2010-2011

Lecture Notes on Operating Systems

Lecture *1: Introduction to Operating Systems*

- An *operating system* is a program that acts as an intermediary between a user of a computer and the computer hardware.
- The purpose of an operating system is to provide an environment in which a user can execute programs. The primary goal of an operating system is thus to make the computer system *convenient* to use.
- A secondary goal is to use the computer hardware in an *efficient* manner.

Lecture *1: Introduction to Operating Systems*

- In brief, an operating system is the set of programs that controls a computer. Some examples of operating systems are UNIX, Mach, MS-DOS, MS-Windows, Windows/NT, OS/2 and MacOS.
- An operating system is an important part of almost every computer system.
- A computer system can be divided roughly into four components: the *hardware,* the *operating system,* the *application programs* and the *users* (Figure 1.1).

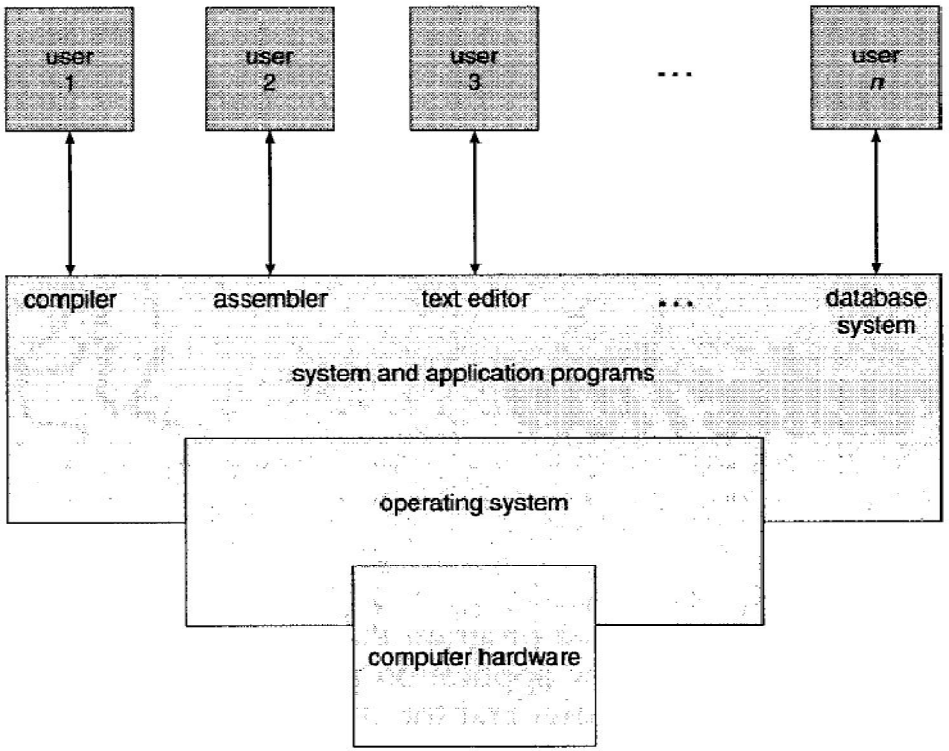


Figure 1.1 Abstract view of the components of a computer system.

Objectives of Operating Systems

- To hide details of hardware by creating abstraction.
- To allocate resources to processes (Manage resources).
- Provide a pleasant and effective user interface.

History of Operating Systems

- **The 1940's - First Generations**

  The earliest electronic digital computers had no operating systems. Machines of the time were so primitive that programs were often entered one bit at time on rows of mechanical switches (plug boards). Programming languages were unknown (not even assembly languages). Operating systems were unheard of.

- **The 1950's - Second Generation**

  By the early 1950's, the routine had improved somewhat with the introduction of punch cards. The General Motors Research Laboratories implemented the first operating systems in early 1950's for their IBM 701. The system of the 50's generally ran one job at a time.

History of Operating Systems

- **The 1960's - Third Generation**

  The systems of the 1960's were also batch processing systems, but they were able to take better advantage of the computer's resources by running several jobs at once.

- **Fourth Generation**

  With the development of LSI (Large Scale Integration) circuits, chips, operating system entered in the personal computer and the workstation age. Microprocessor technology evolved to the point that it becomes possible to build desktop computers as powerful as the mainframes of the 1970s.

Lecture 2: *Operating Systems Structure*

- ***System Components***
- ***Operating Systems Services***
- ***System Calls and System Programs***

System Components

- **Process Management**

A *process* is only ONE instant of a program in execution. There are many processes can be running the same program.

The five major activities of an operating system in regard to process management are:

- *Creation and deletion of user and system processes.*
- *Suspension and resumption of processes.*
- *A mechanism for process synchronization.*
- *A mechanism for process communication.*
- *A mechanism for deadlock handling.*

System Components

- **Main-Memory Management**

  Main-Memory is a large array of words or bytes. Each word or byte has its own address. Main memory is a repository of quickly accessible data shared by the CPU and I/O devices.

The major activities of an operating system in regard to memory-management are:

- *Keep track of which part of memory are currently being used and by whom.*
- *Decide which processes are loaded into memory when memory space becomes available.*
- *Allocate and deallocate memory space as needed.*

System Components

- **File Management**

  A file is a collected of related information defined by its creator. Computer can store files on the disk (secondary storage), which provide long term storage.

- *The creation and deletion of files.*
- *The creation and deletion of directions.*
- *The support of primitives for manipulating files and directions.*
- *The mapping of files onto secondary storage.*
- *The backup of files on stable storage media.*

System Components

- **I/O System Management**

One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user.

- **Secondary-Storage Management**

Generally speaking, systems have several levels of storage, including primary storage, secondary storage and cache storage. Instructions and data must be placed in primary storage or cache to be referenced by a running program.

System Components

- **Networking**

  A distributed system is a collection of processors that do not share memory, peripheral devices, or a clock. The processors communicate with one another through communication lines called network.

- **Protection System**

  Protection refers to mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.

- **Command Interpreter System**

  A command interpreter is an interface of the operating system with the user. The user gives commands with are executed by operating system (usually by turning them into system calls).

Operating Systems Services

- **Program Execution**

  The system must be able to load a program into memory and to run it. The program must be able to end its execution, either normally or abnormally (indicating error).

- **I/O Operations**

  A running program may require I/O. This I/O may involve a file or an I/O device.

- **File System Manipulation**

  The output of a program may need to be written into new files or input taken from some files. The operating system provides this service.

- **Error Detection**

  An error is one part of the system may cause malfunctioning of the complete system. To avoid such a situation the operating system constantly monitors the system for detecting the errors.

System Calls and System Programs

- System calls provide the interface between a process and the operating system. These calls are generally available as assembly-language instructions, and are usually listed in the manuals used by assembly-language programmers.

Lecture 3: ***Process Management***

- The operating system is responsible for the following activities in connection with process management: the creation and deletion of both user and system processes; the scheduling of processes; and the provision of mechanisms for synchronization, communication, and deadlock handling for processes.

Process, on the other hand, includes:

- Current value of Program Counter (PC)
- Contents of the processors registers
- Value of the variables
- The processes stack (SP) which typically contains temporary data such as subroutine parameter, return address, and temporary variables.
- A data section that contains global variables.

- ***Process State***

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states:

- **New State:** The process being created.
- **Running State:** A process is said to be running if it has the CPU, that is, process actually using the CPU at that particular instant.
- **Blocked (or waiting) State:** A process is said to be blocked if it is waiting for some event to happen such that as an I/O completion before it can proceed. Note that a process is unable to run until some external event happens.
- **Ready State:** A process is said to be ready if it is waiting to be assigned to a processor.
- **Terminated state:** The process has finished execution.

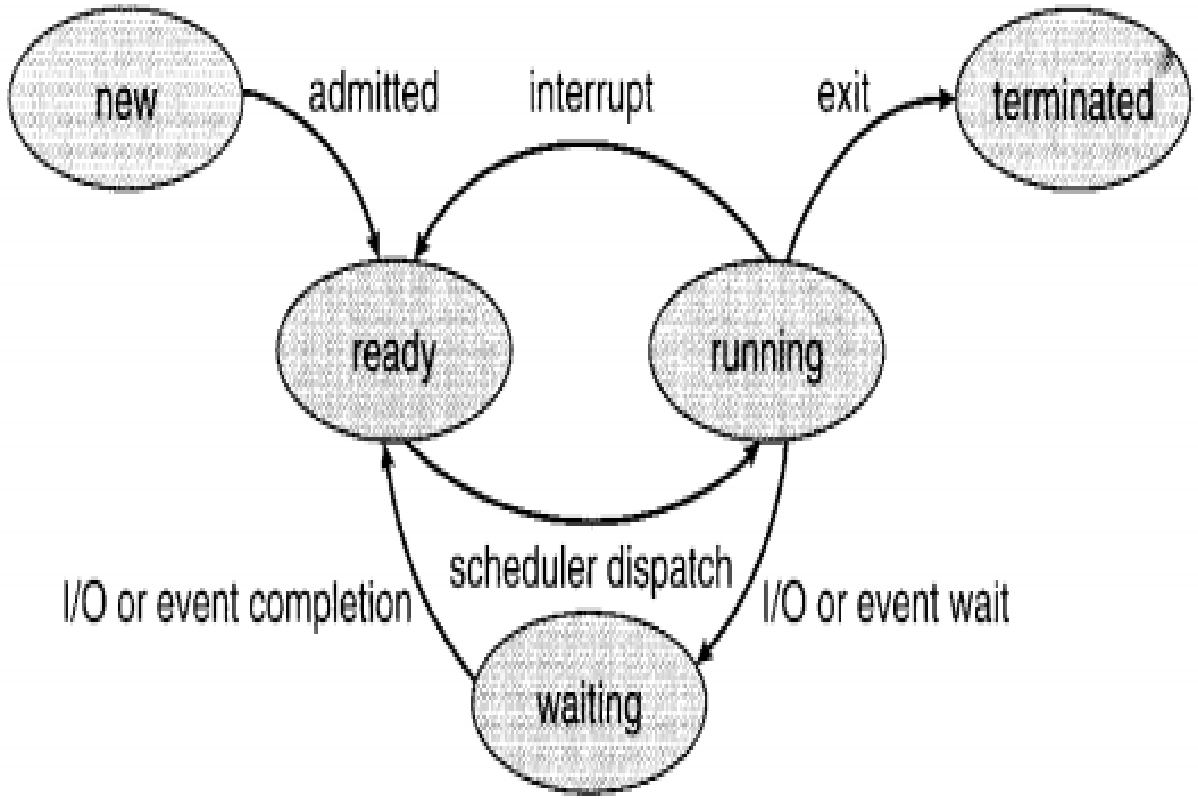


Figure : Diagram of process states.

- ***Process Control Block***
- Each process is represented in the operating system by a process control block PCS)—also called a task control block.

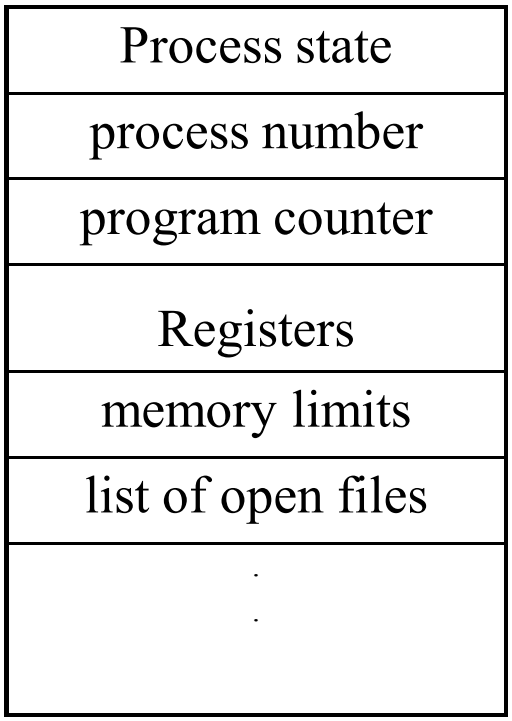


Figure : Process control block.

Lecture 4: *CPU Scheduling*

- CPU scheduling is the basis of multiprogrammed operating systems. By switching the CPU among processes, the operating system can make the computer more productive.

- **Basic Concepts**

The idea of multiprogramming is relatively simple. A process is executed until it must wait, typically for the completion of some I/O request. In a simple computer system, the CPU would then just sit idle.

Scheduling is a fundamental operating-system function. Almost all computer resources are scheduled before use.

- **CPU - I/O Burst Cycle**

  The success of CPU scheduling depends on the following observed property of processes: Process execution consists of a cycle of CPU execution and I/O wait. Processes alternate back and forth between these two states.

- **Context Switch**

  To give each process on a multiprogrammed machine a fair share of the CPU, a hardware clock generates interrupts periodically.

  This allows the operating system to schedule all processes in main memory (using scheduling algorithm) to run on the CPU at equal intervals. Each switch of the CPU from one process to another is called a context switch.

- Preemptive Scheduling

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state (for. example, I/O request, or invocation of wait for the termination of one of the child processes).

2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).

3. When a process switches from the waiting state to the ready state (for example, completion of I/O).

4. When a process terminates.

- Dispatcher

- *Switching context.*
- *Switching to user mode.*
- *Jumping to the proper location in the user program to restart that program*

- Scheduling Criteria

- Different CPU scheduling algorithms have different properties and may favor one class of processes over another. In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms.

- Many criteria have been suggested for comparing CPU scheduling algorithms.

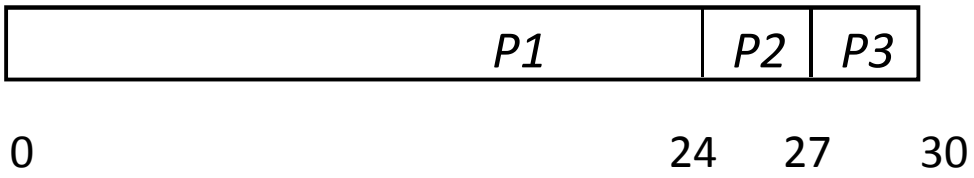
- Criteria that are used include the following:
- ***CPU utilization***.
- ***Throughput.***
- ***Turnaround time.***
- ***Waiting time***.
- ***Response time***.

Lecture 5: ***Scheduling Algorithms***

1. *First-Come, First-Served Scheduling*
2. *Shortest-Job-First Scheduling*
3. *Priority Scheduling*
4. *Round-Robin Scheduling*
5. *Multilevel Queue Scheduling*
6. *Multilevel Feedback Queue Scheduling*

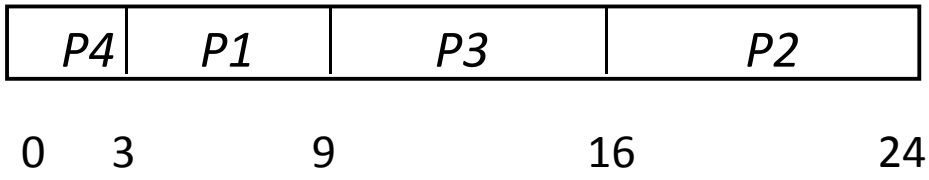
First-Come, First-Served Scheduling

Process	Burst Time
P1	24
P2	3
P3	3



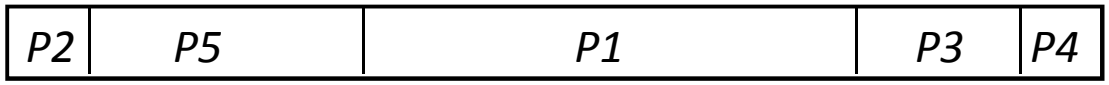
Shortest-Job-First Scheduling

Process	Burst Time
P1	6
P2	8
P3	7
P4	3



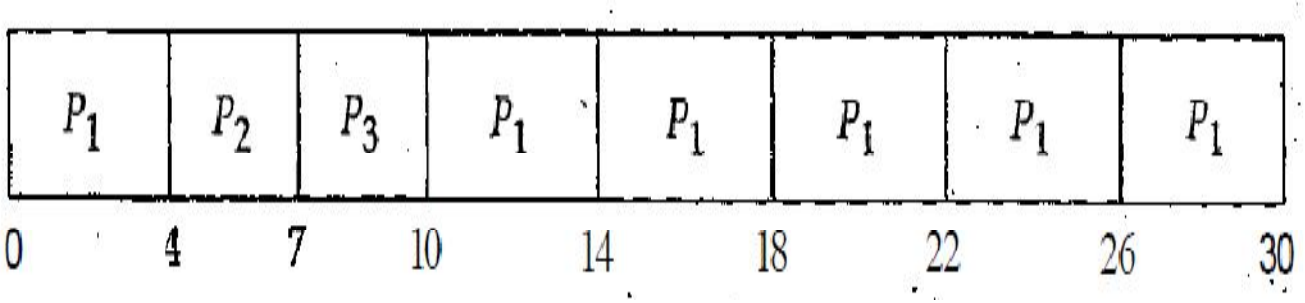
Priority Scheduling

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

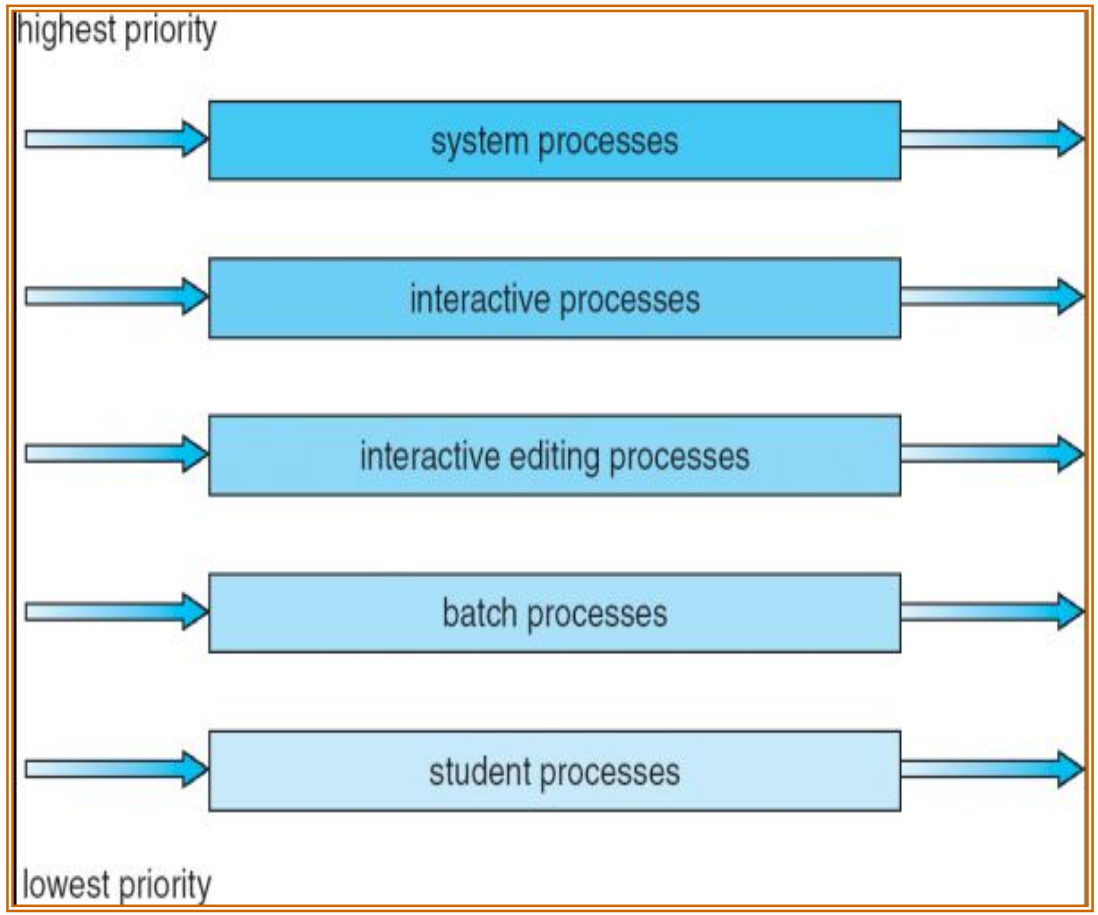


Round-Robin Scheduling

Process	Burst Time
P1	24
P2	3
P3	3



Multilevel Queue Scheduling



- In a multilevel queue scheduling processes are permanently assigned to one queues.
- The processes are permanently assigned to one another, based on some property of the process, such as
- *Memory size*
- *Process priority*
- *Process type*

- Algorithm chooses the process from the occupied queue that has the highest priority, and run that process either
- *Preemptive or*
- *Non-preemptively*

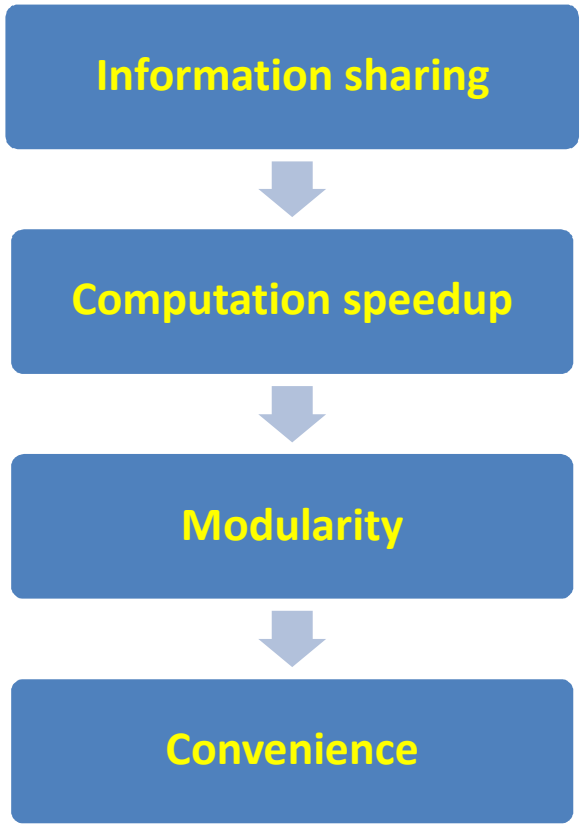
6. Process Synchronization

- A cooperating process is one that can affect or be affected by the other processes executing in the system.
- Cooperating processes may either directly share a logical address space(that is, both code and data), or be allowed to share data only through files. The former case is achieved through the use of *lightweight processes or threads*. *Concurrent access to shared data* may result in data inconsistency.
- In this lecture, we discuss various mechanisms to ensure the orderly execution of cooperating processes that share a logical address space, so that data consistency is maintained.

Cooperating Processes

- The concurrent processes executing in the operating system may be either independent processes or cooperating processes.
- A process is *independent if it cannot affect or be affected by* the other processes executing in the system.
- On the other hand, a process is *cooperating if it can affect or be affected by the other processes executing in the system.*

- There are several reasons for providing an environment that allows process cooperation:



Race condition

- When several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a *race condition*.

The Critical-Section Problem

- The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section.

- Thus, the execution of critical sections by the processes is *mutually exclusive in* time.
- The critical-section problem is to design a protocol that the processes can use to cooperate.
- Each process must request permission to enter its critical section.

- A solution to the critical-section problem must satisfy the following three requirements:

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 sections, then only those processes that are not executing in their remainder section can participate in the decision of which will enter its critical section next, and this selection cannot be postponed indefinitely.
3. **Bounded Waiting:** There exist a bound 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.

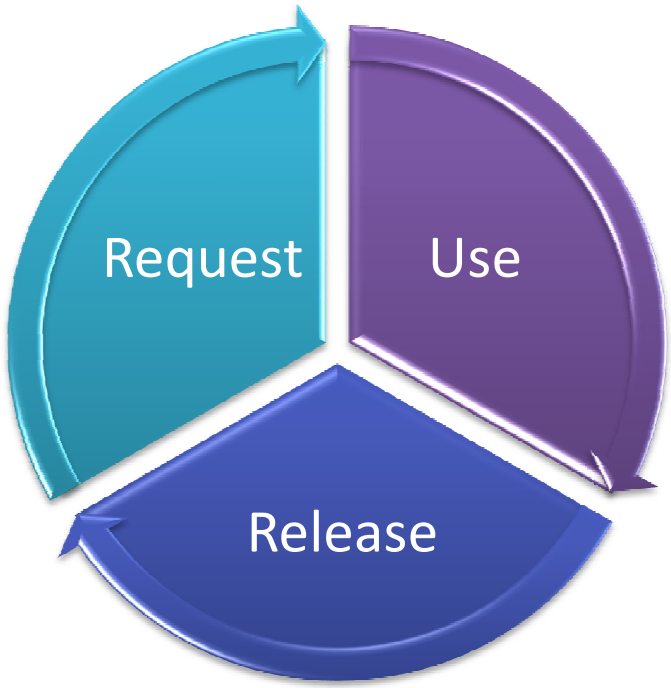
DEADLOCKS

- A process requests resources; if the resources are not available at that time, the process enters a wait state. It may happen that waiting processes will never again change state,
- because the resources they have requested are held by other waiting processes. This situation is called a *deadlock*.

- In this lecture, we describe methods that an operating system can use to deal with the deadlock problem.

Resources

- A process must request a resource before using it, and must release the resource after using it.
- A process may request as many resources as it requires to carry out its designated task.
- a process may utilize a resource in only the following sequence:



Deadlock Characterization

- In a deadlock, processes never finish executing and system resources are tied up, preventing other jobs from ever starting.
- Before we discuss the various methods for dealing with the deadlock problem, we shall describe features that characterize deadlocks.

Necessary Conditions

- A deadlock situation can arise if the following four conditions hold simultaneously in a system:
- Mutual exclusion
- Hold and wait
- No preemption
- Circular wait

Methods for Handling Deadlocks

- Principally, there are three different methods for dealing with the deadlock problem:
- We can use a protocol to ensure that the system will *never enter a deadlock state.*
- We can allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

Deadlock Prevention

- By ensuring that at least one of these conditions cannot hold, we can *prevent the occurrence of a deadlock*

- **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.
- **No Preemption –** o 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.
- **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.

Deadlock Detection

- If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In this environment, the system must provide:

- An algorithm that examines the state of the system to determine whether a deadlock has Occurred.

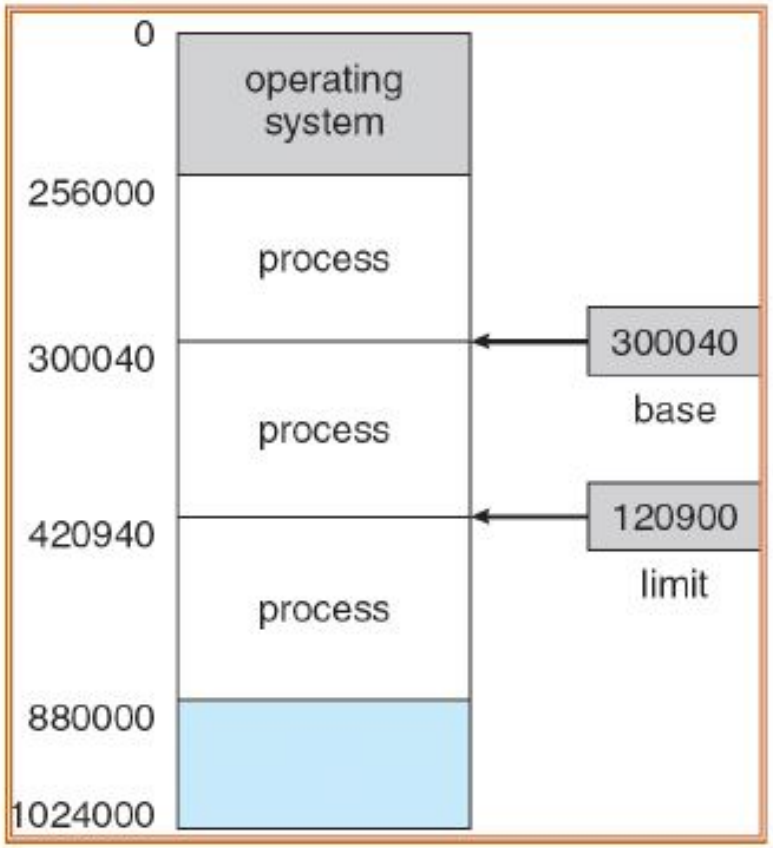
- An algorithm to recover from the deadlock

Memory Management

- 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.



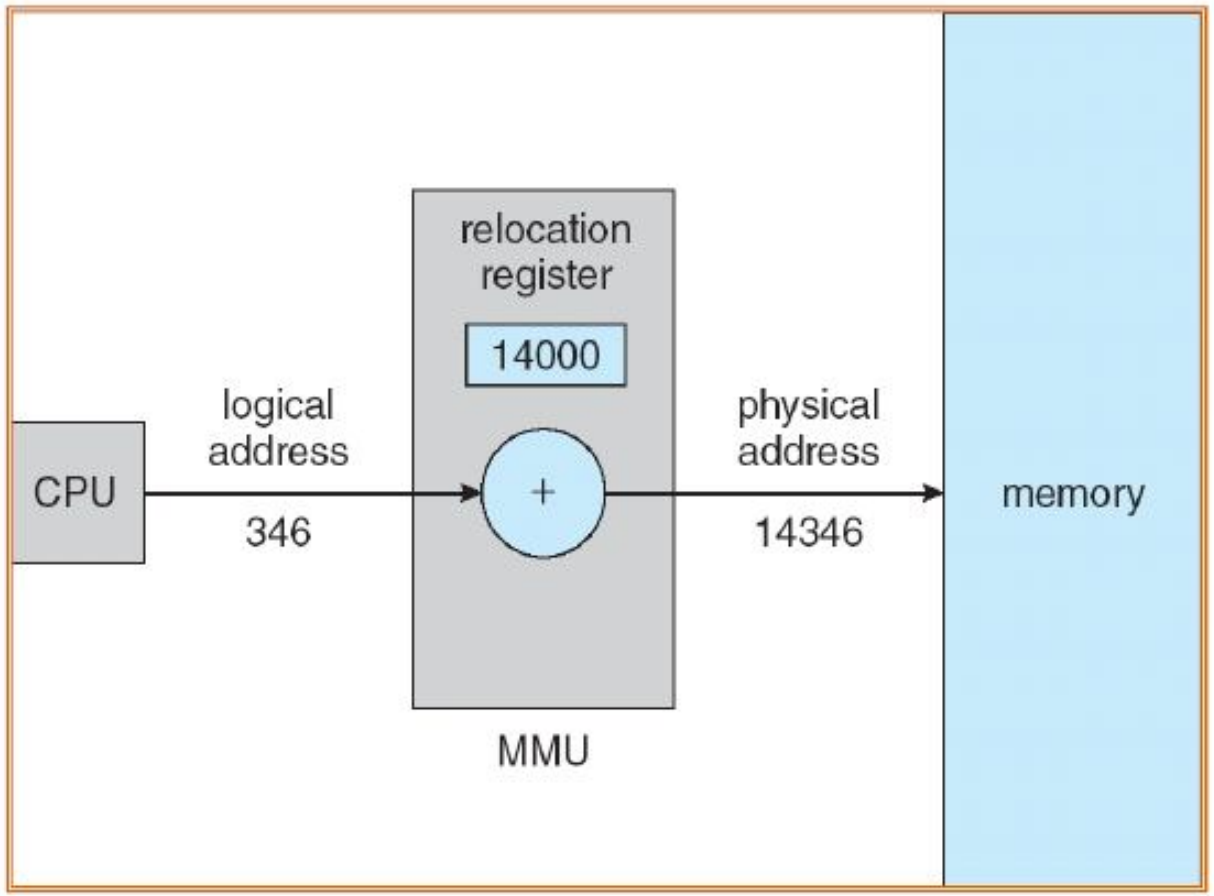
Logical vs. Physical Address Space

- 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

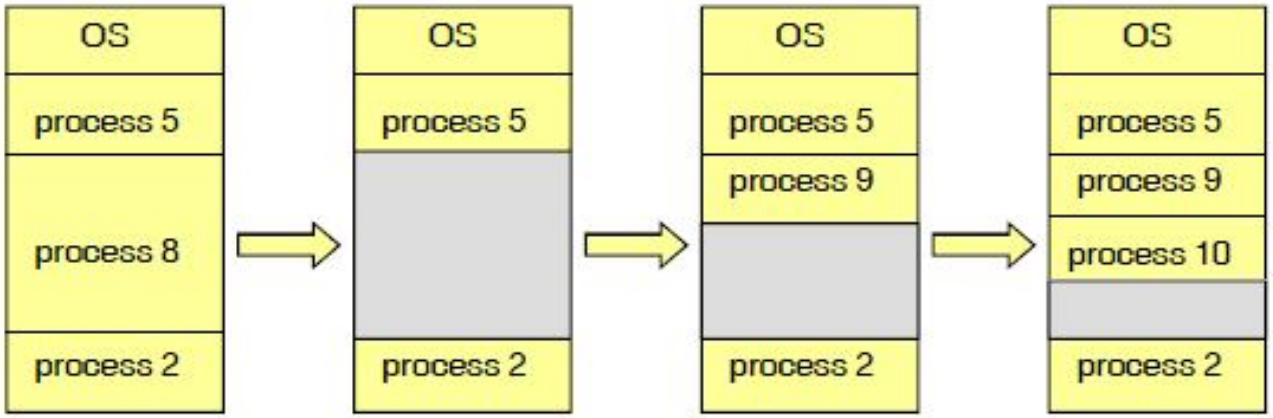


Contiguous Allocation

- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
- Relocation 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*

Contiguous Allocation (Cont.)

- 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

How to satisfy a request of size *n* from a list of free holes

- **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

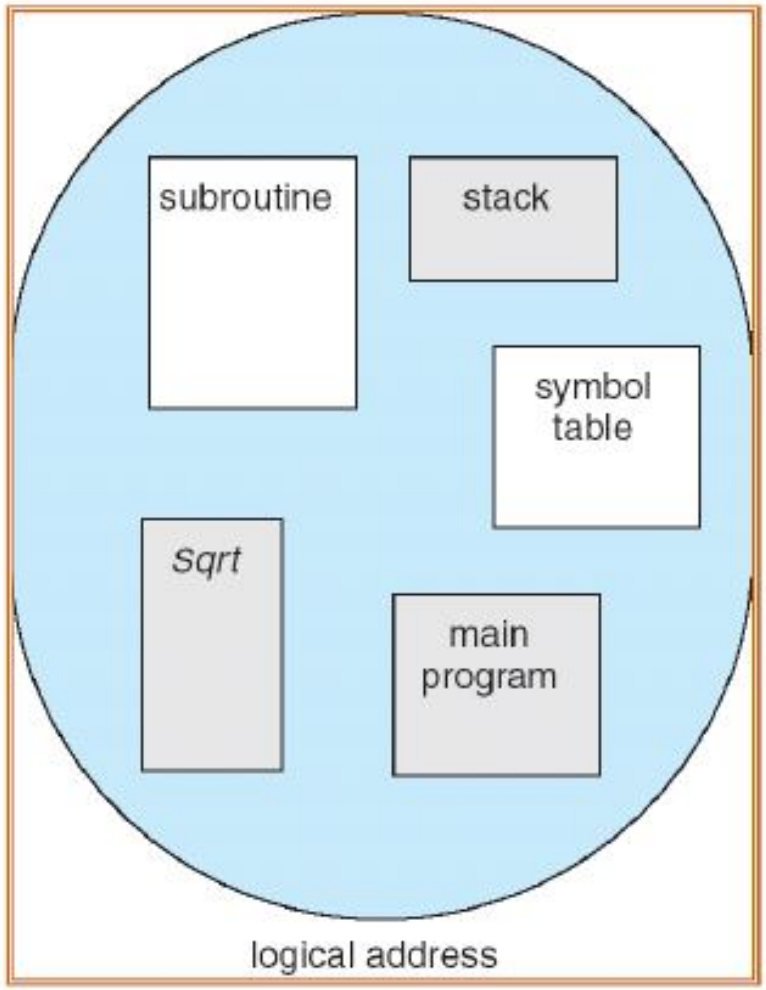
First-fit and best-fit better than worst-fit in terms of speed and storage utilization

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



Segmentation Architecture

- Logical address consists of a two table:

  <segment-number, offset>,

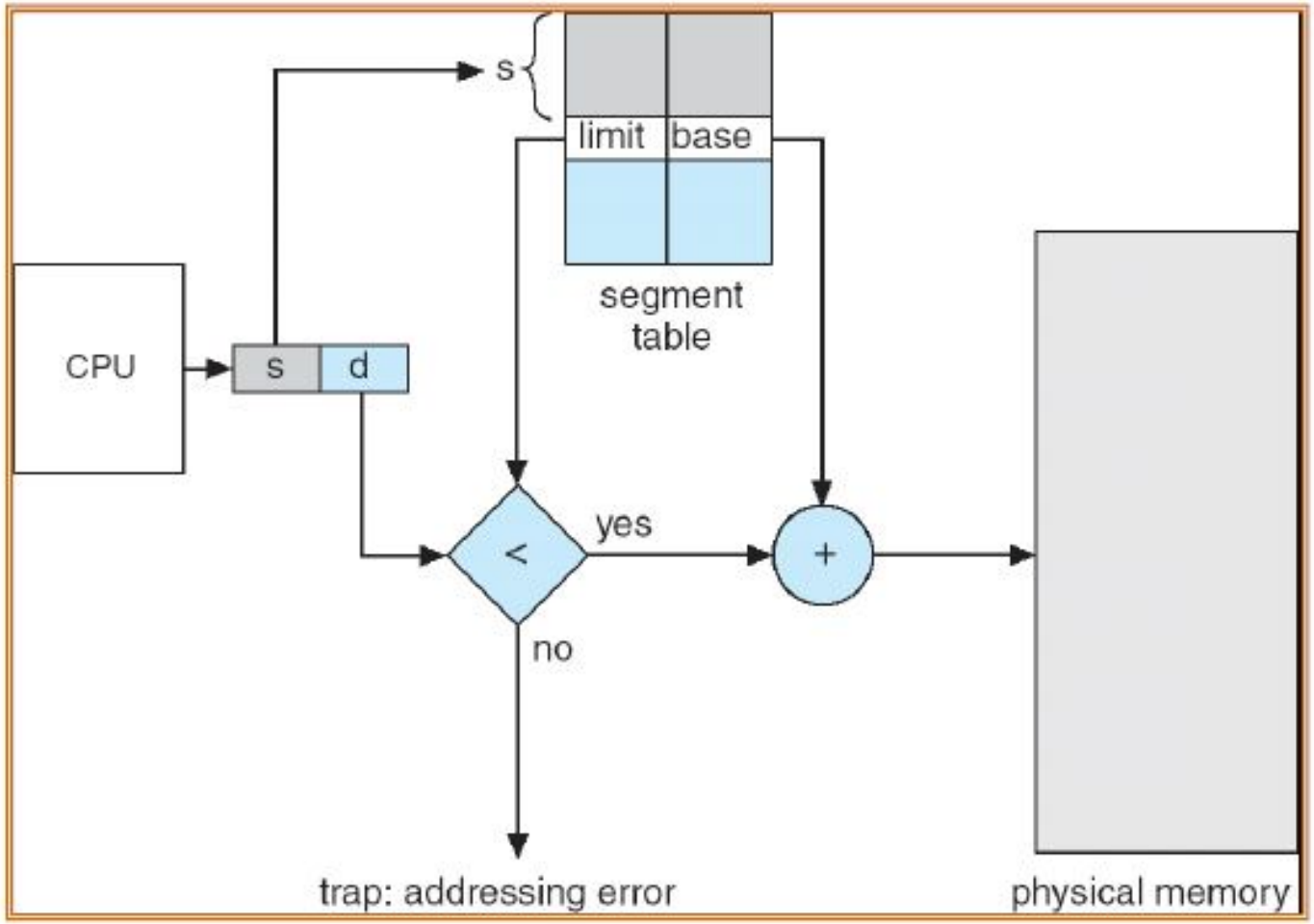
- **Segment table –** maps two-dimensional physical addresses; each table entry has:
  - **base –** contains the starting physical address where the segments reside in memory
  - **limit –** specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

  segment number ***s*** is legal if ***s*** < STLR

Segmentation Architecture (Cont.)

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 ⇒ 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



Virtual Memory

Virtual Memory

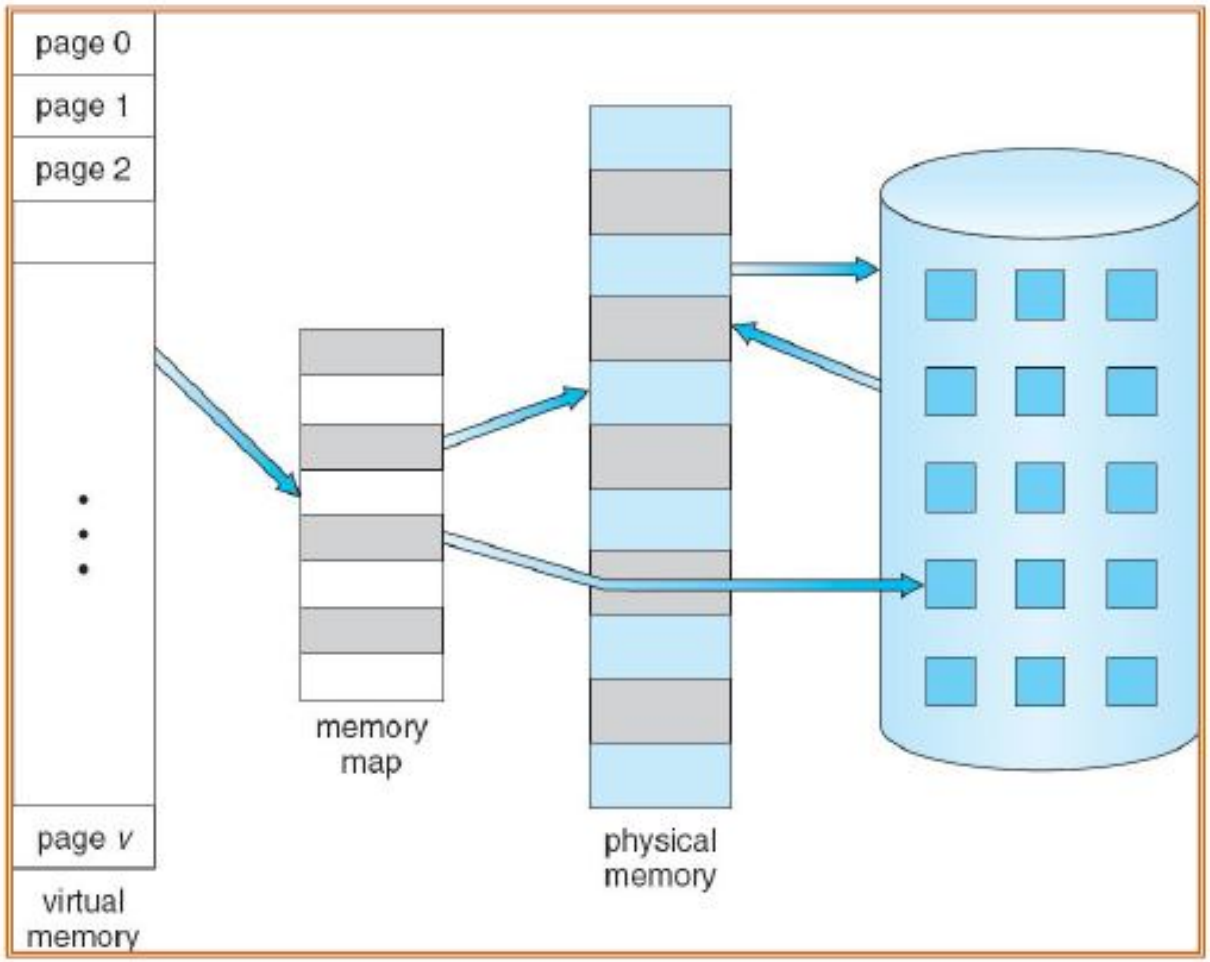
- Background
- Page Replacement
- Memory-Mapped Files

Background

- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation.

- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Virtual Memory That is Larger Than Physical Memory

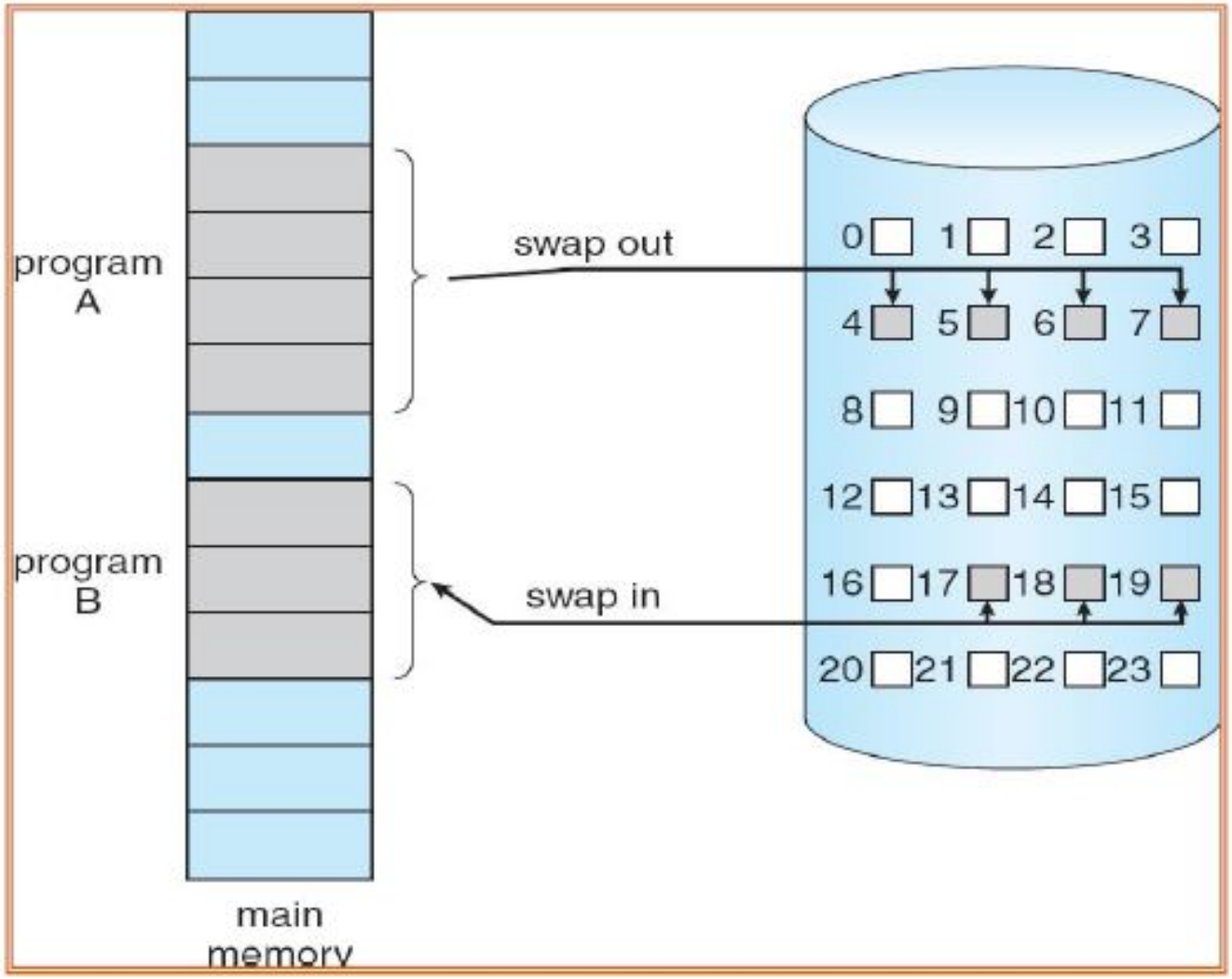


Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response

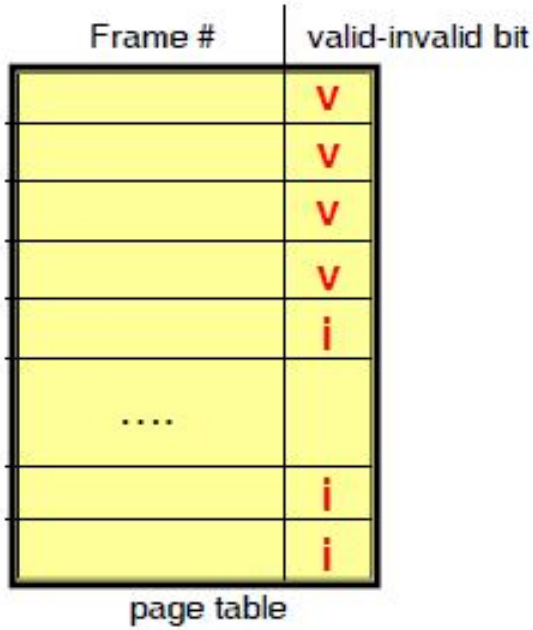
- Page is needed ⇒ reference to it.
  - invalid reference ⇒ abort.
  - not-in-memory ⇒ bring to memory.

Transfer of a Paged Memory to Contiguous Disk Space



Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (**v** ⇒ in-memory, **i** ⇒ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table :



- During address translation, if valid–invalid bit in page table entry is **I** ⇒ page fault

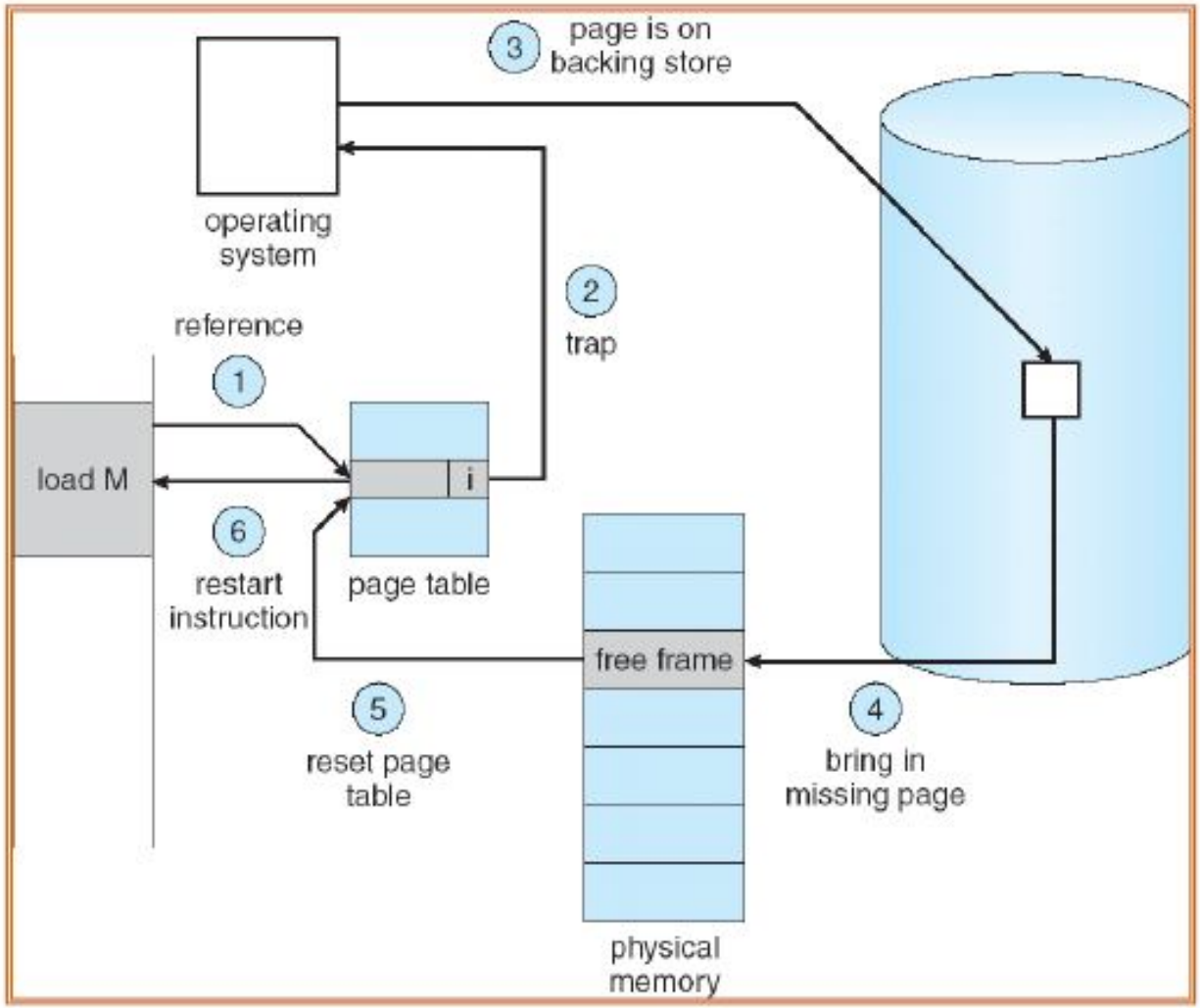
Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system:

  page fault

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = **v**
6. Restart the instruction that caused the page fault

Steps in Handling a Page Fault



Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use **modify bit** to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

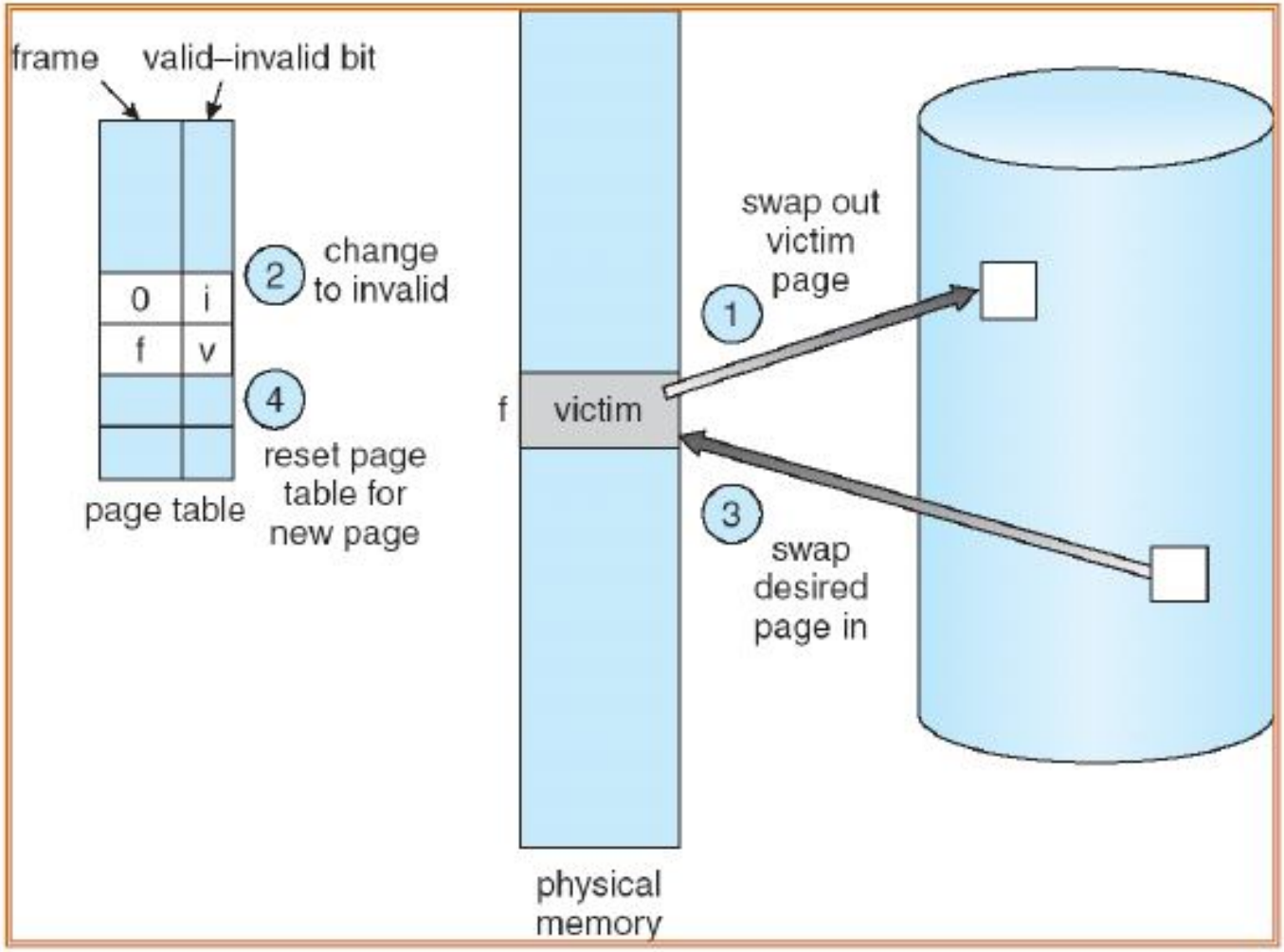
Basic Page Replacement

1. Find the location of the desired page on disk

2. Find a free frame:
   - If there is a free frame, use it.
   - If there is no free frame, use a page replacement algorithm to select a **victim** frame
3. Bring the desired page into the (newly) free frame; update the page and frame tables

4. Restart the process

Page Replacement



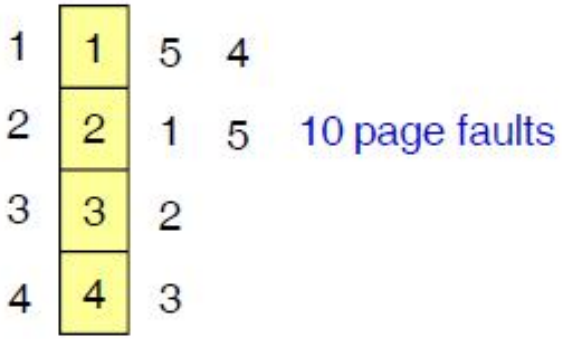
Page Replacement Algorithms

First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

9 page faults

- 4 frames

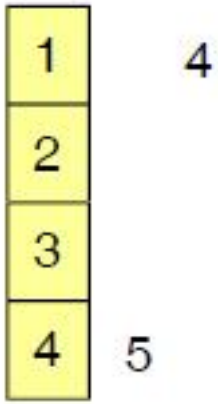


10 page faults

Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

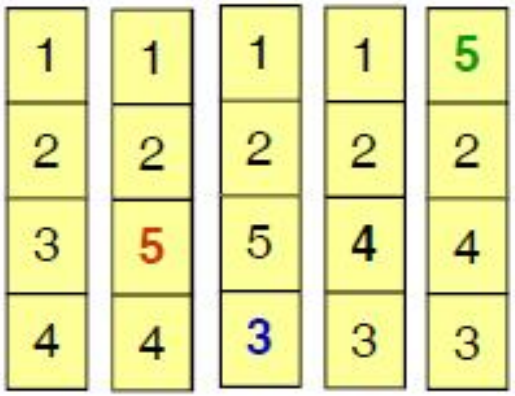


6 page faults

- How do you know this?

Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change

Secondary Storage Systems

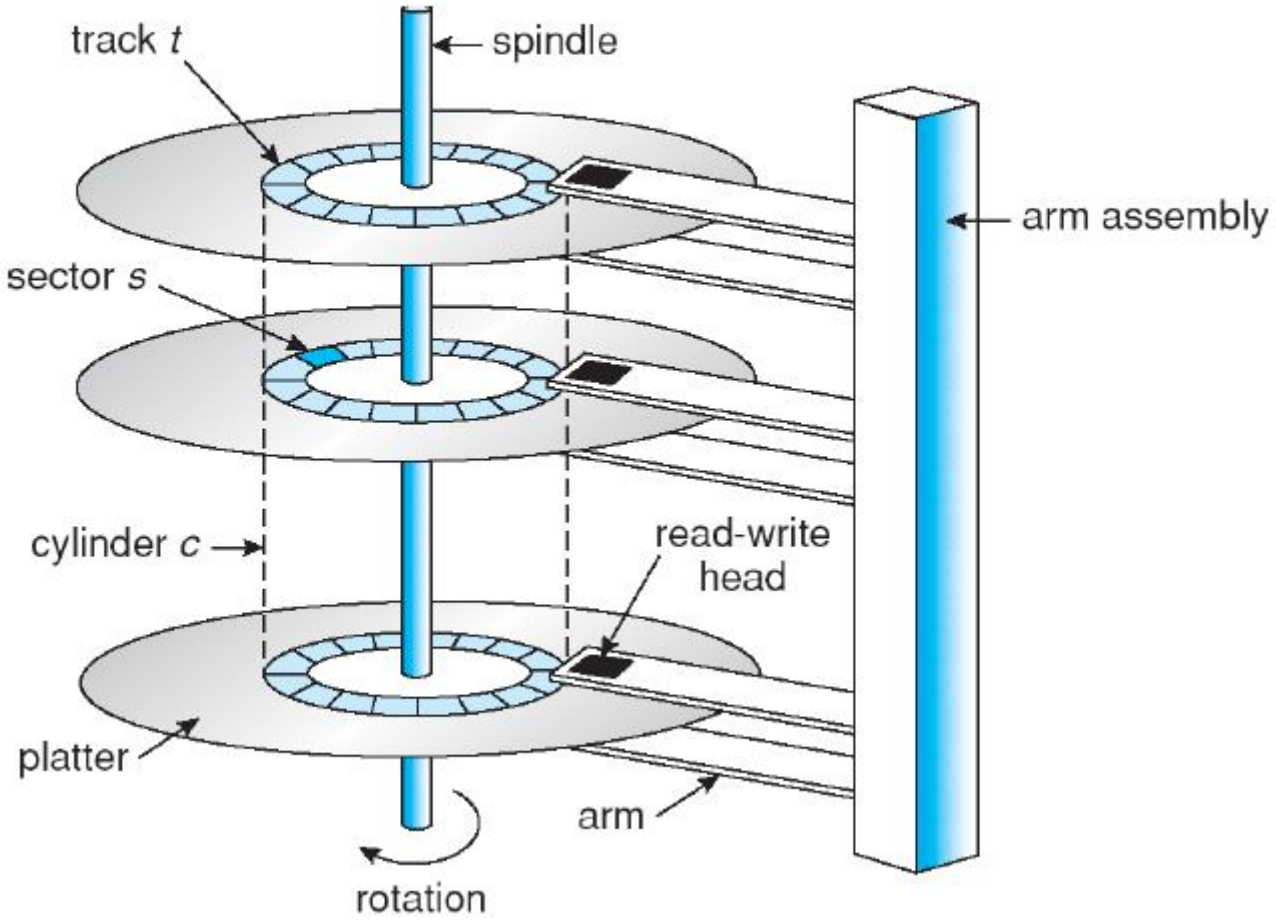
Overview of Secondary Storage Structure

- 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)

Overview of Secondary Storage Structure

- 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



Overview of Secondary Storage Structure

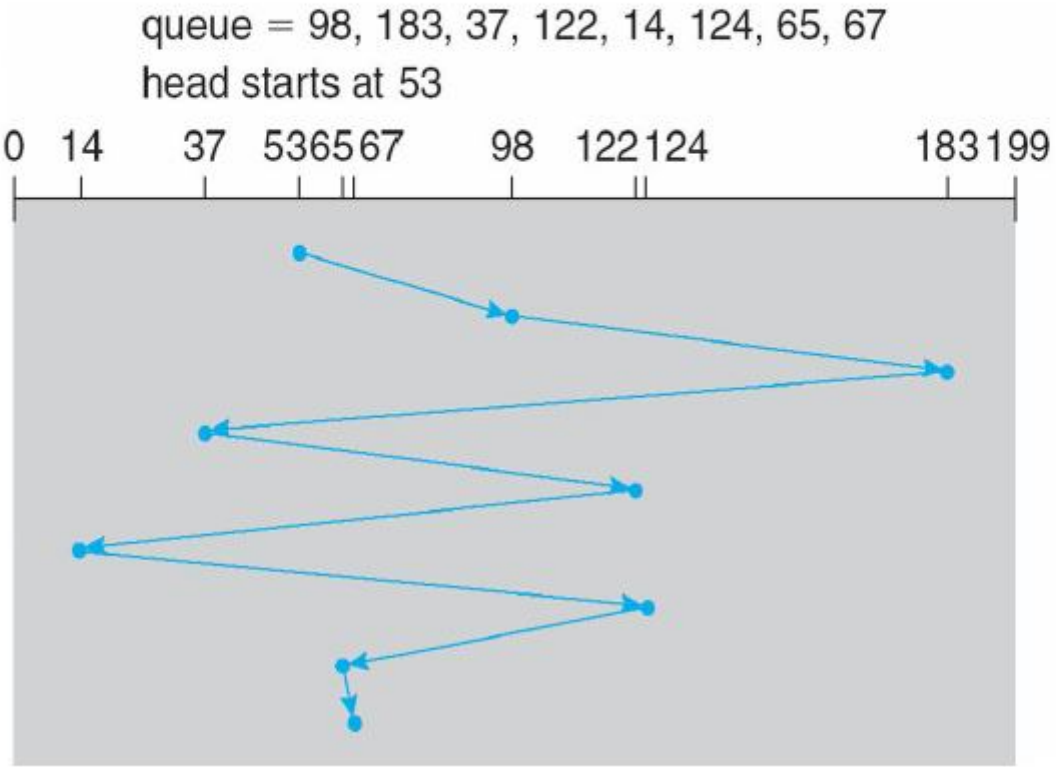
- Magnetic tape
  - Was early secondary-storage medium
  - Relatively permanent and holds large quantities of data
  - Access time slow
  - Random access ~1000 times slower than disk
  - Mainly used for backup, storage of infrequently-used data, transfer medium between systems
  - 20-200GB typical storage

Disk Scheduling

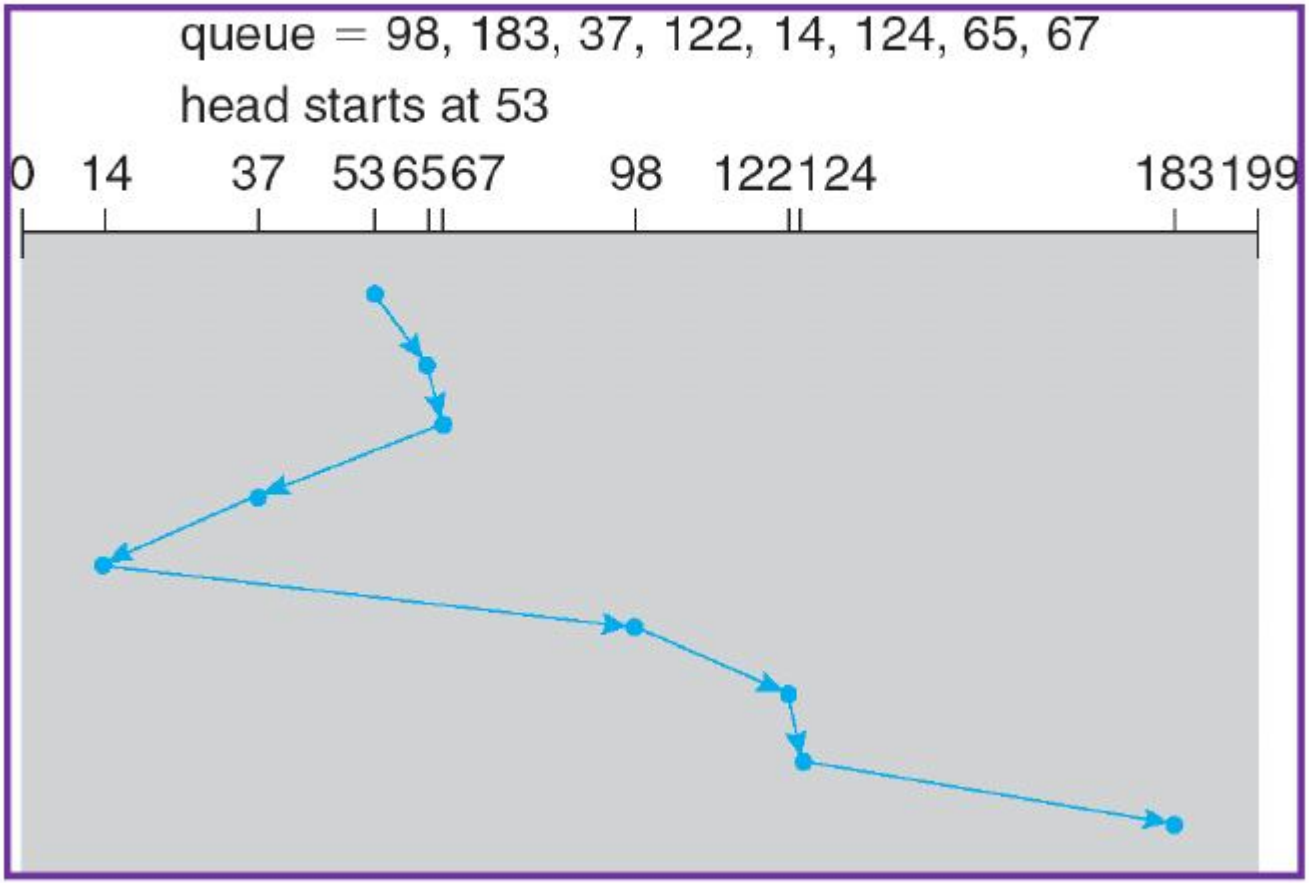
- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Access time has two major components
  - Seek time is the time for the disk to move the heads to the cylinder containing the desired sector
  - Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head
- Minimize seek time
- Seek time $\approx$ 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

FCFS

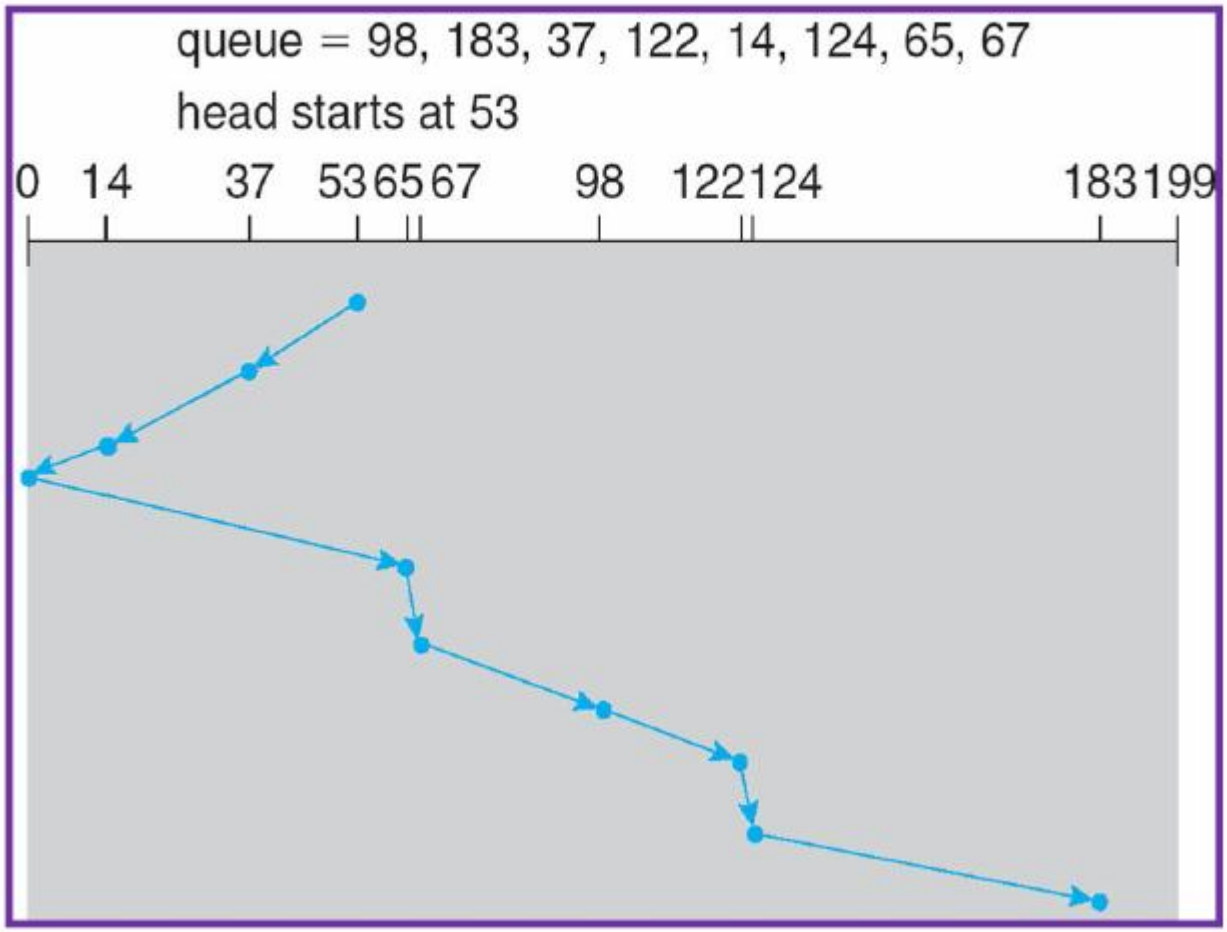
Illustration shows total head movement of 640 cylinders



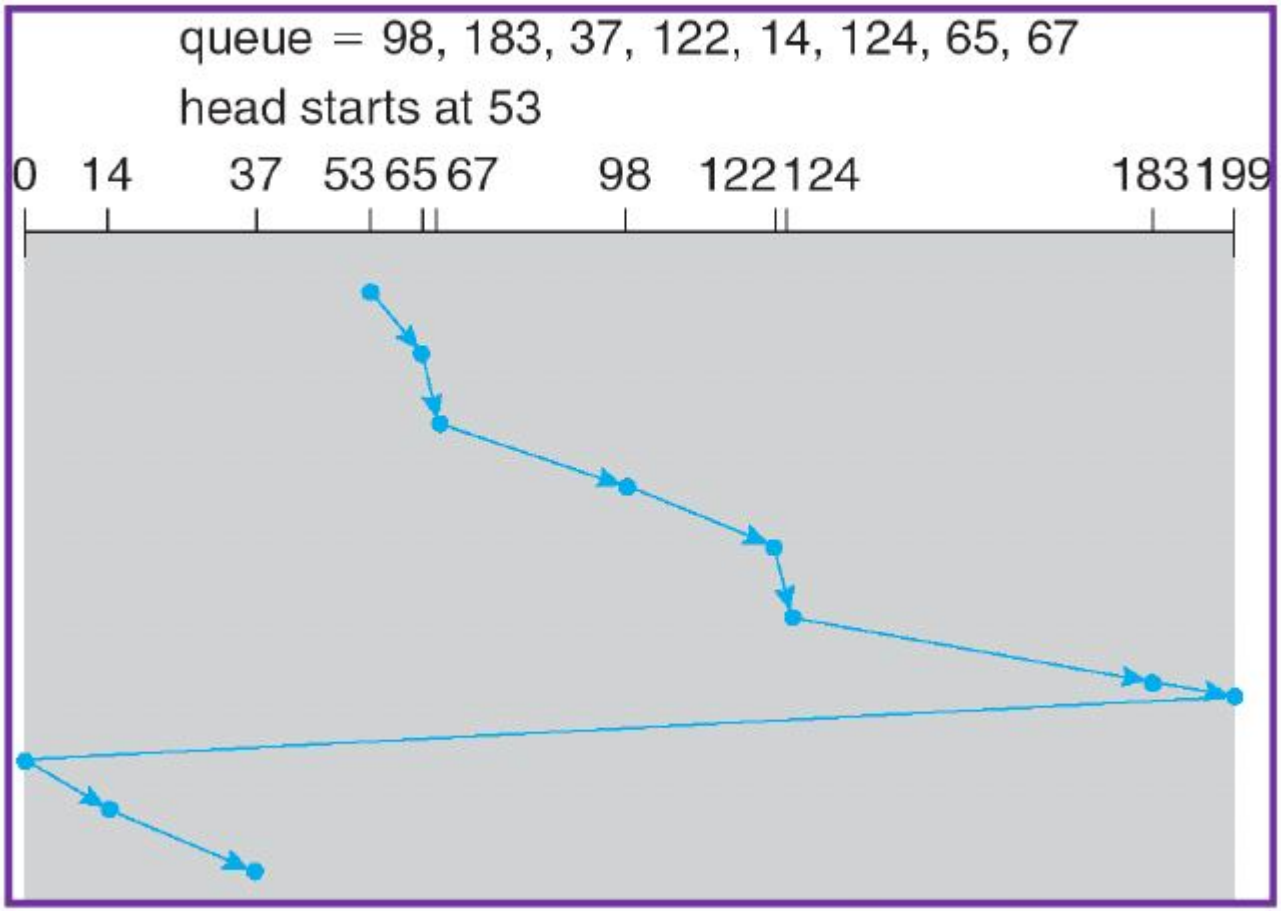
SSTF (Cont)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

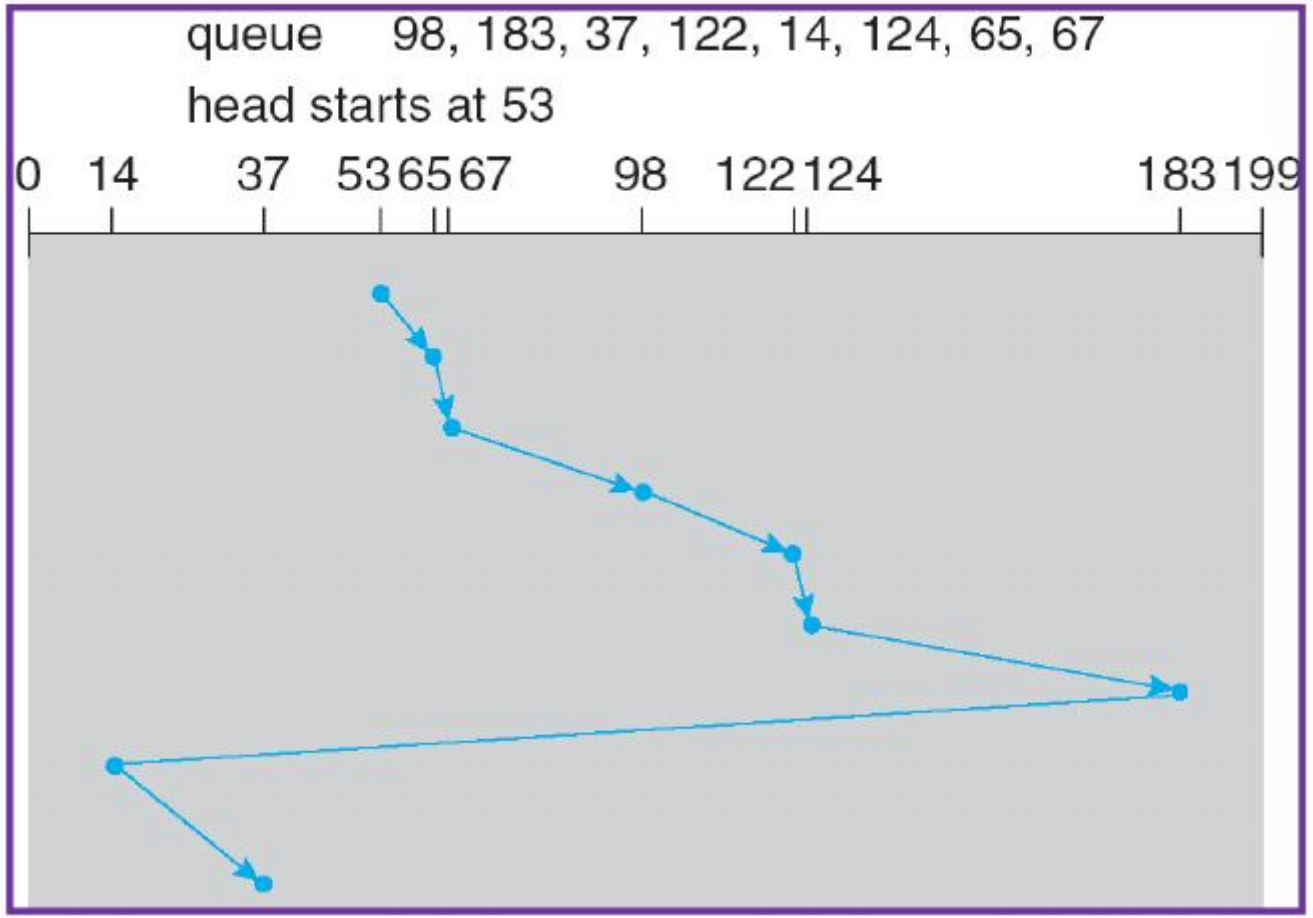
SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-SCAN (Cont)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-LOOK (Cont)


queue 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0 14 37 53 65 67 98 122 124 183 199

File System Implementation

File System Implementation

- File-System Structure
- File-System Implementation
- Allocation Methods

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
- File system organized into layers
- File control block–storage structure consisting of information about a file

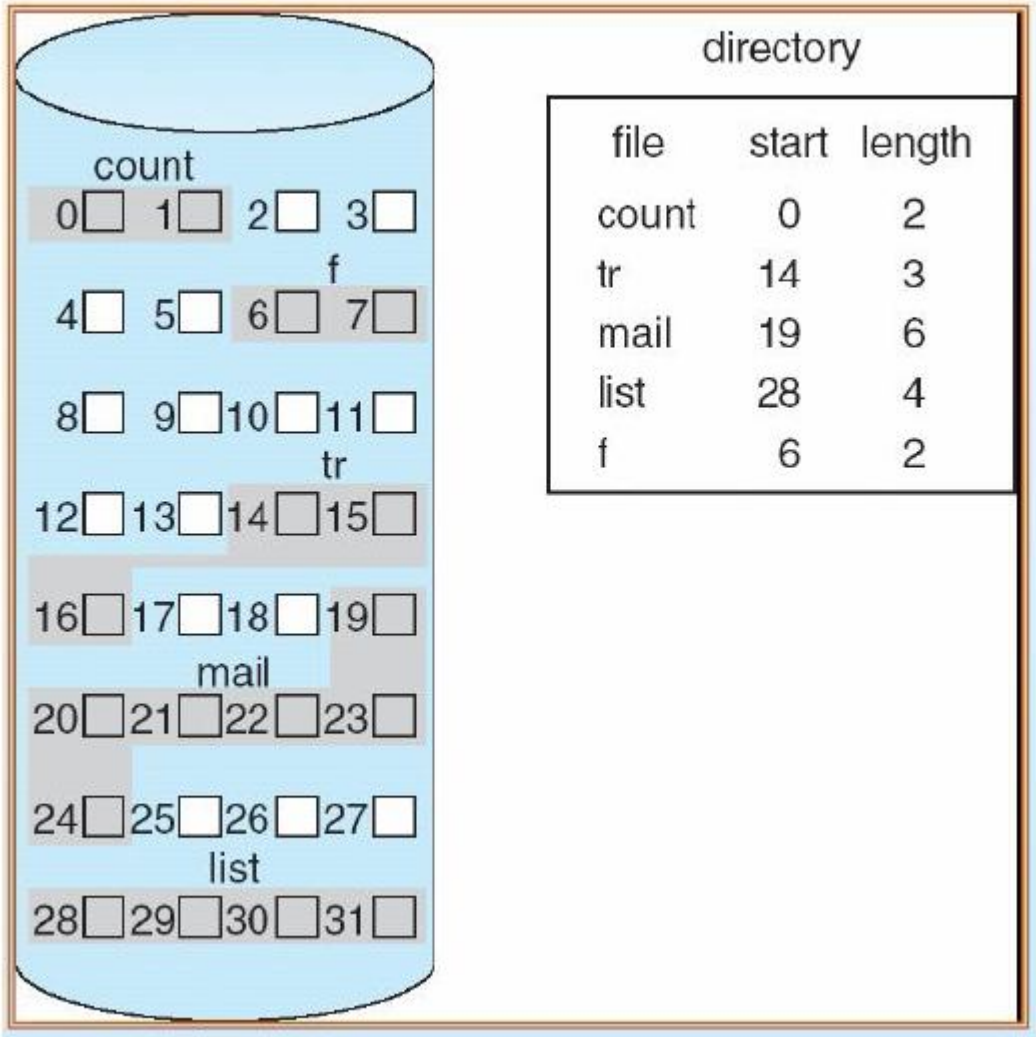
File-System Structure

- the *I/O control, consists of device drivers and interrupt* handlers to transfer information between the memory and the disk system.
- A device driver can be thought of as a translator.

Allocation Methods

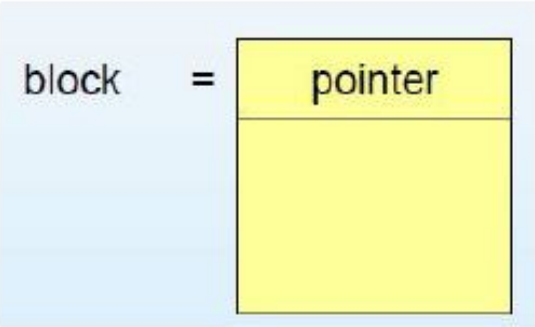
- An allocation method refers to how disk blocks are allocated for files:
- The direct-access nature of disks allows us flexibility in the implementation of files. In almost every case, many files will be stored on the same disk. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.
- Three major methods of allocating disk space are in wide use:
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

Contiguous Allocation of Disk Space

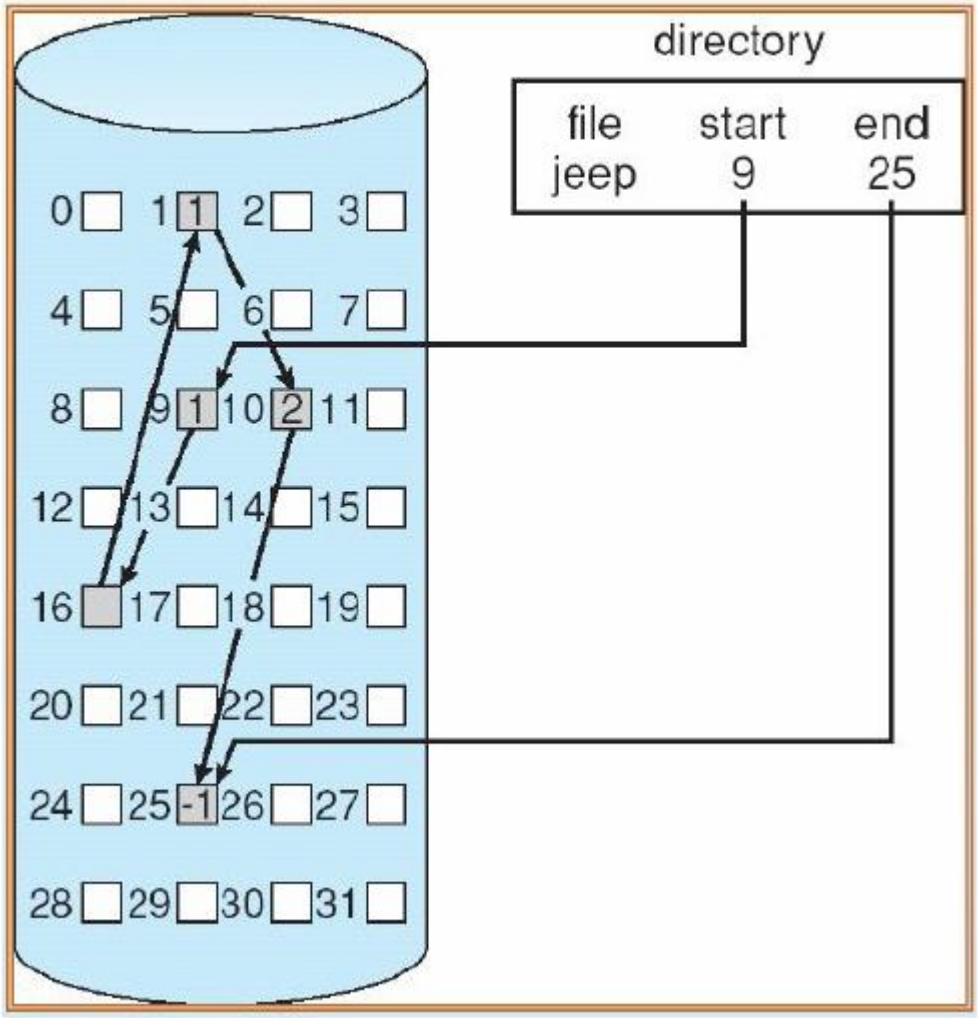


Linked Allocation

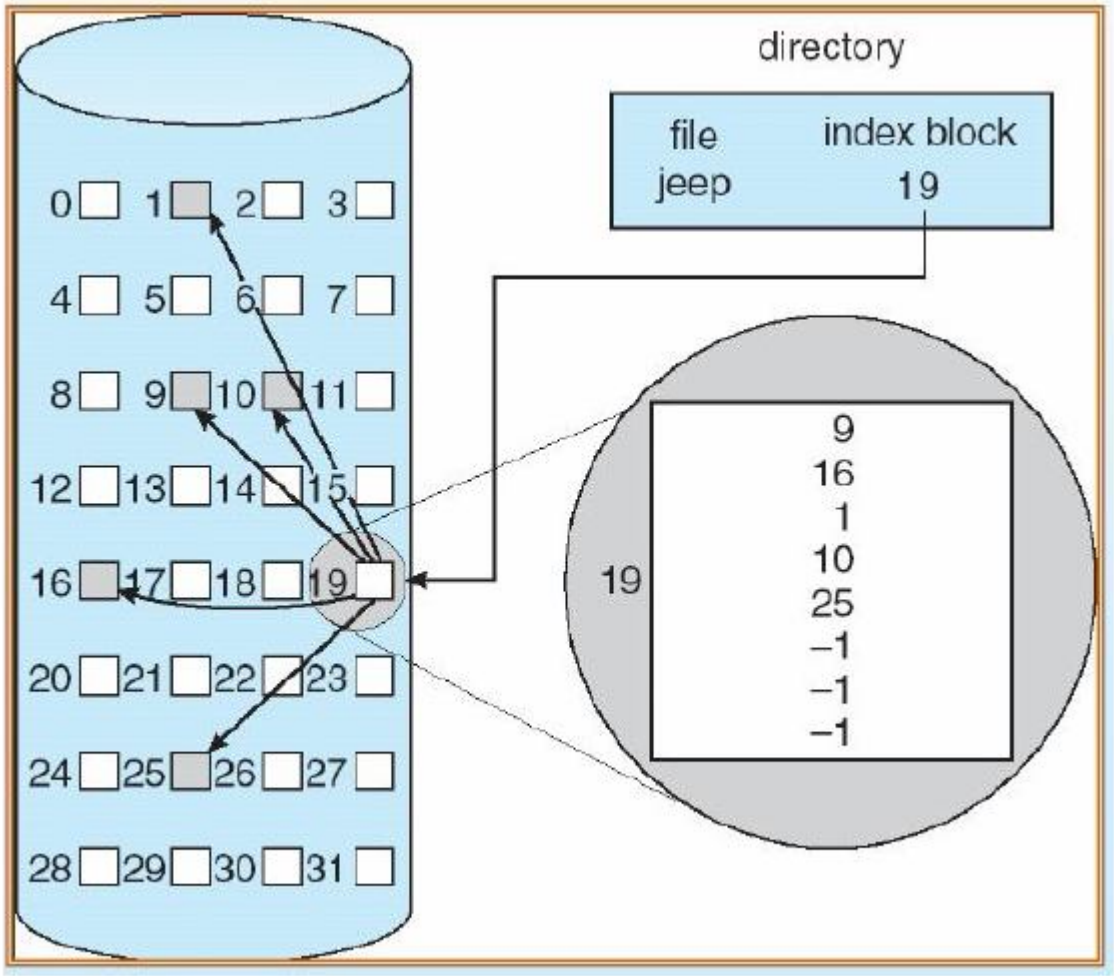
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the file.
- Simple –need only starting address
- Free-space management system –no waste of space



Linked Allocation



Example of Indexed Allocation



Protection

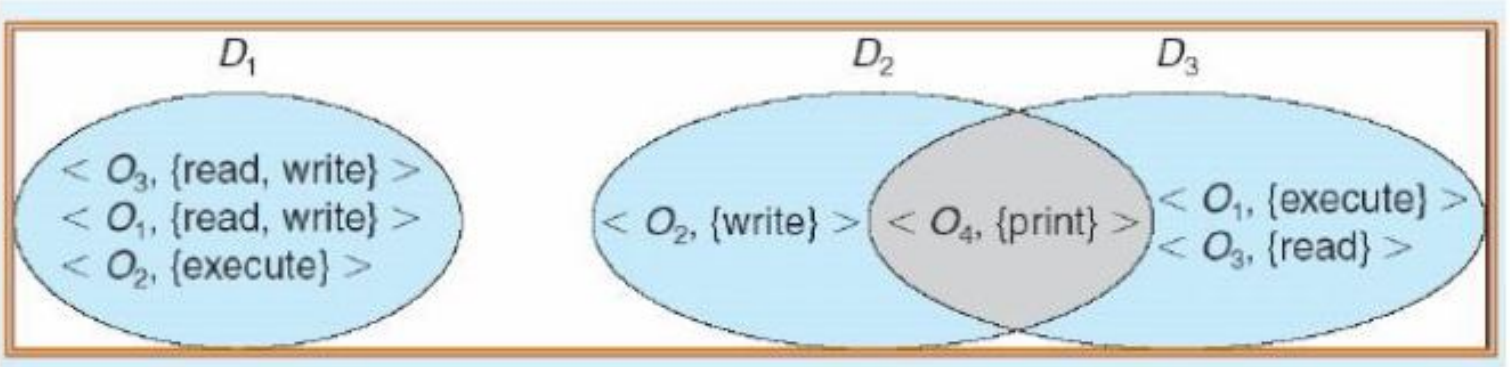
PROTECTION
DOMAIN OF PROTECTION
SECURITY
CRYPTOGRAPHY
AUTHENTICATION

Goals of Protection

- Operating system consists of a collection of objects, hardware or Software.
- Each object has a unique name and can be accessed through a well-defined set of operations.
- Protection problem -ensure that each object is accessed correctly and only by those processes that are allowed to do so.

Domain Structure

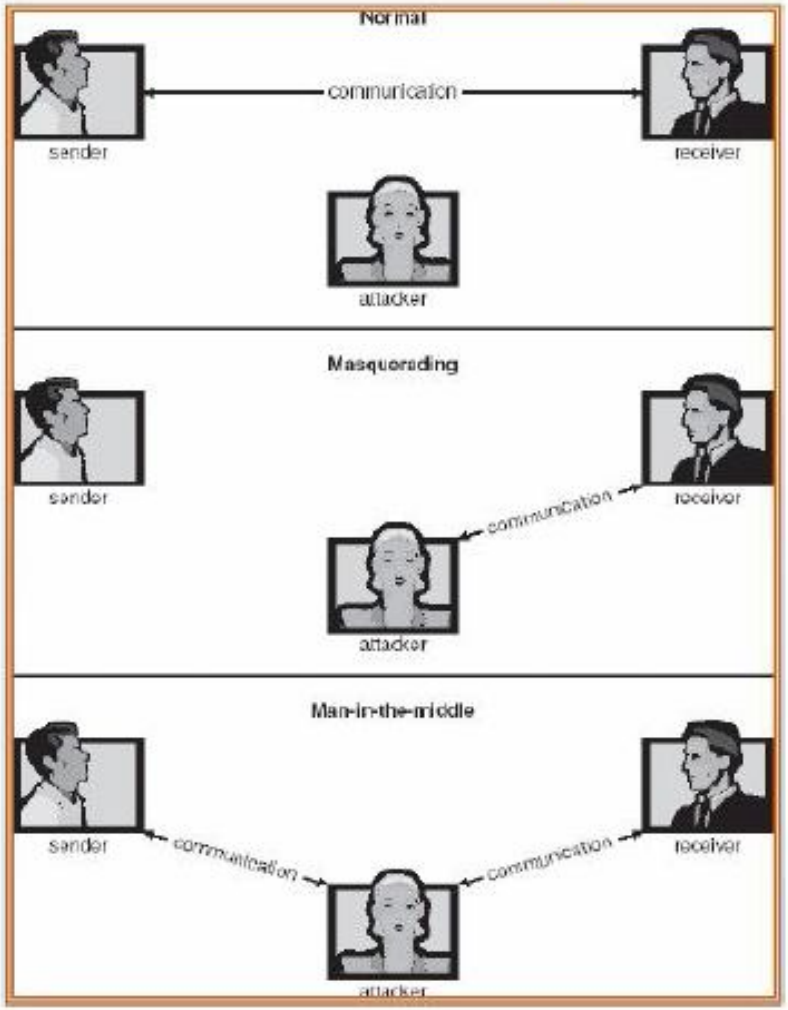
- Access-right = <*object-name, rights-set*>
- where *rights-set is a subset of all valid operations that can be* performed on the object.
- Domain = set of access-rights



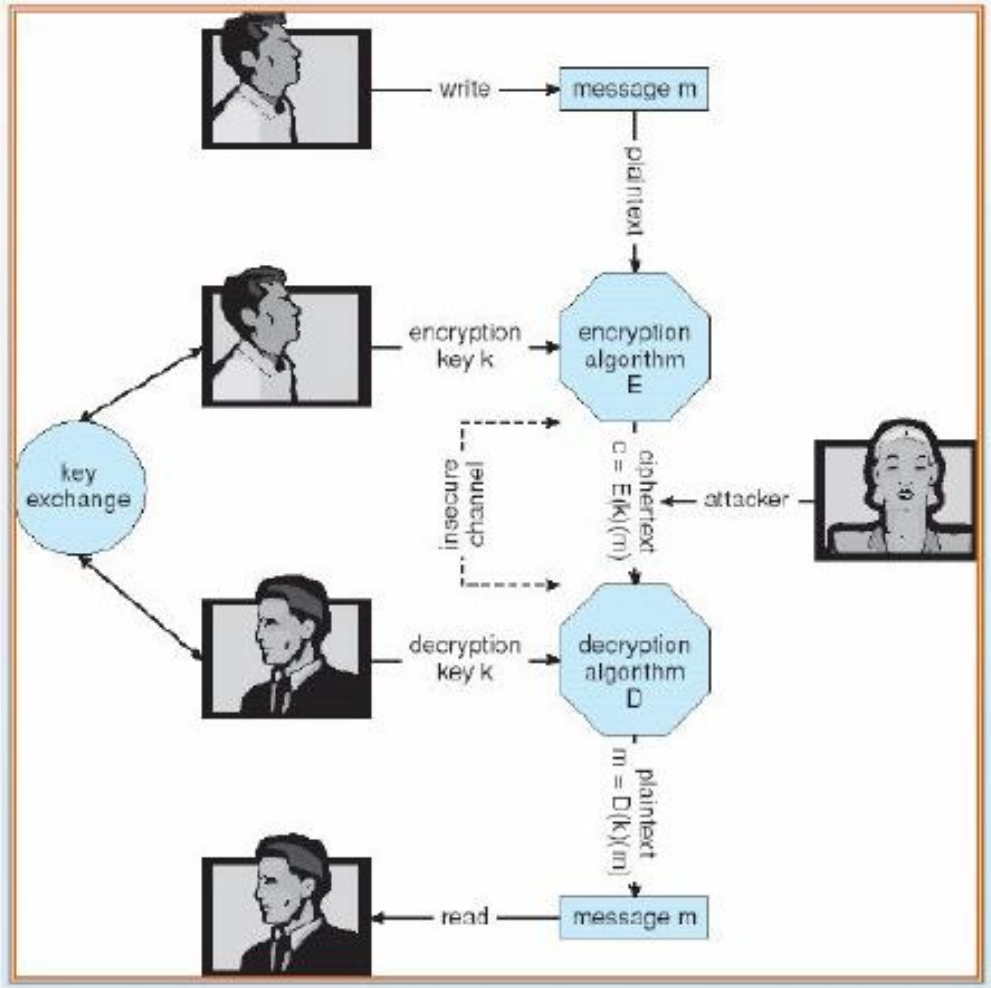
The Security Problem

- Security must consider external environment of the system, and protect the system resources
- Intruders (crackers) attempt to breach security
- **Threat** is potential security violation
- **Attack** is attempt to breach security
- Attack can be accidental or malicious
- Easier to protect against accidental than malicious misuse

Standard Security Attacks

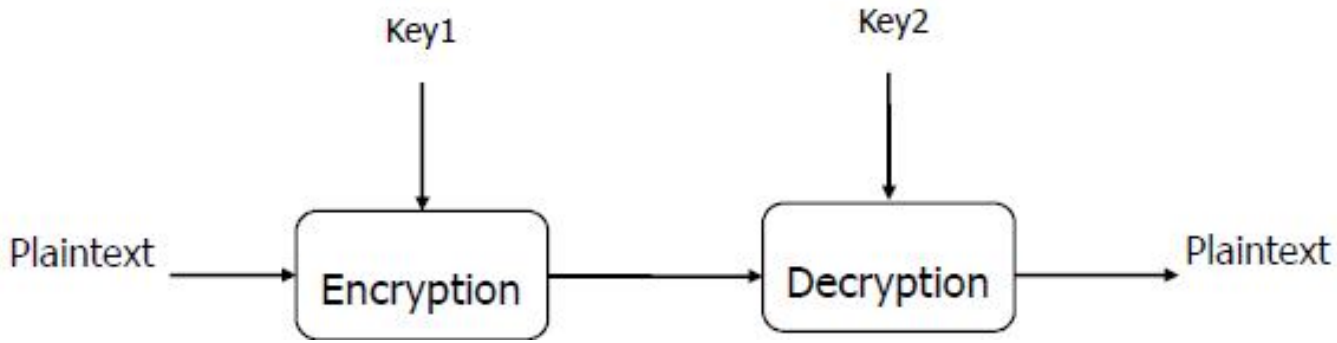


Secure Communication over Insecure Medium



Cryptography as a Security Tool

- "Encryption" is the transformation of data into a form, which is almost impossible to read without some shared secrets. The purpose is to ensure privacy by keeping information hidden from anyone except the intended ones.
- "Decryption" is the reverse of encryption; that is the transformation of encrypted data back to a meaningful form.



Authentication

- The authentication system is the first line to prevent unauthorized users from gaining access to the system[5]. Authentication is mechanism by which a process verifies the communication partner is who it is supposed to be and not an imposter.

User Authentication

- Crucial to identify user correctly, as protection systems depend on user ID
- User identity most often established through *passwords, can be considered a special case of either keys or capabilities*
  - Also can include something user has and /or a user attribute
- Passwords must be kept secret
  - Frequent change of passwords
  - Use of "non-guessable" passwords
  - Log all invalid access attempts
- Passwords may also either be encrypted or allowed to be used only once