Computer System Overview

Operating Systems

2005/S2
What are the objectives of an Operating System?
What are the objectives of an Operating System?

- **convenience & abstraction**
  - the OS should facilitate the task of application and system programmer
  - hardware details should be hidden, uniform interface for different I/O devices provided

- **efficiency**
  - should take up few resources, make good use of resources, and be fast

- **protection**
  - fairness, security, safety
Layers of a Computer System

- Application Programs
- Utilities
- Operating System
- Computer Hardware

End User
Programmer
Operating-System Designer
Basic Elements

Simplified view:

- Processor
BASIC ELEMENTS

Simplified view:

➔ Processor
➔ Main Memory

- referred to as real memory or primary memory
- volatile
**Simplified view:**

- **Processor**
- **Main Memory**
  - referred to as real memory or primary memory
  - volatile
- **I/O modules**
  - secondary memory devices
  - communications equipment
  - terminals
BASIC ELEMENTS

Simplified view:

- Processor
- Main Memory
  - referred to as real memory or primary memory
  - volatile
- I/O modules
  - secondary memory devices
  - communications equipment
  - terminals
- System bus
  - communication among processors, memory, and I/O modules
TOP-LEVEL COMPONENTS

- Monitor
- Keyboard
- Floppy disk drive
- Hard disk drive
- CPU
- Memory
- Video controller
- Keyboard controller
- Floppy disk controller
- Hard disk controller

Bus
EXAMPLE: LARGE PENTIUM SYSTEM

Level 2 cache → CPU → PCI bridge → Main memory

Cache bus → Local bus → Memory bus

PCI bus

SCSI → USB → ISA bridge → IDE disk → Graphics adaptor

Available PCI slot

Mouse → Keyboard → ISA bus

Modem → Sound card → Printer

Available ISA slot

EXAMPLE: LARGE PENTIUM SYSTEM
PROCESSOR

- Fetches instructions from memory, decodes and executes them
- Set of instructions is processor specific
PROCESSOR

➜ Fetches instructions from memory, decodes and executes them
➜ Set of instructions is processor specific
➜ Instructions include:

✳ load value from memory into register
✳ combine operands from registers or memory
✳ branch
PROCESSOR

- Fetches instructions from memory, decodes and executes them
- Set of instructions is processor specific
- Instructions include:
  - load value from memory into register
  - combine operands from registers or memory
  - branch
- All CPU’s have registers to store
  - key variables and temporary results
  - information related to control program execution
**PROCESSOR REGISTERS**

→ Data and address registers
  
  - Hold operands of most native machine instructions
  - Enable programmer to minimize main-memory references by optimizing register use
  - user-visible
**Processor Registers**

- **Data and address registers**
  - Hold operands of most native machine instructions
  - Enable programmer to minimize main-memory references by optimizing register use
  - User-visible

- **Control and status registers**
  - Used by processor to control operating of the processor
  - Used by operating-system routines to control the execution of programs
  - Sometimes not accessible by user (architecture dependent)
USER-VISIBLE REGISTERS

➤ May be referenced by machine language instructions
➤ Available to all programs - application programs and system programs
➤ Types of registers
- Data
- Address
  - Index
  - Segment pointer
  - Stack pointer
- Many architectures do not distinguish different types
CONTROL AND STATUS Registers

=> Program Counter (PC)
**CONTROL AND STATUS REGISTERS**

- **Program Counter (PC)**
  - Contains the address of an instruction to be fetched
CONTROL AND STATUS REGISTERS

➔ Program Counter (PC)
  • Contains the address of an instruction to be fetched

➔ Instruction Register (IR)
CONTROL AND STATUS REGISTERS

- **Program Counter (PC)**
  - Contains the address of an instruction to be fetched

- **Instruction Register (IR)**
  - Contains the instruction most recently fetched
CONTROL AND STATUS REGISTERS

- **Program Counter (PC)**
  - Contains the address of an instruction to be fetched

- **Instruction Register (IR)**
  - Contains the instruction most recently fetched

- **Processor Status Word (PSW)**
CONTROL AND STATUS REGISTERS

- **Program Counter (PC)**
  - Contains the address of an instruction to be fetched

- **Instruction Register (IR)**
  - Contains the instruction most recently fetched

- **Processor Status Word (PSW)**
  - condition codes
  - interrupt enable/disable
  - supervisor/user mode
CONTROL AND STATUS REGISTERS

→ Condition Codes or Flags

- Bits set by the processor hardware as a result of operations
- Can be accessed by a program but not altered
- Examples
  - positive/negative result
  - zero
  - overflow
INSTRUCTION FETCH AND EXECUTE

- Program counter (PC) holds address of the instruction to be fetched next
- The processor fetches the instruction from memory
- Program counter is incremented after each fetch
- Overlapped on modern architectures (pipelining)
INSTRUCTION REGISTER

- Fetched instruction is placed in the instruction register
- Types of instructions
  - Processor-memory
    - transfer data between processor and memory
  - Processor-I/O
    - data transferred to or from a peripheral device
  - Data processing
    - arithmetic or logic operation on data
  - Control
    - alter sequence of execution
INTERACTION BETWEEN PROCESSOR AND I/O DEVICES

- CPU much faster than I/O devices
**Interaction between Processor and I/O Devices**

- CPU much faster than I/O devices
  - waiting for I/O operation to finish is inefficient
  - not feasible for mouse, keyboard
INTERACTION BETWEEN PROCESSOR AND I/O DEVICES

- CPU much faster than I/O devices
  - waiting for I/O operation to finish is inefficient
  - not feasible for mouse, keyboard
- I/O module sends an interrupt to CPU to signal completion
- Interrupts normal sequence of execution
- Interrupts are also used to signal other events
Classes of Interrupts

- Asynchronous (external) events
  - I/O
  - Timer
  - Hardware failure
- Synchronous interrupts or program exceptions
Classes of Interrupts

- Asynchronous (external) events
  - I/O
  - Timer
  - Hardware failure

- Synchronous interrupts or program exceptions caused by program execution:
  - arithmetic overflow
  - division by zero
  - execute illegal instruction
  - reference outside user’s memory space
**Interrupt Cycle**

1. Fetch next instruction
2. Execute instruction
3. Check for interrupt
4. If no interrupts, fetch the next instruction
5. If an interrupt is pending, divert to the interrupt handler
**Interrupt Handler**

- A program that determines nature of the interrupt and performs whatever actions are needed
- Control is transferred to this program by the hardware
- Generally part of the operating system
CONTROL FLOW WITH AND WITHOUT INTERRUPTS

(a) No interrupts

(b) Interrupts; short I/O wait

(c) Interrupts; long I/O wait
MULTIPLE INTERRUPTS

- Interrupt X occurs
- CPU disables all interrupts (only those with lower priority)
- Interrupt handler may enable interrupts
- Interrupt Y occurs
- Sequential or nested interrupt handling
Interrupt X occurs
CPU disables all interrupts (only those with lower priority)
Interrupt handler may enable interrupts
Interrupt Y occurs
Sequential or nested interrupt handling
MULTIPLE INTERRUPTS

Sequential Order:

- Disable interrupts so processor can complete task
- Interrupts remain pending until the processor enables interrupts
- After interrupt handler routine completes, the processor checks for additional interrupts
Multiple Interrupts

Priorities:

- Higher priority interrupts cause lower-priority interrupts to wait
- Causes a lower-priority interrupt handler to be interrupted
- Example: when input arrives from communication line, it needs to be absorbed quickly to make room for more input
Memory

Sould be

- fast
- abundant
- cheap

Unfortunately, that's not the reality...

Solution:

combination of fast & expensive and slow & cheap
Memory

Should be

⇒ fast
⇒ abundant
⇒ cheap

Unfortunately, that’s not the reality...
Memory

Should be

- fast
- abundant
- cheap

Unfortunately, that’s not the reality...

Solution:

- combination of fast & expensive and slow & cheap memory
MEMORY HIERARCHY

Typical access time

- 1 nsec: Registers
- 2 nsec: Cache
- 10 nsec: Main memory
- 10 msec: Magnetic disk
- 100 sec: Magnetic tape

Typical capacity

- <1 KB
- 1 MB
- 64-512 MB
- 5-50 GB
- 20-100 GB
GOING DOWN THE HIERARCHY

- Decreasing cost per bit
- Increasing capacity
- Increasing access time
- Decreasing frequency of access of the memory by the processor

Locality of reference is essential!
Disk Cache

- A portion of main memory used as a buffer to temporarily hold data for the disk
- Disk writes are clustered
- Some data written out may be referenced again. The data are retrieved rapidly from the software cache instead of slowly from disk
- Mostly transparent to operating system
Cache Memory

- Contains a portion of main memory
- Processor first checks cache
- If not found in cache, the block of memory containing the needed information is moved to the cache replacing some other data
Cache/Main Memory System

(a) Cache

- Line Number
- Tag
- Block

- C — 1
- Block Length (K Words)

(b) Main Memory

- Memory address
- Word Length

- Block (K words)
- Block

- $2^n — 1$
CACHE DESIGN

➔ Cache size
  • small caches have a significant impact on performance

➔ Line size (block size)
  • the unit of data exchanged between cache and main memory
  • hit means the information was found in the cache
  • larger line size ⇒ higher hit rate
    until probability of using newly fetched data becomes less than the probability of reusing data that has been moved out of cache
CACHE DESIGN

-> Mapping function
  - determines which cache location the data will occupy

-> Replacement algorithm
  - determines which line to replace
  - Least-Recently-Used (LRU) algorithm

-> Write policy
  - When the memory write operation takes place
  - Can occur every time line is updated (write-through policy)
  - Can occur only when line is replaced (write-back policy)
    - Minimizes memory operations
    - Leaves memory in an obsolete state
**Interaction between I/O Devices and Processor**

- Controller (chip or set of chips) provides a simple interface to OS
  - often, embedded OS running on the controller
- Software that communicates with controller is called
INTERACTION BETWEEN I/O DEVICES AND PROCESSOR

➢ Controller (chip or set of chips) provides a simple interface to OS
  - often, embedded OS running on the controller
➢ Software that communicates with controller is called device driver
➢ Most drivers run in kernel mode
➢ To put new driver into kernel, system may have to
  - be relinked
  - be rebooted
  - dynamically load new driver
PROGRAMMED I/O (Polling)

- I/O module performs the action, not the processor
- Sets appropriate bits in the I/O status register
- No interrupts occur
- Processor checks status until operation is complete
  - Wastes CPU cycles

(a) Programmed I/O

- Issue Read command to I/O module
- Read status of I/O module
- Check status
- Read word from I/O Module
- Write word into memory
- Done?

(b) Interrupt-driven I/O

- Issue Read block command to I/O module
- Read status of DMA module
- Next instruction

(c) Direct memory access

- Issue Read block command to I/O module
- Read status of DMA module
- Next instruction

(a) Programmed I/O
**Interrupt-Driven I/O**

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

(a) Programmed I/O

1. CPU → I/O
2. I/O → CPU
3. I/O → CPU
4. Error condition
5. Ready
6. Read status of I/O module
7. Check status
8. Read word from I/O Module
9. Write word into memory
10. CPU → memory
11. Done?
12. Next instruction

(b) Interrupt-driven I/O

1. Issue Read command to I/O module
2. CPU → I/O
3. Do something else
4. Interrupt
5. Read status of I/O module
6. Error condition
7. Ready
8. Check status
9. Read word from I/O Module
10. I/O → CPU
11. Write word into memory
12. CPU → memory
13. Done?
14. Next instruction
**DIRECT MEMORY ACCESS**

- Transfers a block of data directly to or from memory
- An interrupt is sent when the task is complete
- The processor is only involved at the beginning and end of the transfer

---

**a) Programmed I/O**

- CPU → I/O
- I/O → CPU
- Error condition

---

**b) Interrupt-driven I/O**

- CPU → I/O
- I/O → CPU
- Do something else

---

**c) Direct memory access**

- Issue Read block command to I/O module
- Read status of DMA module
- CPU → DMA
- DMA → CPU
- Interrupt
- Next instruction