Operating Systems Overview

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Why Do We Study OS?

- To make a better OS or system
  - Functionality
  - Performance/cost
  - Reliability
  - Energy efficiency
- To make a new hardware up and running
- To design OS-aware hardware
- To understand computer systems better
- Just for fun
Computer Systems

- Computer system organization
What is OS? (1)

- **Application view**
  - Provides an execution environment for running programs
  - Provides an abstract view of the underlying computer system
    - Processors $\rightarrow$ Processes, Threads
    - Memory $\rightarrow$ Address spaces (virtual memory)
    - Storage $\rightarrow$ Volumes, Directories, Files
    - I/O Devices $\rightarrow$ Files (ioctl)
    - Networks $\rightarrow$ Files (sockets, pipes, ...)
    - ...
What is OS? (2)

- System view
  - Manages various resources of a computer system
    - Sharing
    - Protection
    - Fairness
    - Efficiency
    - ...

Resources
- CPU
- Memory
- I/O devices
- Queues
- Energy
- ...

What is OS? (3)

- Implementation view
  - Highly-concurrent, event-driven software

![Diagram showing trap, System call, Interrupts, and Hardware]
Operating System Services (1)

- Operating systems provide
  - An execution environment to programs
  - Helpful services to programs and users

- Operating-system services
  - User interface (UI)
    - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
  - Program execution
    - Load a program into memory and to run that program
    - End execution, either normally or abnormally
  - I/O operations
    - A running program may require I/O, which may involve a file or an I/O device
Operating System Services (2)

- Operating-system services (cont’d)
  - **File-system manipulation**
    - Programs need to read, write, create and delete files and directories
    - Search, list, and permission management
  - **Communications**
    - Processes may exchange information, on the same computer or between computers over a network
    - Via shared memory or through message passing
  - **Error detection**
    - Being constantly aware of possible errors
    - In CPU, memory, I/O devices and user programs
    - OS should take the appropriate action to ensure correct and consistent computing
    - Debugging support (ex. core dump)
Another set of OS functions

- **Resource allocation**
  - Resources must be allocated to multiple users and jobs running concurrently
  - Many types of resources - CPU cycles, main memory, file storage, I/O devices.

- **Accounting**
  - To keep track of which users use how much and what kinds of computer resources

- **Protection and security**
  - Concurrent processes should not interfere with each other
  - **Protection**
    » All accesses to system resources are controlled
  - **Security**
    » User authentication, depending external I/O devices from invalid access attempts
Computer Systems

- Services
- Applications
- Middleware
  - Software Development Environment
    (compilers, loaders, editors, utilities, command interpreter, libraries)
- Operating System (Kernel)
- Computer System Architecture (Hardware Platform)
OS Internals

User space

Kernel space

System Call Interface

File System Management
I/O Management (device drivers)
Memory Management
Hardware Control (Interrupt handling, etc.)

Process Management
- scheduler
- IPC
- synchronization

Protection

Hardware

shell
ls
shell
ps
trap
Architectural Support for OS

- Interrupts and exceptions
- DMA (Direct Memory Access)
- Timers
- Protected or privileged instructions
- At least two modes of CPU operation
  - Kernel mode vs. user mode
  - Mode change instruction: TRAP
- MMU (Memory Management Unit)
  - Implements virtual memory
  - Memory protection
  - Fast address translation (TLB)
- Atomic instructions
Process Concept

- **Process in memory**

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-4k</td>
<td>Unused</td>
</tr>
<tr>
<td>4k-16k</td>
<td>Read-only segment (.init, .text, .rodata)</td>
</tr>
<tr>
<td>16k-32k</td>
<td>Read/write segment (.data, .bss)</td>
</tr>
<tr>
<td>32k-64k</td>
<td>Run-time heap (managed by malloc)</td>
</tr>
<tr>
<td>64k-1024k</td>
<td>User stack (created at runtime)</td>
</tr>
<tr>
<td>1G-2T</td>
<td>Memory (code, data, heap, stack)</td>
</tr>
</tbody>
</table>

- **brk**: Next address after heap expansion
- **code**
- **data**

**Virtual Memory**

- Kernel virtual memory is invisible to user code.
Process State Transition

- new
- admitted
- interrupt
- exit
- terminated

- ready
  - scheduler dispatch
  - I/O or event completion

- running
  - I/O or event wait

- waiting
Processes vs. Threads (1)

(a) Process 1

(b) Process
Processes vs. Threads (2)

- Processes vs. Threads
  - A thread is bound to a single process.
  - A process, however, can have multiple threads.
  - Sharing data between threads is cheap: all see the same address space.
  - Threads become the unit of scheduling.
  - Processes are now containers in which threads execute.
  - Processes become static, threads are the dynamic entities.
Process Address Space

Address space

0x00000000

0xFFFFFFFF

Stack (dynamically allocated mem)

Heap (dynamically allocated mem)

Static data (data segment)

Code (text segment)

PC

SP
Address Space with Threads

- Address space
  - 0xFFFFFFFF
  - 0x00000000

- Code (text segment)
  - PC (T1)
  - PC (T2)
  - PC (T3)

- Static data (data segment)
  - 

- Heap (dynamically allocated mem)
  - SP (T1)
  - SP (T2)
  - SP (T3)

- Thread stacks
  - Thread 1 stack
  - Thread 2 stack
  - Thread 3 stack

- SP (T1)
- SP (T2)
- SP (T3)
# threads per addr space:  
# of addr spaces: 

<table>
<thead>
<tr>
<th>One</th>
<th>Many</th>
</tr>
</thead>
</table>
| MS/DOS  
Early Macintosh | Traditional UNIX |
| Many embedded Oses  
(VxWorks, uClinux, ..) | Mach, OS/2, Linux, Windows, Mac OS X, Solaris, HP-UX |
System Calls

Application processes

System call functions

Other kernel functions

Hardware
Categories of System Calls

- Process control
  - Create, terminate, wait, load, etc.

- File management
  - Create, delete, open, close, read, write, reposition, etc.

- Device management
  - Request, release, read, write, etc.

- Information management
  - Get/set time/date, get process/file/device attributes, etc.

- Communications
  - Create/delete connection, send/receive messages, etc.

- Etc.
System Call Handling in User-Level

Application

C Library

Kernel

strncpy()

gettimeofday()
What is actually implemented...
Protection

- **User mode**
  - Processes in user mode can access its own instructions and data (but, not kernel instructions and data)

- **Kernel mode**
  - Processes in kernel mode can access kernel and user addresses, and also can execute privileged instructions

- **Kernel distinguishes among processes**

- **Hardware distinguishes mode of execution**

<table>
<thead>
<tr>
<th>Kernel Mode</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>K</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User Mode</th>
<th></th>
<th>U</th>
<th>U</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Privileged Instructions

- Machine instructions that can be executed only in kernel mode

- Examples
  - I/O controls
  - Timer management
  - Interrupt management
  - Switching to user mode
  - Etc.

- Manipulate memory management register
  - LGDT, LLDT, LTR, LIDT, SGDT, SLDT, SIDT, STR
- Load and store control registers
  - MOV {CR0~CR4}
- Invalidate cache and TLB
  - INVD, WBINVD, INVLPG
- Performance monitoring
  - RDPMC, RDTSC, RDTSCP
- Fast System Call
  - SYSEXIT, (SYSENTER)
System Call and Execution Mode

- **User Process**: Mode bit = 0 → Execute system call
- **Kernel**: Mode bit = 1 → Trap mode bit = 0
- **Kernel Mode**: Return mode bit = 1

- **User Mode**: Mode bit = 1 → Calls system call
- **Return from System Call**
Interrupts and Exceptions

- OS is basically interrupt-driven
  - Hardware interrupt
    - Timer
    - I/O
  - Software interrupt
    - System call
    - Exception
Sharing Resources

User

Kernel

Hardware
Synchronization Mechanism

- Disabling interrupts
- Spinlocks
  - Busy waiting
- Semaphores
  - Binary semaphore = mutex (= lock)
  - Counting semaphore
- Monitors
  - Language construct with condition variables
- Mutex + Condition variables
  - Pthreads
CPU Scheduling (1)

- Non-preemptive scheduling
  - The scheduler waits for the running job to voluntarily yield the CPU.
  - Jobs should be cooperative.

- Preemptive scheduling
  - The scheduler can interrupt a job and force a context switch.
  - What happens
    - If a process is preempted in the midst of updating the shared data?
    - If a process in a system call is preempted?
CPU Scheduling (2)

- Priority scheduling
  - Choose job with highest priority to run next
  - SJF = Priority scheduling, where priority = expected length of CPU burst
  - Round-robin or FIFO within the same priority
  - Can be either preemptive or non-preemptive
  - Priority is dynamically adjusted.
  - Modeled as a Multi-level Feedback Queue (MLFQ)
Memory Management

- **Goals**
  - To provide a convenient abstraction of memory for programming.
  - To allocate scarce memory resources among competing processes to maximize performance with minimal overhead.
  - To provide isolation between processes.
Memory Protection

- **Requirements**
  - OS must protect user programs from each other
    - Malicious users
  - OS must also protect itself from user programs
    - Integrity and security
Memory Protection

- Simplest scheme
  - Use base and limit registers
  - Base and limit registers are loaded by OS before starting a program

![Diagram showing base and limit registers for Prog A, B, and C]
Memory Protection

- MMU (memory management unit)
  - Memory management hardware provides more sophisticated memory protection mechanisms
    - Base and limit registers
    - page table pointers, page protection, TLBs
    - virtual memory
    - segmentation
  - Manipulation of memory management hardware are protected (privileged) operations
Paging (1)

Virtual memory

Process B
- Page 3
- Page 2
- Page 1
- Page 0

Process A
- Page 5
- Page 4
- Page 3
- Page 2
- Page 1
- Page 0

Page tables

Physical memory

- Frame 11
- Frame 10
- Frame 9
- Frame 8
- Frame 7
- Frame 6
- Frame 5
- Frame 4
- Frame 3
- Frame 2
- Frame 1
- Frame 0
Paging (2)

- **Address translation architecture**

![Diagram of address translation architecture](image)

- Logical address
- Physical address
- Page table
- Physical memory
- f0000 0000
- f1111 1111
Demand Paging

- Handling a page fault

1. Trap
2. Page is on backing store
3. Operating system
4. Bring in missing page
5. Reset page table
6. Restart instruction
TLBs

- Address translation with TLB
Memory Reference

- **Situation**
  
  - Process is executing on the CPU, and it issues a read to a (virtual) address.

![Memory Reference Diagram](image_url)
A Typical PC Bus Structure

- Monitor
- Processor
- Cache
- Memory
- IDE disk controller
- Expansion bus interface
- Expansion bus
- Parallel port
- Serial port
- Keyboard
- SCSI bus
- Bridge/memory controller
- SCSI controller
- Disk
- Graphics controller
I/O Software Layers

- User-level I/O Software
- Device-independent I/O Software
- Device Drivers
- Interrupt Handlers

Hardware

Network
I/O Subsystems

- One purpose of OS is to hide peculiarities of hardware devices from the user
- General device-driver interface
- Drivers for specific hardware devices
- I/O subsystem includes
  - Buffering (storing data temporarily while it is being transferred)
  - Caching (storing parts of data in faster storage for performance)
  - Spooling (the overlapping of output of one job with input of other jobs)
Memory Hierarchy

- registers
- cache
- main memory
- solid-state disk
- hard disk
- optical disk
- magnetic tapes
Storage: A Logical View

- Abstraction given by block device drivers:

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - file
  - Each medium is controlled by device (i.e., disk drive, tape drive)
  - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)

- Operations
  - Identify(): returns N
  - Read(start sector #, # of sectors)
  - Write(start sector #, # of sectors)

Source: Sang Lyul Min (Seoul National Univ.)
File System Basics (1)

- For each file, we have
  - File contents (data)
    - File systems normally do not care what they are
  - File attributes (metadata)
    - File size
    - Owner, access control lists
    - Creation time, last access time, last modification time, ...
  - File name

- File access begins with...
  - File name
    - open ("/etc/passwd", O_RDONLY);
File System Basics (2)

- File system: A mapping problem
  - \(<\text{filename, data, metadata}> \rightarrow <\text{a set of blocks}>\)
File System Basics (3)

- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and directories
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media
File System Basics (4)

- **Goals**
  - Performance + Reliability

- **Design issues**
  - What information should be kept in metadata?
  - How to locate metadata?
    - Mapping from pathname to metadata
  - How to locate data blocks?
  - How to manage metadata and data blocks?
    - Allocation, reclamation, free space management, etc.
  - How to recover the file system after a crash?
  - ...

File System Internals

System call interface

Virtual File System (VFS)

minix  nfs  ext2  dosfs  ...  mmfs  procfs

buffer cache

device driver

File System

SDE5007: Special Topics on IC Design II | Spring 2016 | Jinkyu Jeong (jinkyu@skku.edu)