Architectural Support for Operating Systems

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Today’s Topics

- Basic computer system architecture
- Interaction between OS and architecture
- Architectural support for OS
Computer Systems (1)

- Computer system organization
Characteristics

- I/O devices and CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- CPU issues specific commands to I/O devices
- CPU should be able to know whether the issued command has been completed or not
OS and Architecture

- Mutual interaction
  - The functionality of an OS is limited by architectural features.
    - Multiprocessing on DOS/8086?
  - The structure of an OS can be simplified by architectural support.
    - Interrupt, DMA, etc.
  - Most proprietary OS’s were developed with the certain architecture in mind.
    - SunOS/Solaris for SPARC architecture
    - IBM AIX for Power/PowerPC architecture
    - HP-UX for PA-RISC architecture
    - ...
Interrupts (1)

- How does the kernel notice an I/O has finished?
  - Polling
  - Hardware interrupt
Interrupts (2)

- **Interrupt handling**
  - Preserves the state of the CPU
    - In a fixed location
    - In a location indexed by the device number
    - On the system stack
  - Determines the type
    - Polling
    - Vectored interrupt system
  - Transfers control to the interrupt service routine (ISR) or interrupt handler
Exceptions (1)

- **Interrupts**
  - Generated by hardware devices
    - Triggered by a signal in INTR or NMI pins (x86)
  - Asynchronous

- **Exceptions**
  - Generated by software executing instructions
    - INT instruction in x86
  - Synchronous
  - Exception handling is same as interrupt handling
Exceptions (2)

- **Further classification of exceptions**
  
  - **Traps**
    - Intentional
    - System calls, breakpoint traps, special instructions, ...
    - Return control to “next” instruction
  
  - **Faults**
    - Unintentional but possibly recoverable
    - Page faults (recoverable), protection faults (unrecoverable), ...
    - Either re-execute faulting (“current”) instruction or abort
  
  - **Aborts**
    - Unintentional and unrecoverable
    - Parity error, machine check, ...
    - Abort the current program
Exceptions (3)

- **System calls**
  - Programming interface to the services provided by OS
  - e.g., system call sequence to copy the contents of one file to another

![Diagram showing system call sequence](image)
### Important system calls (POSIX & Win32)

<table>
<thead>
<tr>
<th>Process Management</th>
<th>Important System Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>fork</td>
<td>CreateProcess</td>
</tr>
<tr>
<td>waitpid</td>
<td>WaitForSingleObject</td>
</tr>
<tr>
<td>execve</td>
<td>(none)</td>
</tr>
<tr>
<td>exit</td>
<td>ExitProcess</td>
</tr>
<tr>
<td>kill</td>
<td>(none)</td>
</tr>
<tr>
<td>open</td>
<td>CreateFile</td>
</tr>
<tr>
<td>close</td>
<td>CloseHandle</td>
</tr>
<tr>
<td>read</td>
<td>ReadFile</td>
</tr>
<tr>
<td>write</td>
<td>WriteFile</td>
</tr>
<tr>
<td>lseek</td>
<td>SetFilePointer</td>
</tr>
<tr>
<td>stat</td>
<td>GetFileAttributesEx</td>
</tr>
<tr>
<td>chmod</td>
<td>(none)</td>
</tr>
</tbody>
</table>

- **Process Management**
  - fork: Create a new process
  - waitpid: Wait for a process to exit
  - execve: CreateProcess = fork + execve
  - exit: Terminate execution
  - kill: Send a signal

- **File Management**
  - open: Create a file or open an existing file
  - close: Close a file
  - read: Read data from a file
  - write: Write data to a file
  - lseek: Move the file pointer
  - stat: Get various file attributes
  - chmod: Change the file access permission

- **File System Management**
  - mkdir: Create a new directory
  - rmdir: Remove an empty directory
  - link: Make a link to a file
  - unlink: Destroy an existing file
  - mount: Mount a file system
  - umount: Unmount a file system
  - chdir: Change the current working directory
Exceptions (5)

- Implementing system calls

```
user application

open()

system call interface

kernel mode

user mode

... ...

Implementation of open()

... ...

return
```
Exceptions (6)

- Implementing system calls (cont’d)

```c
count = read(fd, buffer, nbytes);
```
### DMA (1)

- **Data transfer modes in I/O**
  - **Programmed I/O (PIO)**
    - CPU is involved in moving data between I/O devices and memory
    - By special I/O instructions vs. by memory-mapped I/O
  - **DMA (Direct Memory Access)**
    - Used for high-speed I/O devices able to transmit information at close to memory speeds
    - Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
    - Only an interrupt is generated per block.
DMA (2)

- Processing I/O requests
Timers

- How does the OS take control of CPU from the running programs?
  - Use a hardware timer that generates a periodic interrupt
  - The timer interrupt transfers control back to OS
  - The OS preloads the timer with a time to interrupt.
    - 10ms for Linux 2.4, 1ms for Linux 2.6
    - (cf.) time slice
  - The timer is privileged.
    - Only the OS can load it
Protected Instructions

- Protected or privileged instructions
  - Direct I/O access
    - Use privileged instructions or memory-mapping
  - Memory state management
    - Page table updates, page table pointers
    - TLB loads, etc.
  - Setting special “mode bits”
  - Halt instruction
How does the processor know if a protected instruction should be executed?

- The architecture must support at least two modes of operation: kernel and user mode
  - 4 privilege levels in IA-32: Ring 0 > 1 > 2 > 3
- Mode is set by a status bit in a protected processor register
  - User programs in user mode, OS in kernel mode
  - Current Privilege Level (CPL) in IA-32: CS register
- Protected instructions can only be executed in the kernel mode
OS Protection (2)

- Crossing protection boundaries
  - User programs must call an OS to do something privileged.
    - OS defines a sequence of system calls
  - There must be a system call instruction that:
    - causes an exception, which invokes a kernel handler
    - passes a parameter indicating which system call to invoke
    - saves caller’s state (registers, mode bits) so they can be restored
    - OS must verify caller’s parameters (e.g. pointers)
    - must provide a way to return to user mode when done.
    - (cf.) INT 0x80 in Linux
Making a system call
- System call changes mode to kernel
- Return from system call resets it to user
Memory Protection (1)

- **Requirements**
  - OS must protect user programs from each other
    - Malicious users
  - OS must also protect itself from user programs
    - Integrity and security
Memory Protection (2)

- Simplest scheme
  - Use base and limit registers
  - Base and limit registers are loaded by OS before starting a program
Memory Protection (3)

- **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
Synchronization

- **Problems**
  - Interrupt can occur at any time and may interfere with the interrupted code.
  - OS must be able to synchronize concurrent processes.

- **Synchronization**
  - Turn off/on interrupts
  - Use a special atomic instructions
    - read-modify-write (e.g., INC, DEC)
    - test-and-set
    - LOCK prefix in IA32
    - LL (Load Locked) & SC (Store Conditional) in MIPS