Operating Systems

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
What is an OS?
What is an OS?

- Application view
- System view
- Implementation view
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 space (virtual memory)
  - Storage $\rightarrow$ Volumes, Directories, Files
  - I/O Devices $\rightarrow$ Files
  - ...
System View

▪ Manages various resources of a computer system

▪ Goals
  • Sharing
  • Protection
  • Fairness
  • Efficiency
  • ...

Resources

▪ CPU
▪ Memory
▪ I/O devices
▪ Queues
▪ Energy
▪ ...

Implementation View

- Highly-concurrent, event-driven software
System Calls
OS Structure

User Application

C Library (libc)

System Call Interface

Kernel

Arch-dependent kernel code

Hardware Platform

User Space

Kernel Space
Kernel Internals

System Call Interface

- Shell
- Ls
- Process Management
  - Scheduler
  - IPC
  - Synchronization
- Memory Management
- I/O Management (device drivers)
- File System Management
- Hardware Control (Interrupt handling, etc.)

User space

Kernel space

Protection

Hardware
Processes and Threads
Abstracting CPUs

- **Process**
  - An instance of a program in execution
  - Resource allocation unit

- **Thread**
  - A sequential flow of control
  - Scheduling unit

- **Task**
  - Process or thread
Thread (1)

- A thread of control
- Usually consists of
  - A program counter (PC)
  - A stack to keep track of local variables and return addresses
  - Registers
- Threads share the instructions and most of its data
Thread (2)

- Key abstraction
  - Each thread has its own logical control flow.
  - Thread executions interleaved by the scheduler
  - What is running a thread?

![Thread Diagram]

- Time
  - Thread A
  - Thread B
  - Thread C

 quantum
 or
 time slice
Thread (3)

- Context switching
  - Control flow passes from one thread to another via a context switch.

![Diagram showing context switching between two threads, Thread A and Thread B, with user and kernel code sections and labeled context switches.]
Thread State Transition

Diagram showing the state transitions of a thread, including states such as "new", "admitted", "interrupt", "exit", "terminated", "ready", "running", "waiting", "I/O or event completion", "scheduler dispatch", and "I/O or event wait".
Process (1)

- An instance of a program in execution
- A process includes
  - Hardware execution state (PC, SP, registers, ...)
  - OS resources (e.g., memory, open files)
  - Other information (PID, state, owner, ...)
- Two key abstractions
  - Logical control flow (virtual CPU)
  - Private address space (virtual memory)
Process (2)

- Used to run a user program
- Protection domain
- Creating a new process is costly
- Inter-process communication is costly, since it must usually go through the OS
Private Address Spaces

- Kernel virtual memory (code, data, heap, stack)
- Memory mapped region for shared libraries
- Run-time heap (managed by malloc)
- Read/write segment (.data, .bss)
- Read-only segment (.init, .text, .rodata)
- Unused

Memory invisible to user code
%esp (stack pointer)
brk

Loaded from the executable file
0xffffffff
0x00000000
0x40000000
0x80480000
0
Virtual Memory

- Allows to run programs much larger than the available physical memory size
- Simplifies memory management
- Provides memory protection
Implementing Processes

Process

Kernel Entity
Process Descriptor

- **Linux: struct task_struct**
- **Everything the kernel has to know about a process or a thread**
- **About 1.7KB on a 32-bit machine**
- **Task list: the list of task structures in a circular doubly linked list**
Processes with Threads

- 1:1 model
Threads vs. Processes

- A thread is bound to a single process (or a single address space)
- A process can have multiple threads
- Sharing data between threads is cheap: all see the same address space
- Threads are the unit of scheduling
- Processes are containers in which threads execute
- Processes become static, threads are the dynamic entities
# 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>