU

- Hardware unit that translates *virtual address* to *physical address*
  - Defines the boundaries of kernel/apps
  - Enable efficient use of physical memory
Synchronization

• Synchronization problem with threads

Deposit(account, amount) {
    account->balance += amount;
}

**Thread 1:** Deposit(acc, 10)  
**Thread 2:** Deposit(acc, 10)

LOAD R1, account->balance  
LOAD R1, account->balance

ADD R1, amount  
ADD R1, amount

STORE R1, account->balance  
STORE R1, account->balance
Synchronization Instructions

• Hardware support for synchronization
  – TestAndSet, CompareAndSwap instructions
  – Atomic load and store
  – Used to implement lock primitives
  – New TSX instruction → hardware transaction

• Another methods to implement locks in single-core systems
  – Disabling interrupts
Summary

• OS needs to understand architecture
  – Hardware (CPU, memory, disk) trends and their implications in OS designs

• Architecture needs to support OS
  – Interrupts and timer
  – User/kernel mode and privileged instructions
  – MMU
  – Synchronization instructions
## OS Abstractions

<table>
<thead>
<tr>
<th>Reality</th>
<th>Abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>A single computer</td>
<td>Multiple computers</td>
</tr>
<tr>
<td>Limited RAM capacity</td>
<td>Infinite capacity</td>
</tr>
<tr>
<td>Mechanical disk</td>
<td>File system</td>
</tr>
<tr>
<td>Insecure and unreliable networks</td>
<td>Reliable and secure</td>
</tr>
</tbody>
</table>