Operating Systems Review

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Why Do We Study OS?

- To make a better OS or system.
  - Fine-tune the performance.
  - Add, modify, or enhance functionalities in a transparent way.

- To make a new hardware up and running.

- To design OS-aware hardware.

- To understand computer systems better.

- Just for fun!
What is OS? (1)

- **Application view**
  - Provides an execution environment for running programs
  - Provides an abstract view of the underlying computer system
    - Processors → Processes, Threads
    - Memory → Address spaces (virtual memory)
    - Storage → Volumes, Directories, Files
    - I/O Devices → Files (ioctl)
    - Networks → 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
  - ...

---

ECE5658: Operating System Design | Summer 2010 | Jin-Soo Kim (jinsookim@skku.edu)
What is OS? (3)

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

```
trap

System call

Interrupts

Hardware
```
## OS Classification

<table>
<thead>
<tr>
<th>Feature</th>
<th>MS-DOS</th>
<th>Windows 98</th>
<th>Windows 2000/Vista</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-user</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Multi-task</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Multi-process</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Multi-processor</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Multi-thread</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>
OS Internals

User space

- shell
- ls
- ps

Kernel space

System Call Interface

- File System Management
- Memory Management
- I/O Management (device drivers)
- Hardware Control (Interrupt handling, etc.)
- Process Management
  - scheduler
  - IPC
  - synchronization

Protection

Hardware
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**

![Diagram of memory segments]

- **In memory**
  - **0**: Unused
  - **read-only segment** (.init, .text, .rodata)
  - **read/write segment** (.data, .bss)
  - **run-time heap** (managed by malloc)
  - **user stack** (created at runtime)
  - **kernel virtual memory** (code, data, heap, stack)

- **Program**
  - **Code**
  - **Data**

- **Memory invisible to user code**

---

---
Process State Transition

- New
- Admitted
- Ready
- Running
- Waiting
- I/O or event completion
- Scheduler dispatch
- I/O or event wait
- Interrupt
- Exit
- Terminated
Processes vs. Threads (1)

(a) Processes

(b) Threads
Processes vs. Threads

- 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

0xFFFFFFFF

address space

0x00000000

stack
(dynamic allocated mem)

heap
(dynamically allocated mem)

static data
(data segment)

code
(text segment)

SP

PC
Address Space with Threads

0xFFFFFFFF

address space

0x00000000

thread 1 stack

thread 2 stack

thread 3 stack

heap (dynamically allocated mem)

static data (data segment)

code (text segment)

SP (T1)

SP (T2)

SP (T3)

PC (T2)

PC (T3)

PC (T1)
## Classification

<table>
<thead>
<tr>
<th># threads per addr space:</th>
<th># of addr spaces:</th>
<th>One</th>
<th>Many</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>One</td>
<td>MS/DOS Early Macintosh</td>
<td>Traditional UNIX</td>
</tr>
<tr>
<td>Many</td>
<td>Many embedded Oses (VxWorks, uClinux, ..)</td>
<td>Mach, OS/2, Linux, Windows, Mac OS X, Solaris, HP-UX</td>
<td></td>
</tr>
</tbody>
</table>
Sharing Resources
Synchronization Mechanisms

- Disabling interrupts
- Spinlocks
  - Busy waiting
- Semaphores
  - Binary semaphore = mutex ($\equiv$ 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 for programming.
  - To allocate scarce memory resources among competing processes to maximize performance with minimal overhead.
  - To provide isolation between processes.

- **Why is it so difficult?**
Paging (1)
Paging (2)

- Address translation architecture
Demand Paging

- Handling a page fault

1. Trap
2. Page is on backing store
3. Load M
4. Bring in missing page
5. Reset page table
6. Restart instruction
Two-level Page Tables

- Two-level page tables
  - Virtual addresses have 3 parts:
    - Master page table: master page number → secondary page table.
    - Secondary page table: secondary page number → page frame number.

<table>
<thead>
<tr>
<th>Master page #</th>
<th>Secondary page #</th>
<th>Offset</th>
</tr>
</thead>
</table>

**Diagram:**
- Logical address: \(p_1 \| p_2 \| d\)
- \(p_1\) to outer page table
- \(p_2\) to page of page table
- \(d\)
TLBs

- Address translation with TLB
Memory Reference

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

- Abstraction given by block device drivers:

  - 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**
  - `<filename, data, metadata> \(\rightarrow\) <a set of blocks>`

```
<table>
<thead>
<tr>
<th></th>
<th>meta1</th>
<th>1</th>
<th>4</th>
<th><code>a.out</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>meta2</td>
<td></td>
<td>3</td>
<td>2</td>
<td><code>dog.jpg</code></td>
</tr>
</tbody>
</table>
```

ECE5658: Operating System Design | Summer 2010 | Jin-Soo Kim (jinsookim@skku.edu)
File System Basics (3)

- **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

Virtual File System (VFS)

- minix
- nfs
- ext2
- dosfs
- mmfs
- procfs

System call interface

buffer cache

device driver

File System