Lecture 2 - Fundamental Concepts

Instructor : Bibhas Ghoshal (bibhas.ghoshal@iiita.ac.in)

Autumn Semester, 2015

Lecture Outline

- Operating System Overview : Re-visit
- Computer System Operation
- Storage Structure
- Storage Hierarchy
- Hardware protection
- Interacting with services provided by the OS
  - System calls - link between application programs and OS
  - System programs - users interact using programs
- Installation, customization etc.
  - booting

References and Illustrations have been used from:

- lecture slides of the book - Operating System Concepts by Silberschatz, Galvin and Gagne, 2005
- Modern Operating System by Andrew S. Tanenbaum
- lecture slides of CSE 30341: Operating Systems (Instructor : Surendar Chandra),

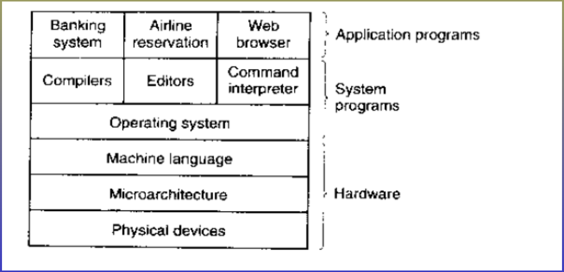
Operating system Overview

- Operating System allows the user to achieve the intended purpose of using the computer system in a fast and efficient manner
- Operating system controls and coordinates use of hardware among various applications and users

Key concerns of Operating System

- Programs
- Resources
- Scheduling
- Protection

Computer System



Operating System Overview

Operating System Structure

- Multiprogramming needed for efficiency
- Timesharing (multitasking)
- Multi-user
- OS should protect the users/processes from each other as well as protect itself from users
- Dual-mode operation allows OS to protect itself and other system components

Hardware Control Aspect of the Operating System

- Acts as an intermediary between user and hardware
- Resource allocator
- Control program
  Operating Systems cannot make hardware go faster. However, OS can make hardware appear faster.

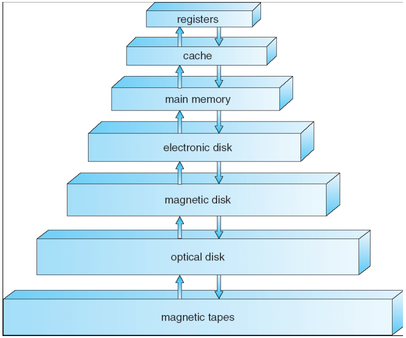
Computer Hardware Review

Hardware issues of concern to OS Designers

Computer System : CPU, Memory and I/O connected through buses to communicate among each other

- CPU : Fetches instruction from memory and executes them
  - CPU has specific instruction set that it executes
  - Registers - General Purpose and Special Purpose (*Program Counter* and *Stack Pointer*)
  - *Program Status Word* : contains condition code bits, CPU priority, mode (kernel/user) etc. PSW plays important role in system calls and I/O
  - Techniques to improve performance : Pipelining, Superscalar CPU etc.
- Memory : Should be extremely fast, abundantly large and cheap
  No technology satisfies it :-(

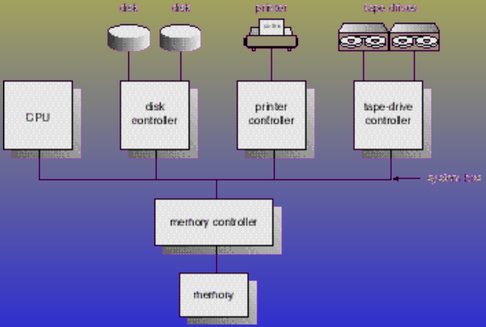
Storage Structure



Caching Principle

- Caching is an important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information "in use" is copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is (cache hit), information used directly from the cache (fast)
  - If not (cache miss), data copied to cache and used there
  - May need to evict some other data (cache replacement)
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policies are important
  - Sometimes bring data before needed (pre-fetch)

Typical Computer Architecture



Computer System Operation

- 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.

Common Functions of Interrupts

- Interrupts transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.
- A trap is a software-generated interrupt caused either by an error or a user request.
- An operating system is interrupt driven.

Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter.
- Determines which type of interrupt has occurred:
    - polling
    - vectored interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt

I/O Structure

- After I/O starts, control returns to user program only upon I/O completion.
  - wait instruction idles the CPU until the next interrupt
  - wait loop (contention for memory access). At most one I/O request is outstanding at a time, no simultaneous I/O processing.
- After I/O starts, control returns to user program without waiting for I/O completion.
  - System call – request to the operating system to allow user to wait for I/O completion.
  - Device-status table contains entry for each I/O device indicating its type, address, and state.
  - Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

Hardware Protection

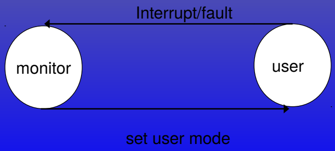
- Dual-Mode Operation
- I/O Protection
- Memory Protection
- CPU Protection

Dual Mode Operation

- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- Provide hardware support to differentiate between at least two modes of operations.
  - User mode – execution done on behalf of a user.
  - Monitor mode (also supervisor mode or system mode) – execution done on behalf of operating system.

Dual Mode Operation contd..

- Mode bit added to computer hardware to indicate the current mode: monitor (0) or user (1).
- When an interrupt or fault occurs hardware switches to monitor mode.



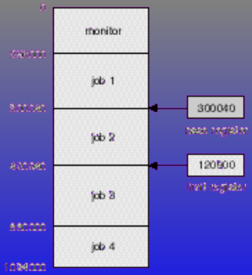
I/O Protection

- All I/O instructions are privileged instructions.
- Must ensure that a user program could never gain control of the computer in monitor mode (I.e., a user program that, as part of its execution, stores a new address in the interrupt vector).

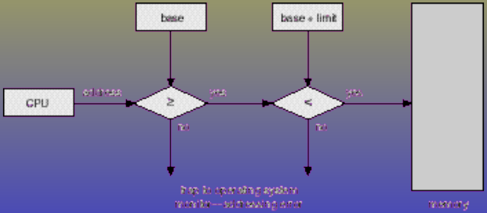
Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
  - Base register : holds the smallest legal physical memory address.
  - Limit register : contains the size of the range
- Memory outside the defined range is protected.

A Base And A Limit Register define A Logical Address Space



Protection Hardware



CPU Protection

- Timer – interrupts computer after specified period to ensure operating system maintains control.
    - Timer is decremented every clock tick.
    - When timer reaches the value 0, an interrupt occurs.
- Timer commonly used to implement time sharing.
- Load-timer is a privileged instruction.

General-System Architecture

- Given the I/O instructions are privileged, how does the user program perform I/O?
- System call – the method used by a process to request action by the operating system.

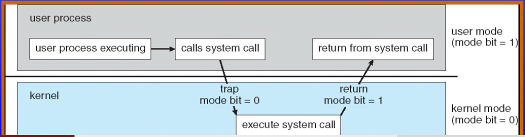
System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (*C, C++*). Maybe in assembly
- Performs basic functions that requires communication with CPU, memory and devices
- All activities related to file handling, memory managemnt and process management are handled by system calls

Examples :
getuid() - get the user ID
fork() - create a child process
exec() - executing a program

System Calls contd...

- System call usually takes the form of a trap to a specific location in the interrupt vector. Control passes through the interrupt vector to a service routine in the OS, and the mode bit is set to monitor mode.
- The OS verifies that the parameters are correct and legal, executes the request, and returns control to the instruction following the system call.

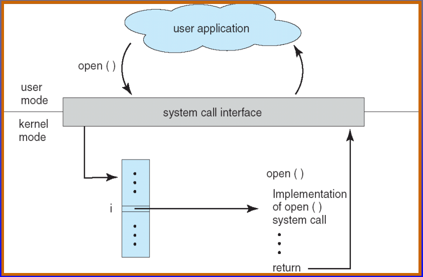


Use of APIs in system calls

- System calls are mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
- Three most common APIs are **Win32 API** for *Windows*, **POSIX API** for *POSIX-based systems* (including virtually all versions of *UNIX, Linux, and Mac OS X*), and **Java API** for the *Java virtual machine (JVM)*
- Why use APIs rather than system calls?
    - Underlying systems calls (error codes) can be more complicated. API gives a uniform, portable interface
    - One need not remeber I/O registers or order of I/O operation

APIs example : BOOL ReadFile c(HANDLE file, LPVOID buffer, DWORD bytes to read, LPDWORD bytes read, LPOVERLAPPED ovl);

API – System Call – OS Relationship



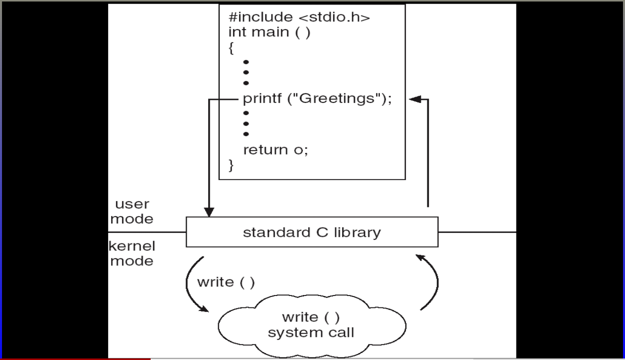
Example of API-System call

- System call sequence to copy the contents of one file to another file (POSIX like C pseudo code) (bold are API system calls)

```
write(1, “Input file”, 11);
read(0, &buffer, 100);
...........
fd = open(buffer,O_RDONLY);
outfd = open(buffer,O_WRONLY|O_CREAT|O_TRUNC,0666);
if(outfd<0)abort(“File creation failed”);
............
closefd;
```

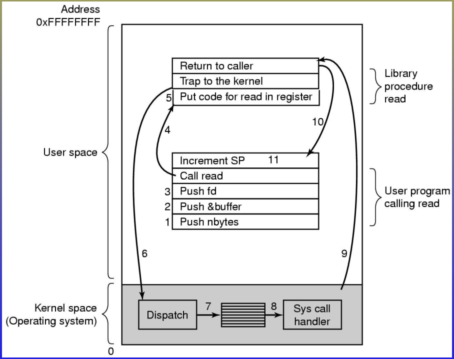
Standard C library example

Some library calls themselves make system calls
ex: C program invoking **printf** library call which calls **write** system call



Steps in Making a System Call, Example : read call

read(fd,buffer,nbytes)



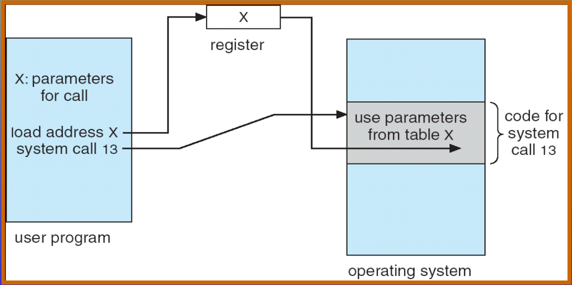
System Call Implementation

- A number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
    Additional info: check /usr/include/sys/syscall.h
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Details of OS interface hidden from programmer by API
  - Managed by run-time support library (set of functions built into libraries included with compiler)

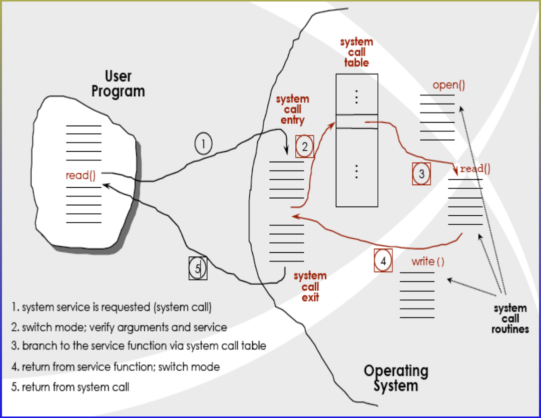
System Call Parameter Passing

- More information is required than simply identity of desired system call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in hardware registers
  - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register (This approach taken by Linux and Solaris)
  - Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system. Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table

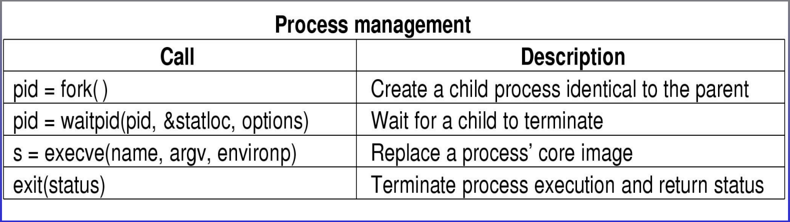


Use of A System Call



1. system service is requested (system call)
2. switch mode; verify arguments and service
3. branch to the service function via system call table
4. return from service function; switch mode
5. return from system call

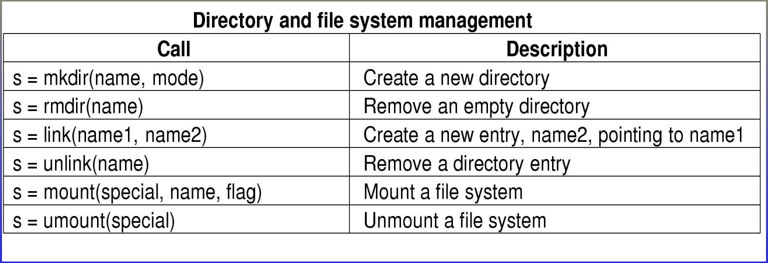
Some System Calls For Process Management



Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

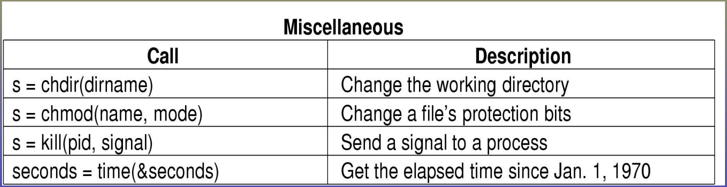
Some System Calls For Directory Management



Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

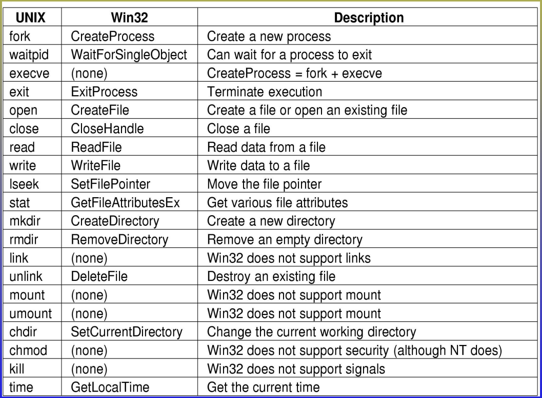
Some System Calls For Miscellaneous Tasks



Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

System Call Example



UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

System programs

- Provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls; others are considerably more complex
  - File management - Create, delete, copy, edit, rename, print, dump, list, and generally manipulate files and directories
  - Programming-language support - Compilers, assemblers, debuggers and interpreters sometimes provided
  - Program loading and execution- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
  - Communications - chat, web browsing, email, remote login, file transfers
  - Status information - system info such as date, time, amount of available memory, disk space, number of users

Interfacing with OS

User Interface

- **Command Line Interface (CLI)** : The command line may itself perform functions or call other system programs to implement functions (e.g. in UNIX, /bin/rm to remove files). Examples are **shell** in *UNIX* and **command.exe** in *Windows*
- **Graphics User Interface (GUI)** : Point and click interface. Examples are *MS windows, MAC OS X Aqua, Unix X* & variants.
- **Batch** : Commands are given using a file/command script to the OS and are executed with little user interaction. Examples are **.bat** files in *DOS*, **shell scripts**

Operating System Generation

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site
- SYSGEN program obtains information concerning the specific configuration of the hardware system
- Booting – starting a computer by loading the kernel
- Bootstrap program – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution

System Boot

Operating system must be made available to hardware so hardware can start it

- Small piece of code – bootstrap loader, locates the kernel, loads it into memory, and starts it
- Sometimes two-step process where boot block at fixed location loads bootstrap loader
- When power initialized on system, execution starts at a fixed memory location - Firmware used to hold initial boot code

Some Fundamental Concepts about OS

- Process
- Memory
- Files

Process : Program in execution

- Address Space - list of address locations which the process can read or write. Address space contains the following:
    - program, program's data and stack
    - registers associated with the program
    - *Program counter* and *Stack Pointer*
    - *PSW*
- When a process is suspended temporarily (such as in time sharing system), it must be restarted in the same state in which it was stopped - information about process needs to be saved
    - information of a suspended process is stored in **process table**

Deadlock : Two interacting process get into a stalemate position through which they cannot get out (Analogous to deadlock in traffic)