What is an Operating System?

- A program that acts as an intermediary between a user of a computer and computer hardware
- Operating system goals
  - Execute user programs
  - Make computer systems convenient to use
  - Use computer hardware in an efficient manner
Computer System Structure

- Computer system can be divided into four components
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating systems
    - Controls and coordinates use of hardware among various applications and users
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers
Four Components of a Computer System

- Users (user 1, user 2, user 3, ..., user n)
- Compiler
- Assembler
- Text editor
- System and application programs
- Operating system
- Computer hardware
- Database system
What Operating Systems Do

- Depends on the point of view
- Users want convenience, ease of use and good performance
  - Don’t care about resource utilization
- But shared computer such as mainframe or minicomputer must keep all users happy
- Users of dedicate systems such as workstations have dedicated resources but frequently use shared resources from servers
- Handheld computers are resource poor, optimized for usability and battery life
- Some computers have little or no user interface, such as embedded computers in devices and automobiles
Operating System Definition

- **OS is a resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use

- **OS is a control program**
  - Controls execution of programs to prevent errors and improper use of the computer
Operating System Definition

- No universally accepted definition
- “Everything a vendor ships when you order an operating system” is a good approximation
  - But varies wildly
- “The one program running at all times on the computer” is the kernel.
- Everything else is either
  - System programs
  - Application programs
bootstrap program is loaded at power-up or reboot

- Typically stored in ROM or EPROM, generally known as firmware
- Initializes all aspects of system
  - Memory, graphic devices and peripheral devices
- Loads operating system kernel and starts execution

 Executing bootstrap program is called booting
Computer System Organization

- One or more CPUs, device controllers connect through common bus providing access to shared memory
- Concurrent execution of CPUs and devices competing for memory cycles
- I/O devices and the 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
- Device controller informs CPU that it has finished its operation by causing an interrupt
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, synchronization, etc.
- Most proprietary OS’s’s were developed with the certain architecture in mind
  - SunOS/Solaris for SPARC architecture
  - Windows for x86 architecture
  - iOS for ARM architecture
Interrupts

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

- 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

- **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 the same as interrupt handling
Exceptions

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

- **System calls**
  - Programming interface to the services provided by OS
    - Example: the system call sequence to copy the contents of one file to another
## Example System Calls

<table>
<thead>
<tr>
<th>Process Management</th>
<th></th>
<th>File System Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>fork</td>
<td>CreateProcess</td>
<td>Create a new process</td>
</tr>
<tr>
<td>waitpid</td>
<td>WaitForSingleObject</td>
<td>Wait for a process to exit</td>
</tr>
<tr>
<td>execve</td>
<td>(none)</td>
<td>CreateProcess = fork + execve</td>
</tr>
<tr>
<td>exit</td>
<td>ExitProcess</td>
<td>Terminate execution</td>
</tr>
<tr>
<td>kill</td>
<td>(none)</td>
<td>Send a signal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File Management</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>CreateFile</td>
<td>Create a file or open an existing file</td>
</tr>
<tr>
<td>close</td>
<td>CloseHandle</td>
<td>Close a file</td>
</tr>
<tr>
<td>read</td>
<td>ReadFile</td>
<td>Read data from a file</td>
</tr>
<tr>
<td>write</td>
<td>WriteFile</td>
<td>Write data to a file</td>
</tr>
<tr>
<td>lseek</td>
<td>SetFilePointer</td>
<td>Move the file pointer</td>
</tr>
<tr>
<td>stat</td>
<td>GetFileAttributesEx</td>
<td>Get various file attributes</td>
</tr>
<tr>
<td>chmod</td>
<td>(none)</td>
<td>Change the file access permission</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File System Management</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mkdir</td>
<td>CreateDirectory</td>
<td>Create a new directory</td>
</tr>
<tr>
<td>rmdir</td>
<td>RemoveDirectory</td>
<td>Remove an empty directory</td>
</tr>
<tr>
<td>link</td>
<td>(none)</td>
<td>Make a link to a file</td>
</tr>
<tr>
<td>unlink</td>
<td>DeleteFile</td>
<td>Destroy an existing file</td>
</tr>
<tr>
<td>mount</td>
<td>(none)</td>
<td>Mount a file system</td>
</tr>
<tr>
<td>umount</td>
<td>(none)</td>
<td>Unmount a file system</td>
</tr>
<tr>
<td>chdir</td>
<td>SetCurrentDirectory</td>
<td>Change the current working directory</td>
</tr>
</tbody>
</table>
System Call Implementation

user application

open ()

user mode

system call interface

kernel mode

i

open ()
Implementation of open ()
system call

return
count = read (fd, buffer, nbytes);
Data Transfer Schemes

- Data transfer modes in I/O
  - Programmed I/O (PIO)
    - CPU is involved in moving data between I/O devices and memory
    - Special I/O instructions VS 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.
I/O Operations via DMA
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
  - Timer is privileged
    - Only the OS can load it