### Lecture Topics

- **Today:** Computer System Overview  
  (Stallings, chapter 1.1-1.8)

- **Next:** Operating System Overview  
  (Stallings, chapter 2.1-2.4, 2.8-2.10)

### Announcements

- Syllabus and calendar available
- Consulting hours posted
- Self-Study Exercise #1 posted
- Self-Study Exercise #2 posted
Operating Systems Overview

Operating system: software that serves as the interface between application programs and computer hardware

Manages resources (hardware):

- Processor (CPU)
- Primary storage (RAM)
- Secondary storage (disk)

Example: Linux Overview
Hardware Support for OS

Desired capabilities of operating system influence hardware – OS and architecture evolve in tandem. Examples:

- Privileged instructions
- Interrupts
- DMA (direct memory access)
Privileged Instructions

- Can only be executed by the OS (user code cannot execute privileged instructions)
  - ex: initialize PC
  - ex: place value in PSW

- Dual-mode operation: status bit for user mode or kernel (supervisor) mode

- Some processors support more than two modes (additional status bits)

Interrupts

- Fundamental to operation of modern computers and integral part of OS design

- Mechanism to allow hardware modules to get the attention of the OS
  - ex: I/O device signals that operation is complete
  - ex: Timer signals that quantum is over

- Interrupts can occur at any time
Modified Instruction Cycle

CPU checks for interrupts after each instruction, invokes interrupt handler

Examples of Interrupts

- I/O (ex: operation completed)
- Timer (ex: time slice completed)
- Hardware failure (ex: memory parity error)
- Program Exception (ex: divide by zero)
- Program Trap (control transfer instruction to request interrupt)
Interrupt Processing

- OS determines the nature of the interrupt and performs the necessary actions
- State of the interrupted process must be saved so that process can be resumed later as if nothing had happened
- Note that some interrupts are completely unrelated to the current process
Interrupt Processing

- The state of a process is all of the bit patterns related to that process:
  - machine language instructions in RAM
  - data objects in RAM
  - contents of CPU registers

- OS must preserve the state of the process which was interrupted: leave RAM alone, save contents of registers

Interrupt Processing

- What process state has to be preserved?
  - PC – address where interrupt occurred
  - PSW – status bits (including user/kernel mode)
  - Any general-purpose registers which are used by the OS to handle the interrupt

- Hardware automatically saves PC and PSW; software (interrupt service routine) saves any general-purpose registers that it uses
Interrupt Processing

- When an interrupt occurs, the hardware does the following simultaneously:
  - acknowledges the interrupt
  - turns on kernel mode
  - saves the PC and the PSW
  - places the address of the ISR into the PC

- Since the PC now contains the entry point of an interrupt service routine, that function is executed (next instruction fetched)

Note: the details for each architecture are different (example is an abstraction)
Interrupt Processing

- The address of the appropriate interrupt service routine is in a table; the hardware uses the interrupt number as the index

<table>
<thead>
<tr>
<th>IR #</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000000080</td>
</tr>
<tr>
<td>1</td>
<td>00000140</td>
</tr>
<tr>
<td>2</td>
<td>00000020</td>
</tr>
<tr>
<td>3</td>
<td>00000238</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Address of ISR for IR #0
Address of ISR for IR #3

Interrupt Service Routines (ISR)

- Each ISR does the following:
  - Finish saving state of interrupted process
  - Process interrupt
  - Restore state of interrupted process
  - Restore old PC and old PSW

- Note: back in interrupted process (old PC)
- Note: back in user mode (old PSW)
Example: SPARC microprocessor

Each entry in the ISR table contains 16 bytes; since each machine language instruction is 4 bytes in size, one table entry contains the first 4 instructions for the ISR (usually a jump to the rest of that ISR).

The ISR table is loaded into a particular area of memory when the machine is booted; that address is the start of the ISR table.

Example: SPARC microprocessor

The TBR (trap base register) contains three fields:

- TBA: upper bits of ISR table address
- tt: trap (interrupt number)
- Last 4 bits are always zero (since each entry is 16 bytes).
Example: SPARC microprocessor

Assume ISR table is at 0x00004000 in memory.

Table entry addresses:

- trap 0 0x00004000
- trap 1 0x00004010
- trap 2 0x00004020
- ...
- ...
- trap ff 0x00004ff0

Interrupt Categories

- Hardware interrupt – signal from device
- Software interrupt – signal from program which is currently executing
  - Exception – error caused by current instruction
  - Trap – specific machine language instruction to cause interrupt in a controlled way
Summary: Interrupt Handling

- An interrupt can occur at any instant
- An interrupt may be related to the current process (software interrupt) or it may be completely unrelated (hardware interrupt)
- OS handles interrupts through a combination of hardware (initial steps) and software (ISRs)
- ISRs should be small and fast

Multiple Interrupts

- More than one interrupt can occur at the same time or while an ISR is executing. Two main strategies:
  - Sequential interrupt processing
  - Nested interrupt processing
- Nested processing allows processing in priority order (each type of interrupt is assigned a relative priority).
Sequential Interrupt Processing

Nested Interrupt Processing
Example

DMA (Direct Memory Access)

Allow I/O modules to transfer blocks of bytes into (or out of) RAM without using the CPU to copy the bytes.

- CPU initiates I/O operation
- I/O module starts processing; CPU used for other purposes
- I/O uses interrupt to signal completion
Evolution of I/O Function

1. CPU directly controls a peripheral device

2. Controller or I/O module is added
   • CPU uses programmed I/O without interrupts
   • CPU does not need to handle details of external devices

3. Controller or I/O module with interrupts
   • CPU does not spend time waiting for an I/O operation to be performed

4. Direct Memory Access
   • blocks of data are moved into memory without involving the CPU
   • CPU involved at beginning and end only
Techniques for Performing I/O

- Programmed I/O
  - I/O command is issued
  - CPU uses "busy waiting" until operation is completed

- Interrupt-driven I/O
  - I/O command is issued
  - CPU continues executing instructions
  - I/O module sends an interrupt when completed

Programmed I/O

- I/O module performs action on behalf of processor
- I/O module sets status bit when done (does not issue interrupt)
- Processor cycles wasted checking status of I/O module
Interrupt-Driven I/O

- Processor interrupted when I/O module ready to exchange data
- Processor is free to do other work -- no needless waiting
- Still consumes a lot of processor cycles because every word read or written passes through the processor

Techniques for Performing I/O

- Direct Memory Access (DMA)
  - DMA module controls exchange of data between main memory and I/O device
  - Processor interrupted only after entire block has been transferred
  - Processor is only involved at the beginning and end of the transfer -- free to perform other tasks during data transfer
Direct Memory Access

- CPU issues request to DMA module (separate module or incorporated into I/O module)
- DMA module transfers a block of data directly to or from memory (without going through CPU)
- An interrupt is issued when the task is complete

CPU to DMA:
- read/write request
- address of I/O device
- address in memory
- number of bytes to transfer

DMA transfers bytes from I/O device to RAM
DMA issues interrupt
**Memory Hierarchy**

- Small cache of expensive but very fast memory interacting with slower but much larger memory
- Processor first checks if word referenced to is in cache
- If not found in cache, a block of memory containing the word is moved to the cache

**Cache Memory**
The Hit Ratio

- Hit ratio = fraction of access where data is in cache
- $T_1$ = access time for fast memory
- $T_2$ = access time for slow memory
- $T_2 \gg T_1$

- When hit ratio is close to 1, the average access time is close to $T_1$

Locality of Reference

- Memory references for both instructions and data values tend to cluster over a long period of time.

- Example: once a loop is entered, there is frequent access to a small set of instructions.

- Hence: once a byte gets referenced, it is likely that nearby bytes will get referenced often in the near future.