**OPERATING SYSTEMS**

**Lecture Notes**

**Prepared By**

**Dr.S.KOTHER MOHIDEEN ,M.Tech .,Ph.D**

**Professor**

**Department of Information System**

**AMBO Univeristy,Woliso Campus**

**Woliso**

**CHAPTER – 1**

OPERATING SYSTEM FUNCTIONS

##### Process Management

* A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.
* Process needs resources to accomplish its task
* CPU, memory, I/O, files
* Initialization data
* Process termination requires reclaim of any reusable resources
* Single-threaded process has one **program counter** specifying location of next instruction to execute
* Process executes instructions sequentially, one at a time, until completion
* Multi-threaded process has one program counter per thread
* Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
* Concurrency by multiplexing the CPUs among the processes / threads

##### Process Management Activities

* The operating system is responsible for the following activities in connection with process management:
* Creating and deleting both user and system processes
* Suspending and resuming processes
* Providing mechanisms for process synchronization
* Providing mechanisms for process communication
* Providing mechanisms for deadlock handling

##### Memory Management

* All data in memory before and after processing
* All instructions in memory in order to execute
* Memory management determines what is in memory when
* Optimizing CPU utilization and computer response to users
* Memory management activities
* Keeping track of which parts of memory are currently being used and by whom
* Deciding which processes (or parts thereof) and data to move into and out of memory
* Allocating and deallocating memory space as needed

##### Storage Management

* OS provides uniform, logical view of information storage
* Abstracts physical properties to logical storage unit - **file**
* Each medium is controlled by device (i.e., disk drive, tape drive)
* Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
* File-System management
* Files usually organized into directories
* Access control on most systems to determine who can access what

##### OS activities include

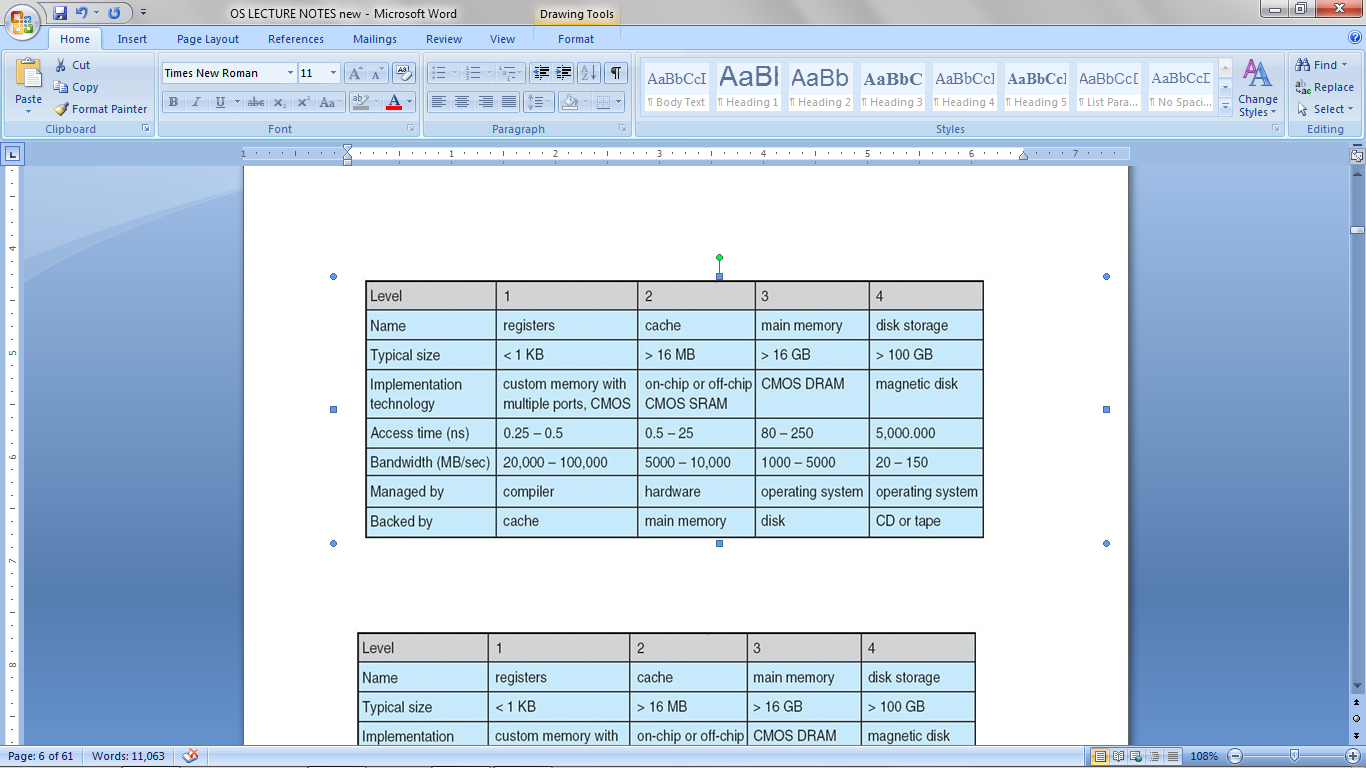
* Creating and deleting files and directories
* Primitives to manipulate files and dirs
* Mapping files onto secondary storage
* Backup files onto stable (non-volatile) storage media

##### Mass-Storage Management

##### Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time

* Proper management is of central importance
* Entire speed of computer operation hinges on disk subsystem and its algorithms
* **MASS STORAGE activities**
* Free-space management
* Storage allocation
* Disk scheduling
* Some storage need not be fast
* Tertiary storage includes optical storage, magnetic tape
* Still must be managed
* Varies between WORM (write-once, read-many-times) and RW (read-write)

##### Performance of Various Levels of Storage



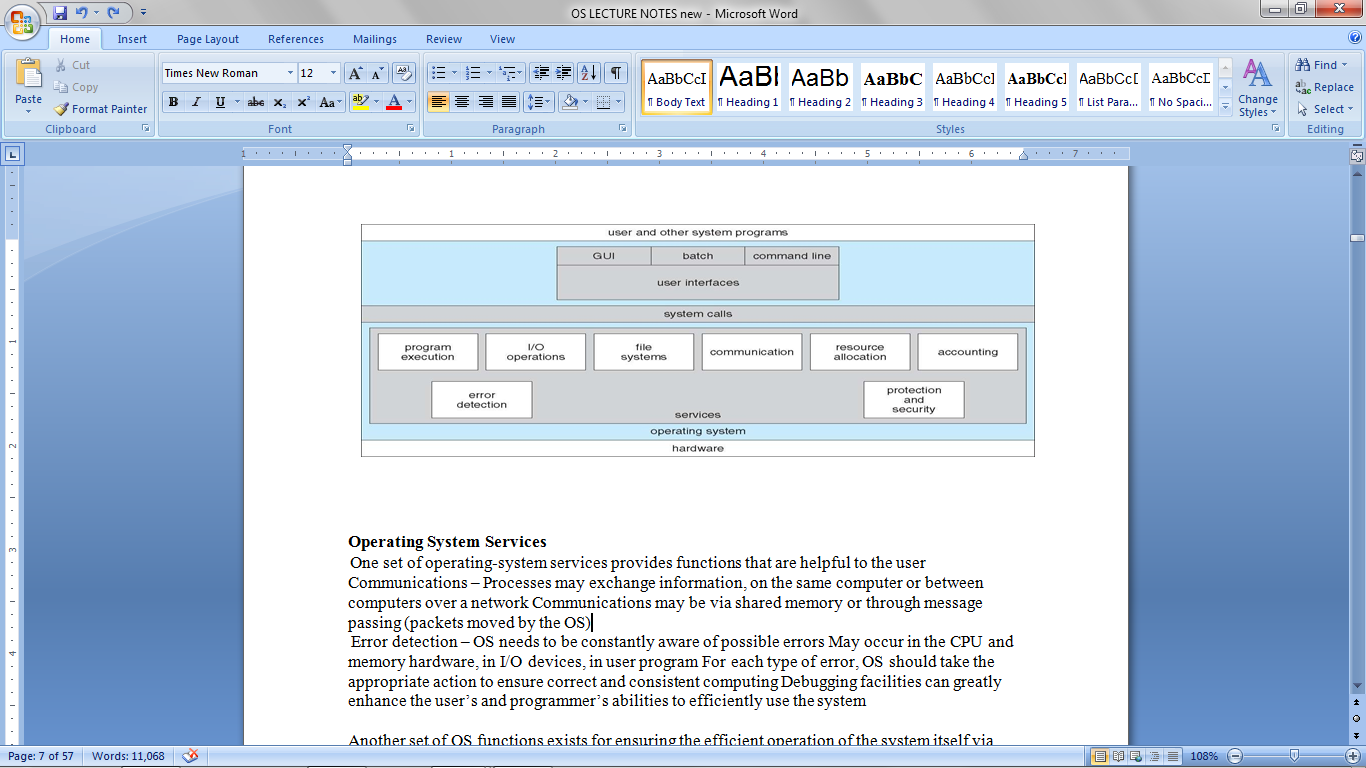
##### Open-Source Operating Systems

* Operating systems made available in source-code format rather than just binary closed-source
* Counter to the copy protection and Digital Rights Management (DRM) movement
* Started by Free Software Foundation (FSF), which has “copyleft” GNU Public License (GPL)
* Examples include GNU/Linux, BSD UNIX (including core of Mac OS X), and Sun Solaris

##### Operating System Services

* One set of operating-system services provides functions that are helpful to the user:
* User interface - Almost all operating systems have a user interface (UI)
* Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
* Program execution - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
* I/O operations - A running program may require I/O, which may involve a file or an I/O device
* File-system manipulation - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

**A View of Operating System Services**

****

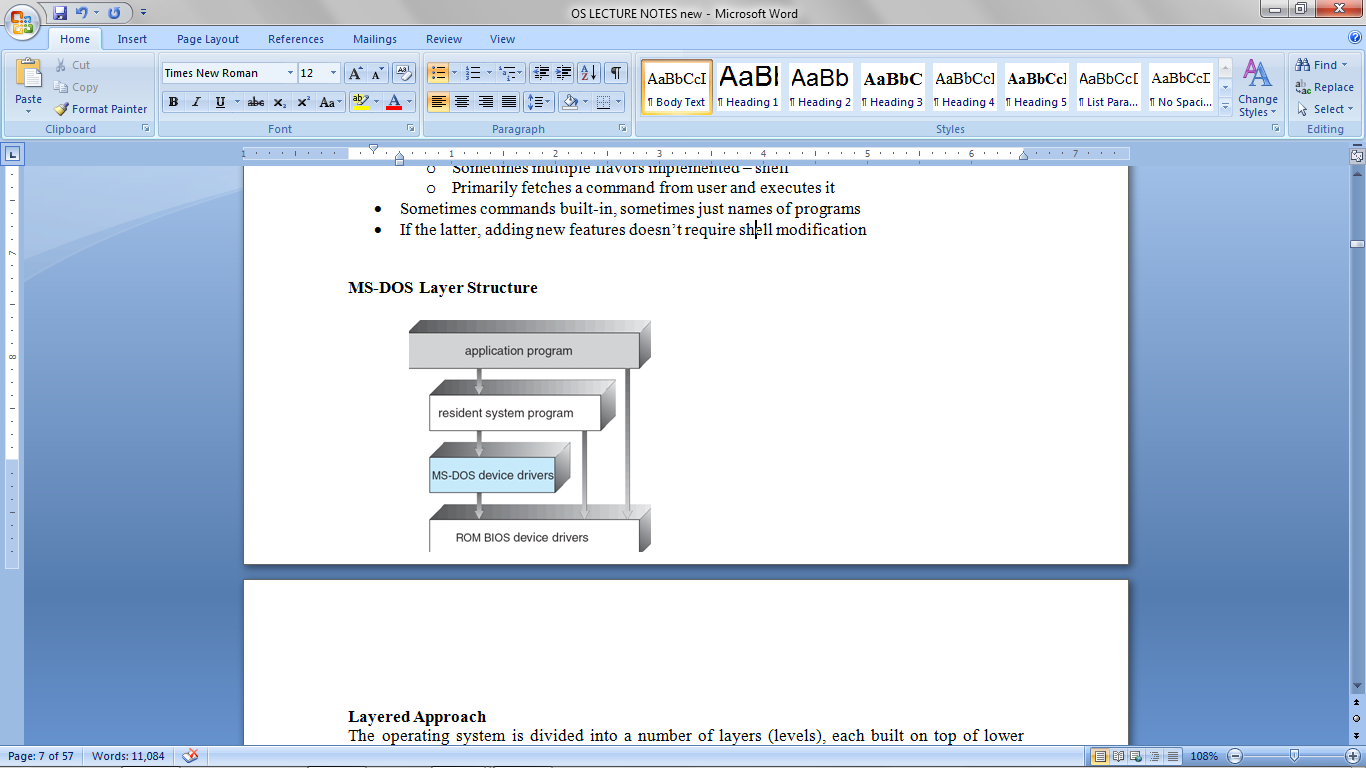
**Operating System Services**

* One set of operating-system services provides functions that are helpful to the user Communications – Processes may exchange information, on the same computer or between computers over a network Communications may be via shared memory or through message passing (packets moved by the OS)
* Error detection – OS needs to be constantly aware of possible errors May occur in the CPU and memory hardware, in I/O devices, in user program For each type of error, OS should take the appropriate action to ensure correct and consistent computing Debugging facilities can greatly enhance the user’s and programmer’s abilities to efficiently use the system
* Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
* **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
* Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code
* **Accounting -** To keep track of which users use how much and what kinds of computer resources
* **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
* **Protection** involves ensuring that all access to system resources is controlled
* **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
* If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

##### User Operating System Interface - CLI

* Command Line Interface (CLI) or command interpreter allows direct command entry
  + Sometimes implemented in kernel, sometimes by systems program
  + Sometimes multiple flavors implemented – shell
  + Primarily fetches a command from user and executes it
* Sometimes commands built-in, sometimes just names of programs
* If the latter, adding new features doesn’t require shell modification

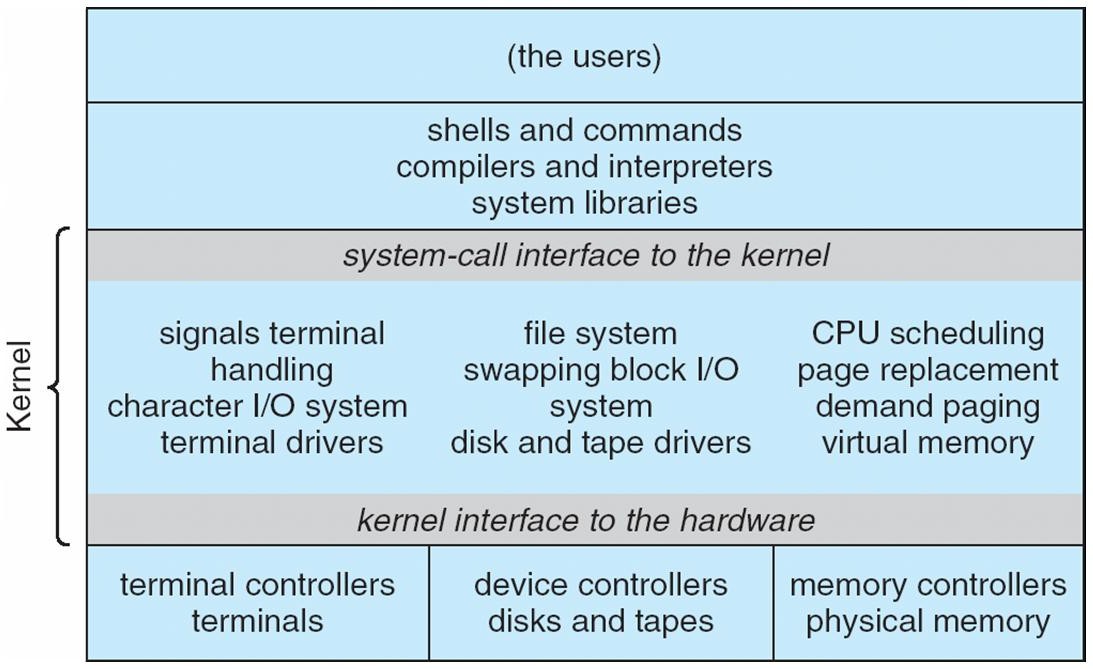
##### MS-DOS Layer Structure



**Layered Approach**

* The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
* With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

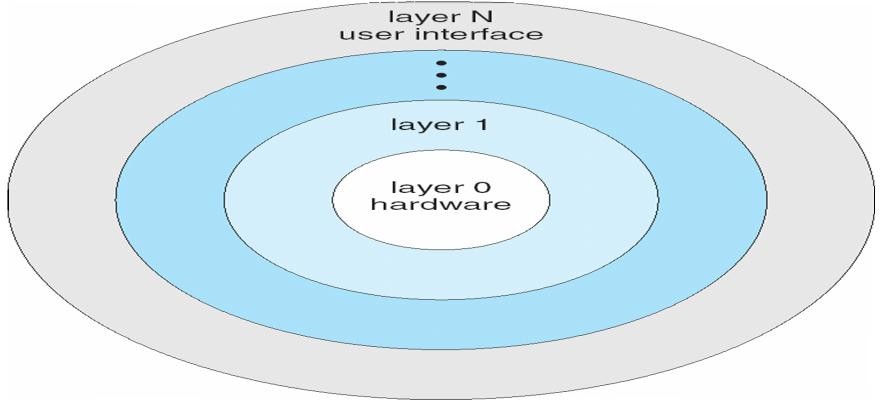
##### Traditional UNIX System Structure



**UNIX**

* UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
* Systems programs
* The kernel
  + Consists of everything below the system-call interface and above the physical hardware
  + Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

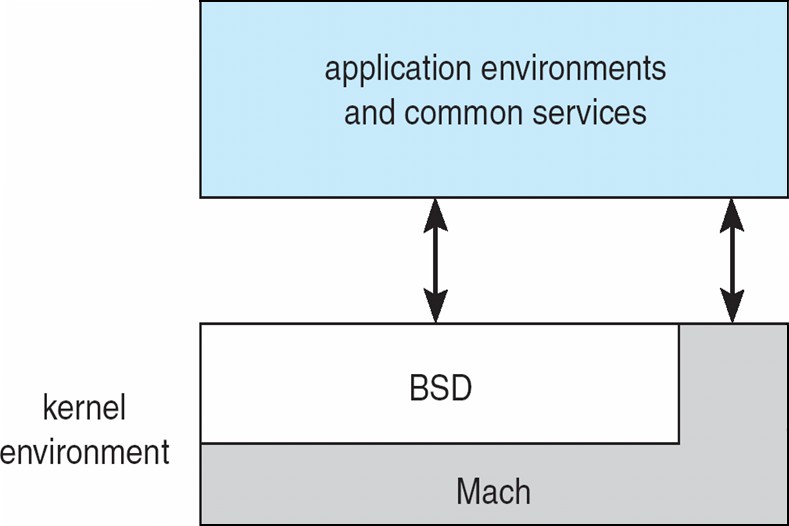
##### Layered Operating System



**Micro kernel System Structure**

* Moves as much from the kernel into “*user*” space
* Communication takes place between user modules using message passing
* Benefits:
* Easier to extend a microkernel
* Easier to port the operating system to new architectures
* More reliable (less code is running in kernel mode)
* More secure
* Detriments:
* Performance overhead of user space to kernel space communication

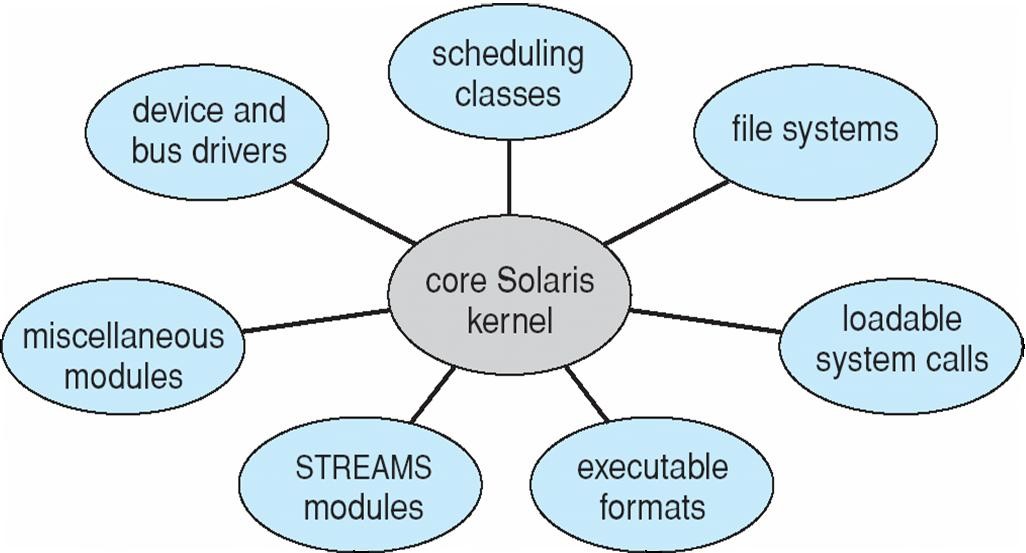
##### Mac OS X Structure



**Modules**

* Most modern operating systems implement kernel modules
* Uses object-oriented approach
* Each core component is separate
* Each talks to the others over known interfaces
* Each is loadable as needed within the kernel
* Overall, similar to layers but with more flexible

##### Solaris Modular Approach

****

**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

**CHAPTER -2**

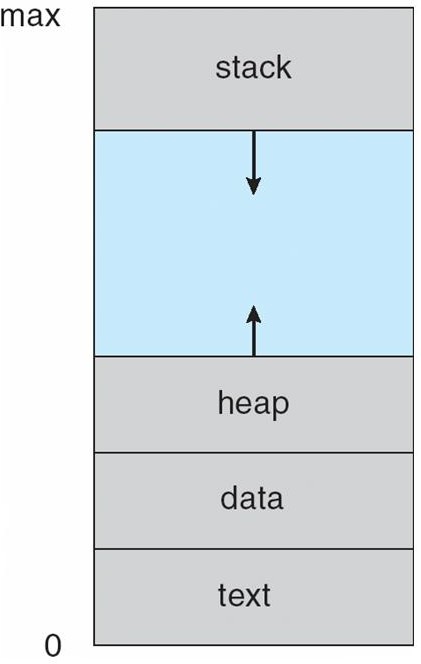
**PROCESS MANAGEMENT**

**Process Concept**

* An operating system executes a variety of programs:
* Batch system – jobs
* Time-shared systems – user programs or tasks
* Textbook uses the terms *job* and *process* almost interchangeably

Process – a program in execution; process execution must progress in sequential fashion

A process includes:

* program counter
* stack
* data section

##### Process in Memory

**Process State**

As a process executes, it changes *stat*e

* **new**: The process is being created
* **running**: Instructions are being executed
* **waiting**: The process is waiting for some event to occur
* **ready**: The process is waiting to be assigned to a processor
* **terminated**: The process has finished execution

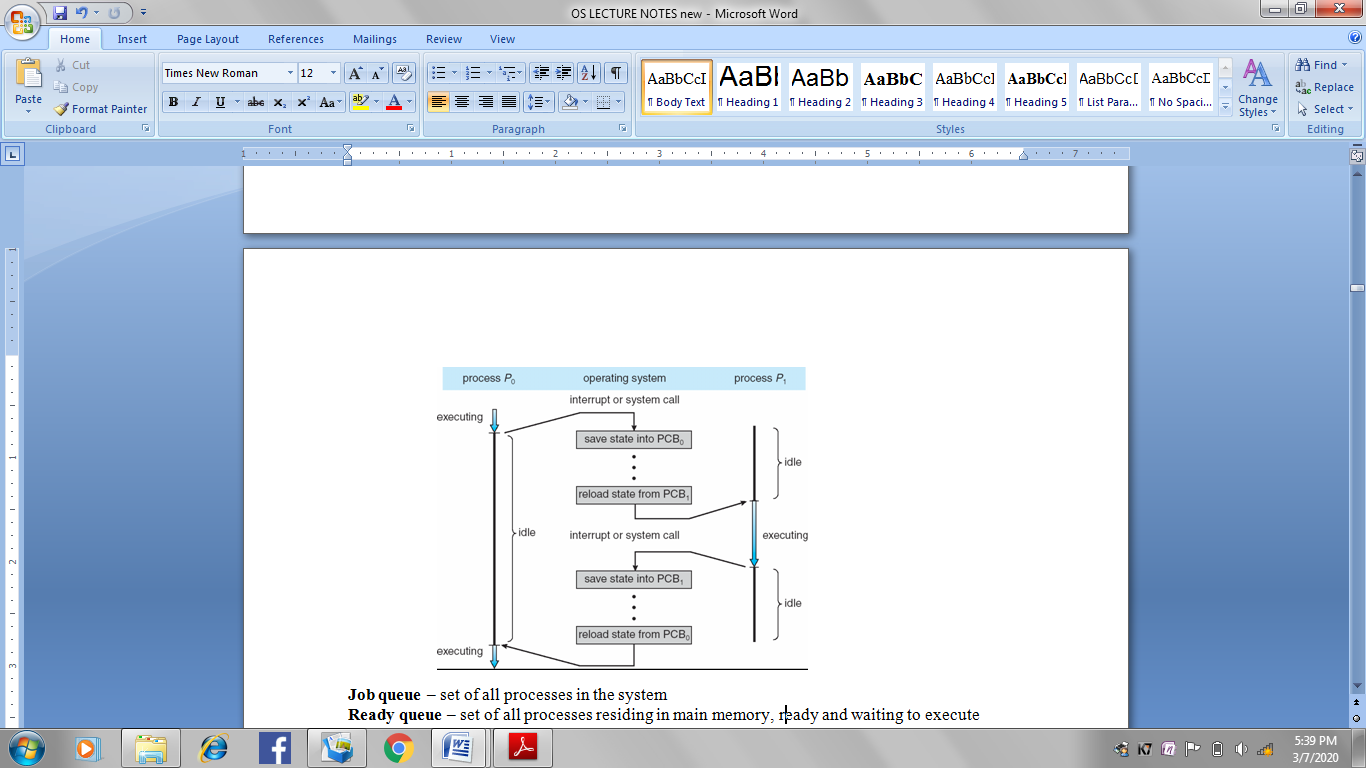
##### Diagram of Process State

**Process Control Block (PCB)**

Information associated with each process

* Process state
* Program counter
* CPU registers
* CPU scheduling information
* Memory-management information
* Accounting information
* I/O status information

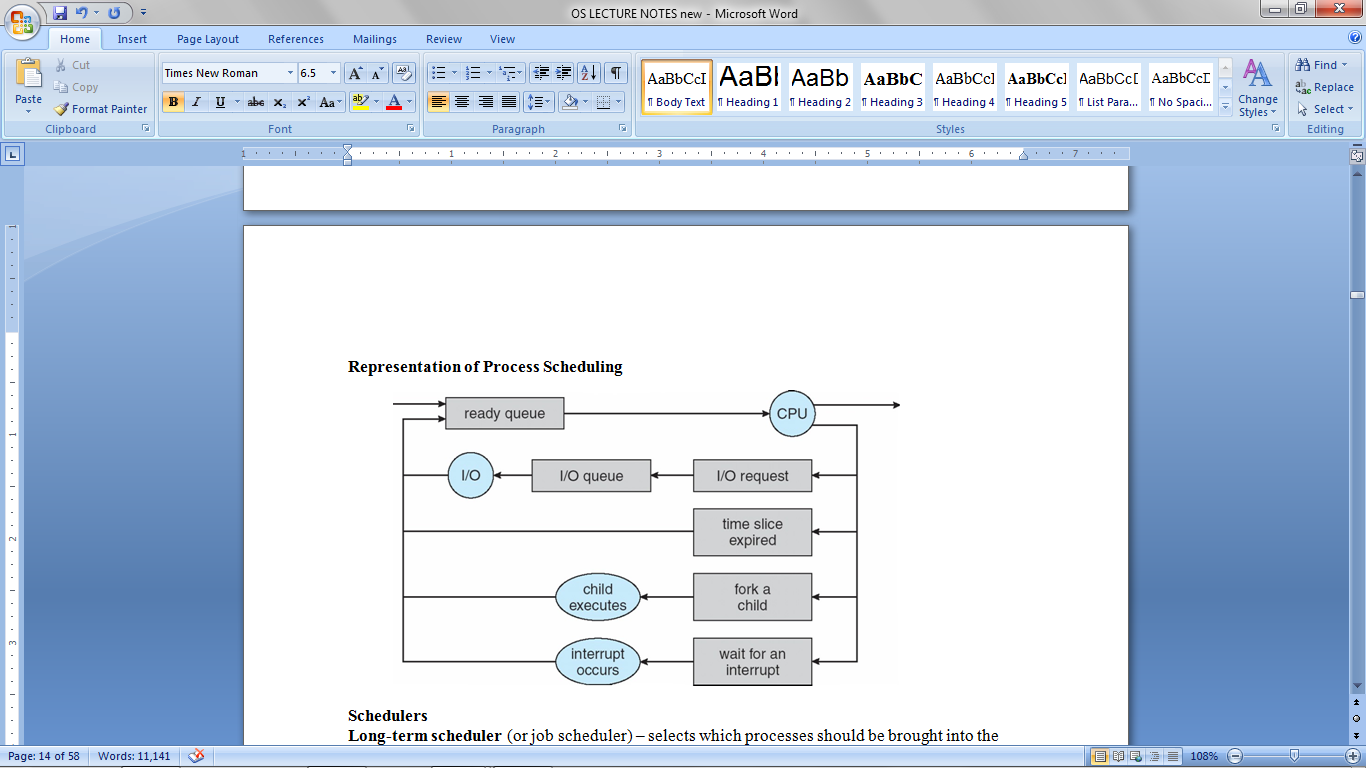
##### CPU Switch From Process to Process

****

**Process Scheduling Queues**

* **Job queue** – set of all processes in the system
* **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
* **Device queues** – set of processes waiting for an I/O device
* Processes migrate among the various queues

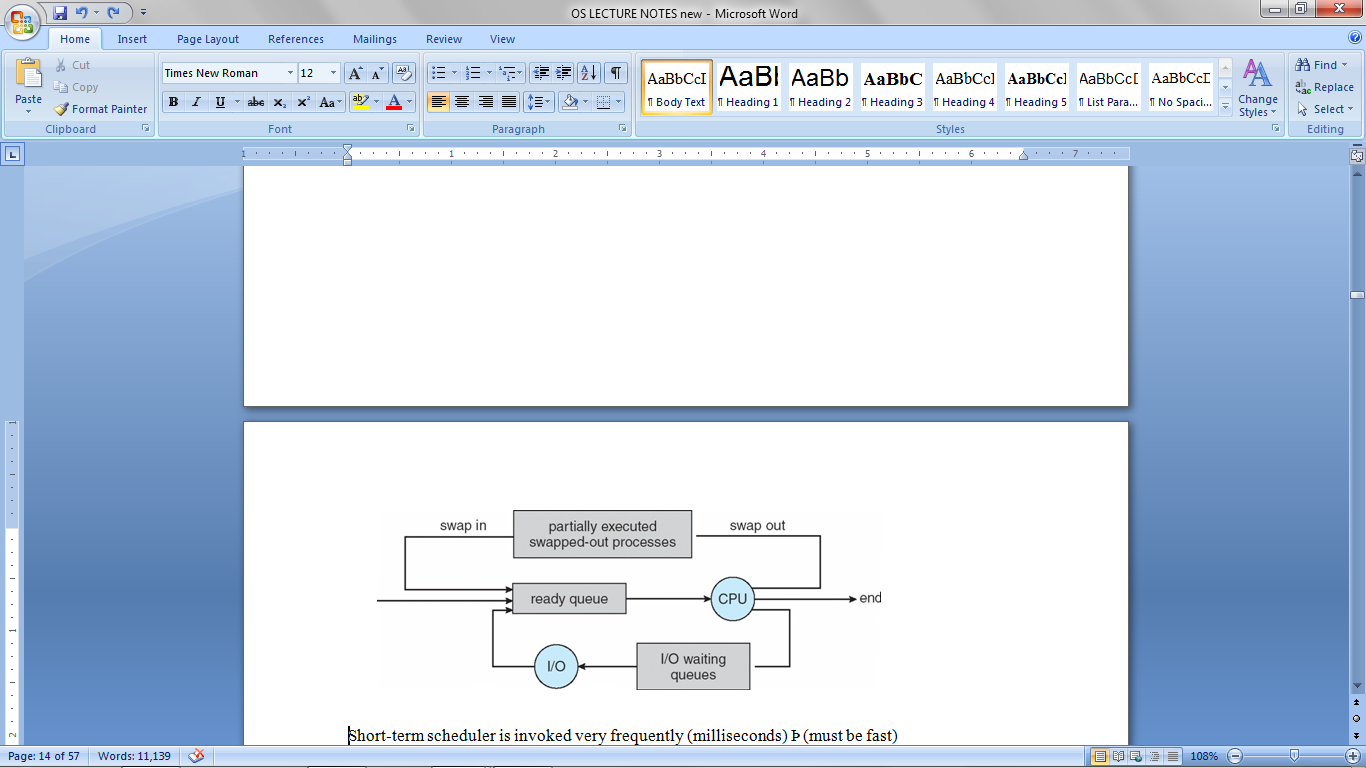
**Representation of Process Scheduling**



**Schedulers**

* **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
* **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU

##### Addition of Medium Term Scheduling

****

* Short-term scheduler is invoked very frequently (milliseconds) Þ (must be fast)
* Long-term scheduler is invoked very infrequently (seconds, minutes) Þ (may be slow
* The long-term scheduler controls the *degree of multiprogramming*
* Processes can be described as either:
* **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
* **CPU-bound process** – spends more time doing computations; few very long CPU bursts

##### Context Switch

##### When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch

* Context of a process represented in the PCB
* Context-switch time is overhead; the system does no useful work while switching
* Time dependent on hardware support

##### Process Creation

* **Parent** process create **children** processes, which, in turn create other processes, forming a tree of processes
* Generally, process identified and managed via **a process identifier** (**pid**)
* Resource sharing
* Parent and children share all resources
* Children share subset of parent’s resources
* Parent and child share no resources
* Execution
* Parent and children execute concurrently
* Parent waits until children terminate
* Address space
* Child duplicate of parent
* Child has a program loaded into it
* UNIX examples
* **fork** system call creates new process
* **exec** system call used after a **fork** to replace the process’ memory space with a new program

##### Process Creation

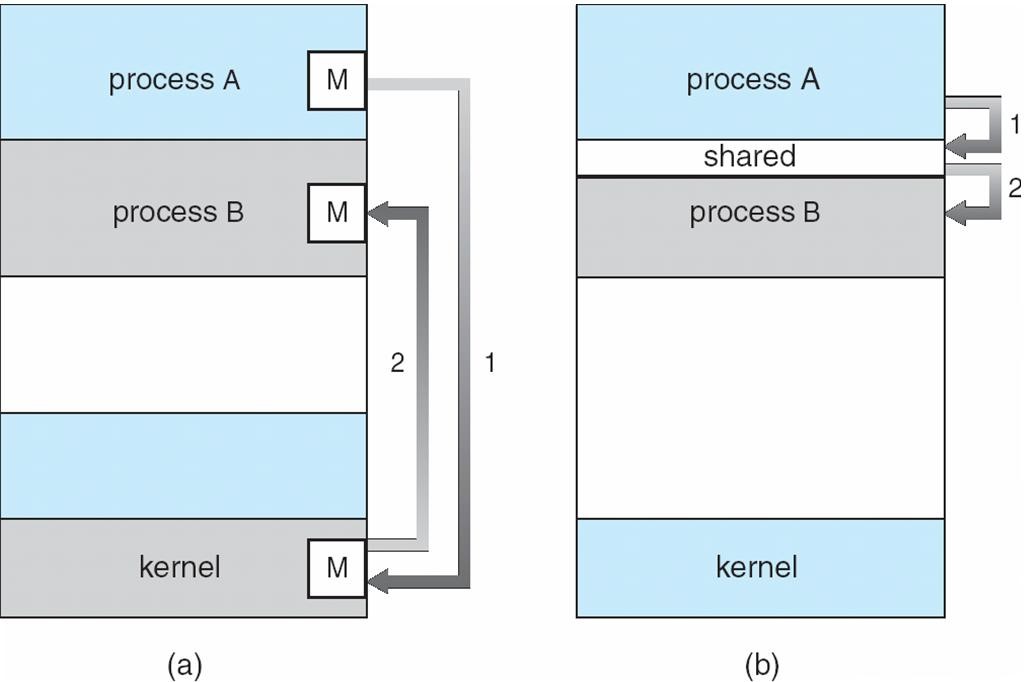
**Process Termination**

* Process executes last statement and asks the operating system to delete it (**exit**)
* Output data from child to parent (via **wait**)
* Process’ resources are deallocated by operating system
* Parent may terminate execution of children processes (**abort**)
* Child has exceeded allocated resources
* Task assigned to child is no longer required
* If parent is exiting Some operating system do not allow child to continue if its parent terminates
* All children terminated - **cascading termination**

##### Interprocess Communication

* Processes within a system may be **independent** or **cooperating**
* Cooperating process can affect or be affected by other processes, including sharing data
* Reasons for cooperating processes:
* Information sharing
* Computation speedup
* Modularity
* Convenience
* Cooperating processes need **interprocess communication** (**IPC**)
* Two models of IPC
* Shared memory
* Message passing

##### Communications Models



**Cooperating Processes**

* **Independent** process cannot affect or be affected by the execution of another process
* **Cooperating** process can affect or be affected by the execution of another process

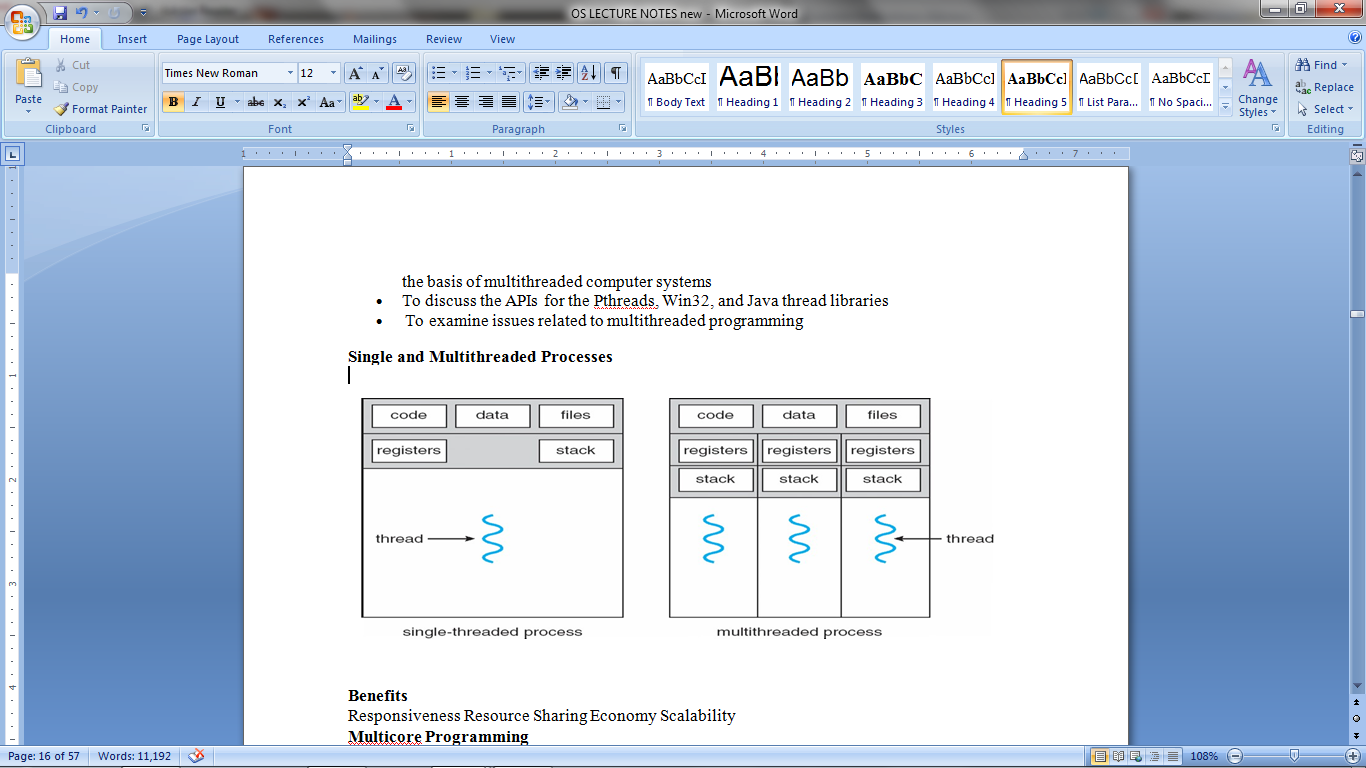
Advantages of process cooperation

* Information sharing
* Computation speed-up
* Modularity
* Convenience

**Threads**

* To introduce the notion of a thread — a fundamental unit of CPU utilization that forms the basis of multithreaded computer systems
* To discuss the APIs for the Pthreads, Win32, and Java thread libraries
* To examine issues related to multithreaded programming

##### Single and Multithreaded Processes

****

**Benefits**

* Responsiveness
* Resource Sharing
* Economy
* Scalability

##### Multicore Programming

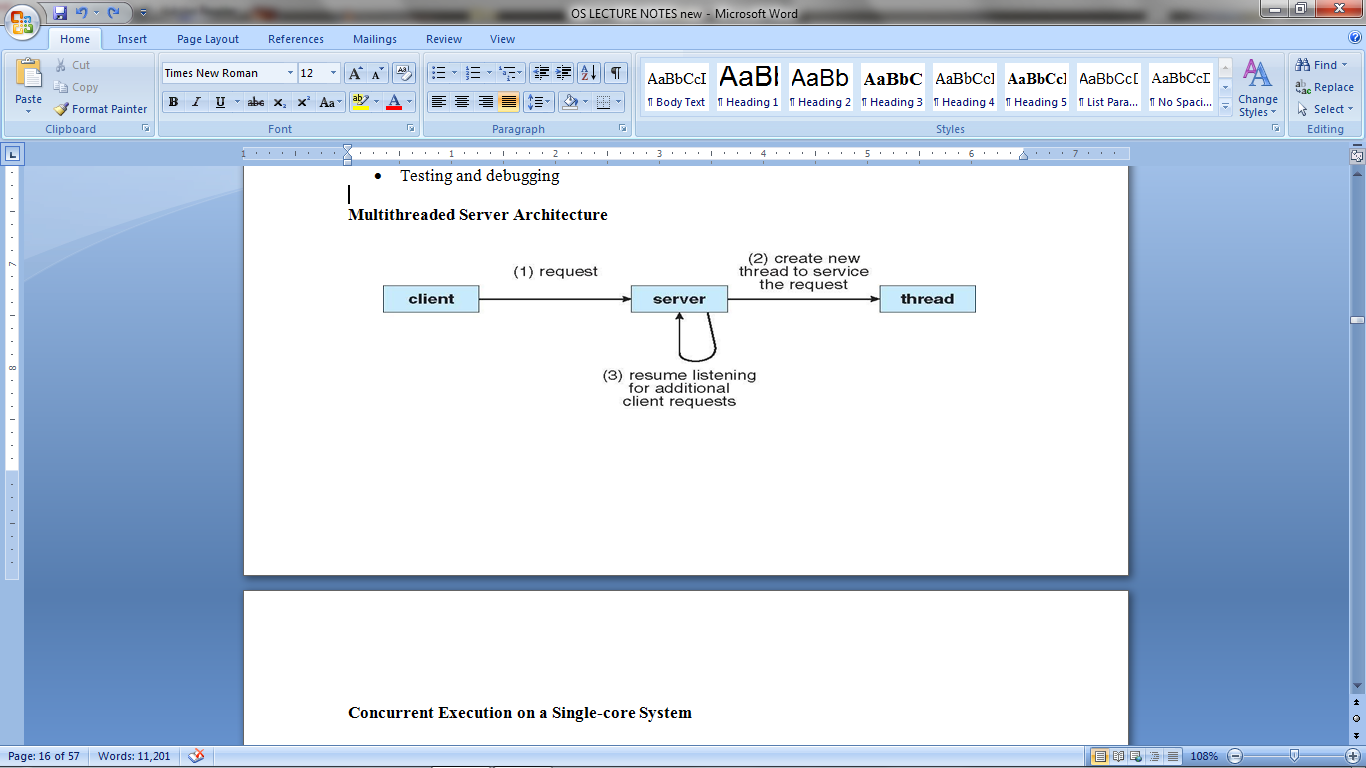
Multicore systems putting pressure on programmers, challenges include

##### Dividing activities

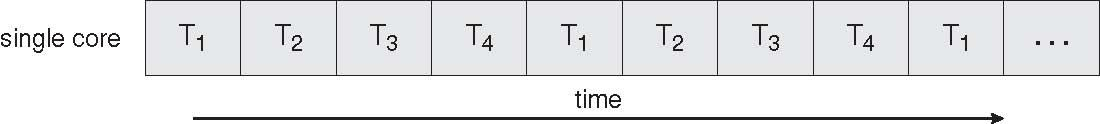
##### Balance

* Data splitting
* Data dependency
* Testing and debugging

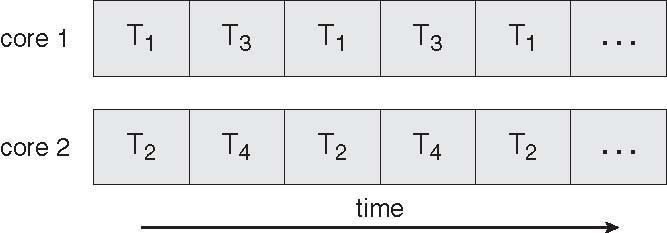
**Multithreaded Server Architecture**



**Concurrent Execution on a Single-core System**



**Parallel Execution on a Multicore System**

****

**User Threads**

* Thread management done by user-level threads librarynThree primary thread libraries:
* POSIX Pthreadsl Win32 threads
* Java threads

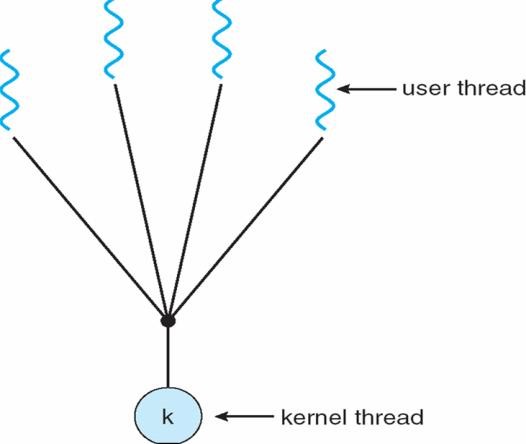
**Kernel Threads**

Supported by the Kernel

Examples

* Windows XP/2000 Solaris
* Linux
* Tru64 UNIX
* Mac OS X

##### Multithreading Models

* Many-to-One
* One-to-One
* Many-to-Many

##### Many-to-One

Many user-level threads mapped to single kernel thread

Examples:

* Solaris Green Threads
* GNU Portable Threads

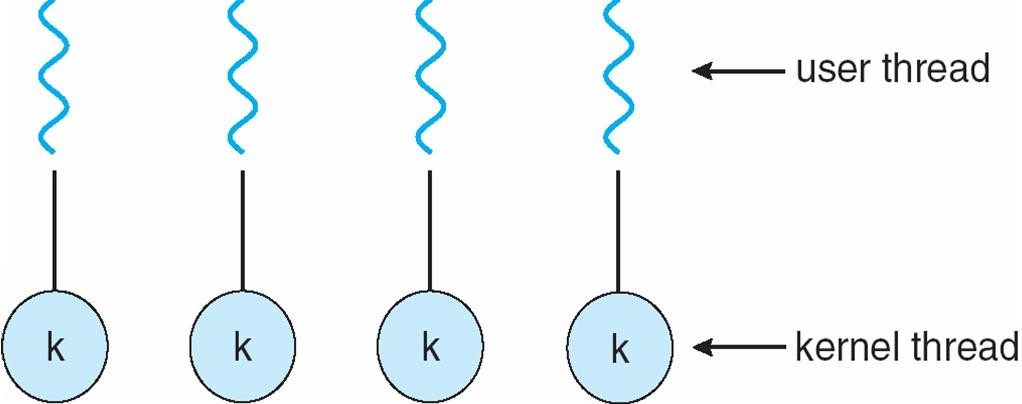
##### One-To-One

Each user-level thread maps to kernel thread

Examples

Windows NT/XP/2000

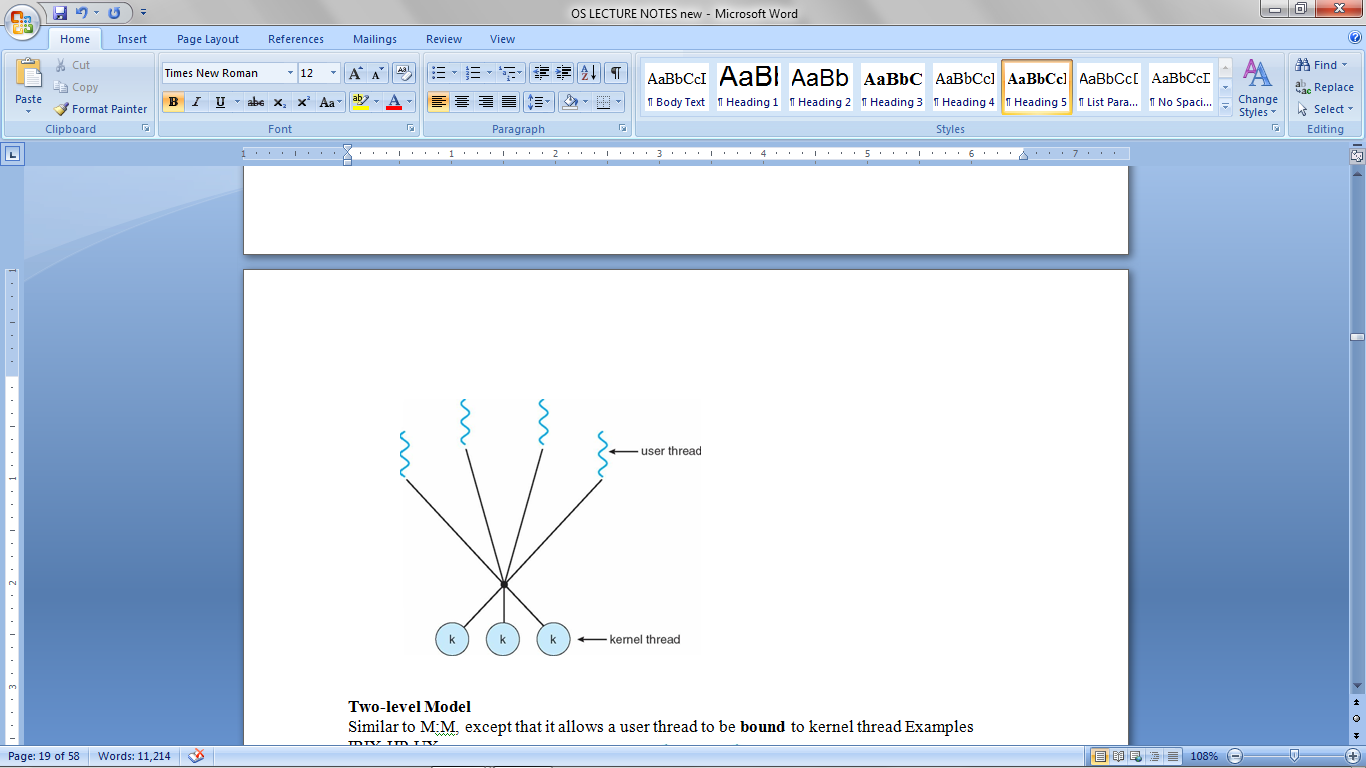
Linux

Solaris 9 and later

##### Many-to-Many Model

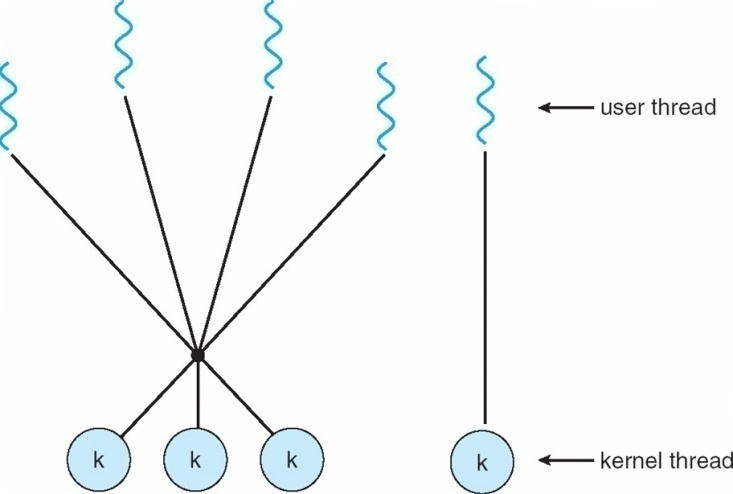
* Allows many user level threads to be mapped to many kernel threads
* Allows the operating system to create a sufficient number of kernel threads
* Solaris prior to version 9

Windows NT/2000 with the *ThreadFiber* package



##### Two-level Model

Similar to M:M, except that it allows a user thread to be **bound** to kernel thread

Examples

* IRIX
* HP-UX
* Tru64 UNIX
* Solaris 8 and earlier

**CPU Scheduler**

Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them CPU scheduling decisions may take place when a process:

1. Switches from running to waiting state
2. Switches from running to ready state
3. Switches from waiting to ready
4. Terminates

Scheduling under 1 and 4 is **nonpreemptive**

All other scheduling is **preemptive**

**Dispatcher**

* Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
* switching context
* switching to user mode
* jumping to the proper location in the user program to restart that program
* **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

##### Scheduling Criteria

* **CPU utilization** – keep the CPU as busy as possible
* **Throughput** – # of processes that complete their execution per time unit
* **Turnaround time** – amount of time to execute a particular process
* **Waiting time** – amount of time a process has been waiting in the ready queue
* **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)
* Max CPU utilization
* Max throughput
* Min turnaround time
* Min waiting time
* Min response time

##### First-Come, First-Served (FCFS) Scheduling

Process Burst Time

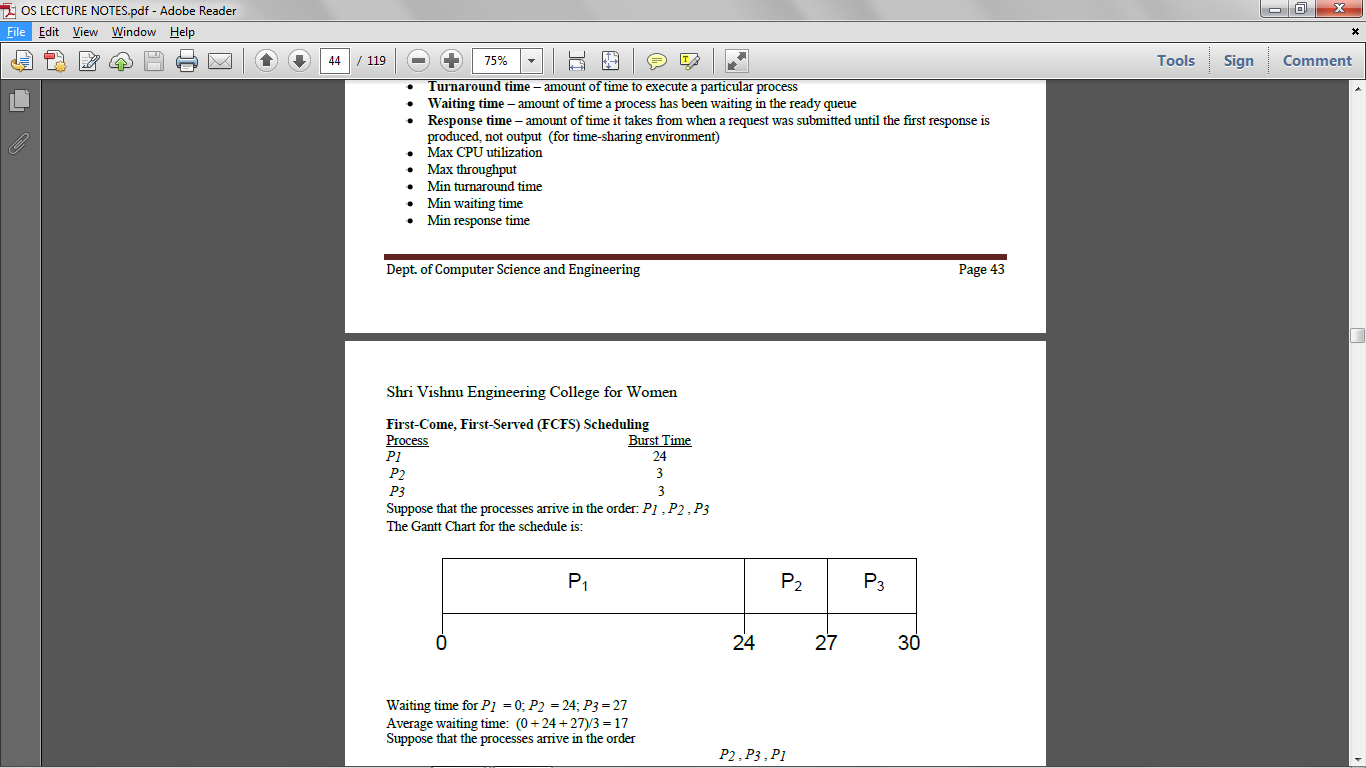
*P1* 24

*P2* 3

*P3* 3

Suppose that the processes arrive in the order: *P1* , *P2* , *P3*

The Gantt Chart for the schedule is:



Waiting time for *P1* = 0; *P2* = 24; *P3* = 27

Average waiting time: (0 + 24 + 27)/3 = 17

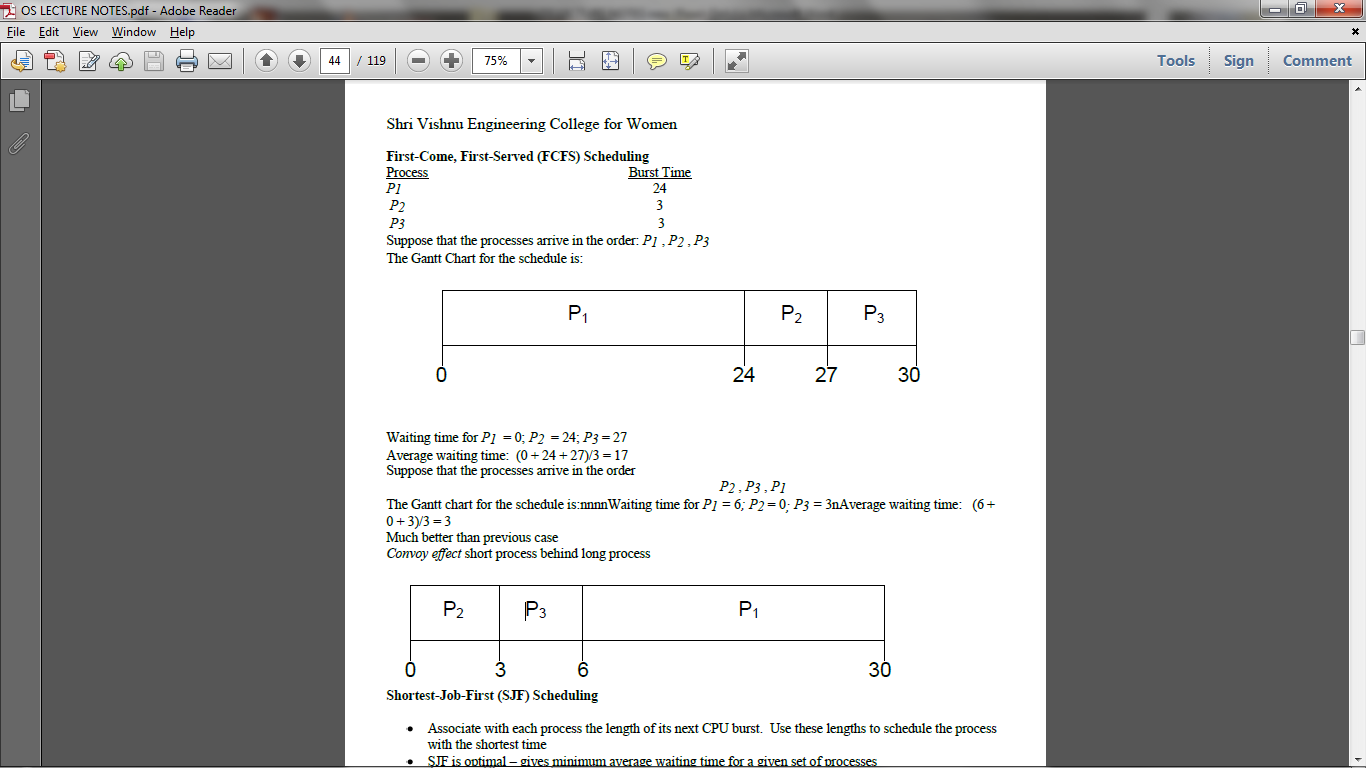
Suppose that the processes arrive in the order

*P2* , *P3* , *P1*

The Gantt chart for the schedule is:nnnnWaiting time for *P1 =* 6*; P2* = 0*; P3 =* 3nAverage waiting time: (6 + 0 + 3)/3 = 3

Much better than previous case

*Convoy effect* short process behind long process



##### Shortest-Job-First (SJF) Scheduling

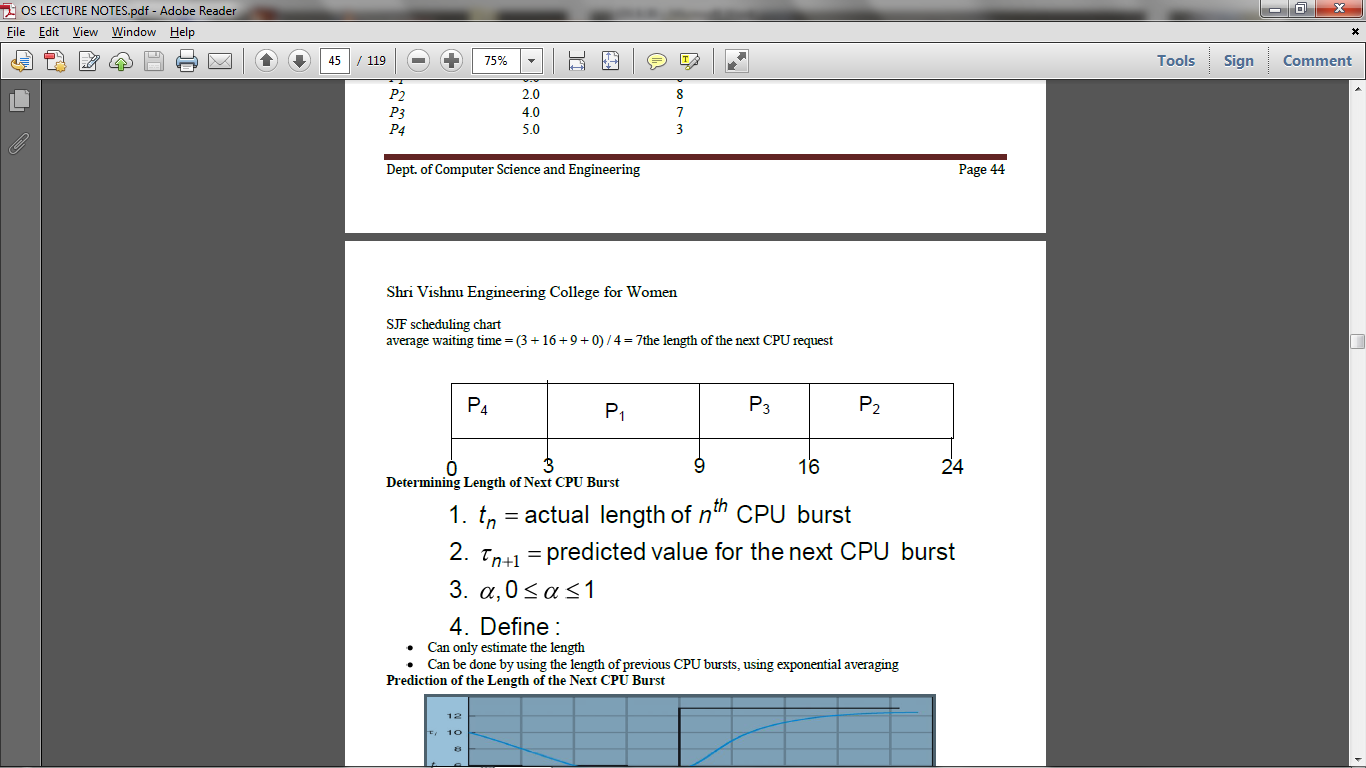
* Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
* SJF is optimal – gives minimum average waiting time for a given set of processes

The difficulty is knowing

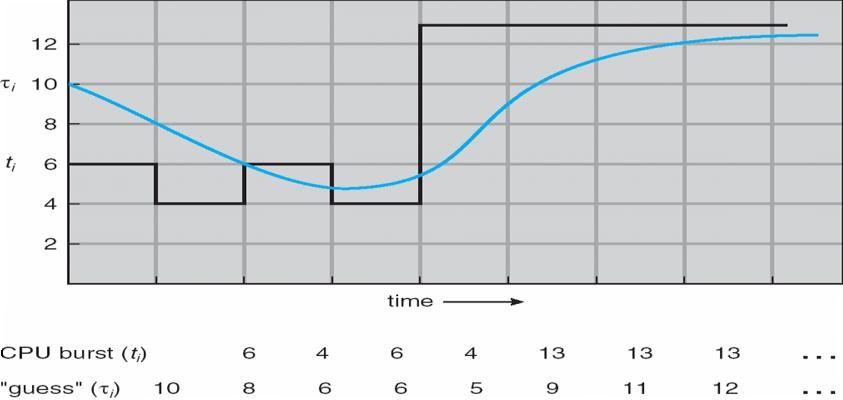
|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst Time |
| *P1* | 0.0 | 6 |
| *P2* | 2.0 | 8 |
| *P3* | 4.0 | 7 |
| *P4* | 5.0 | 3 |

SJF scheduling chart

average waiting time = (3 + 16 + 9 + 0) / 4 = 7the length of the next CPU request



##### Prediction of the Length of the Next CPU Burst



Examples of Exponential Averaging

a =0

tn+1 = tn

Recent history does not count

a =1

tn+1 = a *t*n

Only the actual last CPU burst counts

If we expand the formula, we get:

t*n*+1 = a t*n*+(1 *-* a*)*a *tn* -1 + …

*+(*1 - a *)j* a *tn* -*j* + …

*+(*1 - a *)n* +1 t0

Since both a and (1 - a) are less than or equal to 1, each successive term has less weight than its predecessor

##### Priority Scheduling

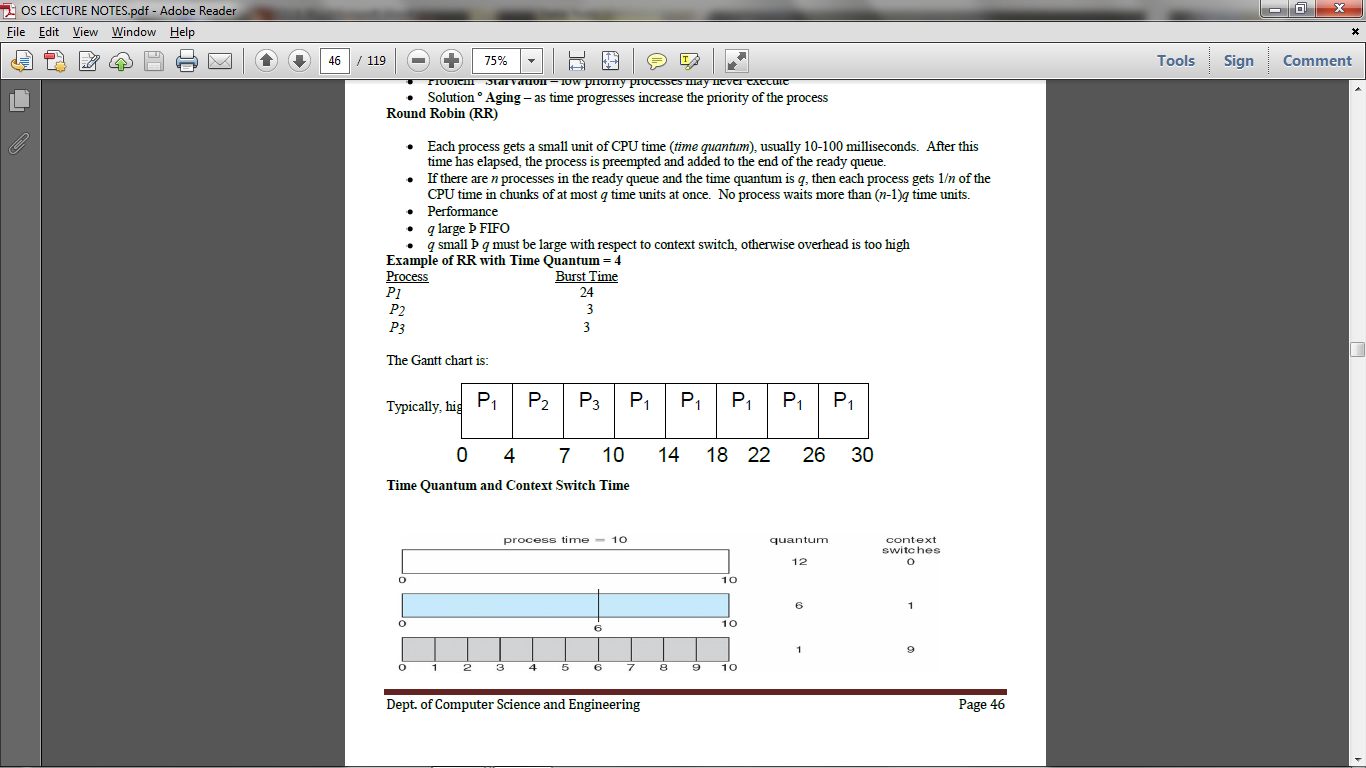
* A priority number (integer) is associated with each process
* The CPU is allocated to the process with the highest priority (smallest integer º highest priority)
* Preemptive
* nonpreemptive
* SJF is a priority scheduling where priority is the predicted next CPU burst time
* Problem º **Starvation** – low priority processes may never execute
* Solution º **Aging** – as time progresses increase the priority of the process

##### Round Robin (RR)

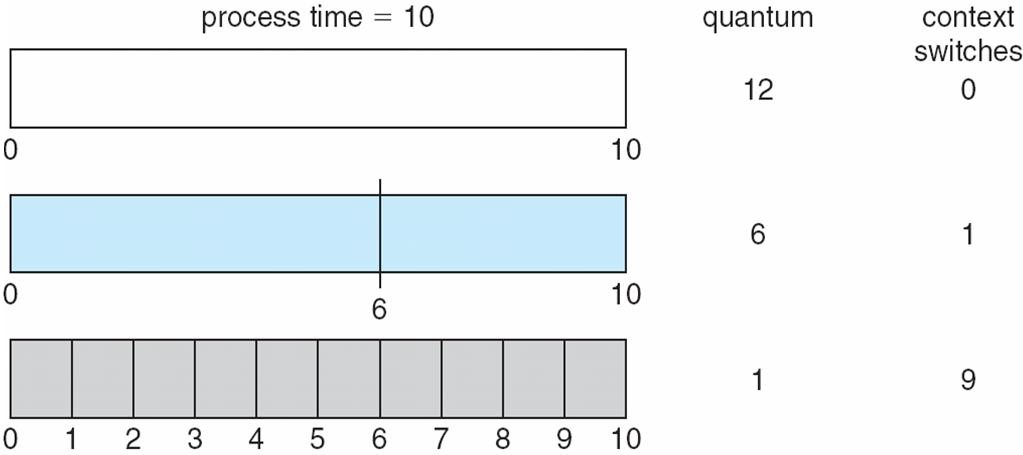
* Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
* If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
* Performance
* *q* large Þ FIFO
* *q* small Þ *q* must be large with respect to context switch, otherwise overhead is too high

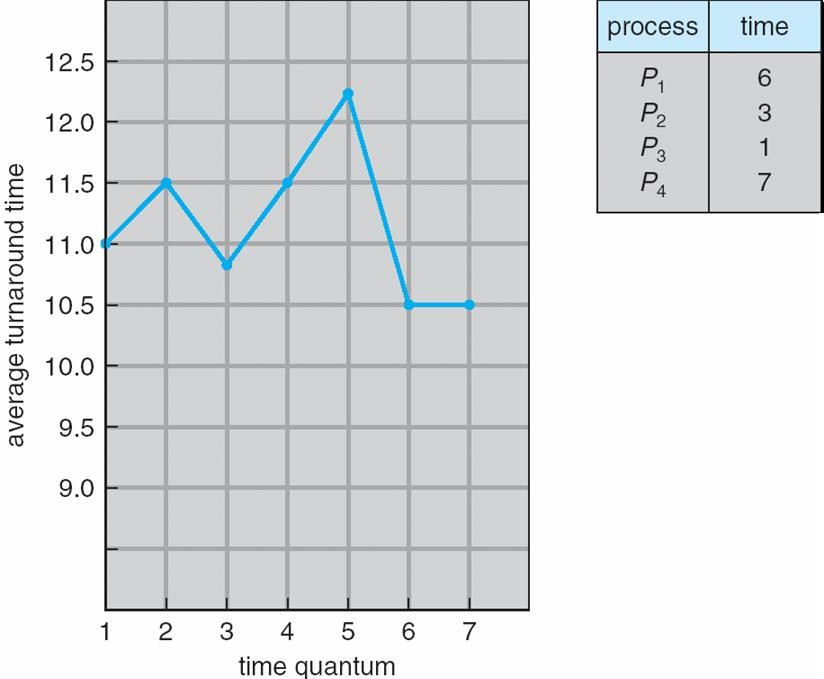
##### Example of RR with Time Quantum = 4

|  |  |
| --- | --- |
| Process | Burst Time |
| *P1* | 24 |
| *P2* | 3 |
| *P3*  The Gantt chart is: | 3 |

****

**Time Quantum and Context Switch Time**



**Turnaround Time Varies With The Time Quantum**

**Multilevel Queue**

* Ready queue is partitioned into separate queues:

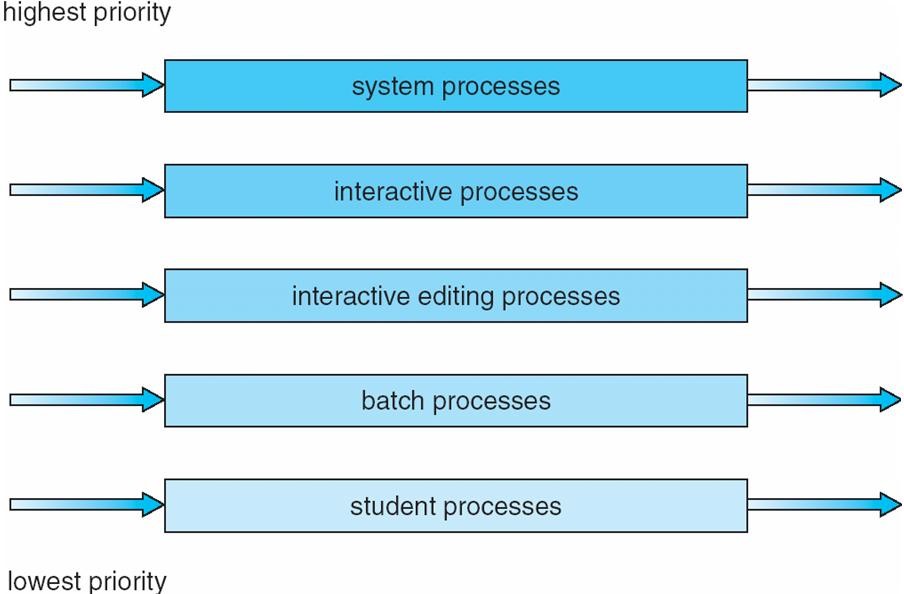
foreground (interactive)

background (batch)

* Each queue has its own scheduling algorithm
* foreground – RR
* background – FCFS
* Scheduling must be done between the queues
* Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
* Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR

20% to background in FCFS

##### Multilevel Queue Scheduling



**CHAPTER -3**

**CONCURRENCY**

**Process Synchronization**

* To introduce the critical-section problem, whose solutions can be used to ensure the consistency of shared data
* To present both software and hardware solutions of the critical-section problem
* To introduce the concept of an atomic transaction and describe mechanisms to ensure atomicity Concurrent access to shared data may result in data inconsistency
* Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
* Suppose that we wanted to provide a solution to the consumer-producer problem that fills all the buffers. We can do so by having an integer count that keeps track of the number of full buffers. Initially, count is set to 0. It is incremented by the producer after it produces a new buffer and is decremented by the consumer after it consumes a buffer

##### Solution to Critical-Section Problem

1. Mutual Exclusion - If process Pi is executing in its critical section, then no other processes can be executing in their critical sections
2. Progress - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely
3. Bounded Waiting - A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted Assume that each process executes at a nonzero speed

No assumption concerning relative speed of the N processes

##### Semaphore

* Synchronization tool that does not require busy waiting nSemaphore *S* – integer variable
* Two standard operations modify S: wait() and signal()
* Originally called P() and V()
* Less complicated
* Can only be accessed via two indivisible (atomic) operations wait (S)

{

while S <= 0

; // no-op

S--;

}

signal (S) {

S++;

}

##### Semaphore as General Synchronization Tool

* Counting semaphore – integer value can range over an unrestricted domain
* Binary semaphore – integer value can range only between 0 and 1; can be simpler to implement
* Also known as mutex locksnCan implement a counting semaphore S as a binary semaphore
* Provides mutual exclusionSemaphore mutex; // initialized to do {

wait (mutex);

// Critical Section

signal (mutex);

// remainder section

} while (TRUE);

##### Semaphore Implementation

* Must guarantee that no two processes can execute wait () and signal () on the same semaphore at the same time
* Thus, implementation becomes the critical section problem where the wait and signal code are placed in the crtical section.
* Could now have busy waiting in critical section implementation

But implementation code is short

Little busy waiting if critical section rarely occupied

* Note that applications may spend lots of time in critical sections and therefore this is not a good solution.

##### Semaphore Implementation with no Busy waiting

* With each semaphore there is an associated waiting queue. Each entry in a waiting queue has two data items:
* value (of type integer)
* pointer to next record in the list
* Two operations:
* block – place the process invoking the operation on the appropriate waiting queue.
* wakeup – remove one of processes in the waiting queue and place it in the ready queue.

Implementation of wait:

wait(semaphore \*S) {

S->value--;

if (S->value < 0) {

add this process to S->list;

block();

}

}

Implementation of signal:

signal(semaphore \*S) {

S->value++;

if (S->value <= 0) {

remove a process P from S->list;

wakeup(P);

}

}

##### Deadlock and Starvation

* Deadlock – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes
* Let S and Q be two semaphores initialized to 1

|  |  |  |
| --- | --- | --- |
| *P*0  wait (S);  wait (Q);  . | *P*1  wait (Q);  wait (S); |  |
| .  .  signal (S); | signal (Q); | .  . |
| signal (Q); | signal (S); |  |

* Starvation – indefinite blocking. A process may never be removed from the semaphore queue in which it is suspended
* Priority Inversion - Scheduling problem when lower-priority process holds a lock needed by higher- priority process

##### Monitors

A high-level abstraction that provides a convenient and effective mechanism for process synchronization Only one process may be active within the monitor at a time

monitor monitor-name

{

// shared variable declarations

procedure P1 (…) { …. }

…

procedure Pn (…) {……}

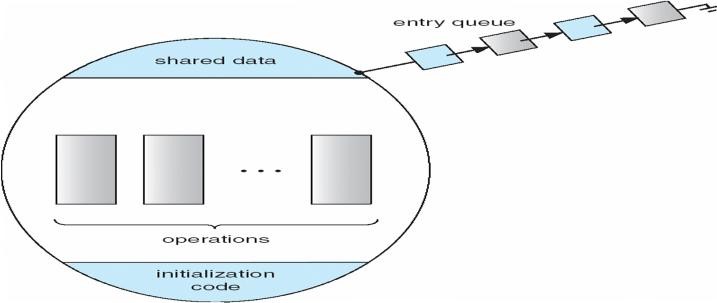
Initialization code ( ….) { … }

…

}

}

##### Schematic view of a Monitor



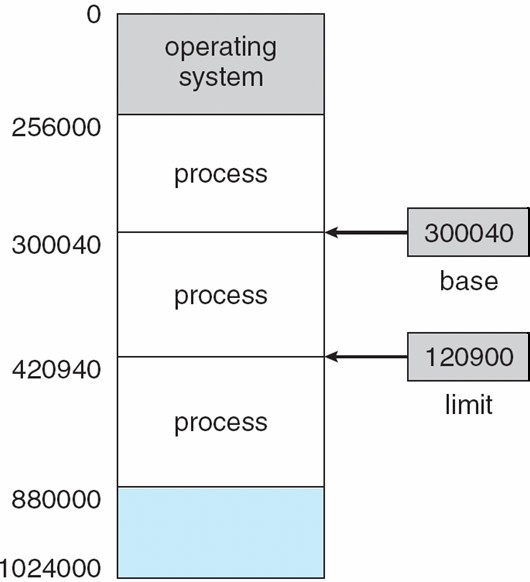
**CHAPTER IV**

**Memory Management**

* To provide a detailed description of various ways of organizing memory hardware
* To discuss various memory-management techniques, including paging and segmentation
* To provide a detailed description of the Intel Pentium, which supports both pure segmentation and segmentation with paging
* Program must be brought (from disk) into memory and placed within a process for it to be run
* Main memory and registers are only storage CPU can access directly
* Register access in one CPU clock (or less)
* Main memory can take many cycles
* **Cache** sits between main memory and CPU registers
* Protection of memory required to ensure correct operation

### Base and Limit Registers

A pair of **base** and **limit** registers define the logical address space



### Binding of Instructions and Data to Memory

* Address binding of instructions and data to memory addresses can happen at three different stages
* **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
* **Load time**: Must generate **relocatable code** if memory location is not known at compile time
* **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers)

### Multistep Processing of a User Program

### 

**Logical vs. Physical Address Space**

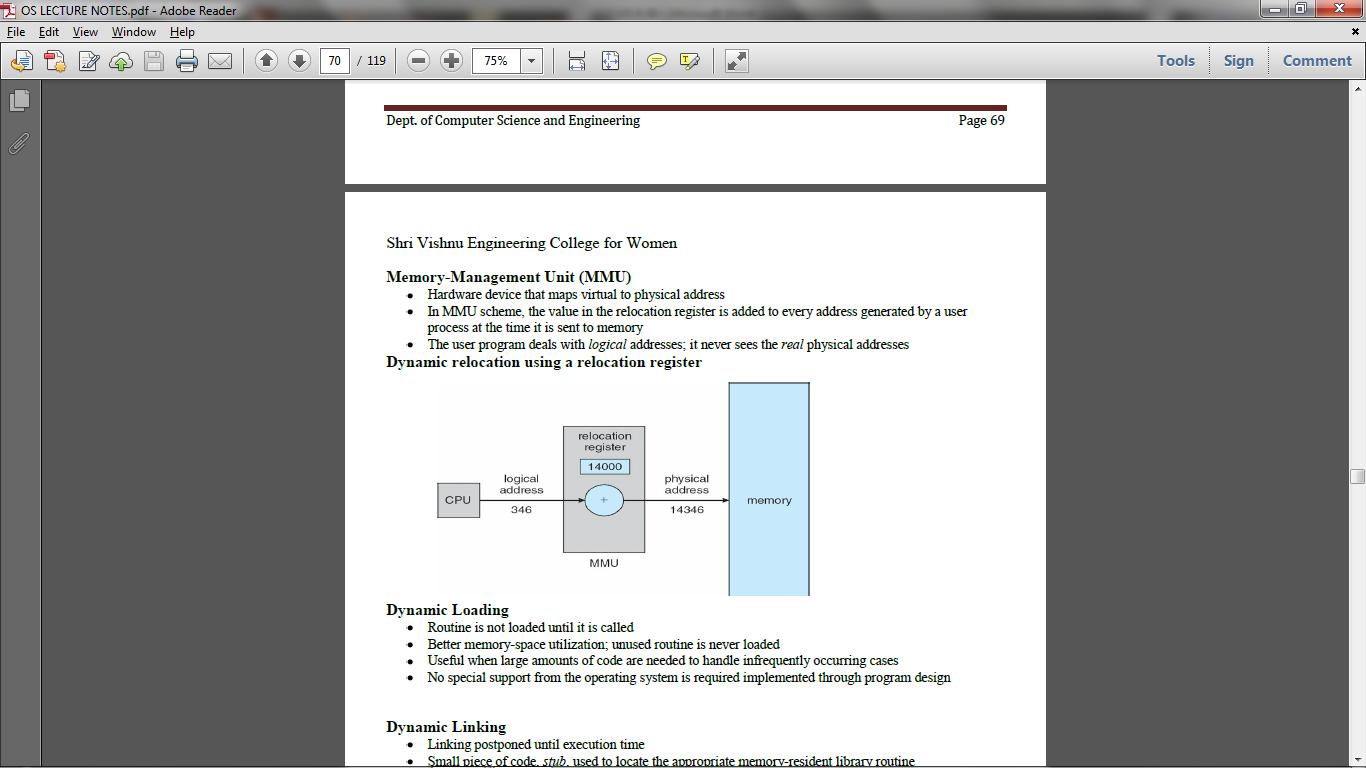
* The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
* **Logical address** – generated by the CPU; also referred to as **virtual address**
* **Physical address** – address seen by the memory unit
* Logical and physical addresses are the same in compile-time and load-time address-binding schemes;

logical (virtual) and physical addresses differ in execution-time address-binding scheme

### Memory-Management Unit (MMU)

* Hardware device that maps virtual to physical address
* In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
* The user program deals with *logical* addresses; it never sees the *real* physical addresses

**Dynamic relocation using a relocation register**

****

**Dynamic Loading**

* Routine is not loaded until it is called
* Better memory-space utilization; unused routine is never loaded
* Useful when large amounts of code are needed to handle infrequently occurring cases
* No special support from the operating system is required implemented through program design

### Dynamic Linking

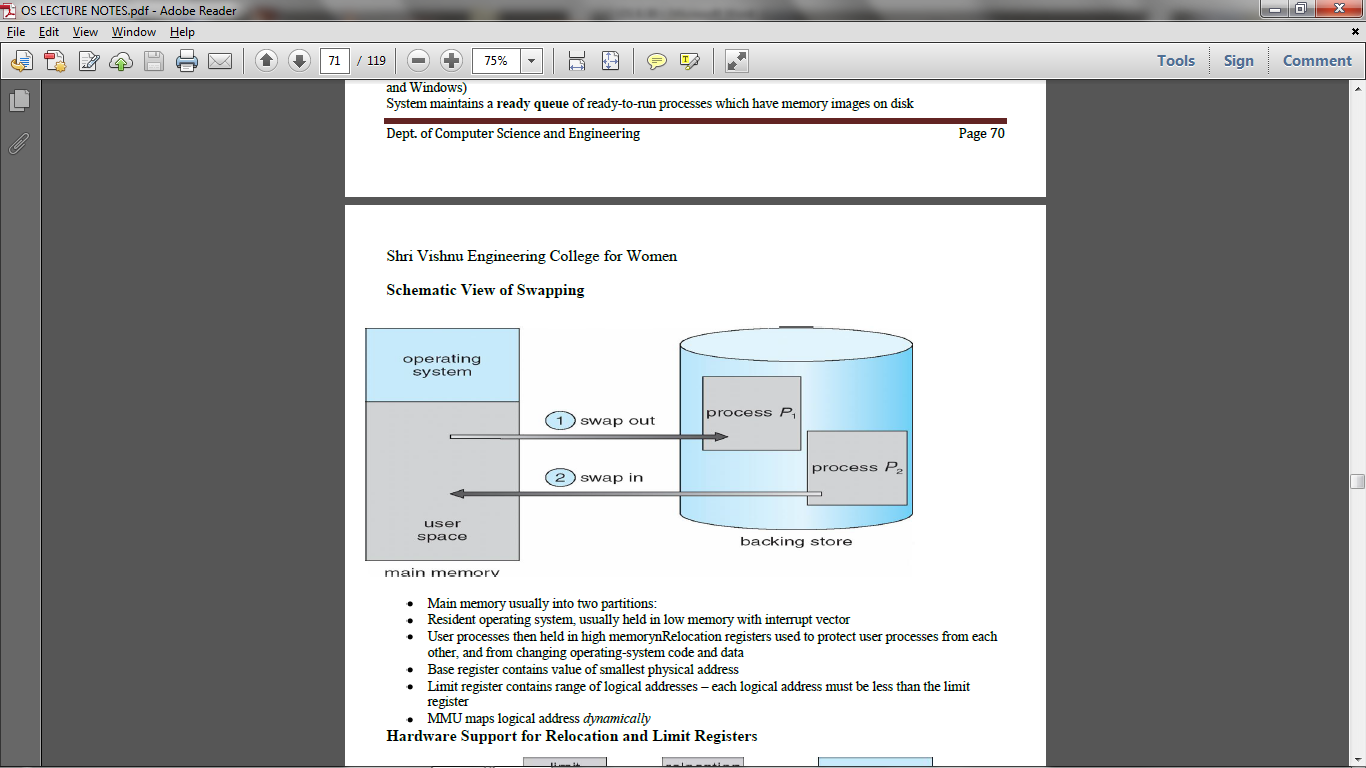
* Linking postponed until execution time
* Small piece of code, *stub*, used to locate the appropriate memory-resident library routine
* Stub replaces itself with the address of the routine, and executes the routine
* Operating system needed to check if routine is in processes’ memory address
* Dynamic linking is particularly useful for libraries
* System also known as **shared libraries**

### Swapping

A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued executionn**Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory imagesn**Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executednMajor part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swappednModified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

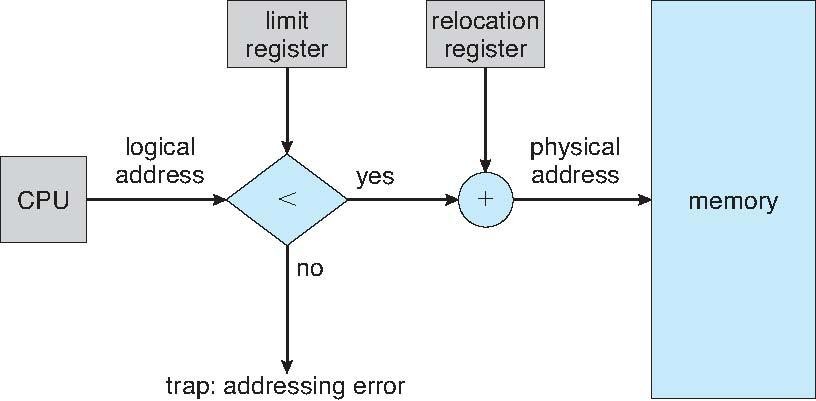
System maintains a **ready queue** of ready-to-run processes which have memory images on disk

**Schematic View of Swapping**

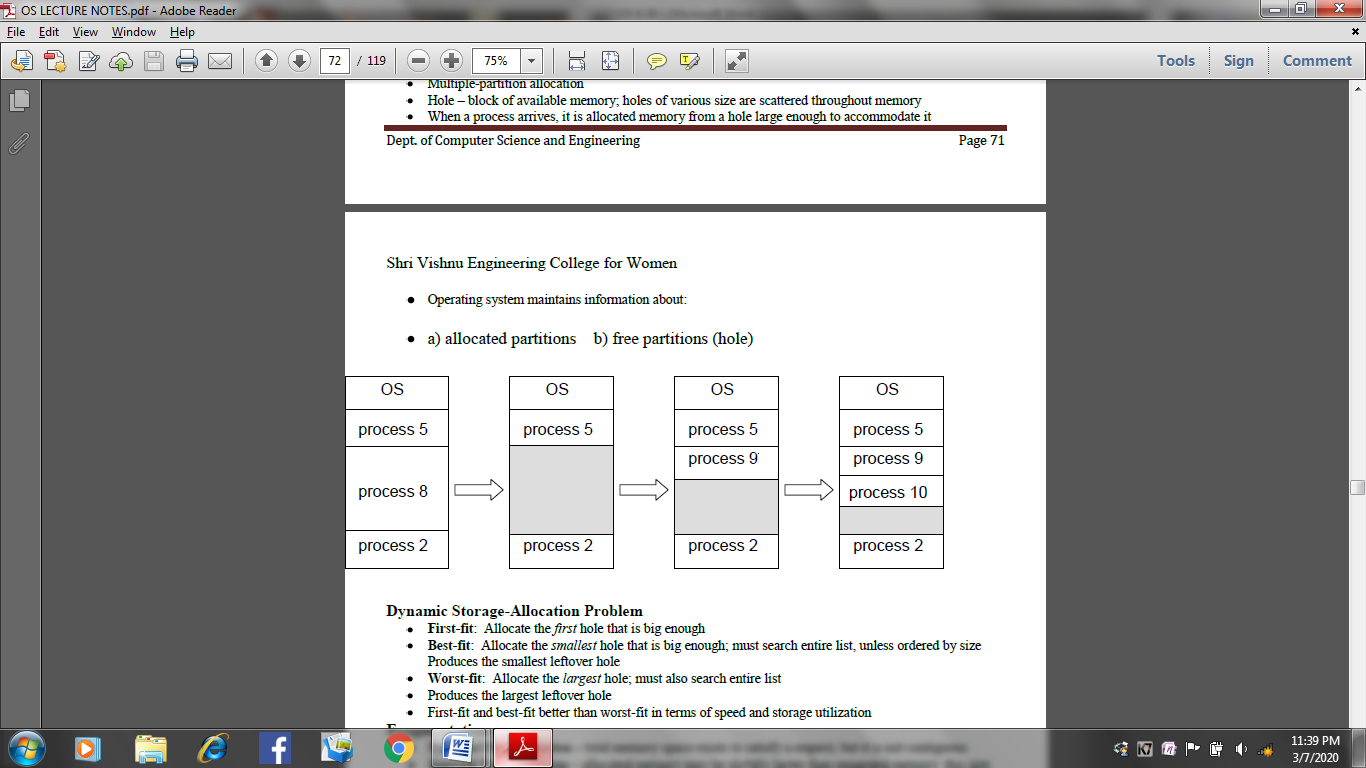
****

* Main memory usually into two partitions:
* Resident operating system, usually held in low memory with interrupt vector
* User processes then held in high memorynRelocation registers used to protect user processes from each other, and from changing operating-system code and data
* Base register contains value of smallest physical address
* Limit register contains range of logical addresses – each logical address must be less than the limit register
* MMU maps logical address *dynamically*

### Hardware Support for Relocation and Limit Registers



* Multiple-partition allocation
* Hole – block of available memory; holes of various size are scattered throughout memory
* When a process arrives, it is allocated memory from a hole large enough to accommodate it
* Operating system maintains information about:
* a) allocated partitions b) free partitions (hole)



**Dynamic Storage-Allocation Problem**

* **First-fit**: Allocate the *first* hole that is big enough
* **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size Produces the smallest leftover hole
* **Worst-fit**: Allocate the *largest* hole; must also search entire list
* Produces the largest leftover hole
* First-fit and best-fit better than worst-fit in terms of speed and storage utilization

### Fragmentation

* **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
* **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
* Reduce external fragmentation by **compaction**
* Shuffle memory contents to place all free memory together in one large block
* Compaction is possible *only* if relocation is dynamic, and is done at execution time.
* I/O problem
  + Latch job in memory while it is involved in I/O
  + Do I/O only into OS buffers

### Paging

* Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
* Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes)
* Divide logical memory into blocks of same size called **pages**nKeep track of all free frames
* To run a program of size ***n*** pages, need to find *n* free frames and load program
* Set up a page table to translate logical to physical addresses
* Internal fragmentation

### Address Translation Scheme

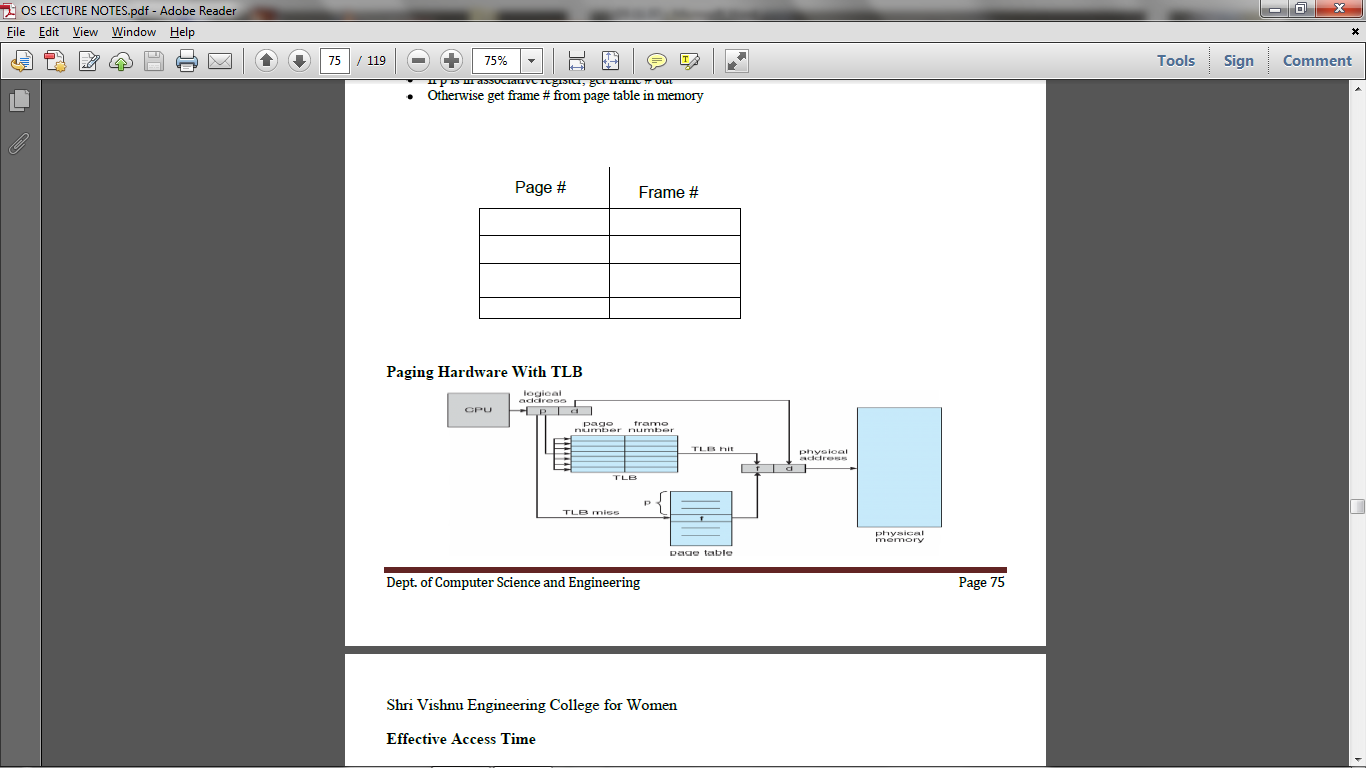
* Address generated by CPU is divided into
* **Page number (*p*)** – used as an index into a *page table* which contains base address of each page in physical memory
* **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit
* For given logical address space 2*m and page size 2*n

**Paging Hardware**

### 

### Associative Memory

* Associative memory – parallel search
* Address translation (p, d)
* If p is in associative register, get frame # out
* Otherwise get frame # from page table in memory



**Effective Access Time**

* Associative Lookup = e time unit
* Assume memory cycle time is 1 microsecond
* Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
* Hit ratio = an **Effective Access Time** (EAT)

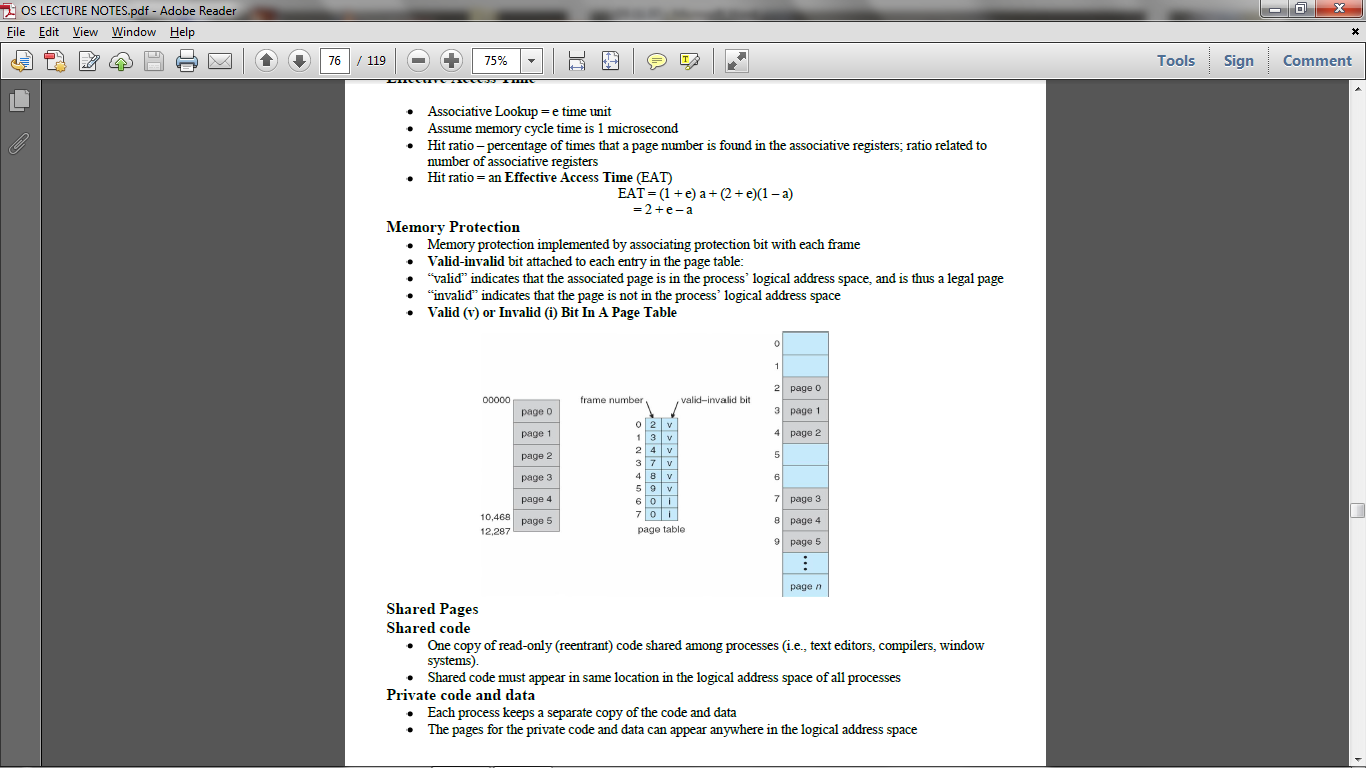
EAT = (1 + e) a + (2 + e)(1 – a)

= 2 + e – a

### Memory Protection

* Memory protection implemented by associating protection bit with each frame
* **Valid-invalid** bit attached to each entry in the page table:
* “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page
* “invalid” indicates that the page is not in the process’ logical address space

##### Valid (v) or Invalid (i) Bit In A Page Table

****

**Shared Pages**

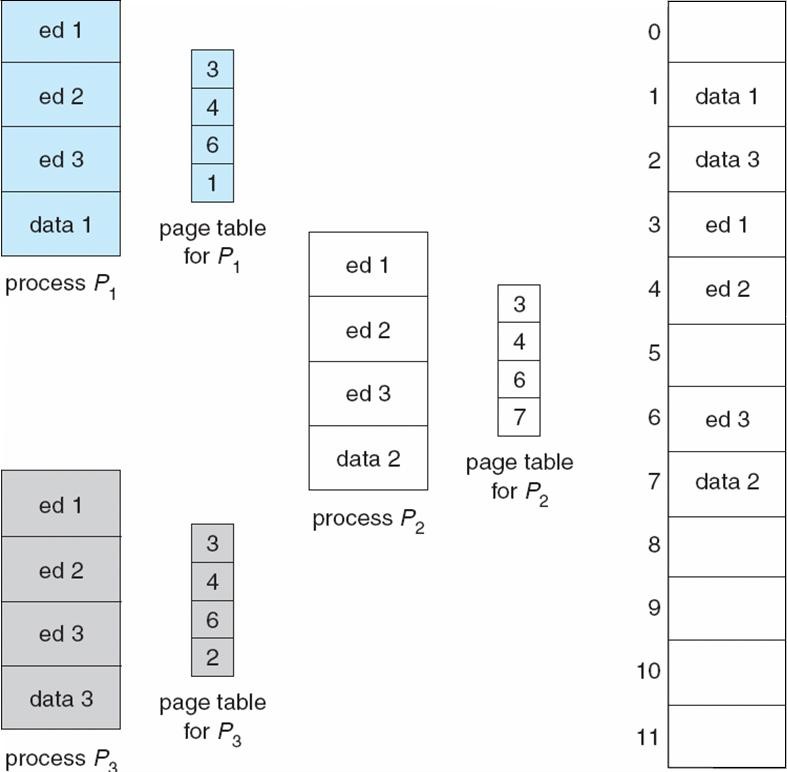
**Shared code**

* One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
* Shared code must appear in same location in the logical address space of all processes

### Private code and data

* Each process keeps a separate copy of the code and data
* The pages for the private code and data can appear anywhere in the logical address space

**Shared Pages Example**



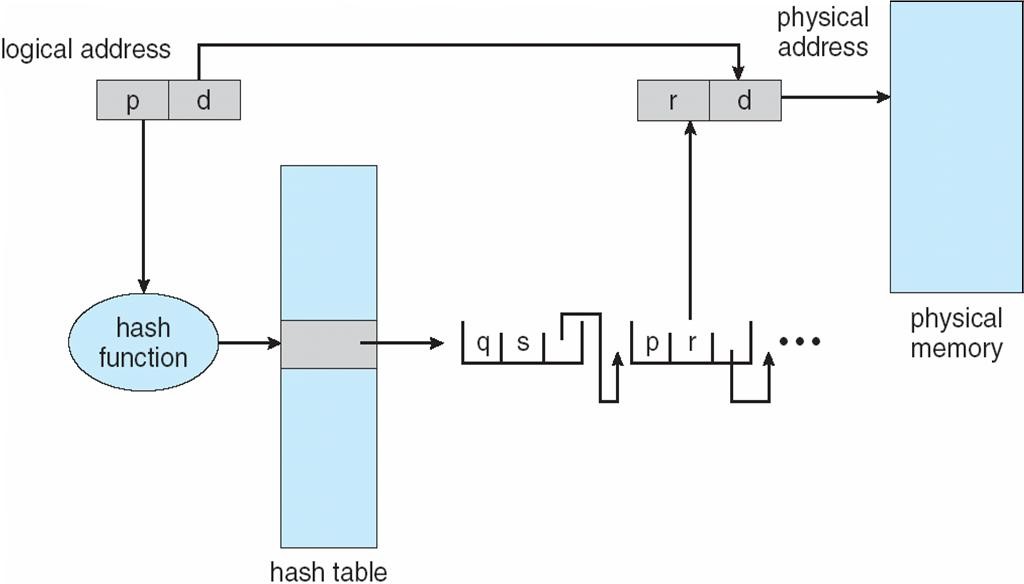
**Structure of the Page Table**

* Hierarchical Paging
* Hashed Page Tables
* Inverted Page Tables

**Hashed Page Tables**

* Common in address spaces > 32 bits
* The virtual page number is hashed into a page table
* This page table contains a chain of elements hashing to the same location
* Virtual page numbers are compared in this chain searching for a match
* If a match is found, the corresponding physical frame is extracted

**Hashed Page Table**

**Segmentation**

Memory-management scheme that supports user view of memory

* A program is a collection of segments
* A segment is a logical unit such as:
* main program
* procedure function
* method
* object
* local variables, global variables
* common block
* stack
* symbol table
* arrays

### User’s View of a Program

**Logical View of Segmentation**

|  |
| --- |
| 1 |
| 4 |
|  |
|  |
| 2 |
| 3 |
|  |

4

3

2

1

user space

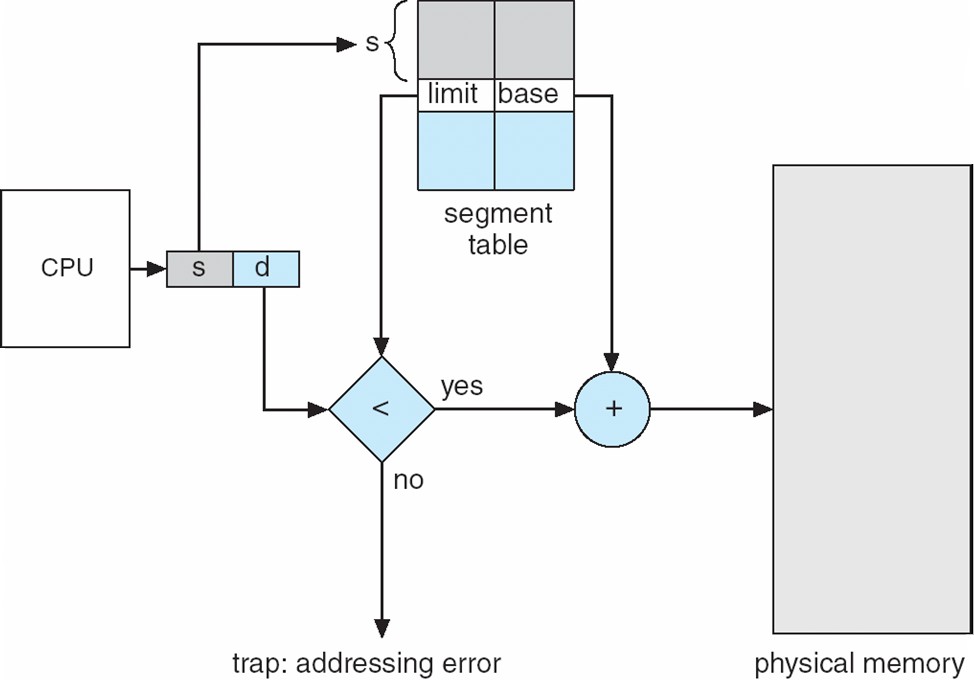
**Segmentation Architecture**

* Logical address consists of a two tuple:

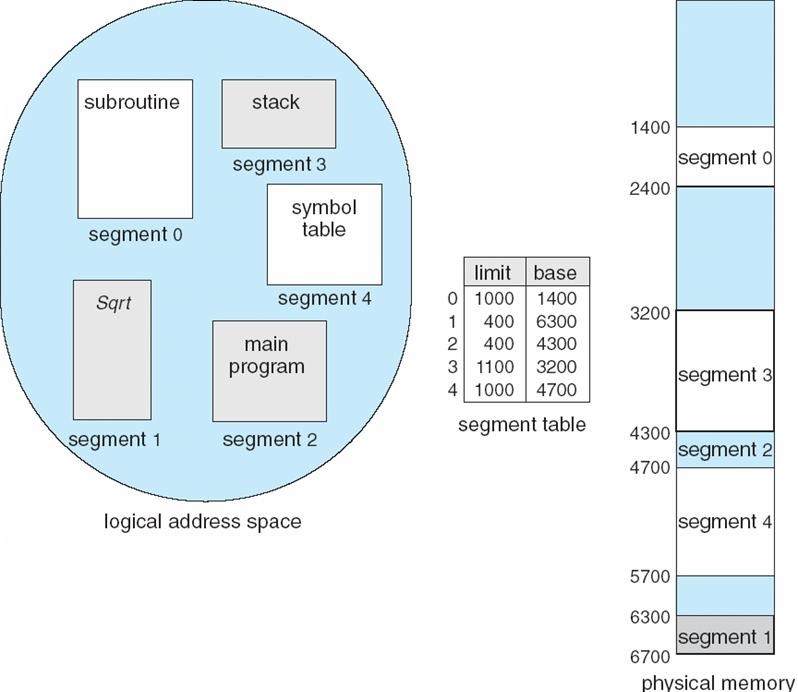
<segment-number, offset>,

* **Segment table** – maps two-dimensional physical adpdrhesysess;iecaachltambleeemntroy rhyas:space
* **base** – contains the starting physical address where the segments reside in memory
* **limit** – specifies the length of the segment
* **Segment-table base register (STBR)** points to the segment table’s location in memory
* **Segment-table length register (STLR)** indicates number of segments used by a program; segment number ***s*** is legal if ***s*** < **STLR**
* Protection
* With each entry in segment table associate:
* validation bit = 0 Þ illegal segment
* read/write/execute privileges
* Protection bits associated with segments; code sharing occurs at segment level
* Since segments vary in length, memory allocation is a dynamic storage-allocation problem
* A segmentation example is shown in the following diagram

### Segmentation Hardware



**Example of Segmentation**



**CHAPTER – V**

**VIRTUAL MEMORY**

**Objective**

* To describe the benefits of a virtual memory system.
* To explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames.
* To discuss the principle of the working-set model.

### Virtual Memory

* Virtual memory is a technique that allows the execution of process that may not be completely in memory. The main visible advantage of this scheme is that programs can be larger than physical memory.
* Virtual memory is the separation of user logical memory from physical memory this separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available ( Fig ).
* Following are the situations, when entire program is not required to load fully.

1. User written error handling routines are used only when an error occurs in the data or computation.
2. Certain options and features of a program may be used rarely.
3. Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.

* The ability to execute a program that is only partially in memory would counter many benefits.

1. Less number of I/O would be needed to load or swap each user program into memory.
2. A program would no longer be constrained by the amount of physical memory that is available.
3. Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

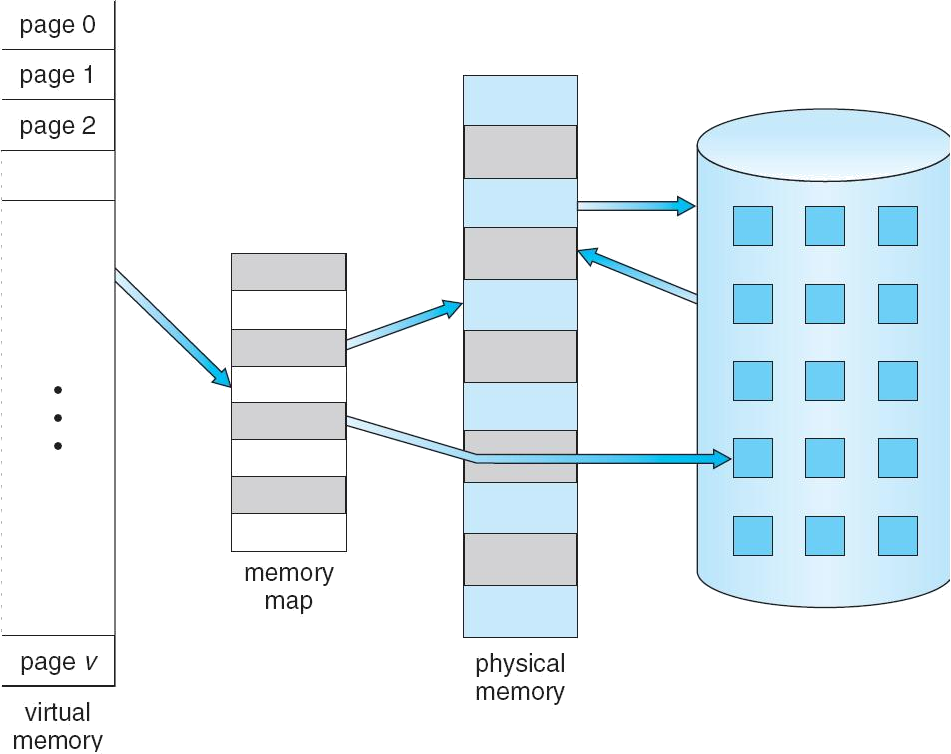


Fig. Diagram showing virtual memory that is larger than physical memory.

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

### Demand Paging

A demand paging is similar to a paging system with swapping(Fig 5.2). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory.

When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked checking the bit and marking a page will have no effect if the process never attempts to access the pages. While the process executes and accesses pages that are memory resident, execution proceeds normally.

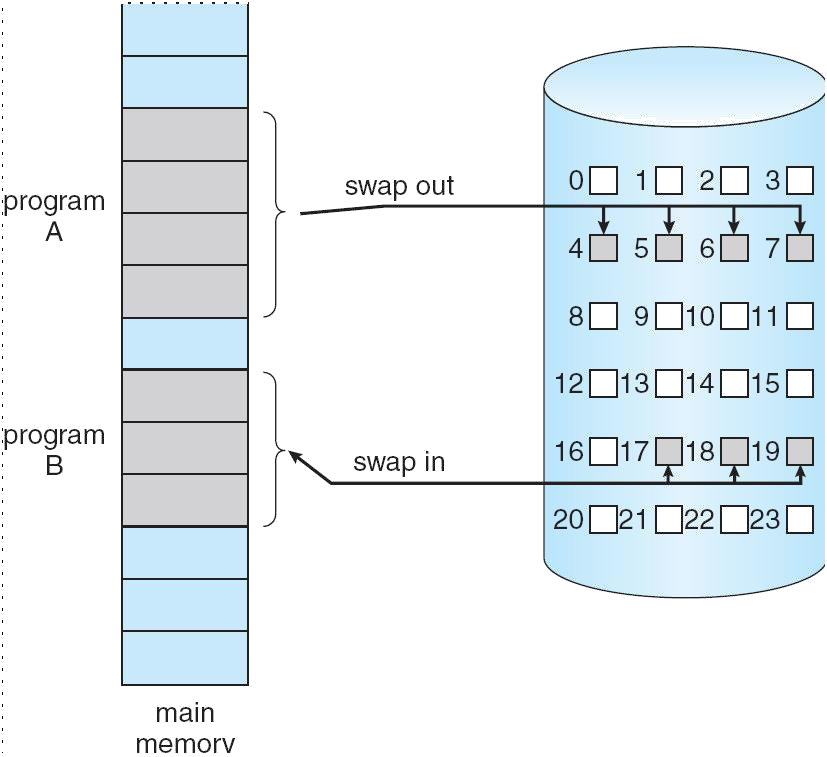


Fig. Transfer of a paged memory to continuous disk space

### Advantages of Demand Paging:

1. Large virtual memory.
2. More efficient use of memory.
3. Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

### Disadvantages of Demand Paging:

1. Number of tables and amount of processor over head for handling page interrupts are greater than in the case of the simple paged management techniques.
2. due to the lack of an explicit constraints on a jobs address space size.

**CHAPTER VI**

**Principles of deadlock**

To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks.To present a number of different methods for preventing or avoiding deadlocks in a computer system

### The Deadlock Problem

A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set

Example

System has 2 disk drives

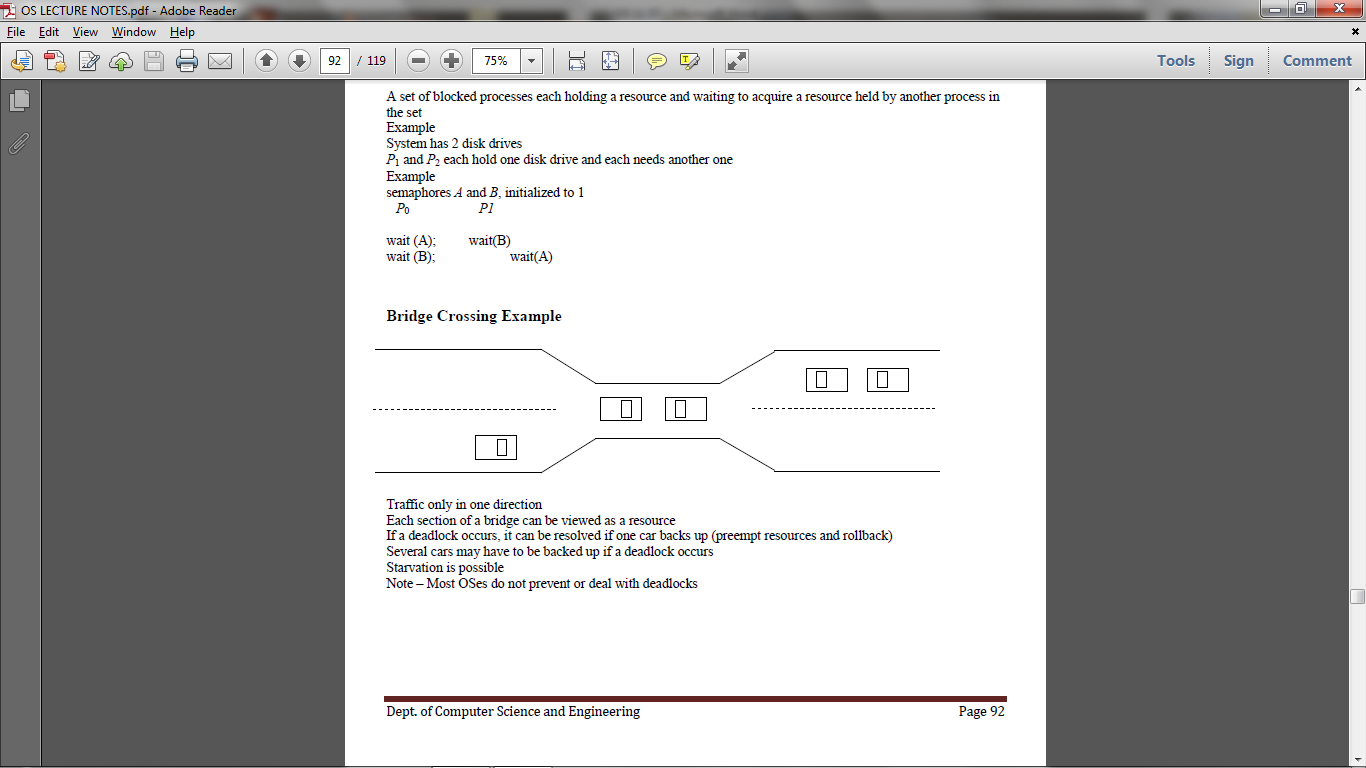
*P*1 and *P*2 each hold one disk drive and each needs another one Example

semaphores *A* and *B*, initialized to 1

*P*0 *P1*

wait (A); wait(B)

wait (B); wait(A)



Traffic only in one direction

Each section of a bridge can be viewed as a resource

If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback) Several cars may have to be backed up if a deadlock occurs

Starvation is possible

Note – Most OSes do not prevent or deal with deadlocks

### System Model

Resource types *R*1, *R*2, . . ., *R*m

*CPU cycles, memory space, I/O devices*

Each resource type *R*i has *W*i instances.

Each process utilizes a resource as follows:

##### request

##### use

##### release

**Deadlock Characterization**

Deadlock can arise if four conditions hold simultaneously

**Mutual exclusion:** only one process at a time can use a resource

**Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes

**No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task

**Circular wait:** there exists a set {*P*0, *P*1, …, *P*0} of waiting processes such that *P*0 is waiting for a resource that is held by *P*1, *P*1 is waiting for a resource that is held by

*P*2, …, *Pn*–1 is waiting for a resource that is held by

*P*n, and *P*0 is waiting for a resource that is held by *P*0.

n

### Resource-Allocation Graph

#### A set of vertices V and a set of edges E

V is partitioned into two types:

*P* = {*P*1, *P*2, …, *Pn*}, the set consisting of all the processes in the system *R* = {*R*1, *R*2, …, *Rm*}, the set consisting of all resource types in the system request edge – directed edge *P*1 ® *Rj*

assignment edge – directed edge *Rj* ® *Pi*

Process

Resource Type with 4 instances

*Pi* requests instance of *Rj*n

*Pi*

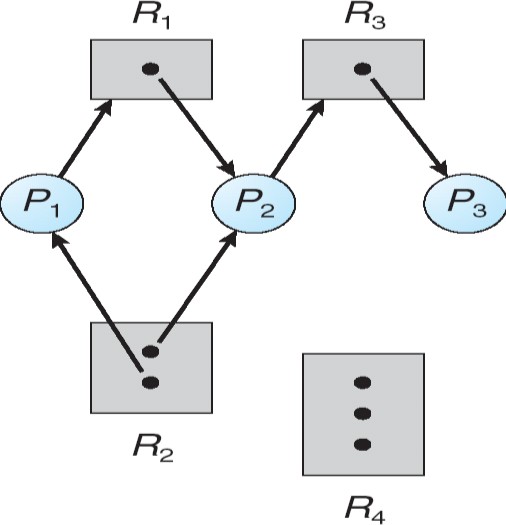
*Rj*

*Pi* is holding an instance of *Rj*

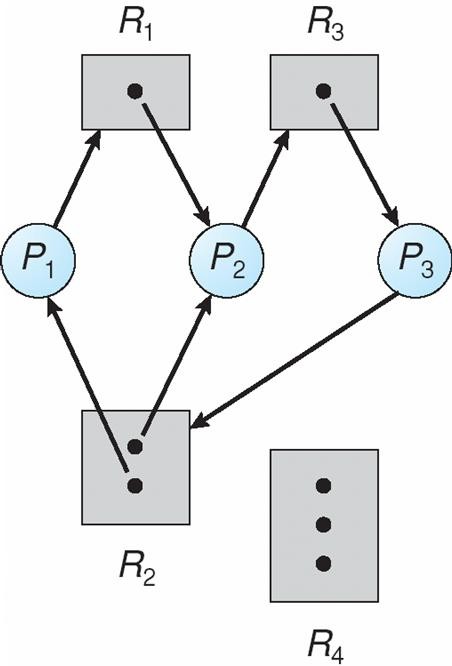
*Pi*

*Rj*

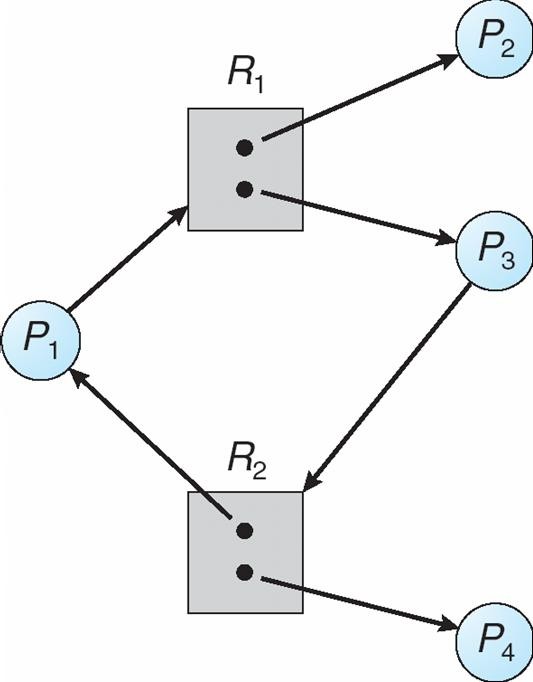
**Example of a Resource Allocation Graph**

****

**Resource Allocation Graph With A Deadlock**

****

**Graph With A Cycle But No Deadlock**



**Basic Facts**

If graph contains no cycles Þ no deadlocknIf graph contains a cycle Þlif only one instance per resource type, then deadlock

if several instances per resource type, possibility of deadlock

### Methods for Handling Deadlocks

Ensure that the system will *never* enter a deadlock statenAllow the system to enter a deadlock state and then recovernIgnore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX

### Deadlock Prevention

Restrain the ways request can be made

**Mutual Exclusion** – not required for sharable resources; must hold for nonsharable resources

**Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources

Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none

Low resource utilization; starvation possible

##### No Preemption –

If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released

Preempted resources are added to the list of resources for which the process is waiting

Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting

**Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

### Deadlock Avoidance

Requires that the system has some additional *a priori* information available

Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need

The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition

Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes

### Safe State

When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state

System is in safe state if there exists a sequence <*P1, P2, …, Pn*> of ALL the processes is the systems such that for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the *Pj*, with *j* < *i*nThat is:

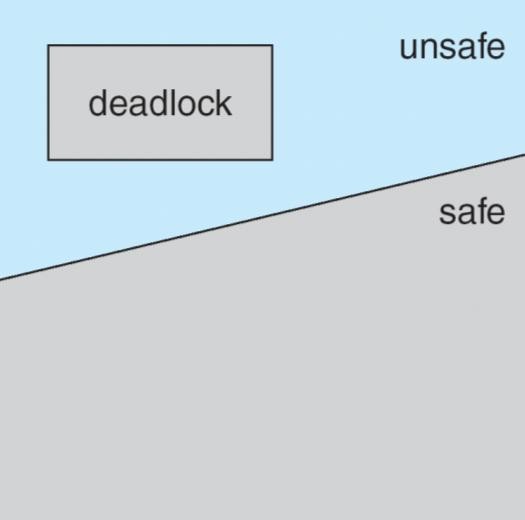
If Pi resource needs are not immediately available, then *Pi* can wait until all *Pj* have finished

When *Pj* is finished, *Pi* can obtain needed resources, execute, return allocated resources, and terminate When *Pi* terminates, *Pi* +1 can obtain its needed resources, and so on

### Basic Facts

If a system is in safe state Þ no deadlocksnIf a system is in unsafe state Þ possibility of deadlocknAvoidance Þ ensure that a system will never enter an unsafe state.

### Safe, Unsafe , Deadlock State



**Avoidance algorithms**

Single instance of a resource type

Use a resource-allocation graph

Multiple instances of a resource type

Use the banker’s algorithm in cost factor

**CHAPTER VII**

**FILE SYSTEM INTERFACE**

**The Concept Of a File**

A file is a named collection of related information that is recorded on secondary storage. The information in a file is defined its creator. Many different types of information may be stored in a file.

**File attributes:-**

A file is named and for the user’s convince is referred to by its name. A name is

usually a string of characters. One user might create file, where as another user might edit that file by specifying its name. There are different types of attributes.

1. **name:-** the name can be in the human readable form.
2. **type:-** this information is needed for those systems that support different types.
3. **location:-** this information is used to a device and to the location of the file on that device.
4. **size:-** this indicates the size of the file in bytes or words.
5. **protection:-**

**6) time,date, and user identifications:-**

the information about all files is kept in the directory structure, that also resides on secondary storage.

**File operations:-**

**Creating a file:-**

Two steps are necessary to create a file first, space in the file system must be found for the file. Second , an entry for the new file must be made in the directory. The directory entry records the name of the file and the location in the system.

**Writing a file:-**

To write a file give the name of the file, the system search the directory to find the location of the file. The system must keep the *writer* pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.

**Reading a file:-** to read from a file, specifies the name of the file and directory is search for the associated directory entry, and the system needs to keep *read* pointer to the location in the file where the next read is to take place. Once the read has taken place, read pointer is updated.

**Repositioning with in a file:-**

The directory is searched for the appropriate entry and the current file position is set to given value. this is also known as a file seek.

**Deleting a file:-** to delete a file , we search the directory for the name file. Found that file in the directory entry, we release all file space and erase the directory entry.

**Truncate a file:-** this function allows all attributes to remain unchanged(except for file length) but for the file to be reset to length zero.

**Appending:-** add new information to the end of an existing file .

**Renaming:-** give new name to an existing file.

**Open a file:-**if file need to be used, the first step is to open the file, using the *open*

system call.

**Close:-** close is a system call used to terminate the use of an already used file.

**File Types:-**

A common technique for implementing file type is to include the type as part of the file name. The name is split in to two parts

1. the name 2) and an extension .

the system uses the extension to indicate the type of the file and the type of operations that can be done on that file.

* 1. **ACCESSMETHODS:-**

**There are** several ways that the information in the file can be accessed.

**1)sequential method 2) direct access method 3) other access methods.**

1)sequential access method:-

the simplest access method is S.A. information in the file is processed in order, one after the other. the bulk of the operations on a file are reads & writes. It is based on a tape model of a file. Fig 10.3

2)Direct access:- or relative access:-

a file is made up of fixed length records, that allow programs to read and write record rapidly in no particular order. For direct access, file is viewed as a numbered sequence of blocks or records. A direct access file allows, blocks to be read & write. So we may read block15, block 54 or write block10. there is no restrictions on the order of reading or writing for a direct access file. It is great useful for immediate access to large amount of information.

The file operations must be modified to include the block number as a parameter. We have read n, where n is the block number.

3)other access methods:-

the other access methods are based on the index for the file. The indexed contain pointers to the various blocks. To find an entry in the file , we first search the index and then use the pointer to access the file directly and to find the desired entry. With large files. The index file itself, may become too large to be kept in memory. One solution is to create an index for the index file. The primary index file would contain pointers to secondary index files which would point to the actual data iteams

* 1. **Directory Structures:-**

operations that are be on a directory (read in text book)

**single level directory:-**

the simple directory structure is the single level directory. All files are contained in

the same directory. Which is easy to understand. Since all files are in same directory, they must have unique names.

In a single level directory there is some limitations. When the no.of files

increases or when there is more than one user some problems can occurs. If the no.of files increases, it becomes difficult to remember the names of all the files. FIG 10.7

**Two-level directory:-**

The major disadvantages to a single level directory is the confusion of file names between different users. The standard solution is to create separate directory for each user.

In 2-level directory structure, each user has her own user file directory(ufd). Each ufd has a similar structure, the user first search the master file directory . the mfd is indexed by user name and each entry point to the ufd for that user.fig 10.8

To create a file for a user, the O.S search only that user’s ufd to find whether another file of that name exists. To delete a file the O.S only search to the local ufd and it can not accidentally delete another user’s file that has the same name.

This solves the name collision problem, but it still have another. This is disadvantages when the user wants to cooperate on some task and to access one another’s file . some systems simply do not allow local user files to be accessed by other user.

Any file is accessed by using path name. Here the user name and a file name defines a path name.

Ex:- user1/ob

In MS-DOS a file specification is C:/directory name/file name **Tree structured directory:-**

This allows users to create their own subdirectories and to organize their files accordingly. here the tree hasa root directory. And every file in the system has a unique path name. A path name is the path from the root, through all the subdirectories to a specified file.FIG 10.9.

A directory contains a set of subdirectories or files. A directory is simply another file, but it is treated in a special way. Here the path names can be of two types. 1)absolute path and 2) relative path.

An absolute path name begins at the root and follows a path down to the specified file, giving the directory name on the path.

Ex:- root/spell/mail/prt/first.

A relative pathname defines a path from the current directory ex:- prt/first is relative path name.

**A cyclic- graph directory:-**

Consider two programmers who are working on a joint project. The files associated with that project can be stored in a sub directory , separating them from

other projects and files of the two programmers. The common subdirectory is shared by both programmers. A shared directory or file will exist in the file system in two places at once. Notice that a shared file is not the same as two copies of the file with two copies, each programmer can view the copy rather than the original but if one programmer changes the file the changes will not appear in the others copy with a shared file there is only one actual file, so any changes made by one person would be immediately visible to the other.

A tree structure prohibits the sharing of files or directories. An acyclic graph allows directories to have shared subdirectories and files

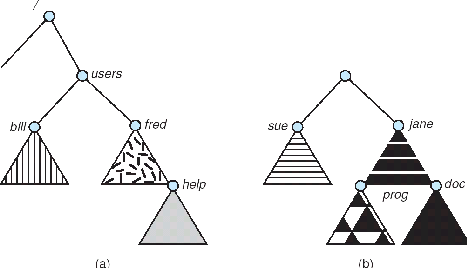
FIG 10.10 . it is more complex and more flexiable. Also several problems may occurs at the traverse and deleting the file contents.

### File System Mounting

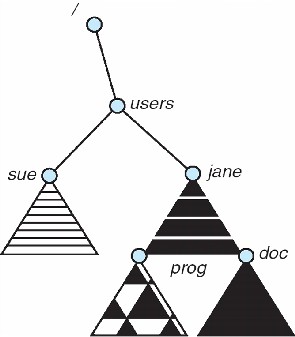
A file system must be **mounted** before it can be accessed

A unmounted file system (i.e. Fig. 11-11(b)) is mounted at a **mount point**

##### (a)Existing. (b) Unmounted Partition

* + 1. 

**Mount Point**



* 1. **File Sharing**

Sharing of files on multi-user systems is desirablenSharing may be done through a **protection** schemenOn distributed systems, files may be shared

across a networknNetwork File System (NFS) is a common distributed filesharing method

##### File Sharing – Multiple Users

**User IDs** identify users, allowing permissions and protections to be perusern**Group IDs** allow users to be in groups, permitting group access rights

##### File Sharing – Remote File Systems

nUses networking to allow file system access between systems

lManually via programs like FTP

lAutomatically, seamlessly using **distributed file systems**

lSemi automatically via the **world wide web**

n**Client-server** model allows clients to mount remote file systems from servers

lServer can serve multiple clients

lClient and user-on-client identification is insecure or complicated

l**NFS** is standard UNIX client-server file sharing protocol

l**CIFS** is standard Windows protocol

lStandard operating system file calls are translated into remote calls nDistributed Information Systems **(distributed naming services)** such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

##### File Sharing – Failure Modes

Remote file systems add new failure modes, due to network failure, server failure

Recovery from failure can involve state information about status of each remote request

Stateless protocols such as NFS include all information in each request, allowing easy recovery but less security

**File Sharing – Consistency Semantics**

n**Consistency semantics** specify how multiple users are to access a shared file simultaneously

lSimilar to Ch 7 process synchronization algorithms

Tend to be less complex due to disk I/O and network latency (for remote file systems

lAndrew File System (AFS) implemented complex remote file sharing semantics

lUnix file system (UFS) implements:

Writes to an open file visible immediately to other users of the same open file

Sharing file pointer to allow multiple users to read and write concurrently

lAFS has session semantics

Writes only visible to sessions starting after the file is closed

### Protection

File owner/creator should be able to control:

lwhat can be done

lby whomnTypes of access

l**Read**

l**Write**

l**Execute**

l**Append**

l**Delete**

l**List**

**Protection:-**]

When the information is kept in the system the major worry is its protection from the both physical damage (Reliability) and improper access(Protection).

The reliability is generally provided by duplicate copies of files.

The protection can be provided in many ways . for some single system user, we might provide protection by physically removing the floppy disks . in a multi-user systems, other mechanism are needed.

1) **types of access:-**

if the system do not permit access to the files of other users, protection is not needed. Protection mechanism provided by controlling accessing. This can be provided by types of file access. Access is permitted or denied depending on several factors. Suppose we mentioned read that file allows only for read .

Read:- read from the file. Write:- write or rewrite the file.

Execute:- load the file in to memory and execute it.

Append:- write new information at the end of the file.

Delete:- delete the file and free its space for possible reuse.

**FILE SYSTEM IMPLEMENTATION**

* 1. **File allocation methods:-**

There are 3 major methods of allocating disk space.

1) **Contiguous allocation:-**

1. The contiguous allocation method requires each file to occupy a set of contiguous block on the disk.
2. Contiguous allocation of a file is defined by the disk address and length of the first block. If the file is ‘n’ block long and starts at location ‘b’ , then it occupies blocks b,b+1,b+2,…..,b+n-1;
3. The directory entry for each file indicates the address of the starting block and length of the area allocated for this file. Fig **11.3**
4. Contiguous allocation of file is very easy to access. For the *sequential access* , the file system remembers the disk address of the last block referenced and, when necessary read next block. For *direct access* to block ‘i’ of a file that starts at block ‘b’ , we can immediately access block b+i. Thus both sequential and direct access can be supported by contagious allocation.
5. One difficulty with this method is finding space for a new file.
6. Also there are many problems with this method
7. **external fragmentation:-** files are allocated and deleted , the free disk space is broken in to little pieces. The E.F exists when free space is broken in to chunks(large piece) and these chunks are not sufficient for a request of new file.

There is a solution for E.F i.e compaction. All free space compact in to one contiguous space. But the cost of compaction is time.

1. Another problem is determining how much space is needed for a file. When file is created the creator must specifies the size of

that file. This becomes to big problem. Suppose if we allocate too little space to a file , some times it may not sufficient.

Suppose if we allocate large space some times space is wasted.

1. Another problem is if one large file is deleted, that large space is becomes to empty. Another file is loaded in to that space whose size is very small then some space is wasted . that wastage of space is called internal fragmentation.

2) **Linked allocation:-**

1. Linked allocation solves all the problems of contagious allocation. With linked allocation , each file is a linked list of disk blocks, the disk block may be scattered any where on the disk.
2. The directory contains a pointer to the first and last blocks of the file. **Fig11.4** Ex:- a file have five blocks start at block 9, continue at block 16,then block 1, block 10 and finally block 25. each block contains a ponter to the next block. These pointers are not available to the user.
3. To create a new file we simply creates a new entry in directory. With linked allocation, each directory entry has a pointer to the first disk block of the file.
4. There is no external fragmentation with linked allocation. Also there is no need to declare the size of a file when that file is created. A file can continue to grows as long as there are free blocks.
5. But it have disadvantage. The major problem is that it can be used only for sequential access-files.
6. To find the I th block of a file , we must start at the beginning of that file, and follow the pointers until we get to the I th block. It can not support the direct access.
7. Another disadvantage is it requires space for the pointers. If a pointer requires 4 bytes out of 512 byte block, then 0.78% of disk is being used for pointers, rather than for information.
8. The solution to this problem is to allocate blocks in to multiples, called clusters and to allocate the clusters rather than blocks.
9. Another problem is reliability. The files are linked together by pointers scattered all over the disk what happen if a pointer were lost or damaged. **FAT( file allocation table):-**

An important variation on the linked allocation method is the use of a file allocation table.

The table has one entry for each disk block, and is indexed by block number. The FAT is used much as is a linked list.

The directory entry contains the block number of the first block of the file. The table entry contains the block number then contains the block number of the next block in the file. This chain continuous until the last block, which has a special end of file values as the table entry. Unused blocks are indicated by a ‘0’ table value. Allocation a new block to a file is a simple. First finding the first 0-value table entry, and replacing the previously end of file value with the address of the new block. The 0 is then replaced with end of file value.

**Fig 11.5**

**3)Indexed allocation:-**

1. linked allocation solves the external fragmentation and size declaration problems of contagious allocation. How ever in the absence of a FAT , linked allocation can not support efficient direct access.
2. The pointers to the blocks are scattered with the blocks themselves all over the disk and need to be retrieved in order.
3. Indexed allocation solves this problem by bringing all the pointers together in to one location i.e *the index block.*
4. Each file has its own index block ,which is an array of disk block addresses. The I th entry in the index block points to the ith block of the file.
5. The directory contains the address of the index block. **Fig 11.6**

To read the ith block we use the pointer in the ith index block entry to find and read the desired block.

1. When the file is created, all pointers in the index block are set to nil. When the ith block is first written, a block is obtained from the free space manager, and

its address is put in the ith index block entry.

1. It supports the direct access with out suffering from external fragmentation, but it suffer from the wasted space. The pointer overhead of the index block is generally greater than the pointer over head of linked allocation.

**CHAPTER VIII**

**MASS-STORAGE STRUCTURE**

**Mass-Storage Systems**

nDescribe the physical structure of secondary and tertiary storage devices and the resulting effects on the uses of the devicesnExplain the performance characteristics of mass-storage devicesnDiscuss operating-system services provided for mass storage, including RAID and HSM

### Overview of Mass Storage Structure

Magnetic disks provide bulk of secondary storage of modern computers Drives rotate at 60 to 200 times per second

Transfer rate is rate at which data flow between drive and computer

Positioning time (random-access time) is time to move disk arm to desired cylinder (seek time) and time for desired sector to rotate under the disk head (rotational latency) Head crash results from disk head making contact with the disk surface

That’s bad

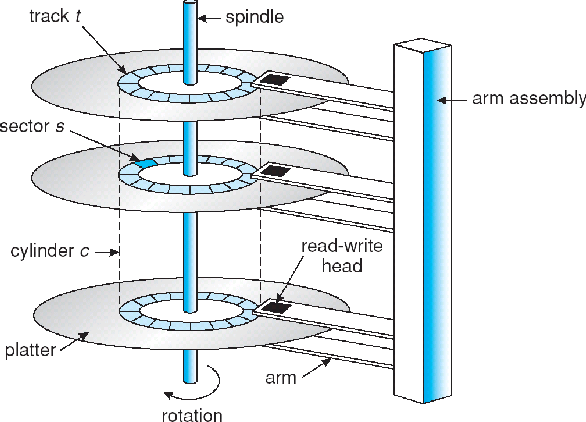
Disks can be removable

Drive attached to computer via I/O bus

Busses vary, including EIDE, ATA, SATA, USB, Fibre Channel, SCSI

Host controller in computer uses bus to talk to disk controller built into drive or storage array

##### Moving-head Disk Mechanism



Magnetic tape

Was early secondary-storage medium

Relatively permanent and holds large quantities of data

Access time slow

Random access ~1000 times slower than disk

Mainly used for backup, storage of infrequently-used data, transfer medium between systems

Kept in spool and wound or rewound past read-write head

Once data under head, transfer rates comparable to disk

20-200GB typical storage

Common technologies are 4mm, 8mm, 19mm, LTO-2 and SDLT

### Disk Structure

Disk drives are addressed as large 1-dimensional arrays of logical

blocks, where the logical block is the smallest unit of transfernThe 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially

Sector 0 is the first sector of the first track on the outermost cylinder

Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost

**Disk Attachment**

Host-attached storage accessed through I/O ports talking to I/O busses SCSI itself is a bus, up to 16 devices on one cable, SCSI initiator requests operation and SCSI targets perform tasks

Each target can have up to 8 logical units (disks attached to device controller FC is high-speed serial architecture

Can be switched fabric with 24-bit address space – the basis of storage area networks (SANs) in which many hosts attach to many storage units

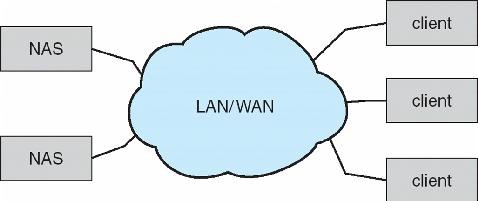
Can be arbitrated loop (FC-AL) of 126 devices

##### Network-Attached Storage

Network-attached storage (NAS) is storage made available over a network rather than over a local connection (such as a bus)

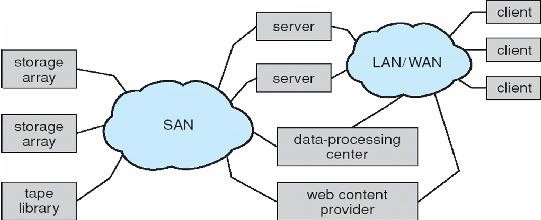
NFS and CIFS are common protocols

Implemented via remote procedure calls (RPCs) between host and storage New iSCSI protocol uses IP network to carry the SCSI protocol



##### Storage Area Network

Common in large storage environments (and becoming more common) Multiple hosts attached to multiple storage arrays – flexible



### Disk Scheduling

The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth

Access time has two major components

Seek time is the time for the disk are to move the heads to the cylinder containing the desired sector

Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head

Minimize seek time

Seek time » seek distance

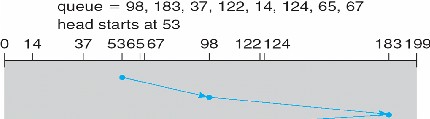
Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer Several algorithms exist to schedule the servicing of disk I/O requests nWe illustrate them with a request queue (0-199)

98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

##### FCFS

Illustration shows total head movement of 640 cylinders



##### SSTF

Selects the request with the minimum seek time from the current head position SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests

nIllustration shows total head movement of 236 cylinders

##### SCAN

The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.nSCAN algorithm Sometimes called the elevator algorithm

Illustration shows total head movement of 208 cylinders

##### C-SCAN

Provides a more uniform wait time than SCAN

The head moves from one end of the disk to the other, servicing requests as it goes

When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip

Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

##### C-LOOK

Version of C-SCAN

Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk **Selecting a Disk-Scheduling Algorithm**

SSTF is common and has a natural appeal

SCAN and C-SCAN perform better for systems that place a heavy load on the disk

Performance depends on the number and types of requests

Requests for disk service can be influenced by the file-allocation method The disk-scheduling algorithm should be written as a separate module of the

operating system, allowing it to be replaced with a different algorithm if necessary Either SSTF or LOOK is a reasonable choice for the default algorithm

##### Disk Management

Low-level formatting, or physical formatting — Dividing a disk into sectors that the disk controller can read and write

To use a disk to hold files, the operating system still needs to record its own data structures on the disk

Partition the disk into one or more groups of cylinders Logical formatting or “making a file system”

To increase efficiency most file systems group blocks into clusters

Disk I/O done in blocks

File I/O done in clusters Boot block initializes system

The bootstrap is stored in ROM Bootstrap loader program

Methods such as sector sparing used to handle bad blocks

### Booting from a Disk in Windows 2000

CHAPTER 9

SECURITY AND PROTECTION

**OVERVIEW OF SECURITY AND PROTECTION SYSTEM**

The operating system security measures we discussed in this chapter are in common use in companies around the globe. The various steps we went over when we discussed hardening operating systems are usually implemented by any competent organization that is building servers for deployment, particularly in cases where these servers will be Internet facing. Depending on the organization in question and its security posture, we may or may not find such measures to have been carried out on client machines. Although such basic hardening measures are a way in which we can increase our security with relative ease, we do so at the potential expense of ease of use and productivity.

The use of anti-malware tools, HIDS, and software firewalls is also rather ubiquitous in many organizations of any appreciable size. We will commonly see anti-malware tools installed on proxy servers filtering Web and mail traffic as it enters from the Internet. Without such tools in place, even if we have very strong border security in the form of firewalls and IDS, when something does manage to make it through these measures, it will cause great havoc on our internal networks.

The tools we discussed in this chapter and in Chapter 10 are some of the staples of the security industry. A huge number and variety of such tools might be used in any given environment for any number of uses, but taking the time to learn some of those that are more commonly seen, such as Nmap and Nessus, will be helpful to anyone entering the security field. We may see larger and costlier commercial tools at use in a given environment, but they will often be in use side by side with the old standbys.

**Operating System Security**

The topic of operating system security is a very lengthy discussion owing to not only the vast amount of technical details but also the number of operating systems and versions. Since our focus is SAP, we'll cover items directly relevant to SAP and Solaris operating system security.

Changing Some Defaults

Let's start by addressing system defaults. When the operating system is installed, there are some default configurations that should be changed. Below the Solaris operating systems there is a firmware called Openboot. The Openboot resides on the Programmable Read Only Memory (PROM) and starts every time the systems start and even determines the boot behavior at startup. Needless to say, we must protect the Openboot from unauthorized access. By default the Openboot is without a password and is set at the lowest level of security. Openboot has three levels of security: *none, command*, and *full*.

The security level *none* requires no password and allows full access to the Openboot commands. The security level *command* requires a password for all commands other than *boot* and *go*. *boot* is the Openboot command to initiate the boot process utilizing any options. *go* resumes processing after halting the system. Security level *full* requires a password for all commands other than *go*. It is best to set the security to command or better with a specified password.

SECURITY

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it. So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc. We're going to discuss following topics in this chapter.

* Authentication
* One Time passwords
* Program Threats
* System Threats
* Computer Security Classifications

Authentication

Authentication refers to identifying each user of the system and associating the executing programs with those users. It is the responsibility of the Operating System to create a protection system which ensures that a user who is running a particular program is authentic. Operating Systems generally identifies/authenticates users using following three ways −

* **Username / Password** − User need to enter a registered username and password with Operating system to login into the system.
* **User card/key** − User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.
* **User attribute - fingerprint/ eye retina pattern/ signature** − User need to pass his/her attribute via designated input device used by operating system to login into the system.

One Time passwords

One-time passwords provide additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used, then it cannot be used again. One-time password are implemented in various ways.

* **Random numbers** − Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding to few alphabets randomly chosen.
* **Secret key** − User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id which is to be generated every time prior to login.
* **Network password** − Some commercial applications send one-time passwords to user on registered mobile/ email which is required to be entered prior to login.

**Program Threats**

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it is known as **Program Threats**. One of the common example of program threat is a program installed in a computer which can store and send user credentials via network to some hacker. Following is the list of some well-known program threats.

* **Trojan Horse** − Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources.
* **Trap Door** − If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door.
* **Logic Bomb** − Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect.
* **Virus** − Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generally a small code embedded in a program. As user accesses the program, the virus starts getting embedded in other files/ programs and can make system unusable for user

**System Threats**

System threats refers to misuse of system services and network connections to put user in trouble. System threats can be used to launch program threats on a complete network called as program attack. System threats creates such an environment that operating system resources/ user files are misused. Following is the list of some well-known system threats.

* **Worm** − Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worms processes can even shut down an entire network.
* **Port Scanning** − Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system.
* **Denial of Service** − Denial of service attacks normally prevents user to make legitimate use of the system. For example, a user may not be able to use internet if denial of service attacks browser's content settings.

**Computer Security Classifications**

As per the U.S. Department of Defense Trusted Computer System's Evaluation Criteria there are four security classifications in computer systems: A, B, C, and D. This is widely used specifications to determine and model the security of systems and of security solutions. Following is the brief description of each classification.

|  |  |
| --- | --- |
|  |  |
| 1 | **Type A**  Highest Level. Uses formal design specifications and verification techniques. Grants a high degree of assurance of process security. |
| 2 | **Type B**  Provides mandatory protection system. Have all the properties of a class C2 system. Attaches a sensitivity label to each object. It is of three types.   * **B1** − Maintains the security label of each object in the system. Label is used for making decisions to access control. * **B2** − Extends the sensitivity labels to each system resource, such as storage objects, supports covert channels and auditing of events. * **B3** − Allows creating lists or user groups for access-control to grant access or revoke access to a given named object. |
| 3 | **Type C**  Provides protection and user accountability using audit capabilities. It is of two types.   * **C1** − Incorporates controls so that users can protect their private information and keep other users from accidentally reading / deleting their data. UNIX versions are mostly Cl class. * **C2** − Adds an individual-level access control to the capabilities of a Cl level system. |

### Memory Protection

Memory protection prevents one process from affecting the confidentiality, integrity, or availability of another. This is a requirement for secure multiuser (ie, more than one user logged in simultaneously) and multitasking (ie, more than one process running simultaneously) systems.

#### Process isolation

Process isolation is a logical control that attempts to prevent one process from interfering with another. This is a common feature among multiuser operating systems such as Linux, UNIX, or recent Microsoft Windows operating systems. Older operating systems such as MS-DOS provide no process isolation, which means a crash in any MS-DOS application could crash the entire system.

#### Hardware segmentation

Hardware segmentation takes process isolation one step further by mapping processes to specific memory locations. This provides more security than logical process isolation alone.

#### Virtual memory

Virtual memory provides virtual address mapping between applications and hardware memory. Virtual memory provides many functions, including multitasking (multiple tasks executing at once on one CPU), swapping, and allowing multiple processes to access the same shared library in memory, among others.

##### Swapping and paging

Swapping uses virtual memory to copy contents of primary memory (RAM) to or from secondary memory (not directly addressable by the CPU, on disk). Swap space is often a dedicated [disk partition](https://www.sciencedirect.com/topics/computer-science/disk-partition) that is used to extend the amount of available memory. If the kernel attempts to access a page (a fixed-length block of memory) stored in swap space, a page fault occurs, which means that the page is not located in RAM and the page is “swapped” from disk to RAM.

##### Basic input/output system

The IBM PC-compatible basic input/output system (BIOS) contains code in firmware that is executed when a PC is powered on. It first runs the [*power-on self-test*](https://www.sciencedirect.com/topics/computer-science/power-on-self-test) (POST), which performs basic tests, including verifying the integrity of the BIOS itself, testing the memory, and identifying system devices, among other tasks. Once the POST process is complete and successful, it locates the boot sector (for systems that boot off disks), which contains the machine code for the [operating system kernel](https://www.sciencedirect.com/topics/computer-science/operating-system-kernel). The kernel then loads and executes, and the operating system boots up.

#### WORM storage

WORM (write once, read many) storage, like its name suggests, can be written to once and read many times. It is often used to support records retention for legal or regulatory compliance. WORM storage helps assure the integrity of the data it contains; there is some assurance that it has not been and cannot be altered, short of destroying the media itself.

**AUTHENTICATION**

Authentication is the process of determining whether someone or something is, in fact, who or what it declares itself to be. Authentication technology provides access control for systems by checking to see if a user's credentials match the credentials in a database of authorized users or in a data authentication server.

Users are usually identified with a user ID, and authentication is accomplished when the user provides a credential, for example a password, that matches with that user ID. Most users are most familiar with using a password, which, as a piece of information that should be known only to the user, is called a knowledge [authentication factor](https://searchsecurity.techtarget.com/definition/authentication-factor). Other authentication factors, and how they are used for two-factor or [multifactor authentication](https://searchsecurity.techtarget.com/definition/multifactor-authentication-MFA) (MFA), are described below.

### Authentication in cyber security

Authentication is important because it enables organizations to keep their networks secure by permitting only authenticated users (or processes) to access its protected resources, which may include computer systems, networks, databases, websites and other network-based applications or services.

Once authenticated, a user or process is usually subjected to an [authorization](https://searchsoftwarequality.techtarget.com/definition/authorization) process as well, to determine whether the authenticated entity should be permitted access to a protected resource or system. A user can be authenticated but fail to be given access to a resource if that user was not granted permission to access it.

The terms authentication and authorization are often used interchangeably; while they may often be implemented together the two functions are distinct. While authentication is the process of validating the identity of a registered user before allowing access to the protected resource, authorization is the process of validating that the authenticated user has been granted permission to access the requested resources. The process by which access to those resources is restricted to a certain number of users is called [access control](https://searchsecurity.techtarget.com/definition/access-control). The authentication process always comes before the authorization process.

### How authentication is used

[User authentication](https://searchsecurity.techtarget.com/definition/user-authentication) occurs within most human-to-computer interactions outside of guest accounts, automatically logged-in accounts and [kiosk](https://whatis.techtarget.com/definition/kiosk) computer systems. Generally, a user has to choose a username or user ID and provide a valid password to begin using a system. User authentication authorizes human-to-machine interactions in operating systems and applications, as well as both wired and wireless networks to enable access to networked and internet-connected systems, applications and resources.

Many companies use authentication to validate users who log into their websites. Without the right security measures, user data, such as credit and debit card numbers, as well as Social Security numbers, could get into the hands of cybercriminals.

Organizations also use authentication to control which users have access to corporate networks and resources, as well as to identify and control which machines and servers have access. Companies also use authentication to enable remote employees to securely access their applications and networks.

For enterprises and other large organizations, authentication may be accomplished using a [single sign-on](https://searchsecurity.techtarget.com/definition/single-sign-on) (SSO) system, which grants access to multiple systems with a single set of login credentials.

### How authentication works

During authentication, credentials provided by the user are compared to those on file in a database of authorized users' information either on the local [operating system](https://whatis.techtarget.com/definition/operating-system-OS) or through an [authentication server](https://searchsecurity.techtarget.com/definition/authentication-server). If the credentials match, and the authenticated entity is authorized to use the resource, the process is completed and the user is granted access. The permissions and folders returned define both the environment the user sees and the way he can interact with it, including hours of access and other rights such as the amount of [resource](https://searchcio.techtarget.com/definition/resource-allocation) storage space.

Traditionally, authentication was accomplished by the systems or resources being accessed; for example, a server would authenticate users using its own password system, implemented locally, using [login](https://searchsecurity.techtarget.com/definition/logon) IDs (user names) and passwords. Knowledge of the login credentials is assumed to guarantee that the user is authentic. Each user registers initially (or is registered by someone else, such as a systems administrator), using an assigned or self-declared password. On each subsequent use, the user must know and use the previously declared password.

However, the web's application protocols, HTTP and HTTPS, are [stateless](https://whatis.techtarget.com/definition/stateless), meaning that strict authentication would require end users reauthenticate each time they access a resource using HTTPS. Rather than burden end users with that process for each interaction over the web, protected systems often rely on token-based authentication, in which authentication is performed once at the start of a session. The authenticating system issues a signed authentication token to the end-user application, and that token is appended to every request from the client.

Entity authentication for systems and processes can be carried out using machine credentials that work like a user's ID and password, except the credentials are submitted automatically by the device in question. They may also use digital certificates that were issued and verified by a certificate authority as part of a public key infrastructure to authenticate an identity while exchanging information over the internet.

### Authentication factors

Authenticating a user with a user ID and a password is usually considered the most basic type of authentication, and it depends on the user knowing two pieces of information: the user ID or username, and the password. Since this type of authentication relies on just one authentication factor, it is a type of [single-factor authentication](https://searchsecurity.techtarget.com/definition/single-factor-authentication-SFA).

[Strong authentication](https://whatis.techtarget.com/definition/strong-authentication) is a term that has not been formally defined, but usually is used to mean that the type of authentication being used is more reliable and resistant to attack; achieving that is generally acknowledged to require using at least two different types of authentication factors.

An authentication factor represents some piece of data or attribute that can be used to authenticate a user requesting access to a system. An old security adage has it that authentication factors can be "something you know, something you have or something you are." These three factors correspond to the knowledge factor, the possession factor and the inherence factor. Additional factors have been proposed and put into use in recent years, with location serving in many cases as the fourth factor, and time serving as the fifth factor.

Currently used authentication factors include:

* [Knowledge factor](https://searchsecurity.techtarget.com/definition/knowledge-factor): "Something you know." The knowledge factor may be any authentication credentials that consist of information that the user possesses, including a personal identification number (PIN), a user name, a password or the answer to a secret question.
* [Possession factor](https://searchsecurity.techtarget.com/definition/possession-factor): "Something you have." The possession factor may be any credential based on items that the user can own and carry with them, including hardware devices like a security token or a mobile phone used to accept a text message or to run an authentication app that can generate a one-time password or PIN.
* [Inherence factor](https://searchsecurity.techtarget.com/definition/inherence-factor): "Something you are." The inherence factor is typically based on some form of biometric identification, including finger or thumb prints, facial recognition, retina scan or any other form of biometric data.
* Location factor: "Where you are." While it may be less specific, the location factor is sometimes used as an adjunct to the other factors. Location can be determined to reasonable accuracy by devices equipped with [GPS](https://whatis.techtarget.com/definition/GPS-tracking), or with less accuracy by checking network routes. The location factor cannot usually stand on its own for authentication, but it can supplement the other factors by providing a means of ruling out some requests. For example, it can prevent an attacker located in a remote geographical area from posing as a user who normally logs in only from home or office in the organization's home country.
* Time factor: "When you are authenticating." Like the location factor, the time factor is not sufficient on its own, but it can be a supplemental mechanism for weeding out attackers who attempt to access a resource at a time when that resource is not available to the authorized user. It may also be used together with location as well. For example, if the user was last authenticated at noon in the U.S., an attempt to authenticate from Asia one hour later would be rejected based on the combination of time and location.

Despite being used as supplemental authentication factors, user location and current time by themselves are not sufficient, without at least one of the first three factors, to authenticate a user. However, the ubiquity of smartphones is helping to ease the burdens of multifactor authentication for many users. Most smartphones are equipped with GPS, enabling reasonable confidence in confirmation of the login location; smartphone [MAC](https://searchnetworking.techtarget.com/definition/MAC-address) addresses may also be used to help authenticate a remote user, despite the fact that MAC addresses are relatively easy to spoof.

### Two-factor and multifactor authentication

Adding authentication factors to the authentication process typically improves security. Strong authentication usually refers to authentication that uses at least two factors, where those factors are of different types. The distinction is important; since both username and password can be considered types of knowledge factor, basic username and password authentication could be said to use two knowledge factors to authenticate -- however, that would not be considered a form of [two-factor authentication](https://searchsecurity.techtarget.com/definition/two-factor-authentication) (2FA). Likewise for authentication systems that rely on "security questions," which are also "something you know," to supplement user ID and passwords.

Two-factor authentication usually depends on the knowledge factor combined with either a biometric factor or a possession factor like a security token. Multifactor authentication can include any type of authentication that depends on two or more factors, but an authentication process that uses a password plus two different types of biometric would not be considered [three-factor authentication](https://searchsecurity.techtarget.com/definition/three-factor-authentication-3FA), although if the process required a knowledge factor, a possession factor and an inherence factor, it would be. Systems that call for those three factors plus a geographic or time factor are considered examples of [four-factor authentication](https://searchsecurity.techtarget.com/definition/four-factor-authentication-4FA).

### Authentication and authorization

Authorization includes the process through which an administrator grants rights to authenticated users, as well as the process of checking user account permissions to verify that the user has been granted access to those resources. The privileges and preferences granted for the authorized account depend on the user's permissions, which are either stored locally or on the authentication server. The settings defined for all these environment variables are set by an administrator.

Systems and processes may also need to authorize their automated actions within a network. Online backup services, patching and updating systems and remote monitoring systems, such as those used in telemedicine and smart grid technologies, all need to securely authenticate before they can verify that it is the authorized system involved in any interaction and not a hacker.

### Types of authentication methods

Traditional authentication depends on the use of a password file, in which user IDs are stored together with hashes of the passwords associated with each user. When logging in, the password submitted by the user is hashed and compared to the value in the password file. If the two hashes match, the user is authenticated.

This approach to authentication has several drawbacks, particularly for resources deployed across different systems. For one thing, attackers who are able to access to the password file for a system can use brute force attacks against the hashed passwords to extract the passwords. For another, this approach would require multiple authentications for modern applications that access resources across multiple systems.

ENCRYPTION

Most search engines, regardless of if they track you, encrypt your search data. This is how search engines, including Google, Yahoo and Search Encrypt, all protect their users’ information. Google, which collects [tons of user data](https://blog.searchencrypt.com/news/does-a-privacy-based-search-engine-exist/), is obligated to protect that information. [SSL encryption](https://blog.searchencrypt.com/tech/search-encrypt-uses-ssl-to-protect-user-data/) is a standard for protecting sensitive information, for search engines and other websites.

# What is Encryption?

Encryption is a process that encodes a message or file so that it can be only be read by certain people. Encryption uses an algorithm to scramble, or encrypt, data and then uses a key for the receiving party to unscramble, or decrypt, the information. The message contained in an encrypted message is referred to as plaintext. In its encrypted, unreadable form it is referred to as ciphertext.

Basic forms of encryption may be as simple as switching letters. As cryptography advanced, cryptographers added more steps, and decryption became more difficult. Wheels and gears would be combined to create complex [encryption systems](https://blog.searchencrypt.com/privacy/7-principles-of-privacy-by-design/). Computer algorithms have now replaced mechanical encryption.

# How Encryption Works

Encryption uses algorithms to scramble your information. It is then transmitted to the receiving party, who is able to decode the message with a key. There are many types of algorithms, which all involve different ways of scrambling and then decrypting information.

**RECOVERY MANAGEMENT**

One price of extensibility and distribution, as implemented in QuickSilver, is a more complicated set of failure modes, and the consequent necessity of dealing with them. In traditional operating systems, services (*e.g.*, file, display) are intrinsic pieces of the kernel. Process state is maintained in kernel tables, and the kernel contains explicit cleanup code (*e.g.*, to close files, reclaim memory, and get rid of process images after hardware or software failures). QuickSilver, however, is structured according to the client-server model, and as in many systems of its type, system services are implemented by user-level processes that maintain a substantial amount of client process state. Examples of this state are the open files, screen windows, address space, etc., belonging to a process. Failure resilience in such an environment requires that clients and servers be aware of problems involving each other. Examples of the way one would like the system to behave include having files closed and windows removed from the screen when a client terminates, and having clients see bad return codes (rather than hanging) when a file server crashes. This motivates a number of design goals:

* Properly written programs (especially servers) should be resilient to external process and machine failures, and should be able to recover all resources associated with failed entities.
* Server processes should contain their own recovery code. The kernel should not make any distinction between system service processes and normal application processes.
* To avoid the proliferation of ad-hoc recovery mechanisms, there should be a uniform system-wide architecture for recovery management.
* A client may invoke several independent servers to perform a set of logically related activities (a unit of work) that must execute atomically in the presence of failures, that is, either all the related activities should occur or none of them should. The recovery mechanism should support this.

----END------