Operating System Concepts

Objectives of the course

- Introduce the underlining principles of an uni-processor operating system.
- Explores the concept of multiprogramming, virtual memory and resource management.
- Case study of some well-known operating systems.

Why Learn about Operating Systems?

Tangible reasons

- Build or modify a real operating system
- Administer and use system well
- Tune application performance

Intangibles reasons

- Intrinsic curiosity
  - Understand how much of your computer system works
- Gain/apply knowledge in other areas of Computer Science
  - Computer architecture and devices
  - Synchronization in programming languages
  - Data structures and algorithms
  - Performance analysis
- Challenge of designing large, complex systems

Books and References

Text Books:

1. Silberschatz, A. and P. B. Galvin, *Operating System Concepts*, Sixth Edition, John-Wiley. (not Java edition)
2. Tanenbaum, A. S., *Modern Operating Systems*, Second Edition, PHI.

References :

1. Research and Technical Papers
2. Stallings, W., *Operating Systems*, Fourth Edition, Pearson.
3. Deitel H. M., *Operating Systems*, Second Edition, Addition Wesley.
4. Tanenbaum, A. S. and Woodhull, A. S., *Operating Systems Design and Implementation*, Second Edition, PHI.

Lab References:

Linux Manual, Linux Programming Manual and C book.

Evaluation system

Total mark (75) = Assessment (15) + Semester (60)

Assessment marks is distributed as follows:

Assignments (5) + Internal Examination1 (5) + Internal Examination2 (5) = 15

- Assignments are basically Lab works and Case Studies, each assignment will have a due date for submission and demonstration, it is mandatory to submit their report with in the due date.
- Home works will be assigned in class only for your practice, not for submission, but sometime it may assigned as an assignment.

Introduction

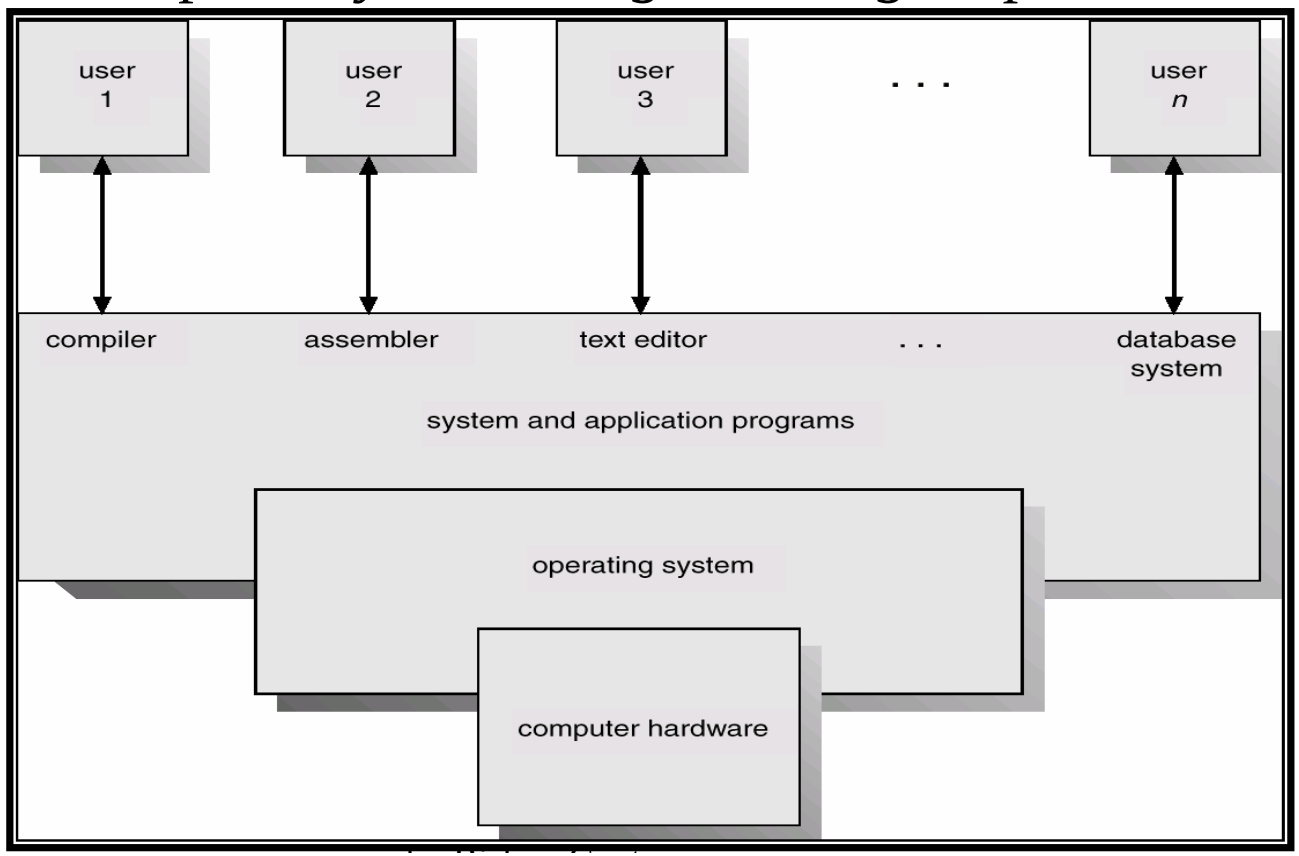
- Computer system
- What is an operating system?
- Variations of OS
- Computer System Operation
- Memory Hierarchy
- Hardware Protection
- Common OS Concepts
- System Calls

Reading:

Chapter 1 of Tanenbaum or chapter 1, 2 and 3 of silberschatz

Computer system

Computer system provide a capability for gathering data, performing computations, storing information, communicating with other computer system and generating output.



Abstract view of computer system

Computer system

1. **Hardware** – provides basic computing resources (CPU, memory, I/O devices).

2. **Operating system** – controls and coordinates the use of the hardware among the various application programs for the various users.

3. **Applications programs** – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).

4. **Users** (people, machines, other computers).

What is an operating system?

An operating system (OS) is a collection of system programs that together control the operation of a computer system.

Operating system goals:

- Execute user programs and make solving user problems easier.
- Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.

Provides an environment within which other programs can do useful work.

Two Functions of OS

- OS as an Extended Machine
- OS as a Resource Manager

OS as an Extended Machine

OS creates higher-level abstraction for programmer

Example: (Floppy disk I/O operation)

-disks contains a collection of named files

-each file must be open for READ/WRITE

-after READ/WRITE complete close that file

-no any detail to deal

OS shields the programmer from the disk hardware and presents a simple file oriented interface.

OS function is to present the user with the equivalent of an extended machine or virtual machine that is easier to program than the underlying hardware.

Challenges:

What are the right abstractions? How to achieve this?

OS as a Resource Manager

-What happens if three programs try to print their output on the same printer at the same time?

-What happens if two network users try to update a shared document at same time?

OS primary function is to manage all pieces of a complex system.

Advantages:

- Virtualizes resource so multiple users/applications can share
- Protect applications from one another
- Provide efficient and fair access to resources

Challenges:

- What mechanisms? What policies?

OS evolution

Computer Generation, Component and OS Types

1st (1945-55)	Vacuum Tubes	User Driven
2nd (1955-65)	Transistor	Batch
3rd (1965-80)	IC	Multiprogramming
4th (1980-present)	PC	Client Server/Distributed

Batch System

During the 2nd generation, operator hired to run computer, the user prepared a job – which consisted of the program, the data, and some control information about the nature of the jobs – and submitted it to the computer operator. The operator perform the function of an OS.

Batch: Group of jobs submitted to machine together

-Operator collects jobs; orders efficiently; runs one at a time

Advantages:

- Amortize setup costs over many jobs
- Operator more skilled at loading tapes
- Keep machine busy while programmer thinks

Problem:

- User must wait for results until batch collected and submitted

(If bug receive, memory and register dump; submit job again!)

Spooling

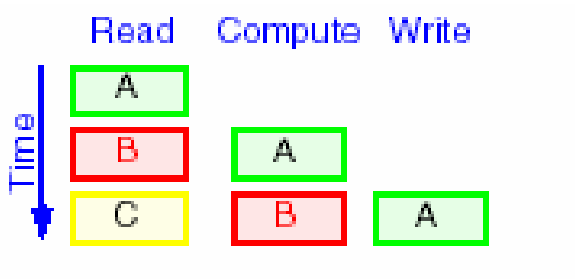
(During 3rd generation)
Simultaneous **P**eripheral **O**peration **O**n **L**ine

Problem: Mechanical I/O devices much slower than CPU.

- Read 17 cards/sec vs. execute 1000s instructions/sec

Spooling:

Overlap I/O (e.g., card reading and line printing) with execution



Advantage: Improves throughput and utilization
Disadvantage: Single job must wait during I/O for data
New OS functionality

- Buffering, DMA, interrupt handling

Multiprogramming

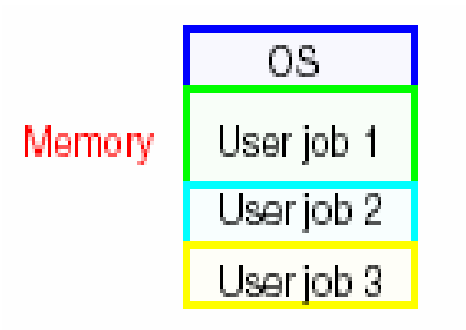
(During 3rd generation)

Problem: -wait time still too long

-long jobs delay everything

Spooling provides pool of ready jobs.

Multiple jobs in memory at the same time. Each memory space protected from each other. OS picks one, execute it for a while, stops (e.g. when programs reads for input or randomly), picks other to run.



Advantage: Improves throughput and utilization

Disadvantage: Still not interactive

New OS functionality

- Job scheduling policies
- Memory management and protection (virtual memory)

Timesharing

(During 3rd generation)

Problem:

-increasing number of users who want to interact with programs while they are running.

-humans' time is expensive don't want to wait.

The CPU executes multiple jobs by switching among them, but the switches occurs so frequently that the user can interact with each program while it is running.

Goal: Improve user's response time

Advantage:

• Users easily submit jobs and get immediate feedback

New OS functionality

-More complex job scheduling, memory management

-Concurrency control and synchronization

Personal Computers

Dedicated machine per user

CPU utilization is not a prime concern

Hence, some of the design decision made for advanced system may not be appropriate for these smaller system. But other design decisions, such as those for security, are appropriate because PCs can now be connected to other computer and users through the network and the web.

There are the different types of OS for PC such as Windows, MacOS, UNIX, Linux.

Conclusion: OS Functionality changes with hardware and users

Real-Time Systems

Time is a key parameter

Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.

Real-Time systems may be either *hard* or *soft* real-time.

Hard Real-Time:

-well defined fixed time constraints, processing must be done within the defined constraints, or the system will fail.

Soft Real-Time:

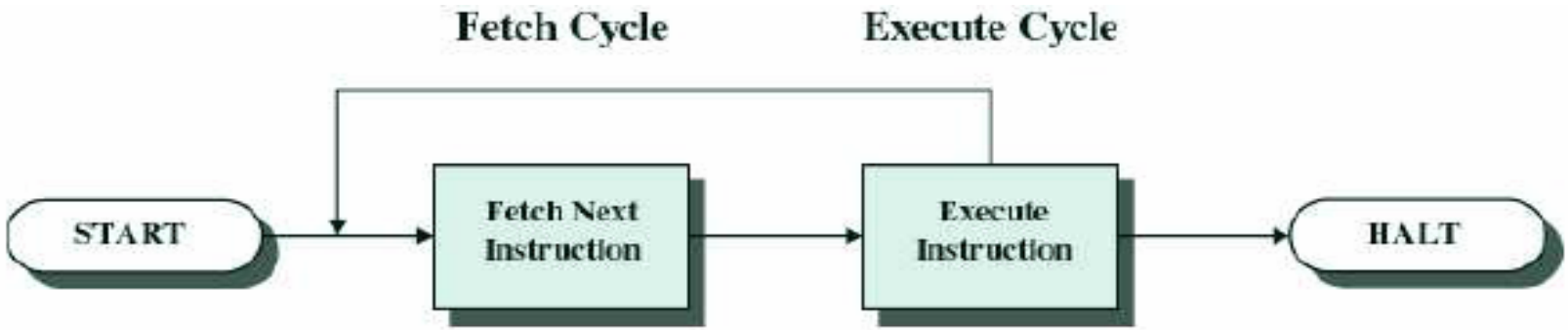
-less stringent timing constraints, and do not support the dead line scheduling.

Handheld Systems

- Personal Digital Assistants (PDAs)
- Cellular telephones
- Issues:
  - Limited memory
  - Slow processors
  - Small display screens.

Computer System Operation

CPU fetches the instructions from memory and executes them. The basic cycle of CPU is to fetch the first instruction from memory, decodes it to determine its type and operands, execute it, and then fetch, decode, and execute subsequent instructions.



Registers are used to hold variables and temporary results.

Program counter – contains the memory address of next instruction to be fetched.

Instruction register – holds the actual instruction.

The instruction is decoded to determine what action to be performed. The action is specified by instruction's opcode bits.

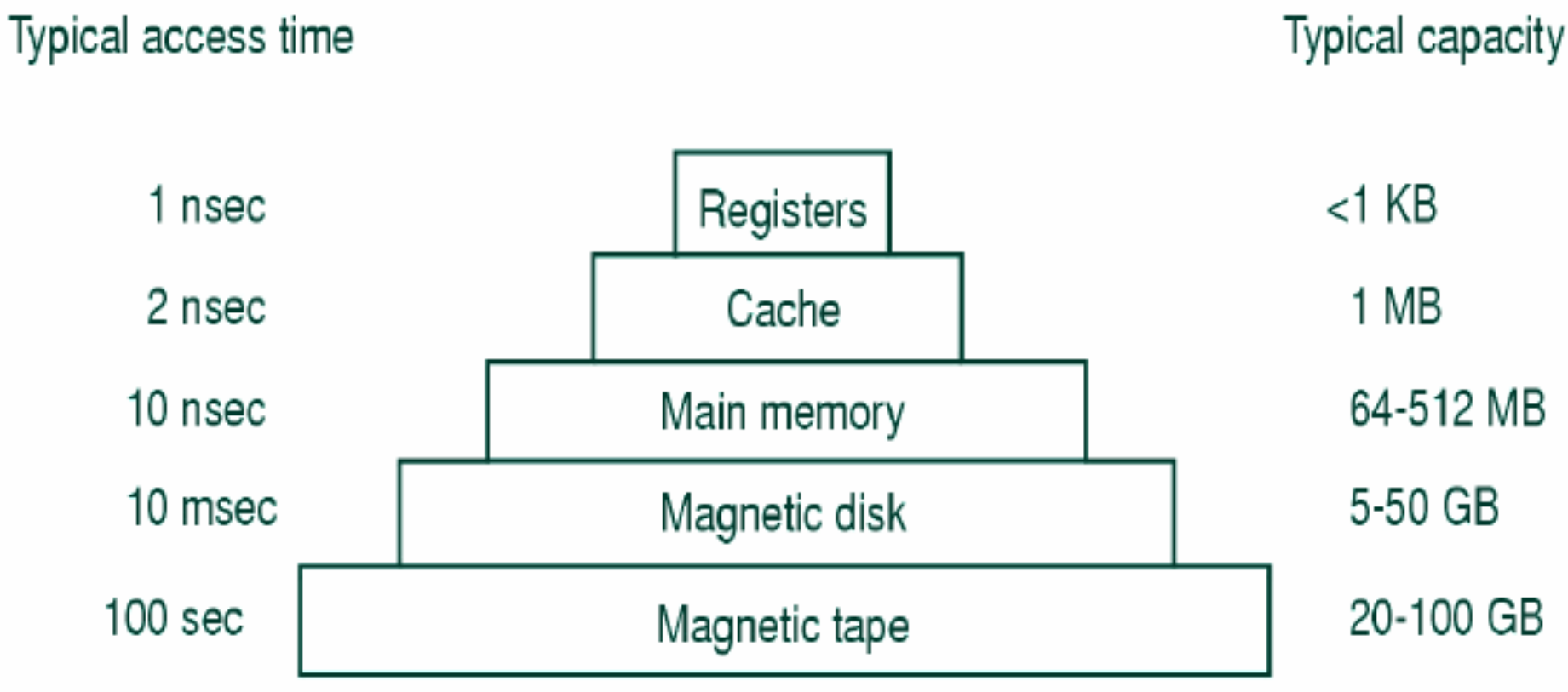
Computer System Operation

The PSW (Program Status Word) - contains the condition code bits, which are set by comparison instruction, the CPU priority, the modes (user or kernel), and various other control bits.

In timesharing system, OS stops a running program to start another one. When stopping a program it save all registers so they can be restored when the programs runs later.

Device controller informs CPU that it has finished its operation by causing an *interrupt* (Signal generated by hardware such as I/O), a *trap* is a software generated interrupt caused either by an error (e.g., division by 0, trying to excess the non existent I/O etc.) or by user (such as by calling System Calls). The CPU responds to the trap and interrupt by saving the current value of program counter.

Memory Structure



A typical memory hierarchy (based on year 2000)

Hardware Protection

Situation: *Multiprogramming put several programs in memory at the same time. When one program contain erroneous data it might modify the program or data of another program.(MS-DOS and MacOS).*

OS must insure that an incorrect program can not cause other programs to execute incorrectly, and must terminate the such erroneous program.

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

Dual-Mode Operation

- Provide hardware support to differentiate between at least two modes of operations.

  1. User mode . execution done on behalf of a user.

  User programs runs in user mode, which permits only the subset of instructions to be executed and subset of features to be accessed.

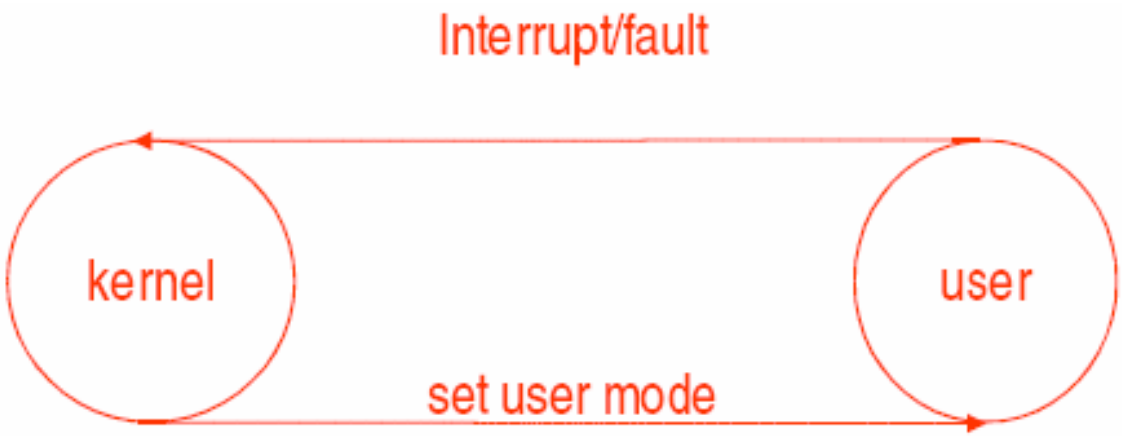
  2. Kernel mode (or supervisor mode) . execution done on behalf of operating system.

  OS runs in kernel mode, giving it access to the complete hardware.

- Mode bit is added to the hardware of the computer to indicate the current mode: kernel(0) and user(1).

Dual-Mode Operation

When an trap or interrupt occurs hardware switches from user mode to the kernel mode. Thus, whenever the OS gains control of computer, it is in kernel mode. The system always switches to user mode before passing control to the user program.



Privileged instructions can be issued only in kernel mode.

Advantages:

protect OS from errant users and errant user from one another.

I/O Protection

Situation: *A user program may disrupt the normal operation of the system by issuing illegal I/O operation, by accessing memory locations within the OS itself.*

- All I/O instructions must be privileged instructions. Thus the user can not issue the I/O instruction directly.
- Must ensure that a user program could never gain control of the computer in kernel mode (i.e., a user program that, as part of its execution, stores a new address in the interrupt vector).

Memory Protection

Problem:

-How to separate each program's memory space ?

-How to handle relocation ?

- 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.

CPU Protection

Situation: *To prevent a user program from getting stuck in an infinite loop or not calling system services, and never returning control to the OS.*

- **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.

  If the timer interrupts, control transfers automatically to the OS, which may treat the interrupt as a fatal error or may give the program more time.

- Timer commonly used to implement time sharing.
- Time also used to compute the current time.
- Load-timer is a privileged instruction.

Common OS Concepts

- Process Management
- Memory Management
- File system Management
- Device Management

Process Management

A process is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.

- The operating system is responsible for the following activities in connection with process management.
  - Process creation and deletion.
  - Process suspension and resumption.
  - Provision of mechanisms for:
    - process synchronization
    - process communication
    - deadlock handling

Memory Management

Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.

- The operating system is responsible for the following activities in connections with memory management:
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes to load when memory space becomes available.
  - Allocate and deallocate memory space as needed.

File Management

A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.

- The operating system is responsible for the following activities in connections with file management:
  - File creation and deletion.
  - Directory creation and deletion.
  - Support of primitives for manipulating files and directories.
  - Mapping files onto secondary storage.
  - File backup on stable (nonvolatile) storage media.

I/O Management

All computers have physical devices for acquiring input and producing output.

The I/O system consists of:

- A memory management component that includes buffering, caching, and spooling.
- A general device-driver interface.
- Drivers for specific hardware devices.
- Disk management.

System calls

System calls provide the interface between a process and the operating system.

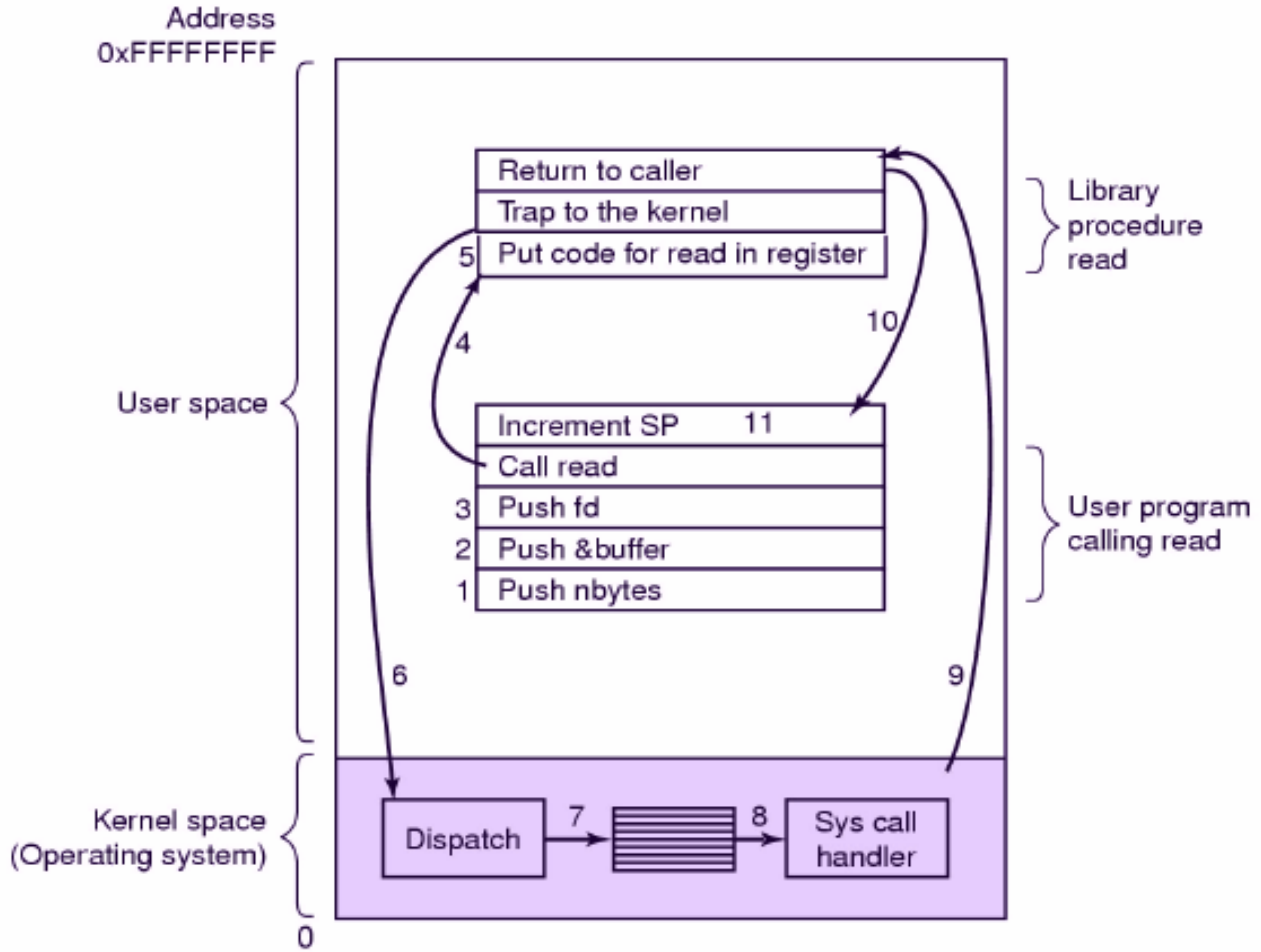
- Generally available as assembly-language instructions.
- Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++)

- System calls performed in a series of steps. Most of the system calls are invoked as the following example system call: read

count = read(fd, buffer, nbytes)

- Push parameters into the stack (1-3)
- Calls library procedure (4)
- Pass parameters in registers.(5)
- Switch from user mode to kernel mode and start to execute.(6)
- Examines the system call number and then dispatches to the correct system call handler via a table of pointer.(7)
- Runs system call handlers (8).
- Once the system call handler completed its work, control return to the library procedure.(9)
- This procedure then return to the user program in the usual way. (10)
- Increments SP before call to finish the job.

System calls



Steps in making the system call
read (fd, buffer, nbytes)

Types of System calls

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

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Types of System calls

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

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

Some Win32 API calls

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

Home Works

HW #1:

1. Textbook: (Ch. 1) 1, 2, 3, 8, 9 &14.
2. What does the CPU do when there are no programs to run?
3. What must user programs be prohibited from writing to the memory locations containing the interrupt vector?
4. Classify the following applications as batch-oriented or interactive.
    a) Word processing
    b) Generating monthly bank statements
    c) Computing pi to a million decimal places
5. What is a purpose of system calls?