Lecture 2 - Fundamental Concepts

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

Autumn Semester, 2020

Lecture Outline

Some Fundamental Concepts about OS :

- Booting
- Process
- Interrupt
- System Calls

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

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
- Protection - Protect data and programs against interference from other users and their programs

Operating System consists of :

- Kernel : OS core that holds the important functionalities (services required by the user programs to access hardware)
- System Programs : utilities to access the services provided by kernel

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**

System Boot

- Operating system must be made available to hardware so that the hardware can start it.
- Bootstrap program (boot loader) : small piece of code stored in ROM that locates the kernel, loads it into memory, and starts it. Sometimes two-step process where boot block at fixed location loads bootstrap loader
- Execution starts at a fixed memory location

Process

Process : Program in execution. OS allocates physical memory to the process (address space)

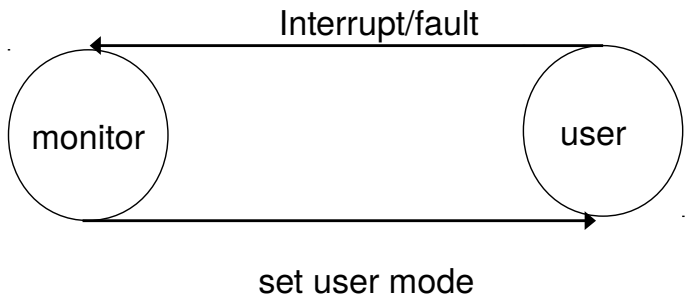
- Address Space (memory image) of a process - list of address locations which the process can read or write. Address space contains the following:
  - program, program's data and stack
  - registers associated with the program
  - *Program counter* and *Stack Pointer*
  - *Program Staus Word (PSW)*
  - *Heap*
  - *Linked libraries*

Virtual Memory of a Process

- Every process has a *Virtual Address* space - range of addresses which is assigned to a process by a compiler
- The range of addresses in the *Virtual Address* space depends on the architecture
- During execution of the process, CPU generates requests for these *Virtual Addresses*
- The *Virtual Addresses* requested by the CPU are converted to actual *Physical Addresses* (the actual location of the process in the physical memory) by the *Memory Management* unit of the OS
- OS code is mapped to *Virtual Address* space of every process - some virtual address in the address space of every process points to the os code

Dual Mode Operation of Processes

- Problem - Sharing the process memory requires operating system to ensure that an incorrect program cannot tamper the OS code or cause other programs to execute incorrectly
- Solution - Restrict the use of os code by using dual mode of operation
- Process has two modes of operation : User mode and Kernel (priviledged / system / monitor) mode
- Hardware support (CPU bit) to differentiate between at least two modes of operations - monitor (0) or user (1)
- When an interrupt or fault occurs hardware switches to monitor mode



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
- Interrupts can be generated either by external events or can be software generated
- 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 CPU changes to **kernel mode** on occurence of interrupt
- Determines which type of interrupt has occurred:
  - polling
  - vectored interrupt system
- Separate segments of code (Interrupt Handler) determine what action should be taken for each type of interrupt. The pointers to each of these segments is maintaned in a table called the Interrupt Descriptor Table
- The operating system preserves the state of the CPU by storing registers and the program counter - Context saving

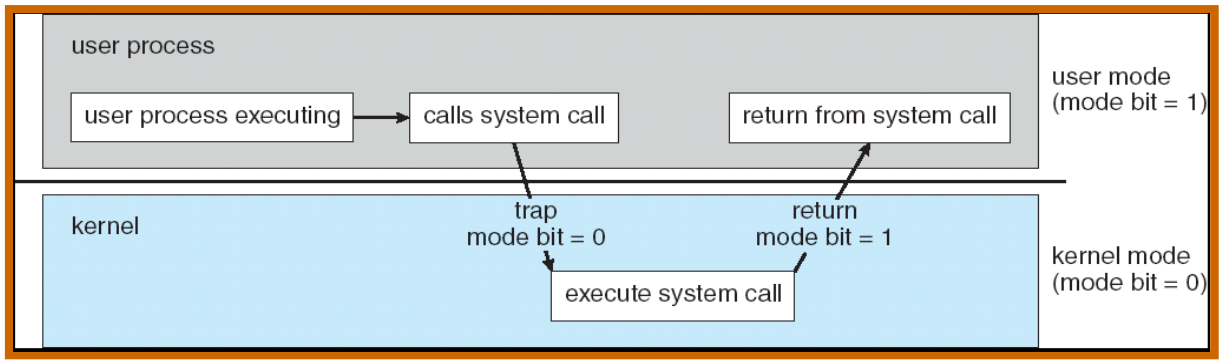
System Calls

- 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

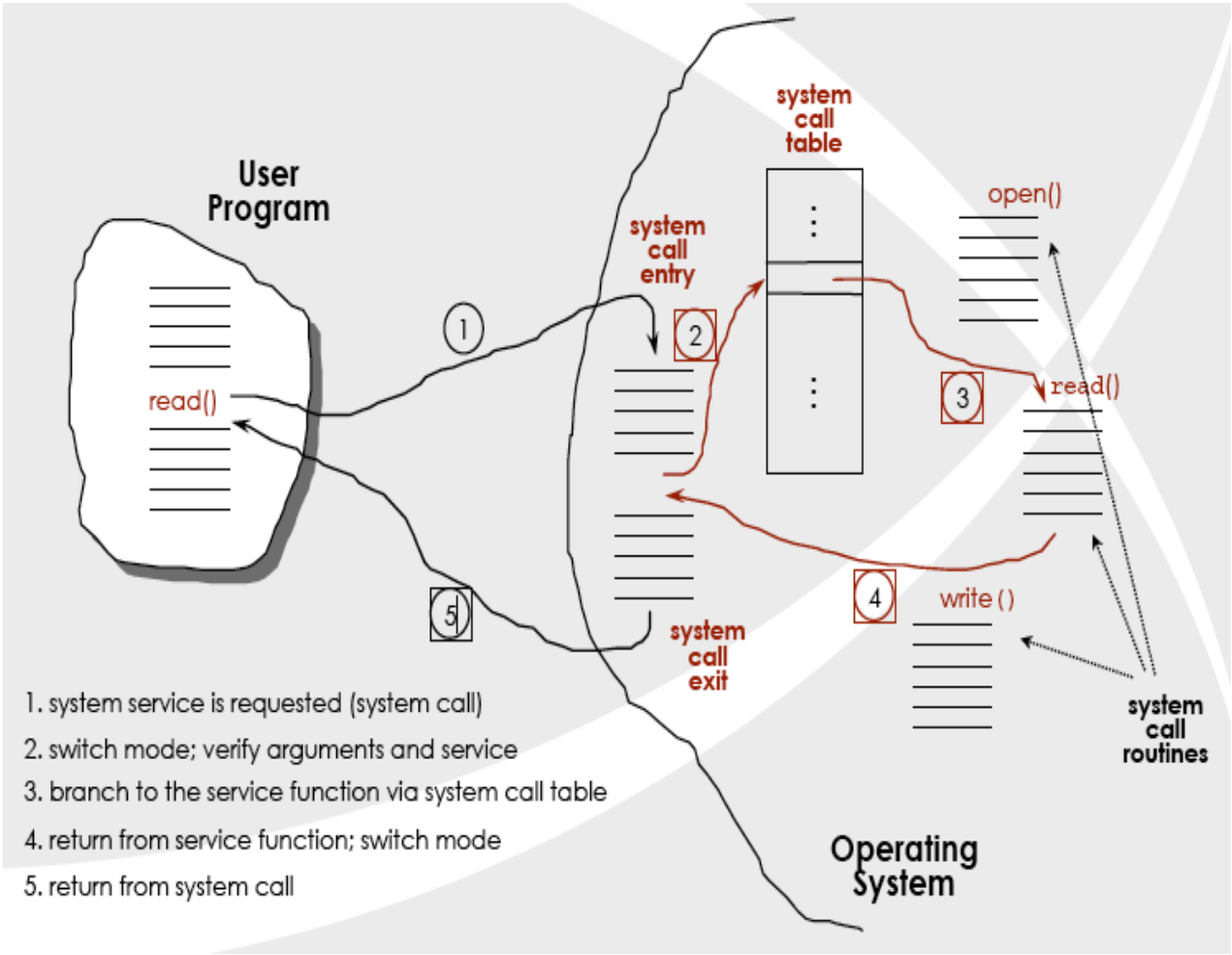
Examples :
getuid() - get the user ID
fork() - create a child process
exec() - executing a program

System Calls

- 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 a System Call

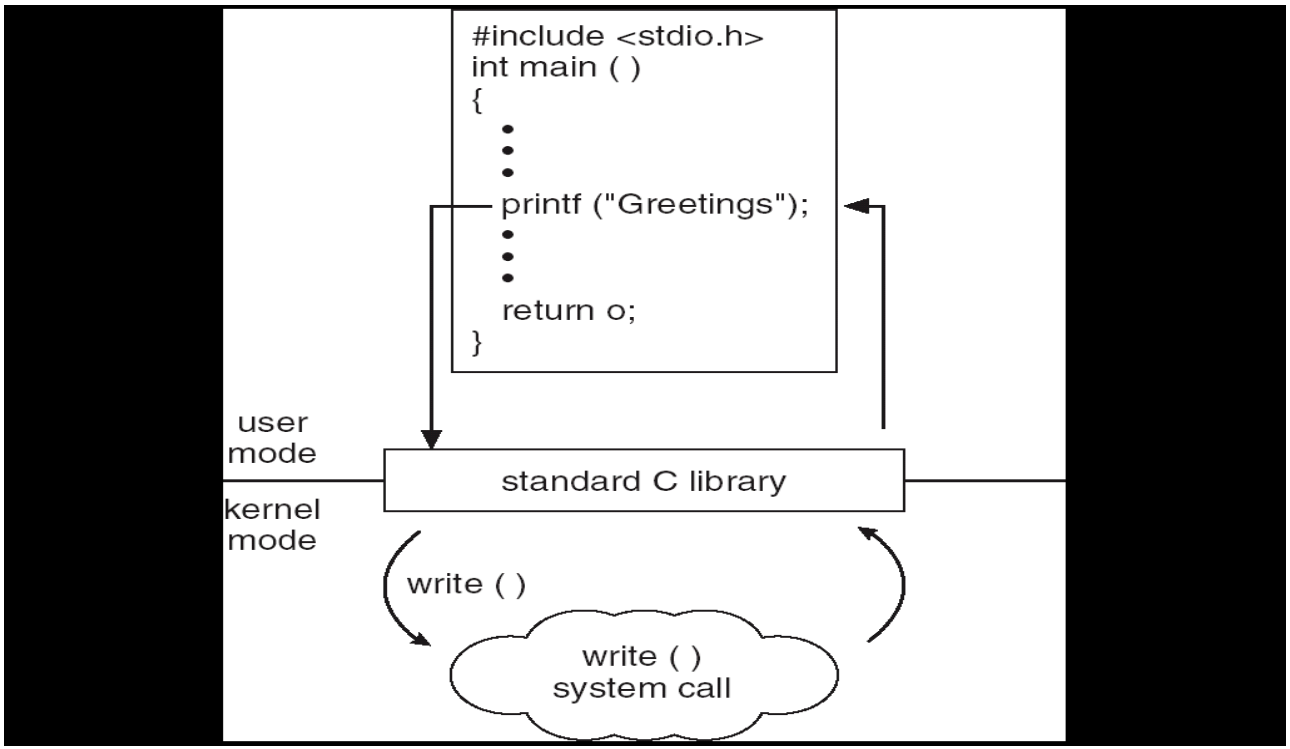


Use of APIs in system calls

- 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

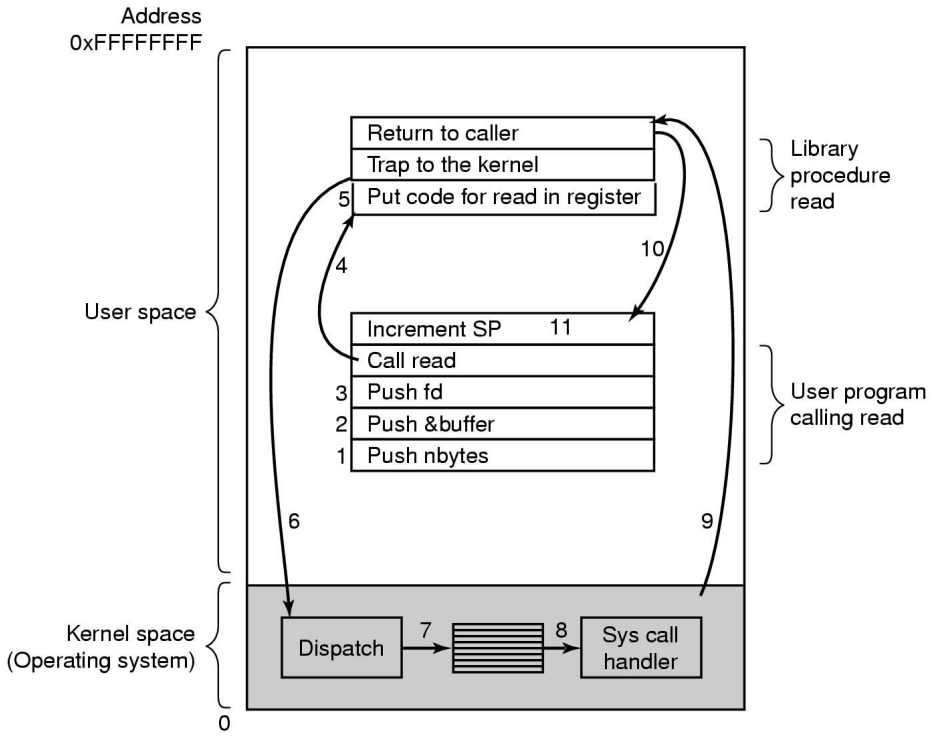
Standard C library example

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)



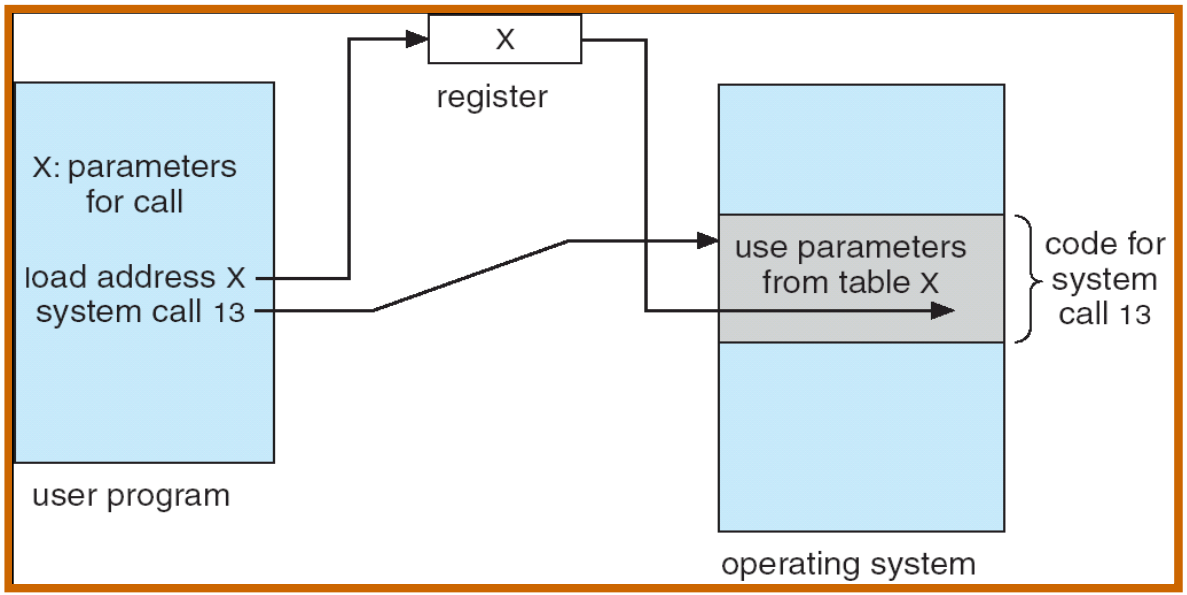
System Call Implementation

- 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



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