Lecture 1: Course Overview

CSE 120: Principles of Operating Systems

UC San Diego: Summer Session I, 2009
Frank Uyeda
Goals for Today

• Introduction to course:
  – Personnel, policies and administration
• Provide high-level overview of course material
  – Specific goals
• Define an operating system
  – Identify its role
• Identify the hardware mechanisms available to the Operating System

• Note: some slide content and images are adapted from Alvin AuYoung, Geoff Voelker, Alex Snoeren (UC San Diego) and Anthony Joseph (UC Berkeley)
Who am I?

• PhD candidate in CSE department
  – UCSD undergrad (Warren college) and UCSD graduate student
  – Working with Professor George Varghese & Amin Vahdat in Systems and Networking group
  – Conduct research and publish papers in area

• I am fascinated by this topic, and would like to share my excitement for it with you.
Basic Information

• Instructor: Frank Uyeda
  – E-mail: fuyeda@cs.ucsd.edu
  – Office Hours: Tues 2-4p, CSE 2126
    • Or by appointment
• TA: Matus Telgarsky
  – E-mail: mtelgars@cs.ucsd.edu
  – Office Hours: Wed?

• Meeting times:
  – Lecture: MWF, 4p-5:50p, Center 201
  – Discussion: Thursday @ ??
  – Final Exam: Saturday, August 1, 3pm
Discussion Section

• Survey: When would you like to have discussion section?
  – Thursday 11a – 12p
  – Thursday 12p – 1p
  – Thursday 1p – 2p
Purpose of this Course

• Goal is to help you understand structure and design of modern operating systems:
  – Lectures and readings to distill information
  – Homework sets and programming projects to reinforce ideas

• At the end of the course you should be able to:
  – Identify common design challenges for an OS
  – Understand traditional mechanisms and policies used to address these challenges
  – Hypothesize how changes in technology will impact existing OS mechanisms or policies
Prerequisites

• Courses
  – CSE 100 (Data Structures), CSE 101 (Algorithms) required
  – CSE 141 (not required but useful)

• Programming
  – Java-based projects. CSE 100 should be adequate.

• Hours per week
  – Students averaged 12 hours/week in regular term (CAPE evaluations)
  – ≈ 24 hours/week in summer?
Homework

• Purpose
  – Reinforce concepts through practice (there are challenging concepts in this class)

• Three homework assignments
  – Assignments due on Monday’s at the beginning of class.
  – No late assignments
  – Collaboration encouraged, but must be completed individually
  – All answers must be legible – if we can’t read it, you don’t get credit.
PeerWise

• Submit potential test questions and answers.
• Read and rate questions submitted by others.
• http://peerwise.cs.auckland.ac.nz/

• Assignment:
  – Submit 1 question and review 10 others.
  – Due every Tuesday and Thursday @ 11:59pm.

• I will show some of the best/worst questions in class.
• At least one question submitted through PeerWise will show up on the Final!
Projects

• Purpose
  – Hands-on experience: implementing is much trickier than discussing concepts.

• Nachos
  – Instructional-based Operating System developed at UC Berkeley (Tom Anderson), running on machine simulator
  – You’ll expand Nachos in 3 project components.
  – Significant portion of course (very time consuming)
  – No late assignments accepted (3 slip days allowed)
Projects (2)

• Rough outline:
  – Project 1: Concurrency and synchronization
  – Project 2: Multiprogramming
  – Project 3: Virtual Memory

• Groups
  – Projects may be done on your own or with a partner.
  – You will have the same partner for the whole course.
  – Please identify your partner early and notify me via email.
Exams

• Midterm
  – Date: Wednesday, 7/17/09, 4p-5:20p
  – Location: Center 201

• Final
  – Date: Saturday, 8/01/08, 3p-6p
  – Location: Center 201

• Make-up policy
  – Only under extreme, university-approved circumstances
Class Resources

- **Class Web page**
  - http://www.cs.ucsd.edu/classes/su09/cse120
  - announcements and schedule updates

- **Message board**
  - http://webct.ucsd.edu
  - Forum to discuss issue related to the projects.

- **Textbooks**
    - Bookstore currently carries the 8th edition (this is fine, too).
Class Resources (2)

• Labs
  – You should have access to a Linux account and lab machines in basement of EBU 3B

• You are free to work from home
  – Nachos works on Eclipse/Windows
  – We will grade on Linux machines, so make sure code runs on lab machines as well
Grading

• Homework assignments (15%)
• PeerWise participation (5%)
• Programming projects (30%)
• Midterm exam (20%)
• Final exam (30%)

• Grades will be assigned on a straight scale:
  – A: >90; B: >80; C: >70; etc...
  – +’s and –’s given at the discretion of instructor
Common CSE 120 Pitfalls

• Don’t spend time on the homework
  – It’s only 15% of my grade
  – Concepts may seem easy, but actually require significant practice to fully understand
  – Mastery of concepts come in handy for projects and exams

• Put off the project
  – I’ve got better things to do, or my partner will do it
  – Hardly anybody finishes them early – testing alone will require significant time
  – Project-related questions may end up on the exam
Common CSE 120 Pitfalls (2)

• Don’t ask questions
  – *I don’t want to look stupid*
  – Matus and I are here to help
  – Chances are many other people have the same question
  – Putting off questions will just create more questions
Modern Computer Systems

- Routers
- Telematics systems
- Sensor nodes
- Cell phones and handheld devices
- Gaming consoles

Lecture 1: Course overview
Computer Organization

What is the Operating System?

CPU

Video controller

USB controller

Disk controller

System bus

Monitor

Scanner

Keyboard

Printer

Disk drive

Memory

Lecture 1: Course overview
What is an Operating System?

The OS is “all the code you didn’t have to write” to implement your applications.
What is an Operating System?

The OS is “the program that is always running”.

Applications

Operating Systems

Hardware

Principles of Operating Systems (CSE 120)
Advanced Operating Systems (CSE 121)

Compilers (CSE 130, 131)
Databases (CSE 132A,B)
Web Apps (CSE 134A,B)
...

Architecture (CSE 141/L)
Digital Systems (CSE 140/L)

Lecture 1: Course overview
Why Study Operating Systems?

• Understand what you use
  – Which operating system is right for me?
  – My computer is slow and I’m not sure why!
  – Am I protected against worms, viruses, and other security vulnerabilities?

• Learn the past before you create the future
  – Pervasive abstractions (e.g., threads and synchronization)
  – Evolution of OSes have followed advances in technology

• Develop experience developing complex system
  – Large system with many moving parts (HW, SW)
  – Many design goals (efficiency, flexibility, security,...)
Course Content

- **Role of an Operating System**
- **Concurrency and Synchronization**
  - What are the challenges to support multitasking?
- **CPU Scheduling**
  - Who gets the CPU and when do they get it?
- **Memory Management**
  - How do programs share finite memory?
- **File Systems and Storage Management**
  - How is data stored and what happens when things fail?
- **Selection of Special Topics (time permitting)**
  - Security, Virtual Machine Monitors, .....
Role of an Operating System

Provide abstractions to ease application development

Applications

Operating Systems

Hardware

Lecture 1: Course overview
Role of an Operating System

Provide abstractions to ease application development

- An application needs to be able to talk to each device:
  - What if a resource is busy/fails?
  - What if the hardware changes?
- Every application will be extremely complex
  - Redundant, error-prone code
  - Not portable (hardware limitations)
Role of an Operating System

- Manages resource access among applications

- OS schedules resource access
  - Significant performance impact
  - Flexible sharing policies

- OS can provide protection mechanisms
  - Privacy (file permissions)
  - Flexible policies (groups)

- OS can provide coordination mechanisms
  - Communication (pipes, cut+paste)
  - Synchronization (monitors, semaphores)
Recap: Role of an Operating System

- Provide abstractions
- Manage resources
- As a result:
  - Hard to properly define boundaries
  - OSes becoming increasingly complex
Break for 5 Minutes

• What belongs in the Operating System vs. Applications?
  – Get user input from peripheral device
    • OS – central management for device drivers and common interface exposed to user programs
  – Set the system time/date
    • OS - all other programs/users share this value.
  – Render a webpage
    • Depends who you ask!
What Isn’t an Operating System?

Difficult to define boundaries

• United States v Microsoft (2000)
  – Microsoft bundled IE with Windows
  – US DOJ brings anti-trust case against Microsoft

• Outcome
  – Basic question: is browser part of the OS?
  – CS academics testified “no”
  – Settlement
    • Microsoft must share API with third-party companies
Role of an Operating System

OSes becoming increasingly complex

Source: Larry O’Brien, knowing.net (2008)
Role of an Operating System

OSES becoming increasingly complex

Hardware Support

• Why do we need to discuss hardware?
  – Key goal of the OS is to provide resource sharing.
  – With good OS design, applications can be oblivious to HW details (abstraction).
  – The OS stands between the HW and the applications, so it must consider the requirements of both.
Hardware Support

• Early processors offered little in the way of protection
  – A process could access any memory location
  – A process could access any hardware device
  – Could implement protection in software, but really slow...

• Hardware features evolve to meet needs of OS
  – Ex. TLB, Test and set instruction, No execute bit
Types of Hardware Support

• Protection
  – OS wants to control hardware access
  – OS wants to keep particular data (registers, memory) private and isolated

• Event handling
  – OS needs to be aware of external events
  – OS needs a way to respond to external events

• Performance improvements Later in the course
  – Commonly used thread synchronization primitives
  – Memory access / Disk access speed-ups
Protection

• Hardware features
  – Protection levels, privileged instructions (CPU)
  – Virtual Memory (MMU)

• What do we need to protect?
  – Direct access to particular devices, such as storage devices, or input/output peripherals. (why?)
  – Memory state, such as page table entries, TLB entries
  – Special operations, such as changing protection level, interrupt handling, halting the machine

Mark particular instructions as privileged
Supporting Privileged Instructions

• Hardware (CPU) offers protection levels
  – Processors actually provide many levels: x86 architecture supports at least four modes ("rings")
  – Linux and most operating systems use two modes: user and kernel mode

• How does an OS or CPU use protection levels?
  – There is a special register on the CPU indicating the mode
  – CPU checks mode bit when executing privileged instructions
  – User programs execute in user mode
  – OS executes in kernel mode
    • For now, think of the kernel as equivalent to an OS
User and Kernel Mode

• Privileged instructions can only be executed while in kernel mode
  – Which instructions are privileged?
• Examples of privileged instructions:
  – Manipulating memory state, direct access to HW
• Examples of non-privileged instructions
  – Anything that can’t bother other programs/apps/processes
• Why don’t we just make every instruction privileged?
## Unix Organization

### User Mode
- **Applications**
  - (the users)
- **Standard Libs**
  - shells and commands
  - compilers and interpreters
  - system libraries

### Kernel Mode
- **Kernel**
  - system-call interface to the kernel
    - signals
    - terminal handling
    - character I/O
    - system
    - terminal drivers
  - file system
  - swapping block I/O system
  - disk and tape drivers
  - CPU scheduling
  - page replacement
  - demand paging
  - virtual memory

### Hardware
- terminal controllers
- terminals
- device controllers
- disks and tapes
- memory controllers
- physical memory
Example: Memory Protection

• Remember: OS must protect programs from each other and protect itself from programs

• We’ve talked about the CPU, but what about Memory?
  – Contains a lot of program state
  – Memory Management Unit (MMU) provides protection
    • Base and limit registers, segments, page tables, TLB, Virtual Memory
    • Manipulation of these structures are protected
  – The particulars of these structures will make more sense when we talk about memory management
Events

• An event is an unnatural change in control flow
  – Events can stop or change current execution path
  – Events can change machine state (protection level, registers, etc...)

• Kernel defines a **handler** for each event type
  – Event handlers run in kernel mode
  – Events are the only entry into the kernel
Events

- Two types of events
  - **Interrupts**: caused by external event
  - **Exceptions**: caused by an instruction

- Events can be:
  - Unexpected
  - Deliberate

<table>
<thead>
<tr>
<th></th>
<th>Unexpected</th>
<th>Deliberate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception (sync)</td>
<td>Fault</td>
<td>Syscall trap</td>
</tr>
<tr>
<td>Interrupt (async)</td>
<td>Interrupt</td>
<td>Software Interrupt</td>
</tr>
</tbody>
</table>
Interrupts

• A signal is sent to the OS on a physical wire (Interrupt-request line)
• Hardware interrupts
  – Generally these events are asynchronous
  – e.g., Network packet arrival, DMA completion
• Hardware/Software Timers
  – Special type of event
  – Allow OS to regain control (why?)
Handling Interrupts

- **Polling**
  - Kernel busy-waits for signal
    - Note: textbook uses slightly different definition and distinguishes it completely from IRQ
    - Can be inefficient depending on frequency

- **Interrupt Handler**
  - Interrupt causes processor to trap into kernel
  - Kernel invokes appropriate interrupt handler
  - Interrupt handler saves process state, processes interrupt, restores state and returns
Issues with Interrupts

• What if an interrupt occurs while in an interrupt handler?
  – Can mask interrupts (see Nachos)
  – Interrupt priority (support different policies)

• What can go wrong?
  – Faulty driver
    • In interrupt chaining, OS queries each driver’s interrupt handler
    • False positive can create flood of interrupts
  – Bad state (Live Lock)
    • Receiver live-lock: too many packets create too much overhead, no progress can be made

– Possible solutions
  • Disable interrupts
  • Polling mode
Exceptions

• Traps
  – System call or execution of a privileged instruction
  – Note: textbook calls this a “software interrupt”

• Faults
  – e.g., Page faults, segmentation fault, divide by zero
  – Similar to Java Exception handling
  – Not needed for correctness, but a performance optimization
System Calls

• Application invokes a **system call** to execute a privileged operation
  – Raises an exception, which invokes kernel handler
  – Passes parameters to handler
    • Predefined System Call code, any other arguments...
  – Kernel handler does the rest
    • saves caller context (state)
    • calls proper system call routine
    • restores called state and returns
System Call example: read()

1. User app: `read(file_reference, buffer_addr, n_bytes)`

User space

4. Trap to kernel

Kernel space

2. Push arguments onto stack

User stack

3. Call Library procedure for read()

read syscall code in register

5. Kernel Handler

6. read syscall handler

7. return to caller

8. Return from read() call
System Calls (2)

- How does the kernel return parameters to user mode?
  - What is file_reference in read() syscall?
    - If it’s the filename, then kernel has to resolve file name and locate file each time
  - What if kernel just passes “address” or “pointer” to the file?
    - User would be referencing a kernel object (bad)
  - Unix solution: file descriptors
    - An object handle unique to a process (literally an integer)
    - Essentially a capability to use a resource
Exceptions

• Traps
  – System call or execution of a privileged instruction
  – Note: textbook calls this a “software interrupt”

• Faults
  – e.g., Page faults, segmentation fault, divide by zero
  – Similar to Java Exception handling
  – Not needed for correctness, but a performance optimization
Faults

• Some faults are fixed silently by the OS
  – Process resumes without knowledge of fault
  – Fault handler handles instruction resumption
• Other faults need to be handled by application
  – Instead of returning control from the faulting instruction, OS fault handler returns control to a user-level handler
  – Applications register user-level handlers with OS
    • Unix signals
    • Windows NT Asynchronous Procedure Calls
A problem has been detected and Windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c
Faults (2)

• Fatal faults
  – Faults that are not handled by OS or user application
    • core files contain process state for crashed programs
  – This type of fault within the kernel results in an OS crash
    (Windows BSOD, Unix kernel panic)
Next Time

• Read Chapters 1 and 2 (this lecture), 3, 13.1-13.5 (next)
• Start thinking about project groups.
  – Use WebCT discussion board to find partners.
• Homework and Project will be posted tomorrow.
• First PeerWise question & reviews due on Thursday.
• Check Web site for course announcements
  – http://www.cs.ucsd.edu/classes/su09/cse120