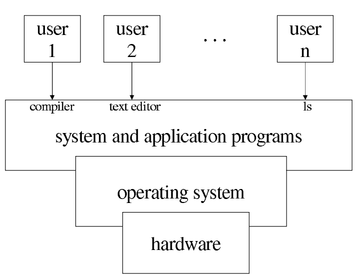
Operating System Basics

CS 537 - Introduction to Operating Systems

Definition

- An operating system is an intermediary between a computer user and the hardware.
- Make the hardware convenient to use.
- Manages system resources.
- Use the hardware in an efficient manner.

System Diagram



Types of Systems

- Batch
  - submit large number of jobs at one time
  - system decides what to run and when
- Time Sharing
  - multiple users connected to single machine
  - few processors, many terminals
- Single User Interactive
  - one user, one machine
  - traditional personal computer

Types of Systems

- Parrallel
  - traditional multiprocessor system
  - higher throughput and better fault tolerance
- Distributed
  - networked computers
- Real Time
  - very strict response time requirements
  - hardware or software

Single Tasking System

- Only one program can perform at a time
- Simple to implement
  - only one process attempting to use resources
- Few security risks
- Poor utilization of the CPU and other resources
- Example: MS-DOS

Multitasking System

- Very complex
- Serious security issues
  - how to protect one program from another sharing the same memory
- Much higher utilization of system resources
- Example: Unix, Windows NT

Hardware Basics

- OS and hardware closely tied together
- Many useful hardware features have been invented to compliment the OS
- Basic hardware resources
  - CPU
  - Memory
  - Disk
  - I/O

CPU

- CPU controls everything in the system
  - if work needs to be done, CPU gets involved
- Most precious resource
  - this is what your paying for
  - want to get high utilization (from useful work)
- Only one process on a CPU at a time
- Hundreds of millions of instructions / sec
  - and getting faster all the time

Memory

- Limitted in capacity
  - never enough memory
- Temporary (volatile) storage
- Electronic storage
  - fast, random access
- Any program to run on the CPU must be in memory

Disk

- Virtually infinite capacity
- Permanent storage
- Orders of magnitude slower than memory
  - mechanical device
  - millions of CPU instructions can execute in the time it takes to access a single piece of data on disk
- All data is accessed in blocks
  - usually 512 bytes

I/O

- Disk is actually part of the I/O subsystem
  - they are of special interest to the OS
- Many other I/O devices
  - printers, monitor, keyboard, etc.
- Most I/O devices are painfully slow
- Need to find ways to hide I/O latency
  - like multiprogramming

Protection and Security

- OS **must** protect itself from users
  - reserved memory only accessible by OS
  - hardware enforced
- OS **may** protect users from one another
  - not all systems do this
  - hardware enforced again

Protection and Security

- Dual -Mode Operation
  - user mode
    - limited set of hardware instr and memory available
    - mode all user programs run in
  - supervisory mode
    - all hardware instr and memory are available
    - mode the OS runs in
- Never let user run in supervisory mode

Interrupts

- Modern OS's are event driven
- Event is signaled by special hardware signal sent to the CPU
- Two types of events
  - interrupts
    - caused by external devices or timers
    - can occur at any moment in time
  - exceptions (traps)
    - caused by software
  - the generic term for both is *Interrupt*
    - sorry for the confusion

Interrupt Philosophy

- One way to handle interrupts is with one standard program
  - big case-switch statement that gets executed on any interrupt
  - inefficient
- Second alternative is to use an interrupt table and special hardware
  - this is the way modern systems operate

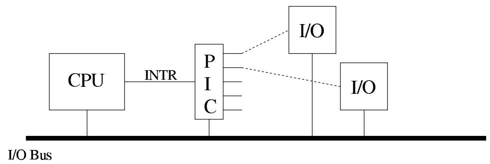
Interrupt Table

- Large array indicating what code to run for a given interrupt
- Each interrupt has a corresponding number associated with it
  - on Intel processors this is from 0 to 255
  - this gives fixed size interrupt table
- Use the interrupt number to index into the array to find out what code to run

Interrupt Hardware

- Programmable Interrupt Controller (PIC)
  - connected to I/O devices via interrupt request lines (IRQ)
    - ideally, one IRQ for each I/O device - doesn't work this way in reality
- PIC connected to CPU by a special signal
- PIC also connected to CPU via I/O bus
- Besides the PIC, interrupts can also be generated by software instructions or errors
  - again, these are usually referred to as exceptions

Interrupt Hardware



The above is a conceptual view of the hardware - not the exact way things are really connected

Hardware Handling of Interrupts

- After each instruction executes, CPU checks to see if interrupt pin has been raised
- If so, the following occurs:
  1) sets the system into kernel mode (if not already there)
  2) determine interrupt number (from PIC or instruction)
  3) read appropriate interrupt table entry
     - special register contains base address of interrupt table
     - each entry in table is fixed size so easy to calculate where to look in memory ($memLoc = idtr + 8 * intNum$)
  4) saves the program counter to the stack (with a couple of others)
  5) saves error code to stack (if it exists)
  6) loads the program counter with the value stored in the interrupt table
     - this starts the CPU executing the interrupt routine

Hardware Handling of Interrupts

- After the interrupt code finishes:
  1) interrupt handler issues an *iret* instruction
  2) reload the program counter from the stack
  3) reload stack pointer with old process
  4) set the system back to user mode
- Steps 3 & 4 may not be executed if the system was running in kernel mode when the interrupt occurred
  – nested exceptions

Software Handling of Interrupts

- The following 6 steps are common to all interrupt handlers:

  1) save IRQ to kernel mode stack
  2) save registers to kernel mode stack
  3) send acknowledgement to PIC
     - this allows PIC to then handle other interrupts on IRQ line
  4) execute the appropriate handler code
  5) restore registers
  6) issue an *iret* instruction

- The steps for exception handlers are almost identical
  - simply remove steps 1 & 3 above

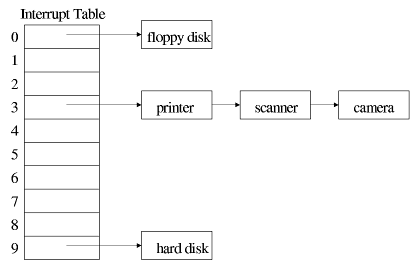
More on Exceptions

- As indicated earlier, exceptions are caused by software
  - divide by zero error
  - page fault
  - *int* instruction
  - etc
- Some of these cause the program to stop executing
- Some of them invoke special operating system code that is invisible to the user
- Some of them invoke operating system code at the user's request
  - system calls

Single IRQ for Multiple Devices

- The number of IRQ lines is usually limited
- May have more I/O devices than IRQ's
- Solution: let multiple devices share an IRQ
- Interrupt table contains a pointer to a linked list of interrupt handlers
  - instead of the address of the interrupt handler
- On interrupt, execute all of the handlers associated with an IRQ
  - requires handlers to recognize if interrupt is really for them

Example



Example

- Assume an interrupt from the scanner arrives at the PIC
  1) PIC raises INTR signal on CPU
  2) CPU reads PIC over I/O bus to determine IRQ
  3) CPU then accesses array in memory to get the first interrupt handler
     - printer
  4) CPU executes printer handler code
  5) printer handler queries printer and notices there is no interrupt pending
     - handler returns immediately
  6) next the scanner handler gets executed
  7) scanner handler queries scanner and notices there is an interrupt pending
     - proceeds to handle the interrupt
     - then returns

System Calls

- An OS's system calls are called the Application Programmers Interface (API)
- System calls are routines run by the OS on behalf of the user
- They are run in supervisory mode
- Allow user to access I/O, create processes, get system information, etc.
- How many system calls an OS has varies
  – Unix: around a hundred
  – Windows: around a thousand

System Startup

- On power up
  - everything in system is in random, unpredictable state
  - special hardware circuit raises RESET pin of CPU
    - sets the program counter to 0xfffffff0
      - this address is mapped to ROM (Read-Only Memory)
- BIOS (Basic Input/Output Stream)
  - set of programs stored in ROM
  - some OS's use only these programs
    - MS DOS
  - many modern systems use these programs to load other system programs
    - Windows, Unix, Linux

BIOS

- General operations performed by BIOS
  1) find and test hardware devices
     - POST (Power-On Self-Test)
  2) initialize hardware devices
     - creates a table of installed devices
  3) find *boot sector*
     - may be on floppy, hard drive, or CD-ROM
  4) load boot sector into memory location 0x00007c00
  5) sets the program counter to 0x00007c00
     - starts executing code at that address

Boot Loader

- Small program stored in boot sector
- Loaded by BIOS at location 0x00007c0
- Configure a basic file system to allow system to read from disk
- Loads kernel into memory
- Also loads another program that will begin kernel initialization

Initial Kernel Program

- Determines amount of RAM in system
  - uses a BIOS function to do this
- Configures hardware devices
  - video card, mouse, disks, etc.
  - BIOS may have done this but usually redo it
    - portability
- Switches the CPU from *real* to *protected* mode
  - real mode: fixed segment sizes, 1 MB memory addressing, and no segment protection
  - protected mode: variable segment sizes, 4 GB memory addressing, and provides segment protection
- Initializes paging (virtual memory)

Final Kernel Initialization

- Sets up page tables and segment descriptor tables
  - these are used by virtual memory and segmentation hardware (more on this later)
- Sets up interrupt vector and enables interrupts
- Initializes all other kernel data structures
- Creates initial process and starts it running
  - *init* in Linux
  - *smss* (Session Manager SubSystem) in NT

OS Philosophy

- Microkernel versus Macrokernel
  - few services versus lots of services
- Hard to agree on what should go in OS
- More functionality, less generality
- More services, more complexity, more bugs
  - longer time to market, too
- But macrokernels dominate the market
  - Unix, Linux, Windows, Mac

OS Philosophy

System Programs

- Application programs included with the OS
- Highly trusted programs
- Perform useful work that most users need
  – listing and deleting files, configuring system
  – ls, rm, Windows Explorer and Control Panel
  – may include compilers and text editors
- Not part of the OS
  – run in user space
- Very useful