JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR
(Established by Govt. of A.P., ACT No.30 of 2008)
ANANTHAPURAMU – 515 002 (A.P) INDIA

Computer Science & Engineering

Course Code OPERATING SYSTEMS 20A05402T
(Common to CSE, IT, CSE(DS), CSE (IoT), CSE (AI), CSE (AI & ML) and AI & DS)

L	T	P	C
3	0	0	3

Course Objectives:

The course is designed to

- Understand basic concepts and functions of operating systems
- Understand the processes, threads and scheduling algorithms.
- Provide good insight on various memory management techniques
- Expose the students with different techniques of handling deadlocks
- Explore the concept of file-system and its implementation issues
- Familiarize with the basics of the Linux operating system
- Implement various schemes for achieving system protection and security

Course Outcomes (CO):

After completion of the course, students will be able to

- Realize how applications interact with the operating system
- Analyze the functioning of a kernel in an Operating system.
- Summarize resource management in operating systems
- Analyze various scheduling algorithms
- Examine concurrency mechanism in Operating Systems
- Apply memory management techniques in the design of operating systems
- Understand the functionality of the file system
- Compare and contrast memory management techniques.
- Understand deadlock prevention and avoidance.
- Perform administrative tasks on Linux based systems.

UNIT - I **Operating Systems Overview, System Structures** 8Hrs

Operating Systems Overview: Introduction, Operating system functions, Operating systems
operations, Computing environments, Open-Source Operating Systems

System Structures: Operating System Services, User and Operating-System Interface, systems calls,
Types of System Calls, system programs, Operating system Design and Implementation, Operating
system structure, Operating system debugging, System Boot.

UNIT - II **Process Concept, Multithreaded Programming,Process Scheduling, Inter-process Communication**
10Hrs

Process Concept: Process scheduling, Operations on processes, Inter-process communication,
Communication in client server systems.

Multithreaded Programming: Multithreading models, Thread libraries, Threading issues, Examples

Process Scheduling: Basic concepts, Scheduling criteria, Scheduling algorithms, Multiple processor scheduling, Thread scheduling, Examples.

Inter-process Communication: Race conditions, Critical Regions, Mutual exclusion with busy waiting, Sleep and wakeup, Semaphores, Mutexes, Monitors, Message passing, Barriers, Classical IPC Problems - Dining philosophers problem, Readers and writers problem.

UNIT - III **Memory-Management Strategies, Virtual Memory Management**

Lecture 8Hrs

Memory-Management Strategies: Introduction, Swapping, Contiguous memory allocation, Paging, Segmentation, Examples.

Virtual Memory Management: Introduction, Demand paging, Copy on-write, Page replacement, Frame allocation, Thrashing, Memory-mapped files, Kernel memory allocation, Examples.

UNIT - IV **Deadlocks, File Systems** Lecture 9Hrs

Deadlocks: Resources, Conditions for resource deadlocks, Ostrich algorithm, Deadlock detection And recovery, Deadlock avoidance, Deadlock prevention.

File Systems: Files, Directories, File system implementation, management and optimization.

Secondary-Storage Structure: Overview of disk structure, and attachment, Disk scheduling, RAID structure, Stable storage implementation.

UNIT - V **System Protection, System Security** Lecture 8Hrs

System Protection: Goals of protection, Principles and domain of protection, Access matrix, Access control, Revocation of access rights.

System Security: Introduction, Program threats, System and network threats, Cryptography as a security, User authentication, implementing security defenses, firewalling to protect systems and networks, Computer security classification.

Case Studies: Linux, Microsoft Windows.

Textbooks:

1. Silberschatz A, Galvin P B, and Gagne G, Operating System Concepts, 9th edition, Wiley, 2016.
2. Tanenbaum A S, Modern Operating Systems, 3rd edition, Pearson Education, 2008. (Topics: Inter-process Communication and File systems.)

Reference Books:

1. Tanenbaum A S, Woodhull A S, Operating Systems Design and Implementation, 3rd edition,

PHI, 2006.

2. Dhamdhere D M, Operating Systems A Concept Based Approach, 3rd edition, Tata McGraw-Hill, 2012.

3. Stallings W, Operating Systems -Internals and Design Principles, 6th edition, Pearson Education, 2009

4. Nutt G, Operating Systems, 3rd edition, Pearson Education, 2004

Online Learning Resources:

https://nptel.ac.in/courses/106/106/106106144/
http://peterindia.net/OperatingSystems.html

UNIT-1

Operating System Overview

OVER VIEW OF OPERATING SYSTEM

What is an Operating System?

A program that acts as an intermediary between a user of a computer and the computer hardware

Operating system goals:

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

Computer System Structure

- Computer system can be divided into four components
- Hardware – provides basic computing resources
- CPU, memory, I/O devices

Operating system

Controls and coordinates use of hardware among various applications and users

Application programs – define the ways in which the system resources are used to solve the computing problems of the usersWord processors, compilers, web browsers, database systems, video games

Users

People, machines, other computers

Four Components of a Computer System

Operating System Definition

- OS is a resource allocator
- Manages all resources
- Decides between conflicting requests for efficient and fair resource use
- OS is a control program
- Controls execution of programs to prevent errors and improper use of the computer
- No universally accepted definition
- Everything a vendor ships when you order an operating system" is good approximation

  But varies wildly.
- "The one program running at all times on the computer" is the kernel. Everything else is either a system program (ships with the operating system) or an application program

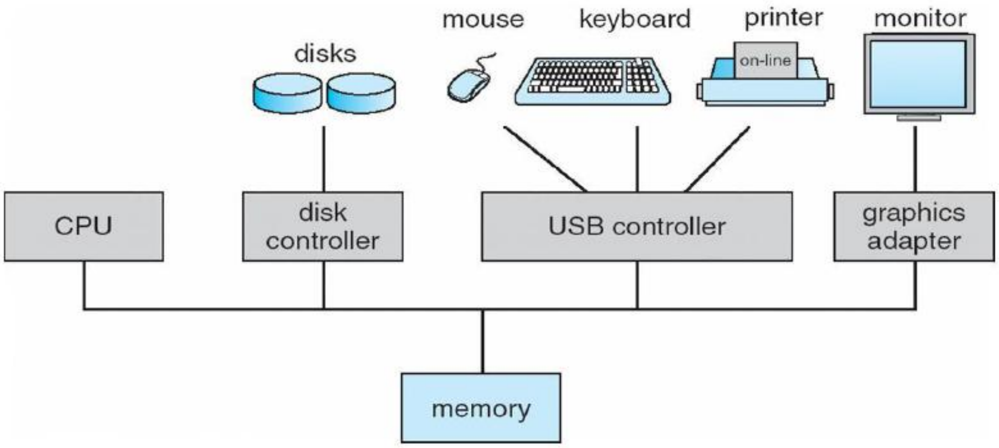
Computer Startup

- bootstrap program is loaded at power-up or reboot
- Typically stored in ROM or EPROM, generally known as firmware
- Initializes all aspects of system
- Loads operating system kernel and starts execution

Computer System Organization

- Computer-system operation

- One or more CPUs, device controllers connect through common bus providing access to shared memory
- Concurrent execution of CPUs and devices competing for memory cycles



<u>**Operating-System Operations**</u>

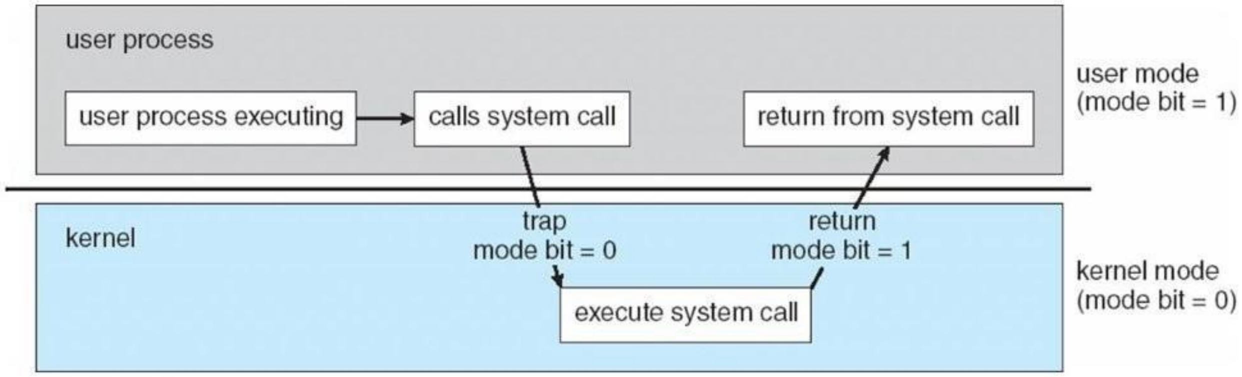
- Interrupt driven by hardware
- Software error or request creates **exception** or**trap**
- Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system
- **Dual-mode** operation allows OS to protect itself and other systemcomponents
    - **User mode** and **kernelmode**
    - **Mode bit** provided byhardware
    - Providesabilitytodistinguishwhensystemisrunningusercodeorkernel code
    - Someinstructionsdesignatedas**privileged**,onlyexecutableinkernel mode
    - System call changes mode to kernel, return from call resets it to user

Transition from User to Kernel Mode

✓ Timer to prevent infinite loop / process hogging resources

- Set interrupt after specific period
- Operating system decrements counter
- When counter zero generate an interrupt

Set up before scheduling process to regain control or terminate program that exceeds allotted



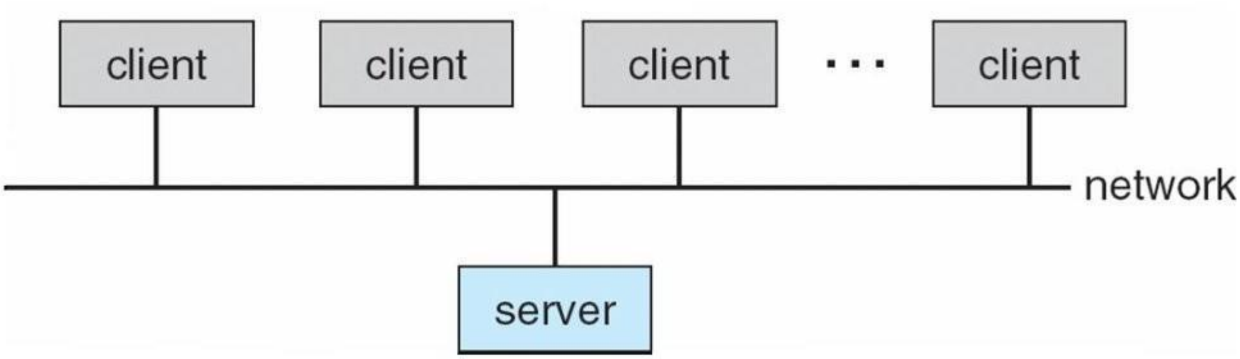
Protection and Security:

- ✓ **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- ✓ **Security** – defense of the system against internal and externalattacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
  - Systems generally first distinguish among users, to determine who can dowhat
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process,file
  - **Privilege escalation** allows user to change to effective ID with morerights

Computing Environments:

Client-Server Computing

- Dumb terminals supplanted by smart PCs
- Many systems now **servers**, responding to requests generated by**clients**
  - **Compute-serve**rprovidesaninterfacetoclienttorequestservices(i.e., database)
  - **File-server** provides interface for clients to store and retrieve files



Peer to Peer:

- ✓ P2P does not distinguish clients and servers
  - o Instead all nodes are considered peers
  - o May each act as client, server orboth
  - o Node must join P2P network
    - o Registers its service with central lookup service on network, or
    - o Broadcast request for service and respond to requests for service via **discovery protocol**
  - o Examples include *Napster* and*Gnutella*

Web-Based Computing

- ✓ Web has become ubiquitous
- ✓ PCs most prevalent devices
- ✓ More devices becoming networked to allow webaccess
- ✓ New category of devices to manage web traffic among similar servers: **load balancers**
- ✓ Use of operating systems like Windows 95, client-side, have evolved into Linux and Windows XP, which can be clients andservers

<u>Open-Source Operating Systems:</u>

- ✓ 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** and **BSD UNIX**(including core of **Mac OS X**), and many more

Operating System Services:

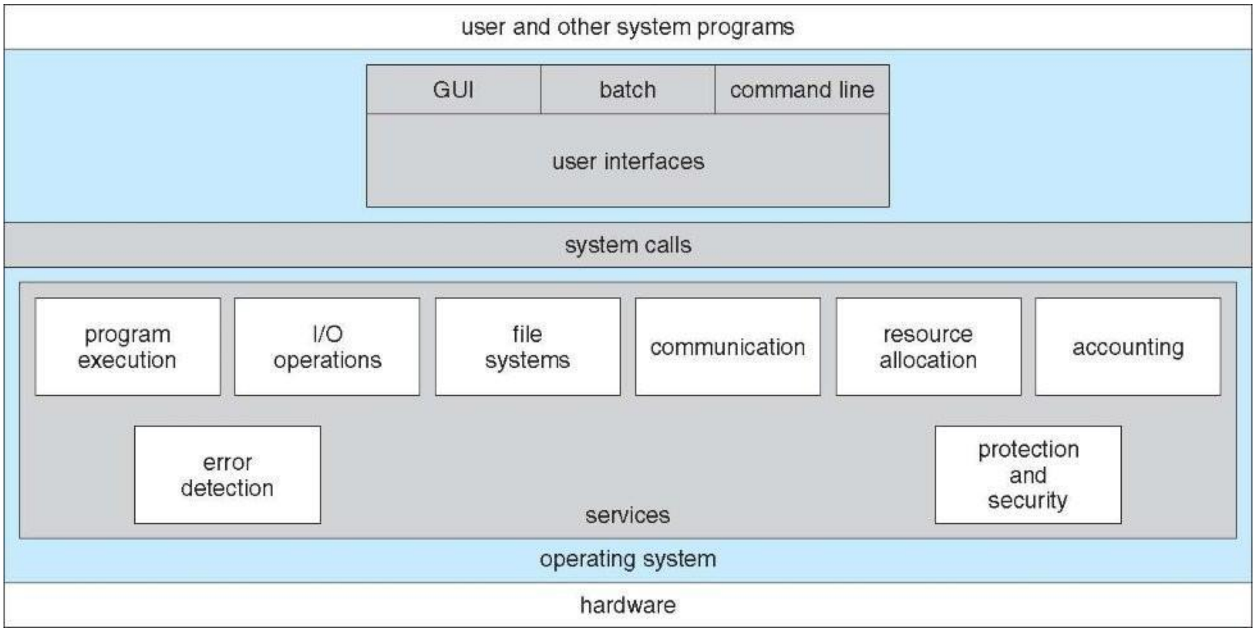
- ✓ Operating systems provide an environment for execution of programs and services to programs and users
- ✓ One set of operating-system services provides functions that are helpful to theuser:
    - o **User interface** - Almost all operating systems have a user interface(UI).
        - ⊿ Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **Batch**
    - o **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)
    - o **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
    - o **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
    - o **Communications** – Processes may exchange information, on the same computer or between computers over a network
        - ⊿Communicationsmaybeviasharedmemoryorthroughmessagepassing (packets moved by theOS)
    - o **Error detection** – OS needs to be constantly aware of possibleerrors
        - ⊿MayoccurintheCPUandmemoryhardware,inI/Odevices,inuser program
        - ⊿Foreachtypeoferror,OSshouldtaketheappropriateactiontoensure correct and consistentcomputing
        - ⊿Debuggingfacilitiescangreatlyenhancetheuser'sandprogrammer's abilities to efficiently use thesystem
- ✓ Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
    - o **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
        - ⊿Manytypesofresources-Some(suchasCPUcycles,mainmemory,and filestorage)mayhavespecialallocationcode,others(suchasI/Odevices) may have general request and releasecode

- o **Accounting -** To keep track of which users use how much and what kinds of computer resources
- o **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 eachother

  4**Protection**involvesensuringthatallaccesstosystemresourcesis controlled

  4 **Security** of the system from outsiders requires user authentication, extendstodefendingexternalI/Odevicesfrominvalidaccessattempts

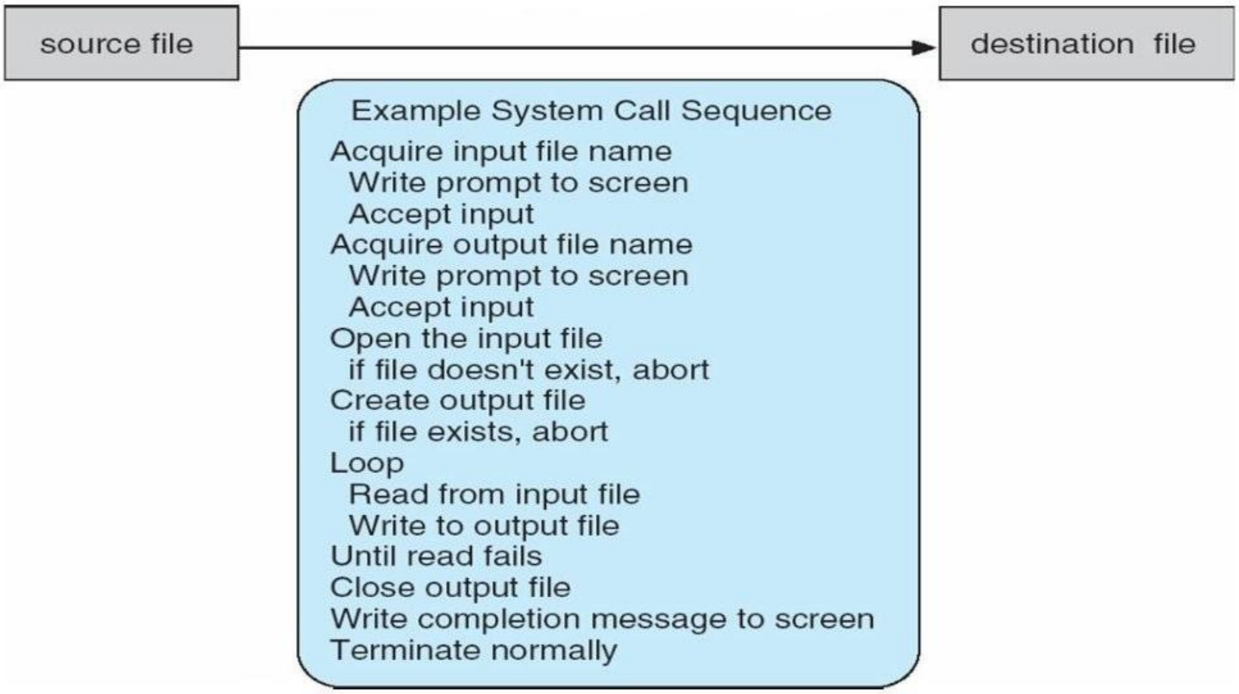
  4Ifasystemistobeprotectedandsecure,precautionsmustbeinstituted throughoutit.Achainisonlyasstrongasitsweakestlink.



System Calls:

- ✓ Programming interface to the services provided by theOS
- ✓ Typically written in a high-level language (C or C++)
- ✓ Mostly accessed by programs via a high-level **Application Program Interface (API)**rather than direct system calluse
- ✓ Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine(JVM)
- ✓ Why use APIs rather than systemcalls?

(Note that the system-call names used throughout this text are generic)



Example System Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Types of System Calls:

- ✓ Process control
    - end, abort
    - load, execute
    - create process, terminate process
    - get process attributes, set processattributes
    - wait for time
    - wait event, signal event
    - allocate and free memory
- ✓ File management
    - create file, delete file
    - open, close file
    - read, write, reposition
    - get and set file attributes
- ✓ Device management
    - request device, release device

  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- ✓ Information maintenance
  - get time or date, set time ordate
  - get system data, set systemdata
  - get and set process, file, or device attributes
- ✓ Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach and detach remote devices

System Programs:

- ✓ System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Statusinformation
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
  - Most users' view of the operation system is defined by system programs, not the actual system calls
- ✓ Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
  - **File management -** Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- ✓ **Status information**
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugginginformation

- Typically, these programs format and print the output to the terminal or other output devices
- Some systems implement a registry - used to store and retrieve configuration information

✓ **File modification**

- Text editors to create and modify files
- Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided

✓ **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machinelanguage

✓ **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computersystems

- Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

Operating-System Debugging:

✓ **Debugging**is finding and fixing errors, or**bugs**

✓ OSes generate **log files**containing errorinformation

✓ Failure of an application can generate **core dump**file capturing memory of theprocess

✓ Operating system failure can generate **crash dump**file containing kernel memory

✓ Beyond crashes, performance tuning can optimize systemperformance

✓ Kernighan's Law: "Debugging is twice as hard as writing the code in the firstplace. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

✓ DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems

- **Probes** fire when code is executed, capturing state data and sending it to consumers of those probes

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 startit
    - o Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it
    - o Sometimes two-step process where **boot block** at fixed location loads bootstrap loader
    - o When power initialized on system, execution starts at a fixed memorylocation

Firmware used to hold initial boot code

UNIT-2

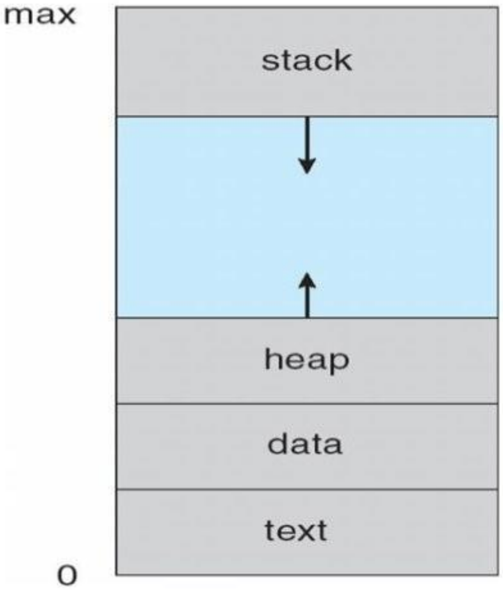
PROCESS THREADS, PROCESS SYNCHRONISATON, CPUSCHEDULING

Process Concept:

- ✓ An operating system executes a variety ofprograms:
  - Batch system –jobs
  - Time-shared systems – user programs ortasks
  - Textbook uses the terms *job* and *process* almostinterchangeably
- ✓ Process – a program in execution; process execution must progress in sequentialfashion
- ✓ A process includes:
  - program counter
  - stack
  - data section

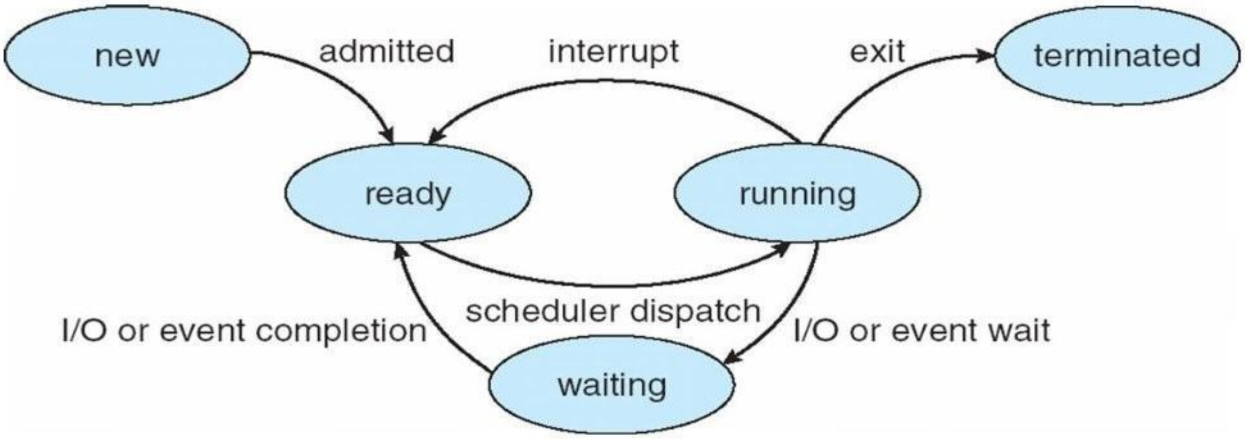
The Process:

- ✓ Multiple parts
  - The program code, also called **textsection**
  - Current activity including **program counter**, processorregisters
  - **Stack** containing temporary data
  - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during runtime
- ✓ Program is passive entity, process isactive
  - Program becomes process when executable file loaded intomemory
- ✓ Execution of program started via GUI mouse clicks, command line entry of its name,etc
- ✓ One program can be several processes
  - Consider multiple users executing the same program



Process State:

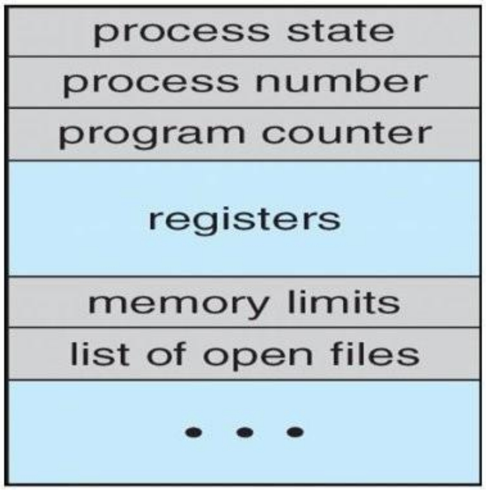
- ✓ As a process executes, it changes*state*
  - o **new**: The process is beingcreated
  - o **running**: Instructions are being executed
  - o **waiting**: The process is waiting for some event tooccur
  - o **ready**: The process is waiting to be assigned to aprocessor
  - o **terminated**: The process has finishedexecution



Process Control Block (PCB):

Information associated with each process

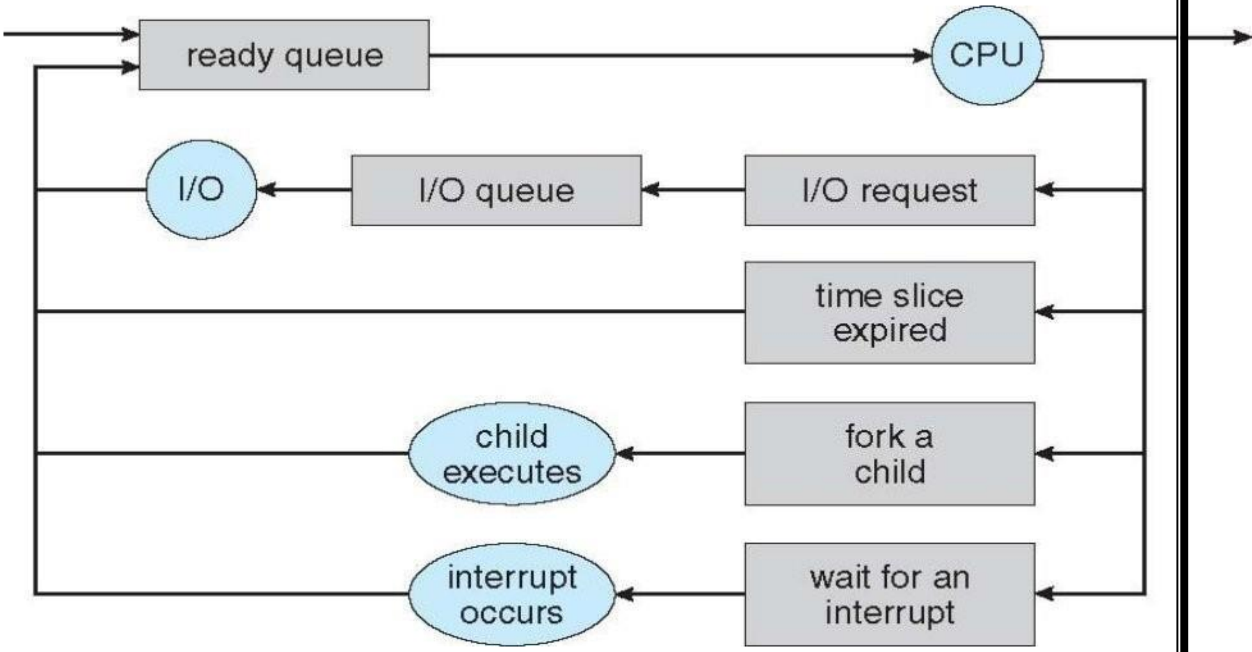
- ✓ Process state
- ✓ Program counter
- ✓ CPU registers
- ✓ CPU scheduling information
- ✓ Memory-management information
- ✓ Accounting information
- ✓ I/O status information



Process Scheduling:

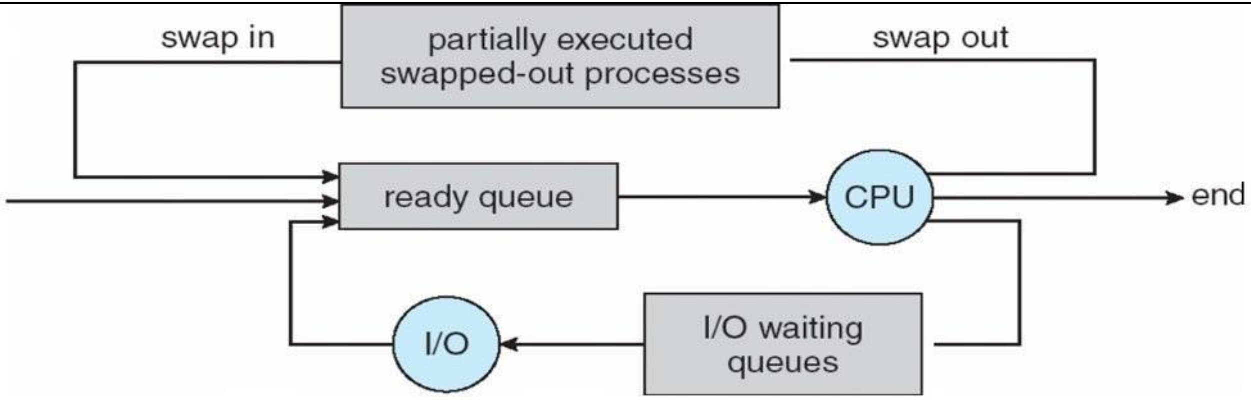
- ✓ Maximize CPU use, quickly switch processes onto CPU for timesharing
- ✓ Process scheduler selects among available processes for next execution onCPU
- ✓ Maintains scheduling queues of processes

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



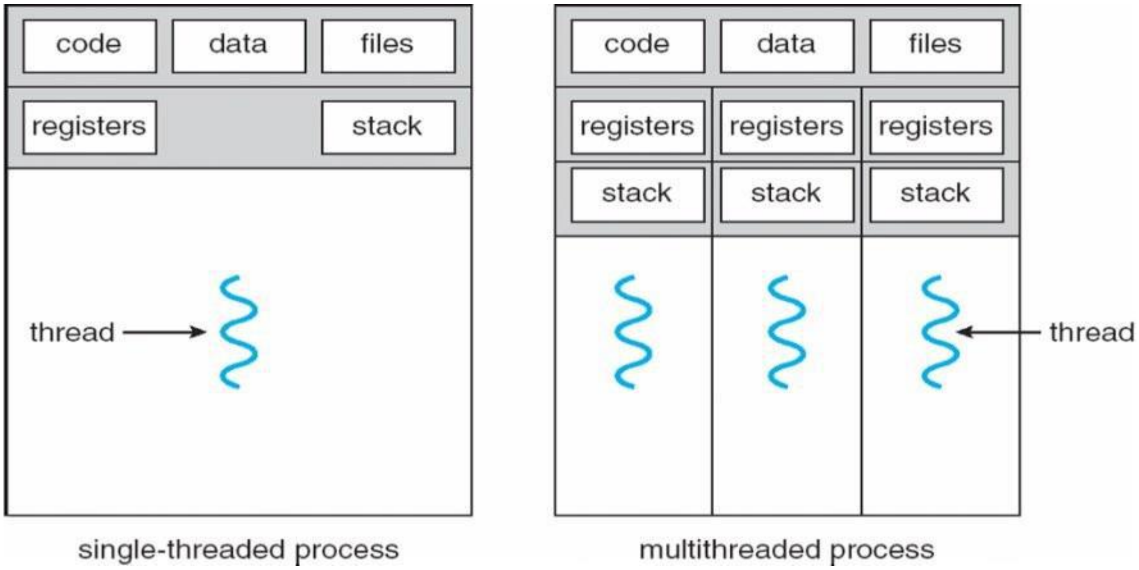
Schedulers:

- ✓ **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
  - Sometimes the only scheduler in a system
- ✓ Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)
- ✓ Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may beslow)
- ✓ The long-term scheduler controls the *degree ofmultiprogramming*
- ✓ Processes can be described aseither:
  - **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

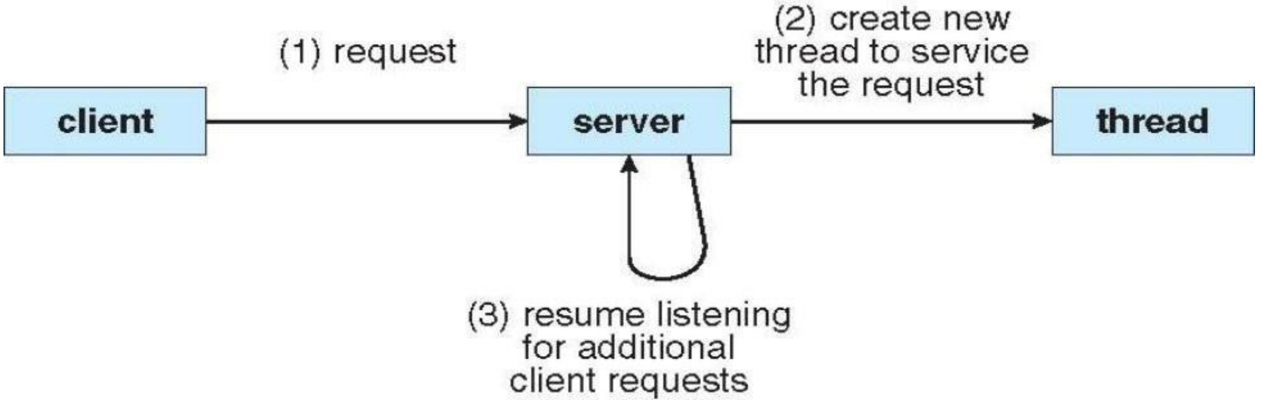


Threads

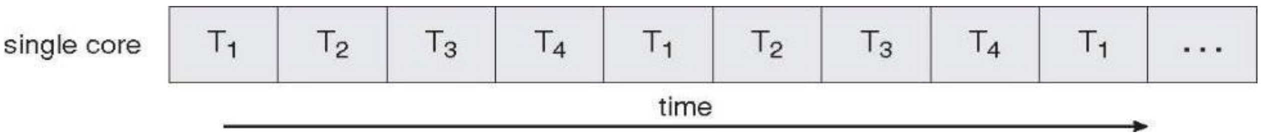
- ✓ Threads run within application
- ✓ Multiple tasks with the application can be implemented by separatethreads
  - ○ Update display
  - ○ Fetch data
  - ○ Spell checking
  - ○ Answer a network request
- ✓ Process creation is heavy-weight while thread creation islight-weight
- ✓ Can simplify code, increase efficiency
- ✓ Kernels are generally multithreaded



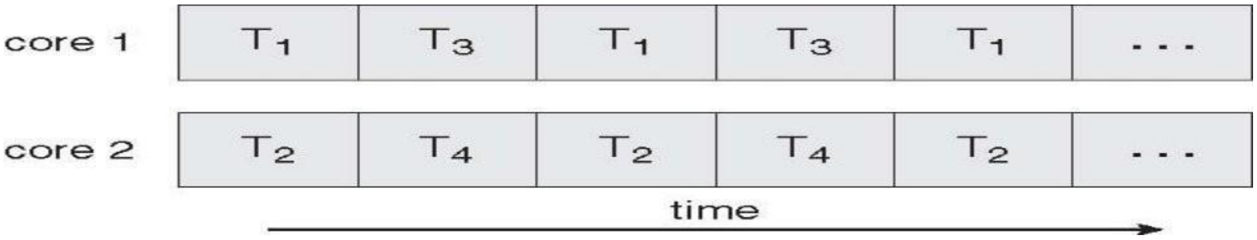
Multithreaded Server Architecture:


client
(1) request
server
(2) create new thread to service the request
thread
(3) resume listening for additional client requests

Concurrent Execution on a Single-core System



Parallel Execution on a Multicore System



User Threads:

- Thread management done by user-level threads library
- Three primary thread libraries:
  - POSIX **Pthreads**
  - Win32 threads
  - Java threads

Kernel Threads:

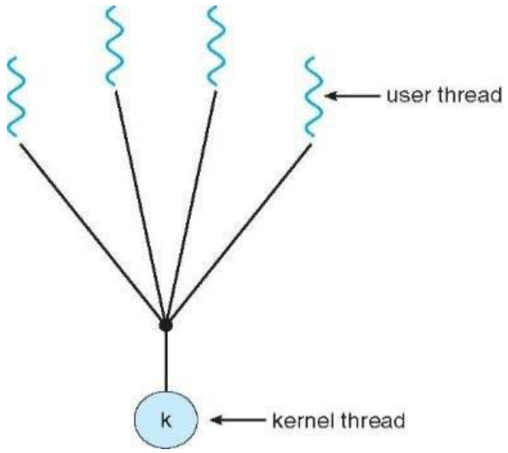
- Supported by the Kernel
- Examples
  - Windows XP/2000
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Multithreading Models:

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

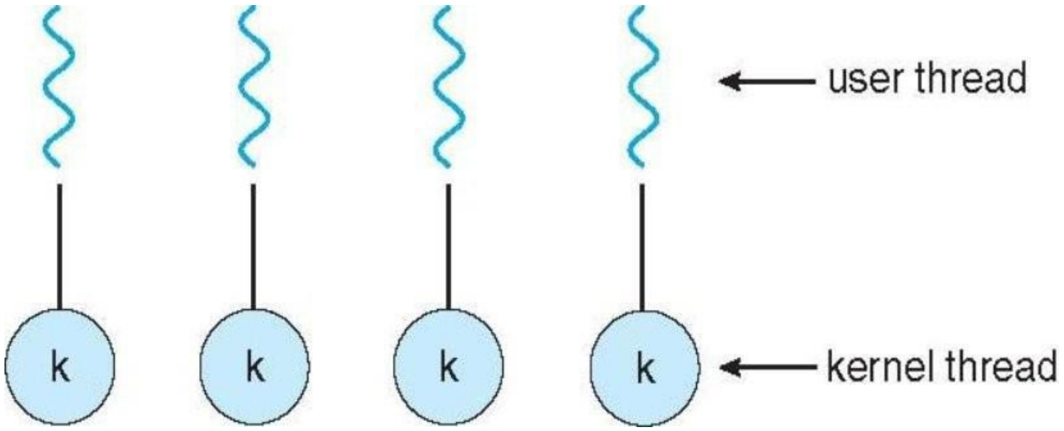
Many-to-One

- Many user-level threads mapped to single kernel thread
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



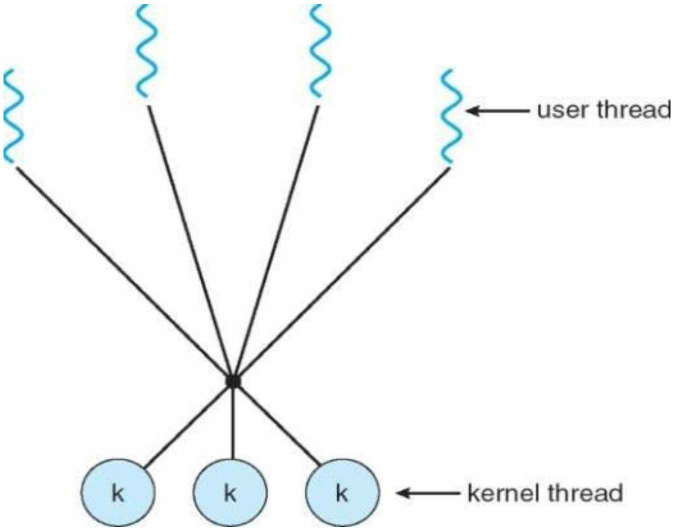
One-to-One:

- 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 kernelthreads
- Allows the operating system to create a sufficient number of kernelthreads
- Solaris prior to version 9
- Windows NT/2000 with the *ThreadFiber*package

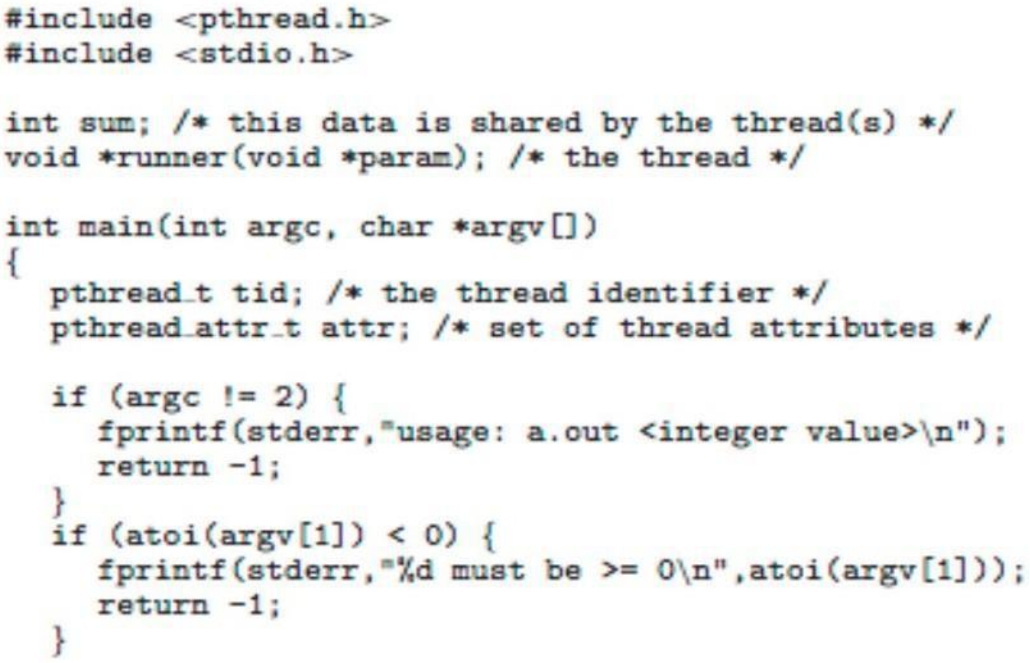


Thread Libraries:

- **Thread library** provides programmer with API for creating and managingthreads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Pthreads

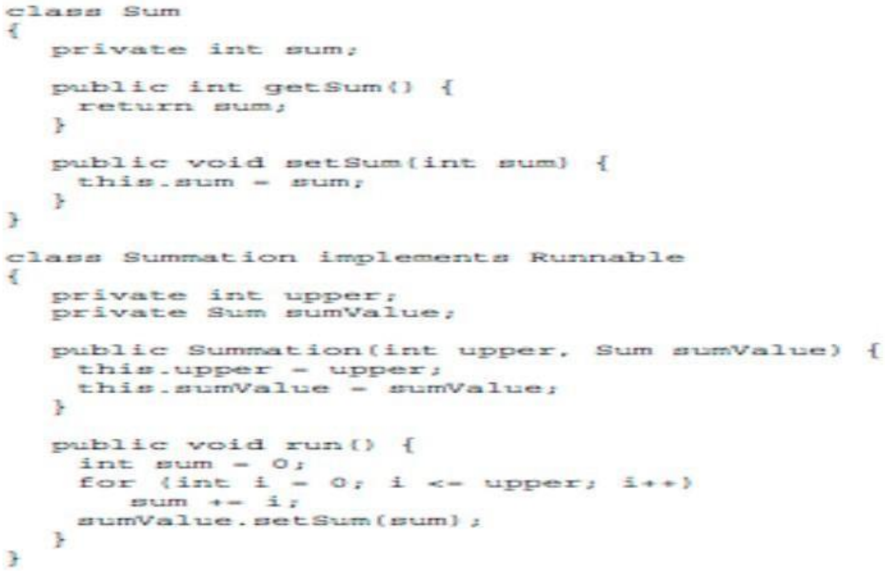
- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation andsynchronization
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OSX)



```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
  pthread_t tid; /* the thread identifier */
  pthread_attr_t attr; /* set of thread attributes */

  if (argc != 2) {
    fprintf(stderr,"usage: a.out <integer value>\n");
    return -1;
  }
  if (atoi(argv[1]) < 0) {
    fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
    return -1;
  }
```

Java Threads:

- Java threads are managed by the JVM
- Typically implemented using the threads model provided by underlyingOS
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface



```
class Sum
{
   private int sum;

   public int getSum() {
    return sum;
   }

   public void setSum(int sum) {
    this.sum = sum;
   }
}

class Summation implements Runnable
{
   private int upper;
   private Sum sumValue;

   public Summation(int upper, Sum sumValue) {
    this.upper = upper;
    this.sumValue = sumValue;
   }

   public void run() {
    int sum = 0;
    for (int i = 0; i <= upper; i++)
       sum += i;
    sumValue.setSum(sum);
   }
}
```

Threading Issues:

- Semantics of **fork()** and **exec()** systemcalls

- **Thread cancellation** of **targetthread**
  - Asynchronous or deferred
  - **Signal**handling
  - Synchronous and asynchronous
- **Thread pools**
- **Thread-specific data**
  - Create Facility needed for data private to thread
- **Scheduler activations**

Thread Cancellation:

- Terminating a thread before it has finished
- Two general approaches:
  - Asynchronous cancellation terminates the target thread immediately.
  - Deferred cancellation allows the target thread to periodically check if it should be cancelled.

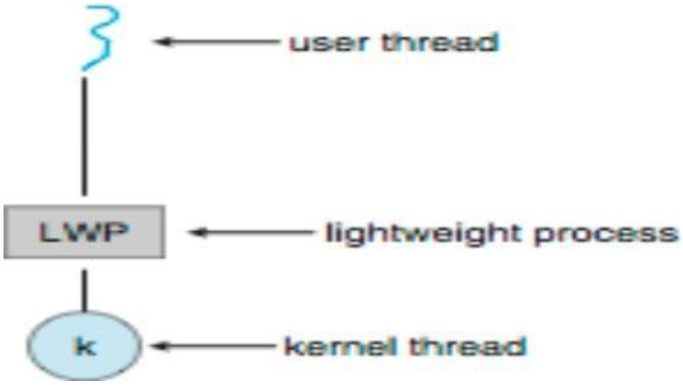
Thread Pools:

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool

Scheduler Activations:

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Scheduler activations provide upcalls - a communication mechanism from the kernel to the thread library
- This communication allows an application to maintain the correct number kernelthreads

Lightweight Processes



Critical Section Problem:

∑ Consider system of n processes {p_0, p_1, …p_{n-1}}

∑ Each process has **critical section** segment ofcode

o Process may be changing common variables, updating table, writing file,etc

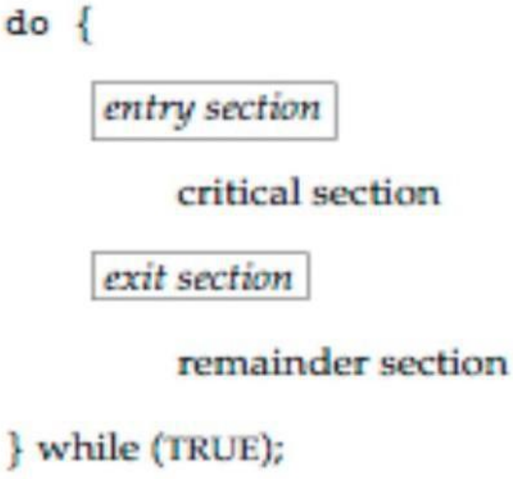
o When one process in critical section, no other may be in its criticalsection

∑ Critical section problem is to design protocol to solve this

∑ Each process must ask permission to enter critical section in **entry section**, may follow critical section with **exit section**, then **remaindersection**

∑ Especially challenging with preemptive kernels

General structure of process $p_i$is



```
do {
    entry section
        critical section
    exit section
        remainder section
} while (TRUE);
```

Figure 6.1 General structure of a typical process P_i.

Solution to Critical-Section Problem:

1. Mutual Exclusion - If process P_i 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

ó Assume that each process executes at a nonzerospeed

ó No assumption concerning relative speed of the nprocesses

Peterson's Solution:

∑ Two process solution

∑ Assume that the LOAD and STORE instructions are atomic; that is, cannot beinterrupted

∑ The two processes share two variables:

o int**turn**;

o Boolean**flag[2]**

o The variable **turn** indicates whose turn it is to enter the criticalsection

∑ The **flag** array is used to indicate if a process is ready to enter the critical section.**flag[i]** = true implies that process **P$_i$**is ready!

Algorithm for Process P$_i$

```
do {
        flag[i] = TRUE;
        turn = j;
        while (flag[j] && turn == j);
                critical section
        flag[i] = FALSE;
                remainder section
} while (TRUE);
```

∑ Provable that

1. Mutual exclusion is preserved
2. Progress requirement is satisfied
3. Bounded-waiting requirement is met

Synchronization Hardware:

∑ Many systems provide hardware support for critical sectioncode

- Uniprocessors – could disableinterrupts
    - Currently running code would execute without preemption
    - Generally too inefficient on multiprocessor systems
        - Operating systems using this not broadly scalable
        - Modern machines provide special atomic hardware instructions
        - Atomic = non-interruptable
    - Either test memory word and set value
    - Or swap contents of two memorywords

```
do {
          acquire lock
                    critical section
          release lock
                    remainder section
     } while (TRUE);
```

Semaphore:

- Synchronization tool that does not require busy waiting
- Semaphore *S* – integervariable
- Two standard operations modify S: wait() and signal()
    - Originally called P() andV()
- 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 unrestricteddomain

∑ **Binary** semaphore – integer value can range only between 0 and 1; can be simpler to implement

l Also known as **mutexlocks**

∑ Can implement a counting semaphore S as a binarysemaphore

∑ Provides mutual exclusion

```
Semaphore mutex;    // initialized to 1

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

o Could now have **busy waiting** in critical section implementation

4 But implementation code is short

4 Little busy waiting if critical section rarely occupied

4Notethatapplicationsmayspendlotsoftimeincriticalsectionsand therefore this is not a goodsolution

Deadlock and Starvation

∑ **Deadlock** – two or more processes are waiting indefinitely for an event that can be caused by only one of the waitingprocesses

∑ Let S and Q be two semaphores initialized to1

```
            P0P1

        wait(S);                    wait (Q);

        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
  - Solved via **priority-inheritance protocol**

Classical Problems ofSynchronization:

- Classical problems used to test newly-proposed synchronization schemes
  - Bounded-Buffer Problem
  - Readers and Writers Problem
  - Dining-Philosophers Problem

Bounded-Buffer Problem

- *N* buffers, each can hold one item
- Semaphore mutex initialized to the value 1
- Semaphore full initialized to the value 0
- Semaphore empty initialized to the value N

- The structure of the producer process

```
do {
                // produce an item in
     nextpwait (empty);
     wait(mutex);
                   // add the item to the buffer
     signal (mutex);
     signal (full);
          } while (TRUE);
```

∑ The structure of the consumer process

```
          do {
wait (full);
wait (mutex);
                    // remove an item from buffer to nextc
signal (mutex);
signal (empty);
                  // consume the item in nextc
               } while (TRUE);
```

Readers-Writers Problem:

∑ A data set is shared among a number of concurrentprocesses

- Readers – only read the data set; they do **not** perform anyupdates
- Writers – can both read and write

∑ Problem – allow multiple readers to read at the same time

- Only one single writer can access the shared data at the sametime
- Several variations of how readers and writers are treated – all involvepriorities

∑ Shared Data

- Data set
- Semaphore mutex initialized to 1
- Semaphore wrt initialized to 1
- Integer readcount initialized to 0

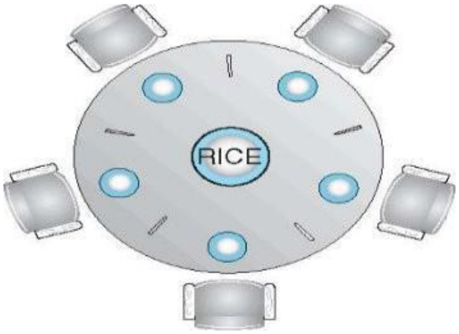
∑ The structure of a writer process

```
do {
wait (wrt) ;

                    //    writing is performed
signal (wrt) ;
         } while (TRUE);
```

∑ The structure of a reader process

```
wait (mutex)

;readcount ++

;

if (readcount == 1)     wait (wrt) ;


signal (mutex)

   // reading is performed

wait (mutex) ;

readcount  - -;

if (readcount == 0)

                            signal (wrt) ;

signal (mutex) ;

           } while (TRUE);
```

Dining-Philosophers Problem



∑ Philosophers spend their lives thinking and eating

∑ Don’t interact with their neighbors, occasionally try to pick up 2 chopsticks (one at a time) to eat from bowl

 o Need both to eat, then release both when done

∑ In the case of 5 philosophers

 o Shared data

⊿Bowlofrice(dataset)

Semaphorechopstick[5]initializedto1

- The structure of Philosopher*i*:

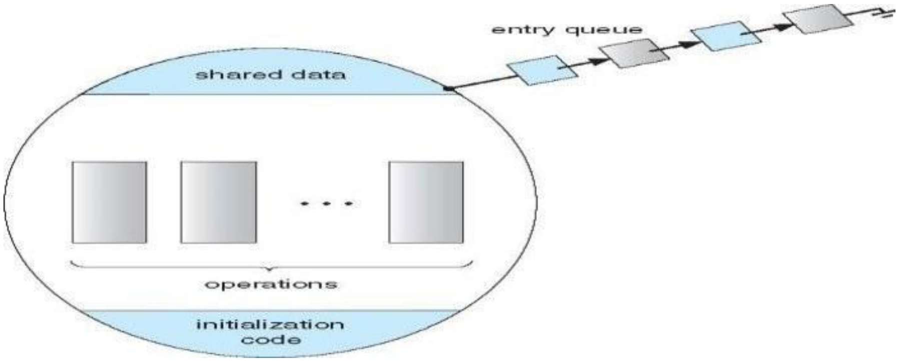
```
wait ( chopstick[i] );
        wait ( chopStick[ (i + 1) % 5] );
          // eat
        signal ( chopstick[i] );
        signal (chopstick[ (i + 1) % 5] );
   // think
} while (TRUE);
```

Monitors

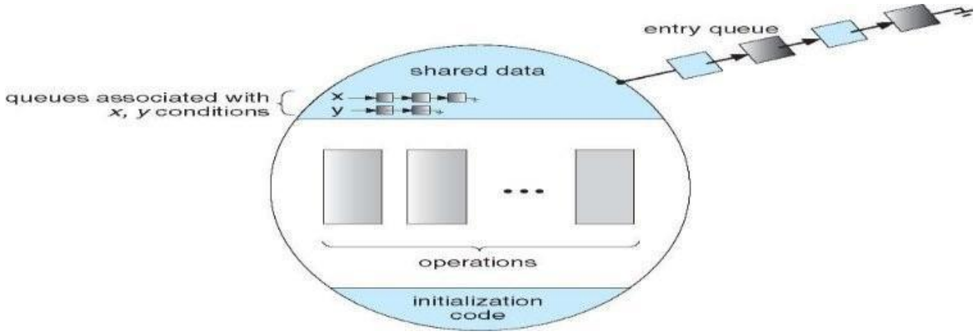
- A high-level abstraction that provides a convenient and effective mechanism for process synchronization
- *Abstract data type*, internal variables only accessible by code within theprocedure
- Only one process may be active within the monitor at atime
- But not powerful enough to model some synchronization schemes

```
monitor monitor-name
{
        // shared variable declarations
        procedure P1 (…) { …. }
        procedurePn (…) {……}
   Initialization code (…) { … }
        }
}
```

Schematic view of a Monitor



Monitor with Condition Variables



Scheduling Criteria:

- **CPU utilization** – keep the CPU as busy aspossible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particularprocess
- **Waiting time** – amount of time a process has been waiting in the readyqueue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharingenvironment)

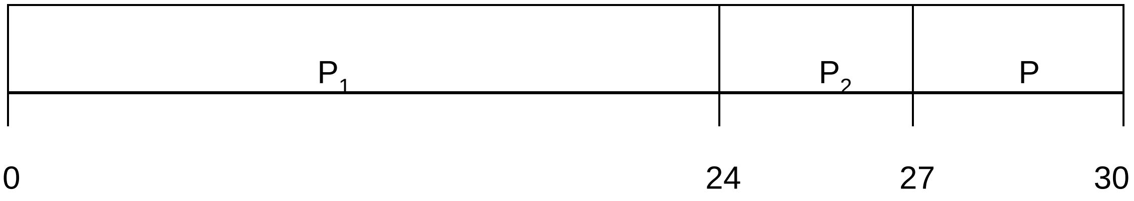
Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

First-Come, First-Served (FCFS) Scheduling

Process	BurstTime
P_1	24
P_2	3
P_3	3

Suppose that the processes arrive in the order: *P_1*,*P_2*, *P_3*
The Gantt Chart for the schedule is:

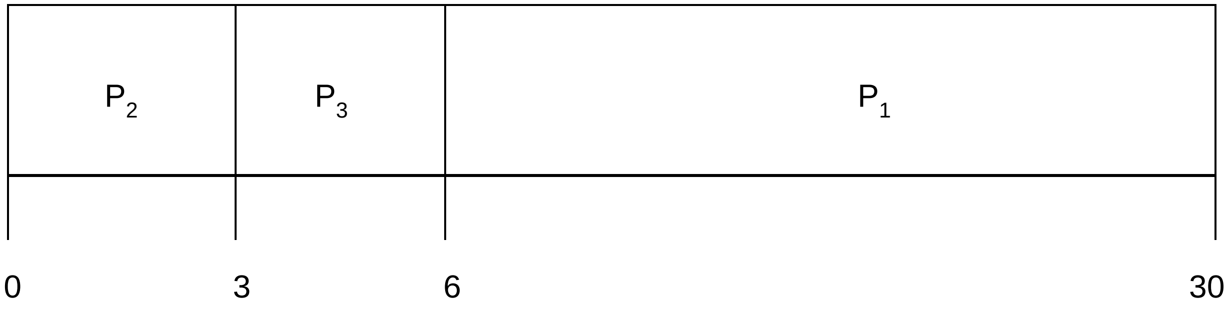


Waiting time for P_1= 0; P_2= 24; P_3= 27

- Average waiting time: (0 + 24 + 27)/3 = 17
- Suppose that the processes arrive in the order:

  P_2,P_3, P_1

- The Gantt chart for the schedule is:



- Waitingtime forP_1=6;P_2= 0;P_3=3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- **Convoy effect** - short process behind longprocess
  - Consider one CPU-bound and many I/O-boundprocesses

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPUburst
  - Use these lengths to schedule the process with the shortesttime
- SJF is optimal – gives minimum average waiting time for a given set ofprocesses
  - The difficulty is knowing the length of the next CPUrequest
  - Could ask the userProcessArrival

    Time BurstTime

P_1	0.0	6
P_2	2.0	8
P_3	4.0	7
P_4	5.0	3

SJF scheduling chart

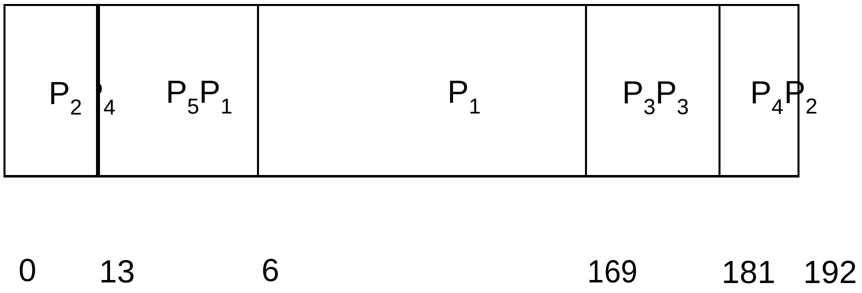
- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Priority Scheduling

- A priority number (integer) is associated with eachprocess
- TheCPUisallocatedtotheprocesswiththehighestpriority(smallestinteger∫highest priority)
  - Preemptive
  - Nonpreemptive
  - SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ∫**Starvation**– low priority processes may never execute
- Solution∫**Aging**– as time progresses increase the priority of the process

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Priority scheduling Gantt Chart



Average waiting time = 8.2 msec

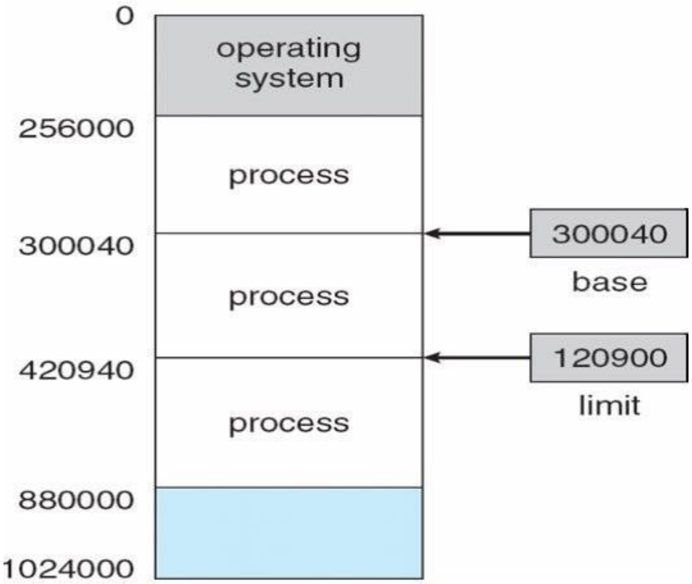
UNIT-3

Virtual Memory, Main Memory, Deadlocks

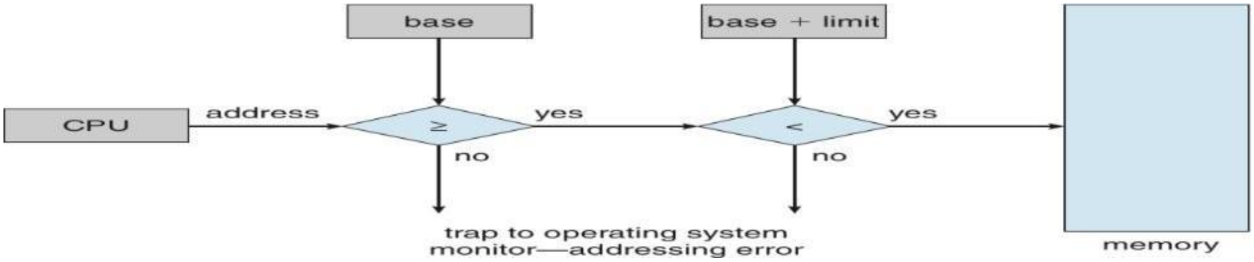
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can accessdirectly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (orless)
- Main memory can take many cycles
- **Cache** sits between main memory and CPUregisters
- Protection of memory required to ensure correct operation

Base and Limit Registers

- A pair of **base** and **limit** registers define the logical addressspace



Hardware Address Protection with Base and Limit Registers



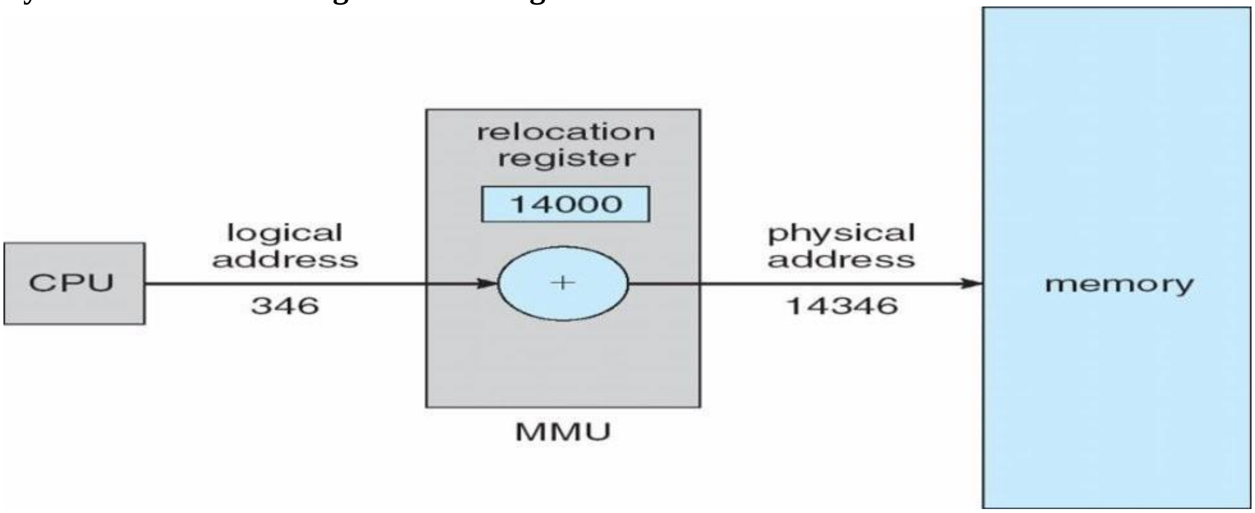
Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical addressspace** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memoryunit
  - 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
- **Logical address space** is the set of all logical addresses generated by aprogram
- **Physical address space** is the set of all physical addresses generated by aprogram

Memory-Management Unit (MMU)

- Hardware device that at run time maps virtual to physicaladdress
- Many methods possible, covered in the rest of this chapter
- To start, consider simple scheme where the value in the relocation register is added to every address generated by a user process at the time it is sent tomemory
  - Base register now called **relocationregister**
  - MS-DOS on Intel 80x86 used 4 relocation registers
- The user program deals with *logical* addresses; it never sees the *real* physicaladdresses
  - Execution-time binding occurs when reference is made to location inmemory
  - Logical address bound to physicaladdresses

Dynamic relocation using relocation register



Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurringcases
- No special support from the operating system isrequired
  - Implemented through programdesign
  - OS can help by providing libraries to implement dynamicloading

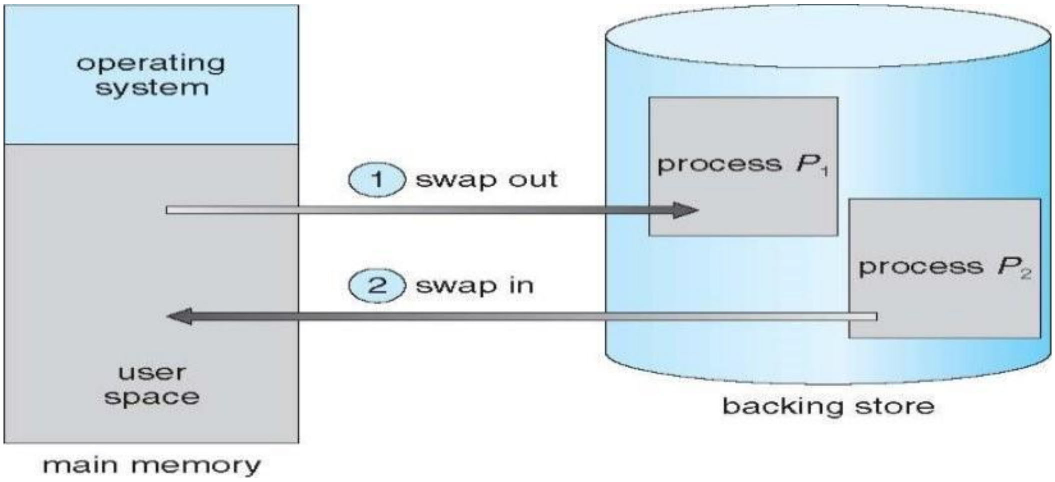
Dynamic Linking

- Static linking – system libraries and program code combined by the loader into the binary program image
- Dynamic linking –linking postponed until execution time
- Small piece of code, *stub*, used to locate the appropriate memory-resident libraryroutine
- Stub replaces itself with the address of the routine, and executes theroutine
- Operating system checks if routine is in processes' memoryaddress
  - If not in address space, add to addressspace
- Dynamic linking is particularly useful for libraries
- System also known as **sharedlibraries**
- Consider applicability to patching system libraries
  - Versioning may be needed

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physicalmemory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memoryimages
- **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 andexecuted
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped

- System maintains a **ready queue** of ready-to-run processes which have memory images on disk
- Does the swapped out process need to swap back in to same physicaladdresses?
- Depends on address binding method
    - Plus consider pending I/O to / from process memoryspace
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
    - Swapping normally disabled
    - Started if more than threshold amount of memory allocated
    - Disabled again once memory demand reduced below threshold

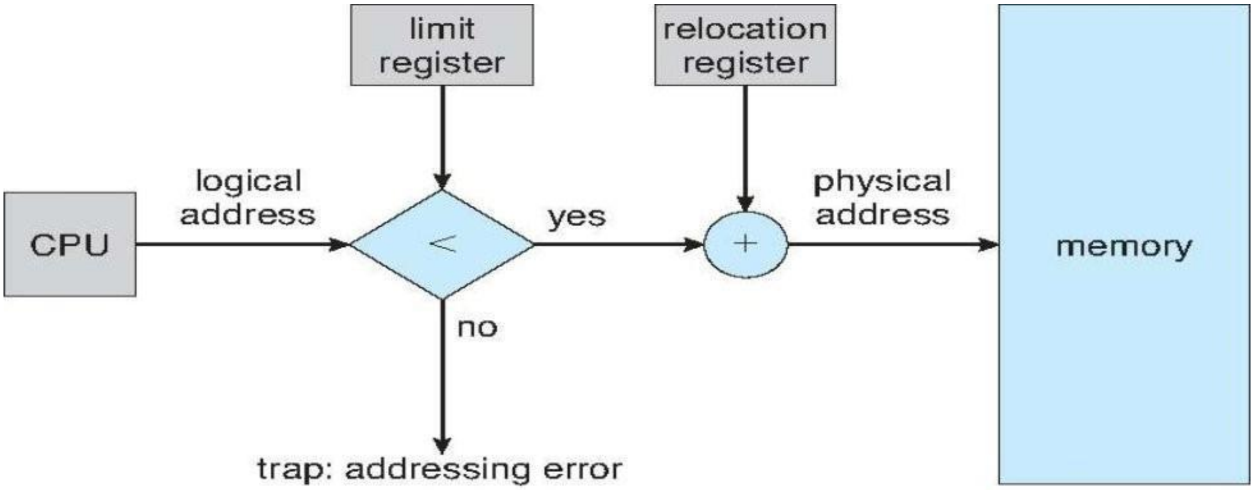


Contiguous Allocation

- Main memory usually into two partitions:
    - Resident operating system, usually held in low memory with interruptvector
    - User processes then held in highmemory
    - Each process contained in single contiguous section ofmemory
- Relocation registers used to protect user processes from each other, and fromchanging 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*

- Can then allow actions such as kernel code being **transient** and kernel changing size

Hardware Support for Relocation and Limit Registers



- ∑ Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - 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
  - Process exiting frees its partition, adjacent free partitionscombined
  - Operating system maintains information about:
    a) allocatedpartitionsb) free partitions (hole)

Dynamic Storage-Allocation Problem

- ∑ **First-fit**: Allocate the *first* hole that is bigenough
- ∑ **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 entirelist
  - Produces the largest leftover hole

Fragmentation

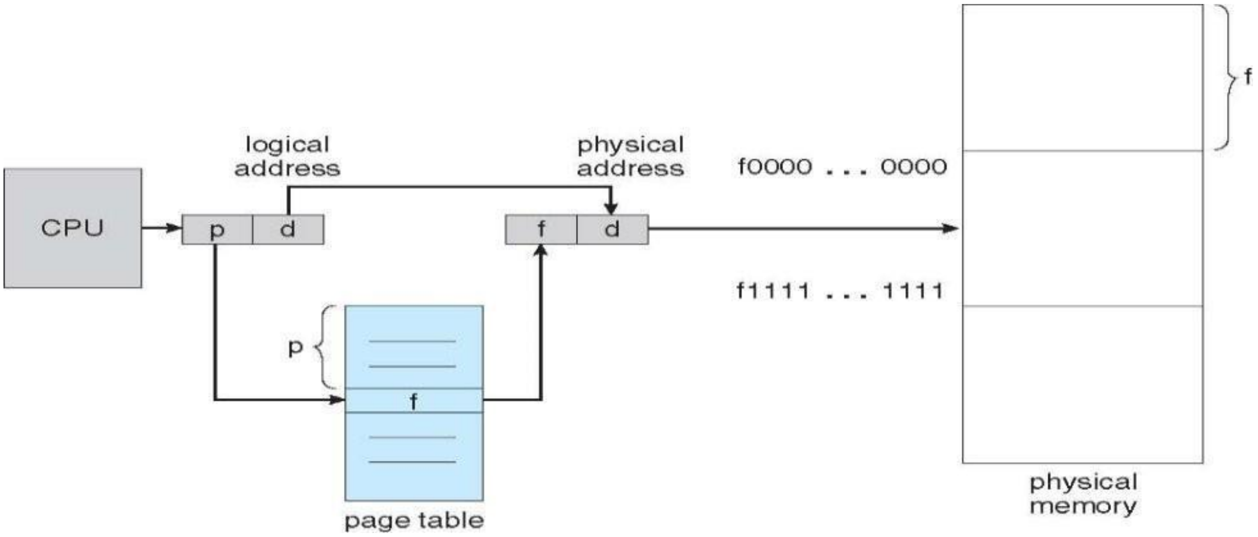
- ∑ **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- ∑ **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not beingused

- First fit analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost tofragmentation
    - 1/3 may be unusable ->**50-percentrule**

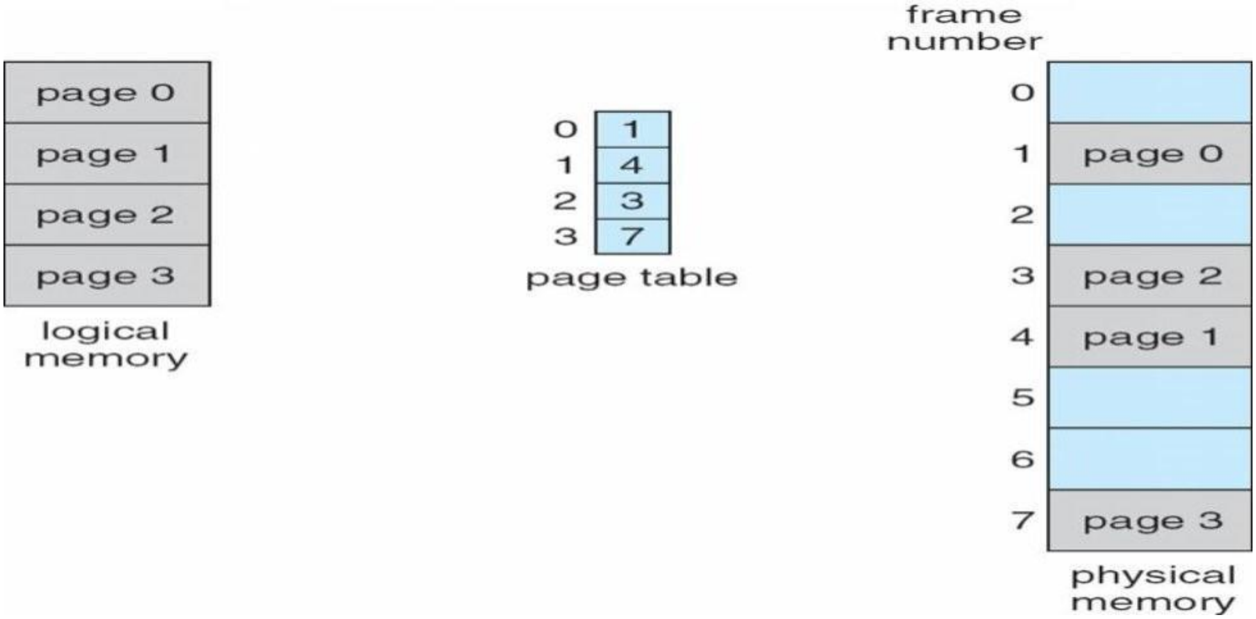
Paging

- Physical 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 16Mbytes
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size *N* pages, need to find *N* free frames and loadprogram
- Set up a **page table** to translate logical to physicaladdresses
- Backing store likewise split into pages
- Still have Internal fragmentation
- Address generated by CPU is dividedinto:
    - **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 memoryunit

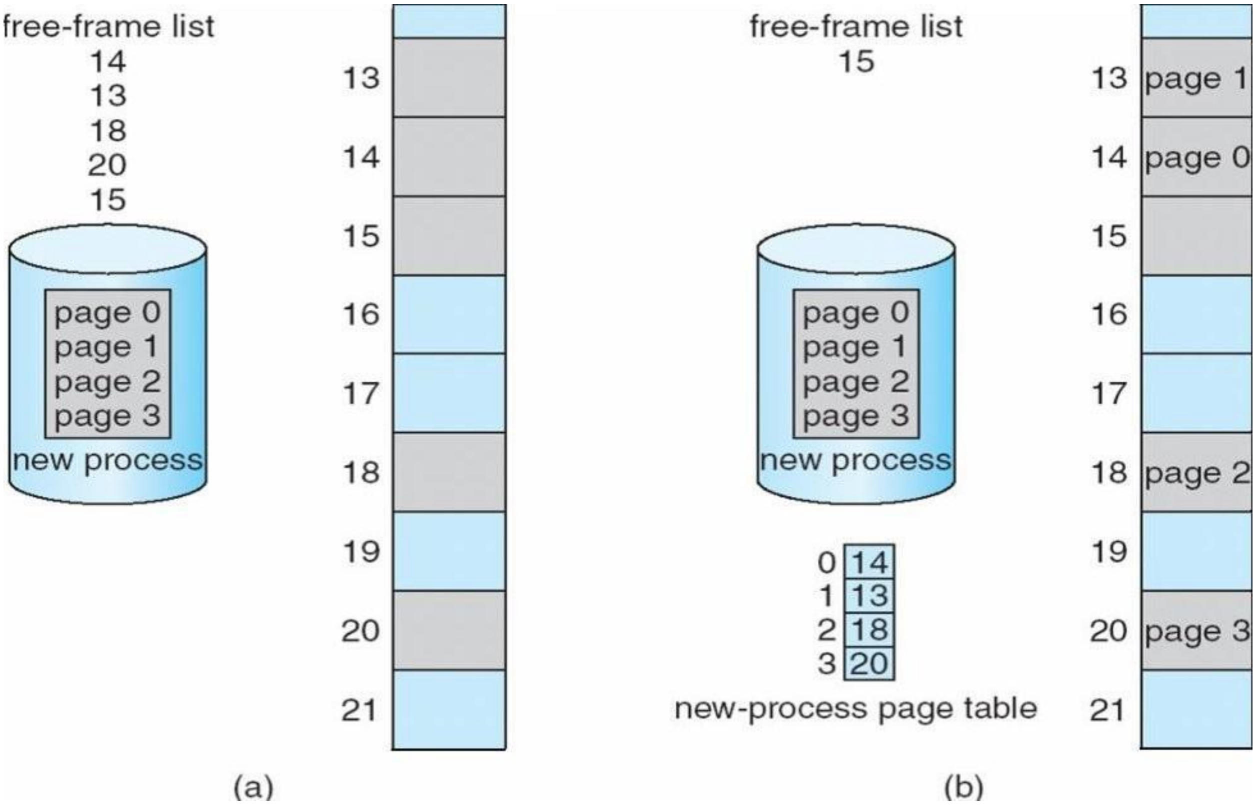
Paging Hardware



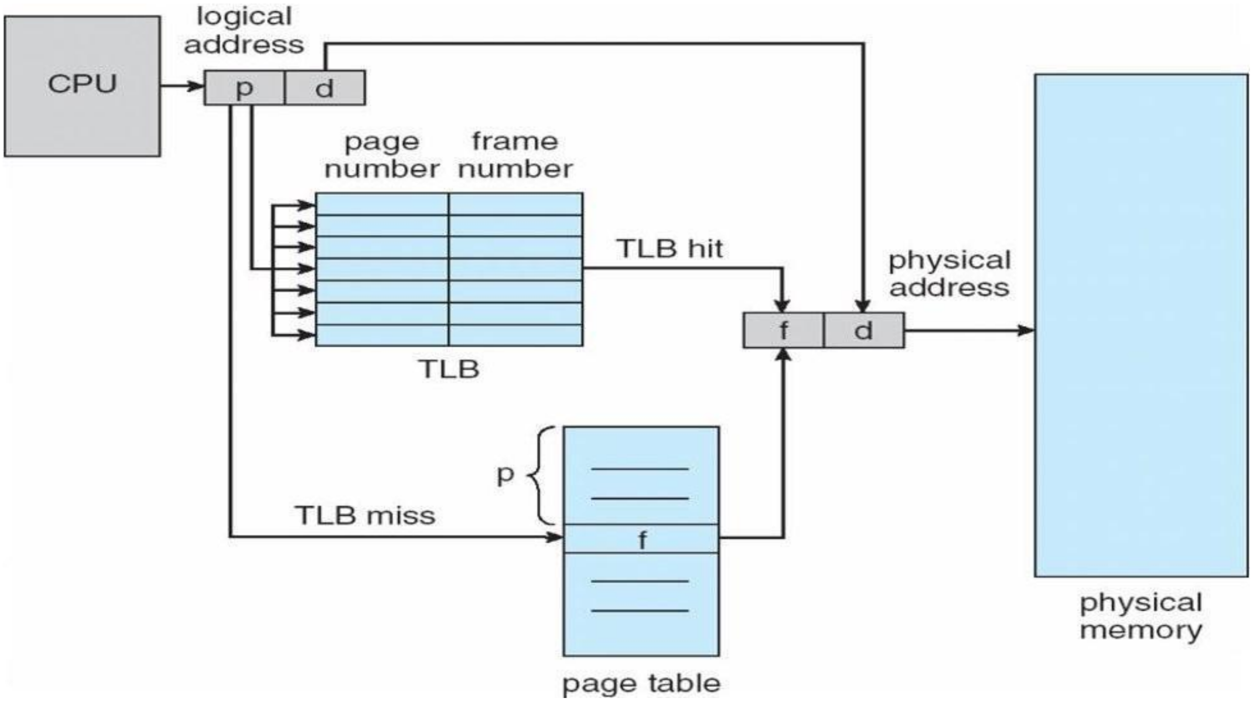
Paging Model of Logical and Physical Memory



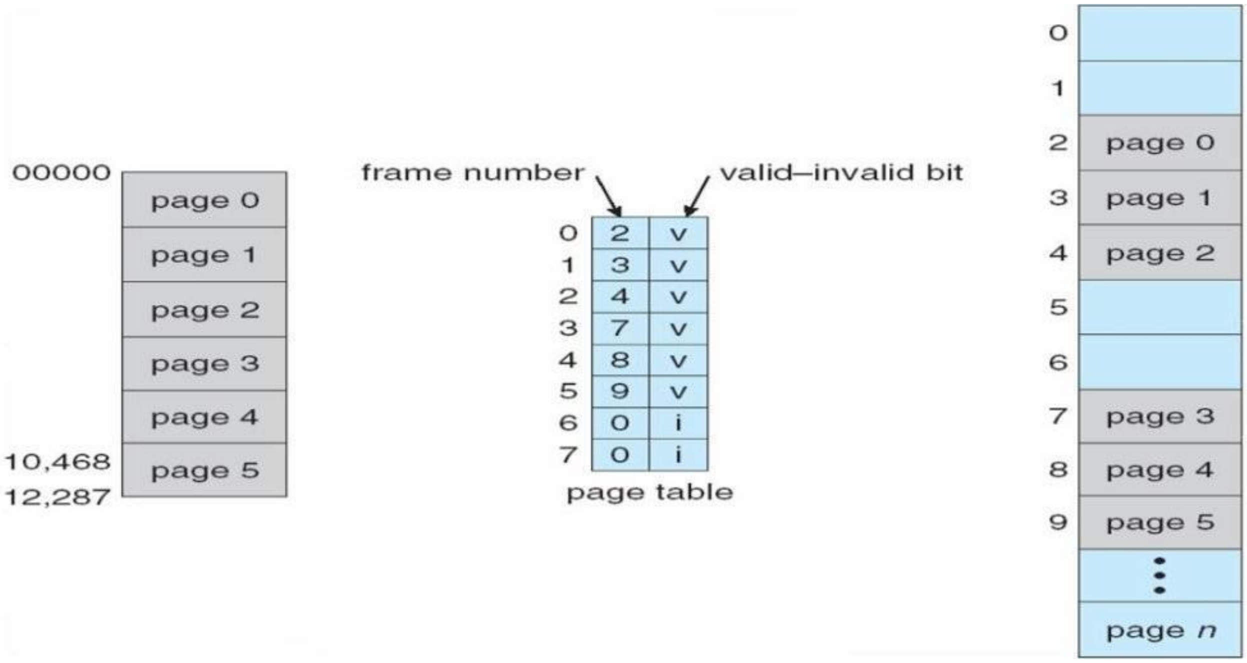
Free Frames



Paging Hardware With TLB

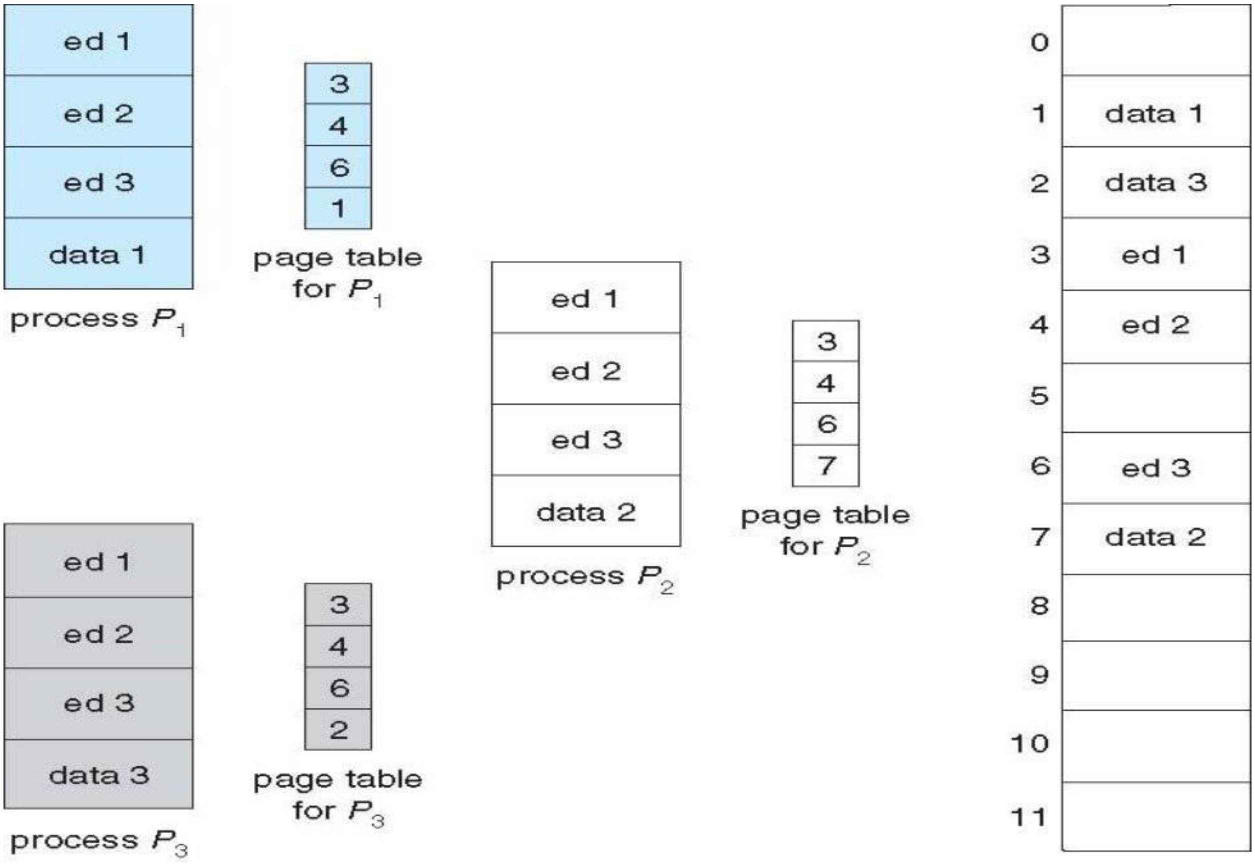


Valid (v) or Invalid (i) Bit In A Page Table



	frame number	valid–invalid bit
0	2	v
1	3	v
2	4	v
3	7	v
4	8	v
5	9	v
6	0	i
7	0	i

Shared Pages Example

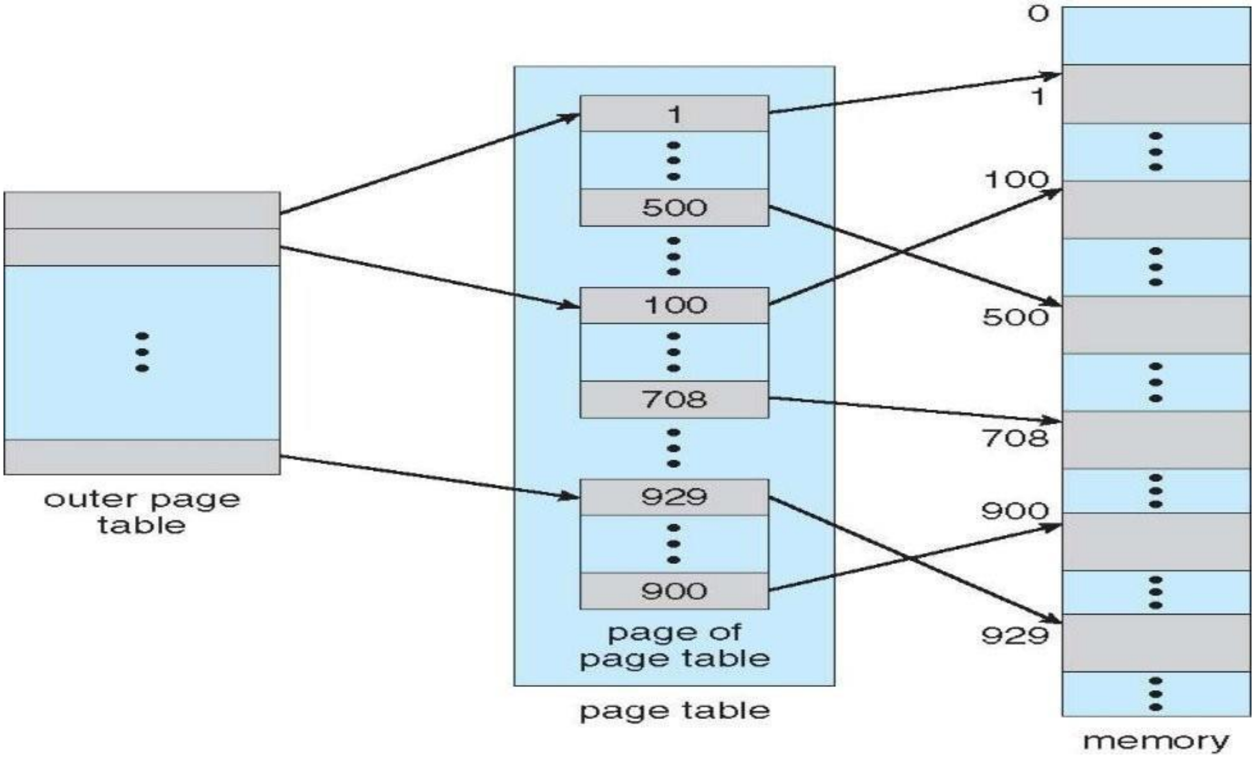


Structure of the Page Table

- ∑ Memory structures for paging can get huge using straight-forward methods
  - o Consider a 32-bit logical address space as on moderncomputers
  - o Page size of 4 KB(2^{12})
  - o Page table would have 1 million entries (2^{32} /$2^{12)}$
  - o If each entry is 4 bytes -> 4 MB of physical address space / memory for page table alone
    - 4 That amount of memory used to cost a lot
    - 4 Don’t want to allocate that contiguously in main memory
- ∑ Hierarchical Paging
- ∑ Hashed PageTables
- ∑ Inverted Page Tables

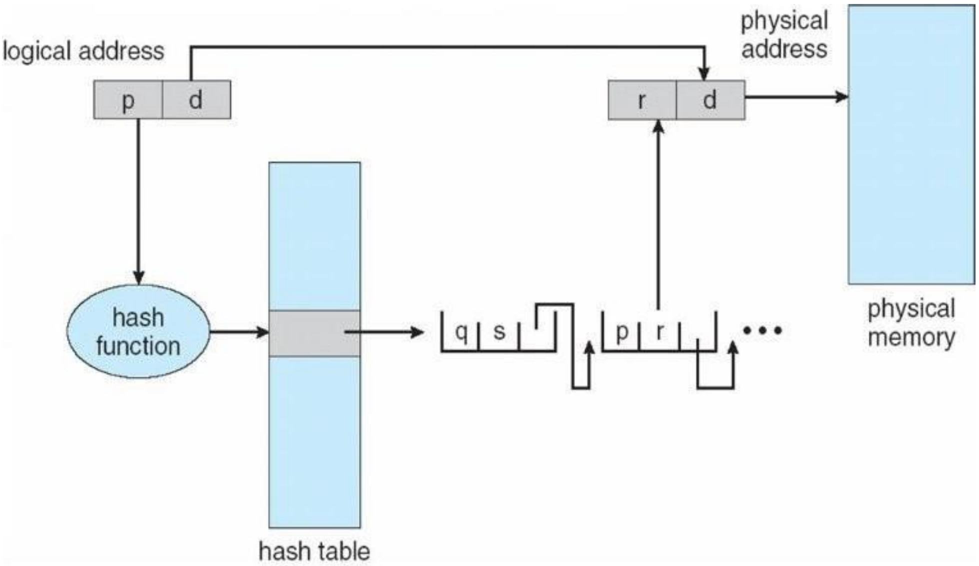
Hierarchical Page Tables

- Break up the logical address space into multiple pagetables
- A simple technique is a two-level page table
- We then page the page table



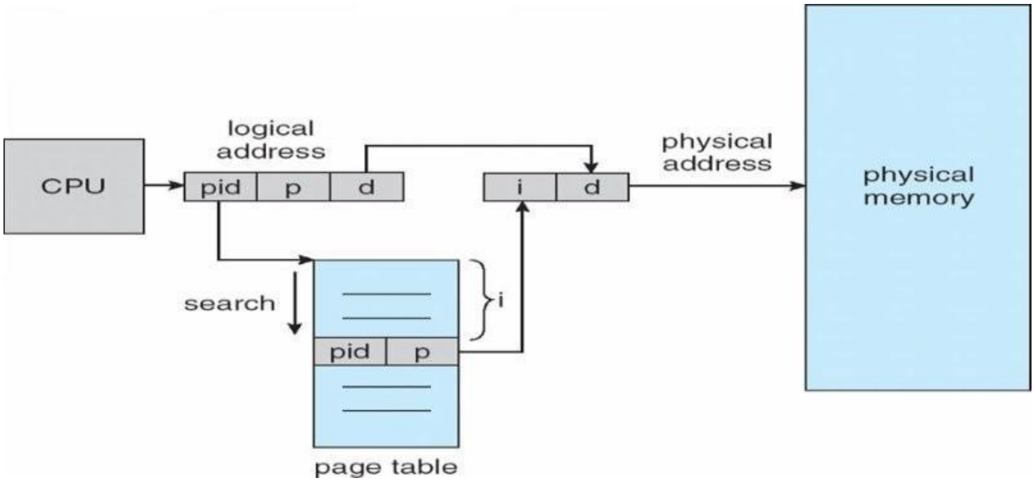
Hashed Page Tables

- Common in address spaces > 32bits
- The virtual page number is hashed into a page table
  - This page table contains a chain of elements hashing to the same location
  - Each element contains (1) the virtual page number (2) the value of the mapped page frame (3) a pointer to the next element
- Virtual page numbers are compared in this chain searching for a match
  - If a match is found, the corresponding physical frame isextracted



Inverted Page Table

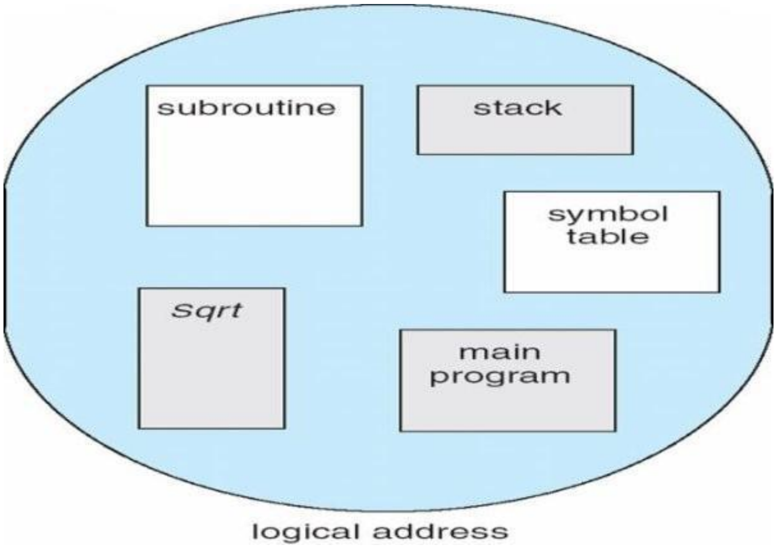
- ∑ Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
- ∑ One entry for each real page of memory
- ∑ Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns thatpage
- ∑ Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- ∑ Use hash table to limit the search to one — or at most a few — page-tableentries
  - o TLB can accelerate access
- ∑ But how to implement sharedmemory?
  - o One mapping of a virtual address to the shared physicaladdress



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



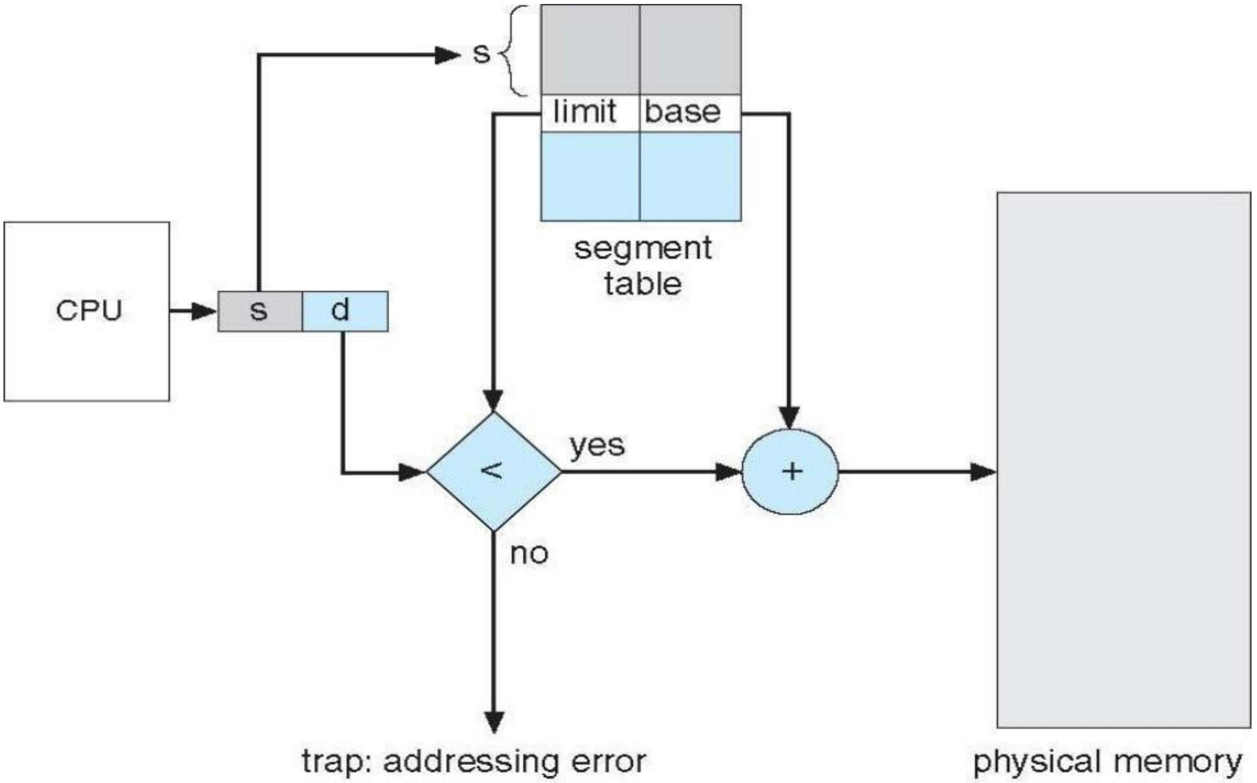
Segmentation Architecture

- Logical address consists of a twotuple:

  <segment-number, offset>,

- **Segment table** – maps two-dimensional physical addresses; each table entryhas:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of thesegment
  - **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**

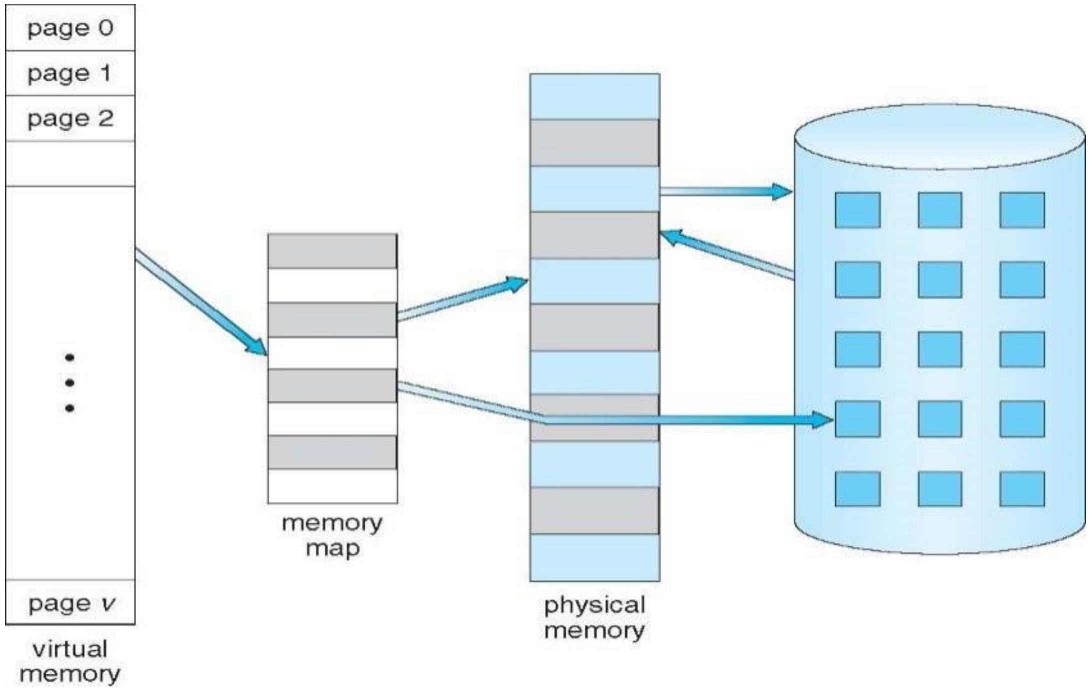


Virtual memory – separation of user logical memory from physical memory

- Only part of the program needs to be in memory for execution
- Logical address space can therefore be much larger than physical addressspace
- Allows address spaces to be shared by severalprocesses
- Allows for more efficient processcreation
- More programs running concurrently
- Less I/O needed to load or swapprocesses

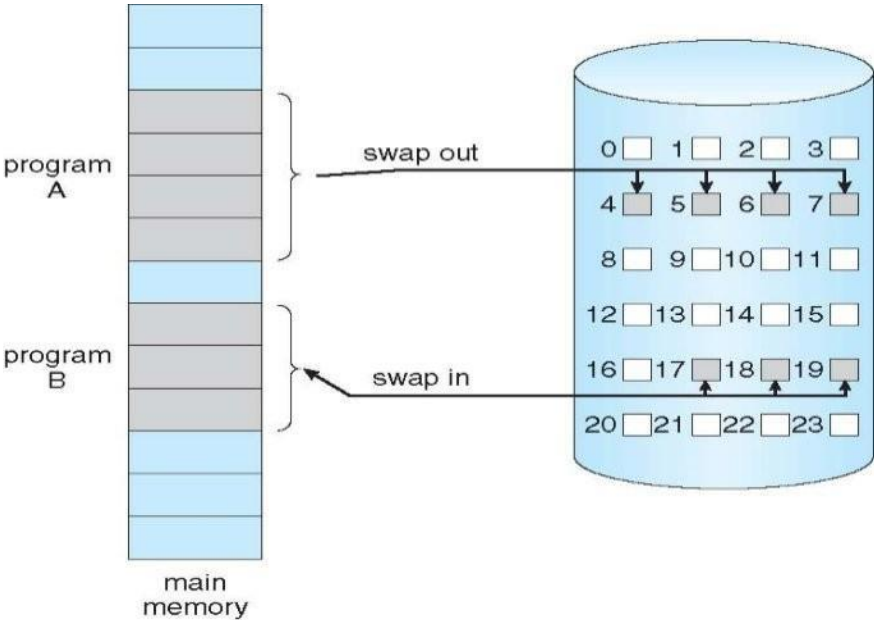
∑ Virtual memory can be implemented via:

- Demand paging
- Demand segmentation



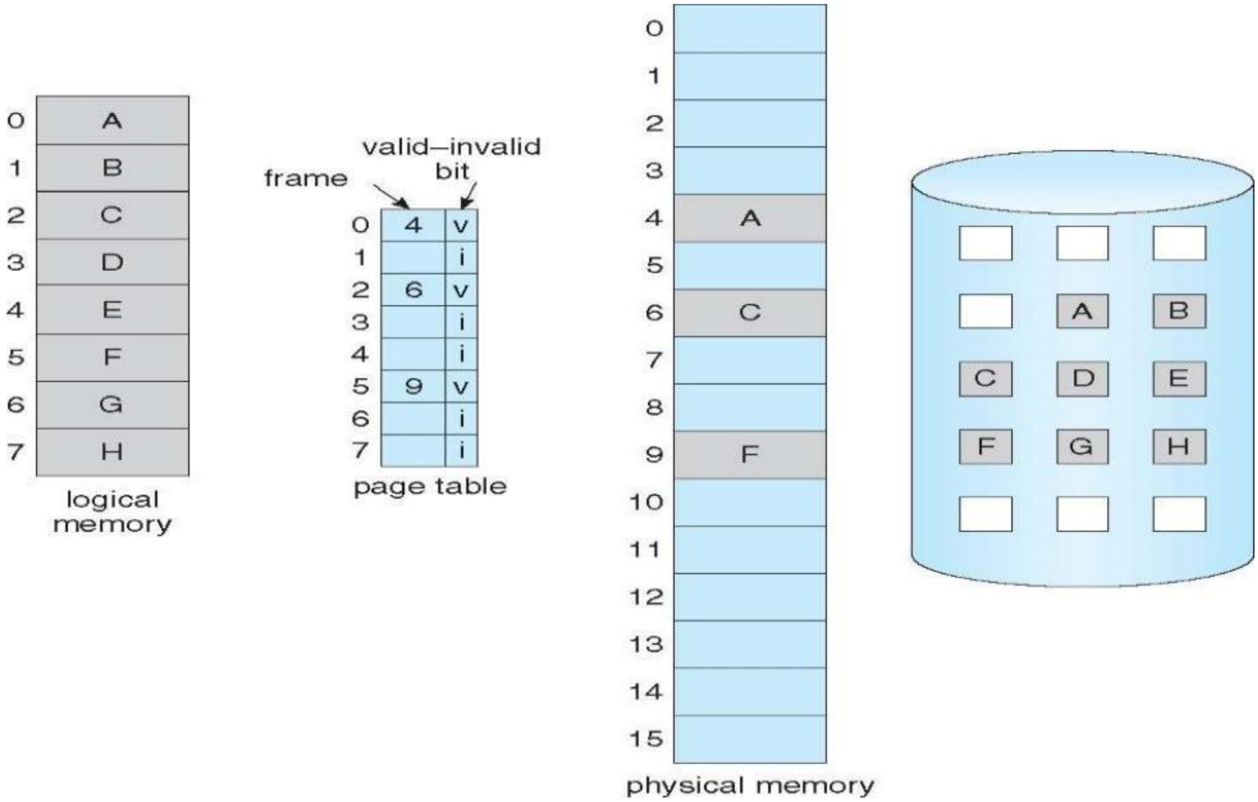
Demand Paging

- ∑ Could bring entire process into memory at loadtime
- ∑ Or bring a page into memory only when it isneeded
    - o Less I/O needed, no unnecessary I/O
  - o Less memory needed
  - o Faster response
  - o More users
- ∑ Page is needed ⇒ reference toit
  - o invalid reference ⇒abort
  - o not-in-memory ⇒ bring tomemory
- ∑ Lazy swapper – never swaps a page into memory unless page will beneeded
  - o Swapper that deals with pages is apager



Valid-Invalid Bit

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

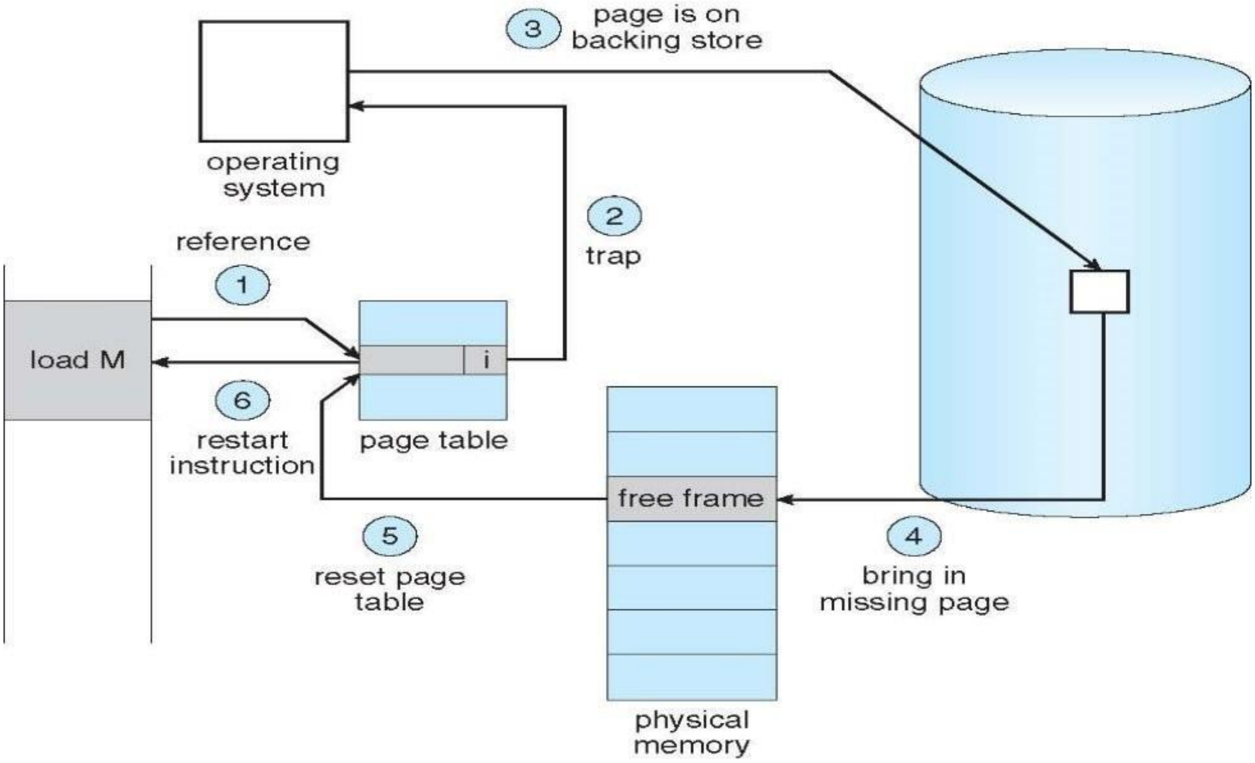


Page Fault

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

page fault

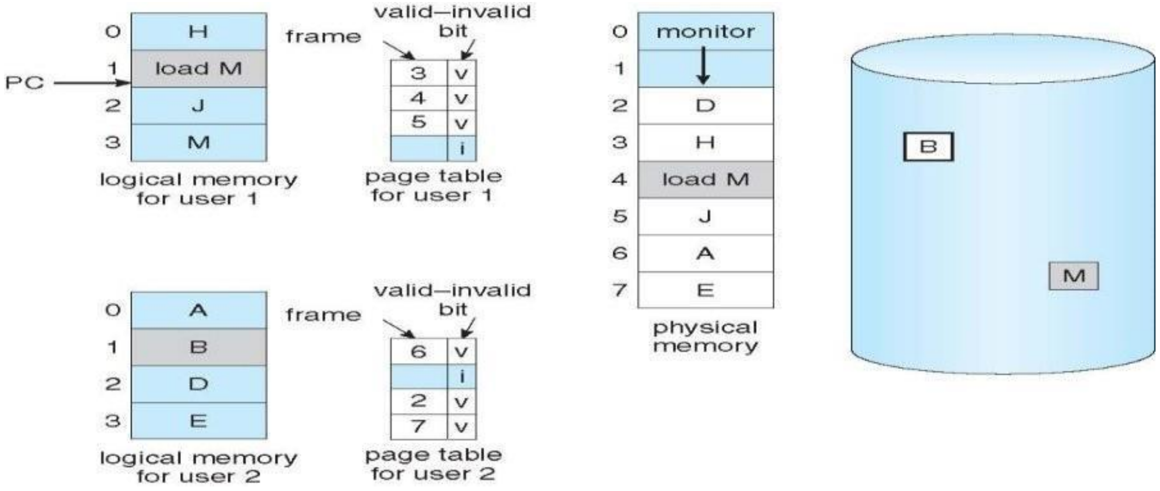
1. Operating system looks at another table to decide:
    - Invalid reference ⇒abort
    - Just not in memory
2. Get empty frame
3. Swap page into frame via scheduled disk operation
4. Reset tables to indicate page now in memory
   Set validation bit = **v**
5. Restart the instruction that caused the page fault
6. Extreme case – start process with *no* pages in memory
    - OS sets instruction pointer to first instruction of process, non-memory-resident ->page fault
    - And for every other process pages on firstaccess
    - **Pure demand paging**
7. Actually, a given instruction could access multiple pages -> multiple pagefaults
    - Pain decreased because of **locality ofreference**
8. Hardware support needed for demand paging
    - Page table with valid / invalidbit
    - Secondary memory (swap device with **swapspace**)
    - Instruction restart

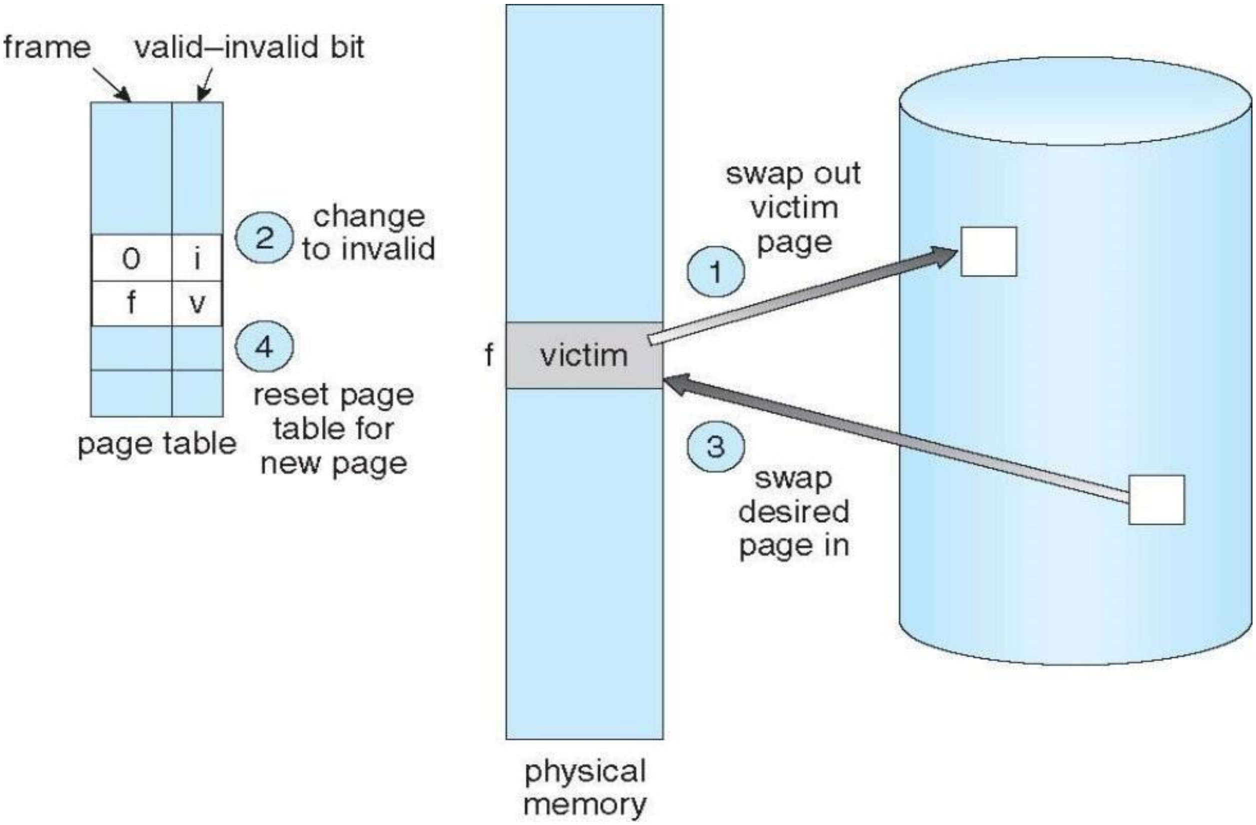


Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement
- Use **modify (dirty) 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

Need For Page Replacement





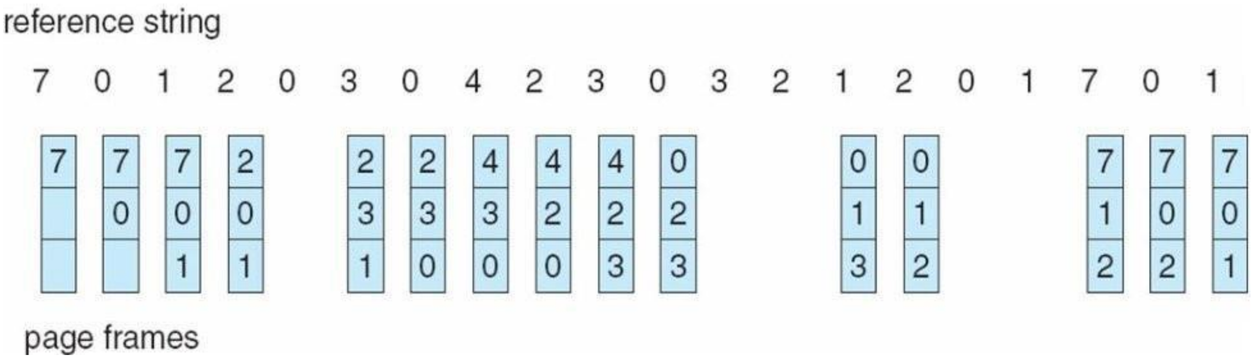
Page and Frame Replacement Algorithms

- **Frame-allocation algorithm**determines
  - How many frames to give eachprocess
  - Which frames to replace
- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access andre-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not fulladdresses
  - Repeated access to the same page does not cause a pagefault
- In all our examples, the reference string is

  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

First-In-First-Out (FIFO) Algorithm:

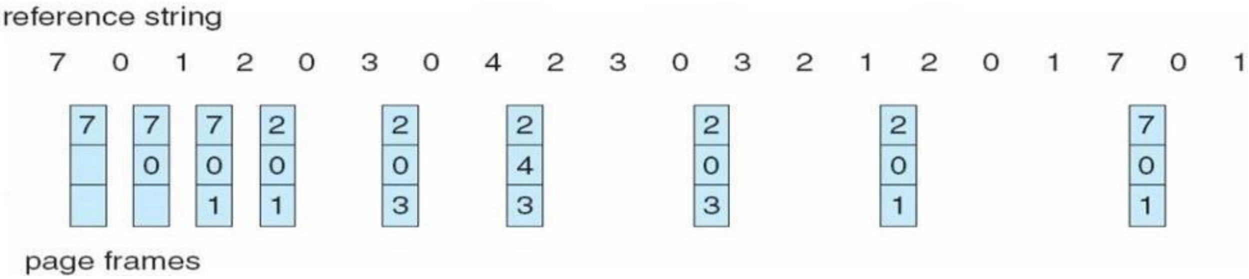
- Reference string:**7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time perprocess)



15 page faults

Optimal Algorithm:

∑ Replace page that will not be used for longest period of time



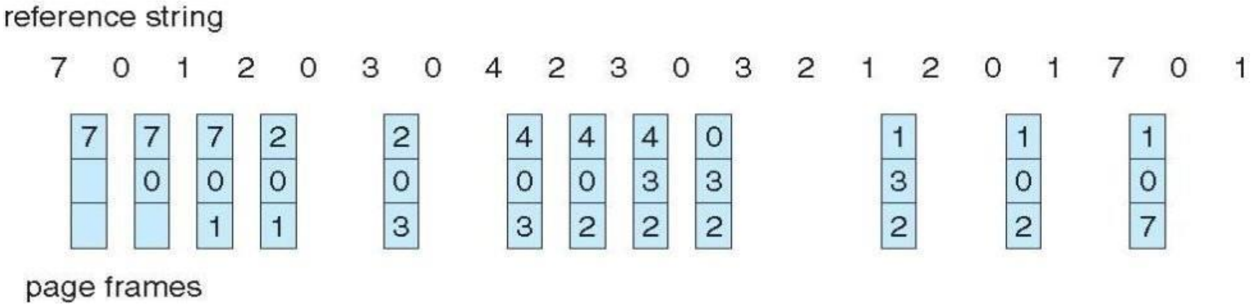
No of page faults: 9

Least Recently Used (LRU) Algorithm:

∑ Use past knowledge rather than future

∑ Replace page that has not been used in the most amount of time

∑ Associate time of last use with eachpage



Page faults:12

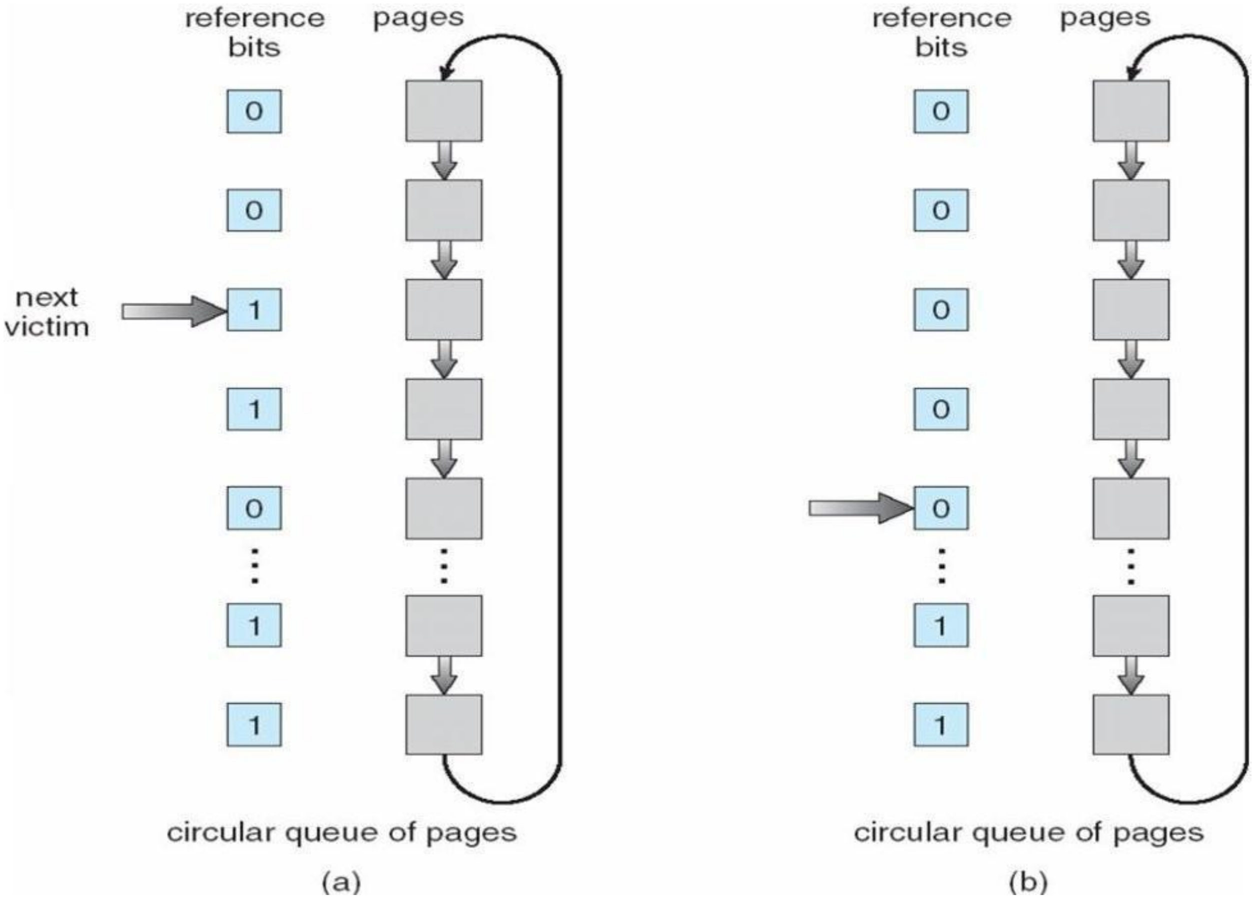
LRU Approximation Algorithms

∑ LRU needs special hardware and still slow

∑ **Reference bit**

  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace any with reference bit = 0 (if one exists)
    - We do not know the order, however

- **Second-chance algorithm**
  - Generally FIFO, plus hardware-provided reference bit
  - Clock replacement
  - If page to be replaced has
    - Reference bit = 0 -> replace it
    - reference bit = 1 then:
      - set reference bit 0, leave page in memory
      - replace next page, subject to same rules



Counting Algorithms

- Keep a counter of the number of references that have been made to each page
  - Not common
- **LFU Algorithm**: replaces page with smallestcount
- **MFU Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to beused

Applications and Page Replacement

- All of these algorithms have OS guessing about future pageaccess
- Some applications have better knowledge – i.e.databases
- Memory intensive applications can cause double buffering
  - OS keeps copy of page in memory as I/Obuffer
  - Application keeps page in memory for its own work
- Operating system can given direct access to the disk, getting out of the way of the applications
  - **Raw disk**mode
- Bypasses buffering, locking, etc

Allocation of Frames

- Each process needs *minimum* number offrames
- Example: IBM 370 – 6 pages to handle SS MOVEinstruction:
  - instruction is 6 bytes, might span 2 pages
  - 2 pages to handle*from*
  - 2 pages to handle*to*
- *Maximum* of course is total frames in thesystem
- Two major allocation schemes
  - fixed allocation
  - priority allocation
- Many variations

Fixed Allocation

- Equal allocation – For example, if there are 100 frames (after allocating frames for the OS) and 5 processes, give each process 20frames
  - Keep some as free frame buffer pool

- Proportional allocation – Allocate according to the size ofprocess
    - Dynamic as degree of multiprogramming, process sizeschange

s_i size of processp_i $S \Re s_i$

m totalnumberofframes

a_i allocation forp_i $\frac{s_i}{S} \infty m$

m 64

s_1 10

s_2 127

$a_1 \; \frac{10}{137} \infty 64 \spadesuit 5$

$a_2 \; \frac{127}{137} \infty 64 \spadesuit 59$

Priority Allocation

- Use a proportional allocation scheme using priorities rather thansize
- If process $P_i$generates a pagefault,
    - select for replacement one of its frames
    - select for replacement a frame from a process with lower prioritynumber

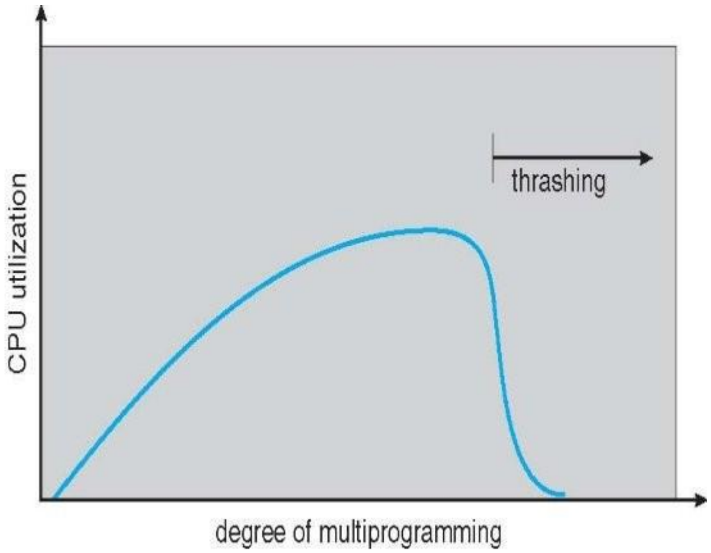
Global vs. Local Allocation

- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
    - But then process execution time can vary greatly
    - But greater throughput so more common
- **Local replacement** – each process selects from only its own set of allocatedframes
    - More consistent per-process performance
    - But possibly underutilized memory

Thrashing

- If a process does not have "enough" pages, the page-fault rate is veryhigh
    - Page fault to get page
    - Replace existingframe
    - But quickly need replaced frame back
    - This leads to:
        - Low CPU utilization
        - Operatingsystemthinkingthatitneedstoincreasethedegreeof multiprogramming
        - Another process added to the system

- **Thrashing∫** a processis busy swapping pages in and out



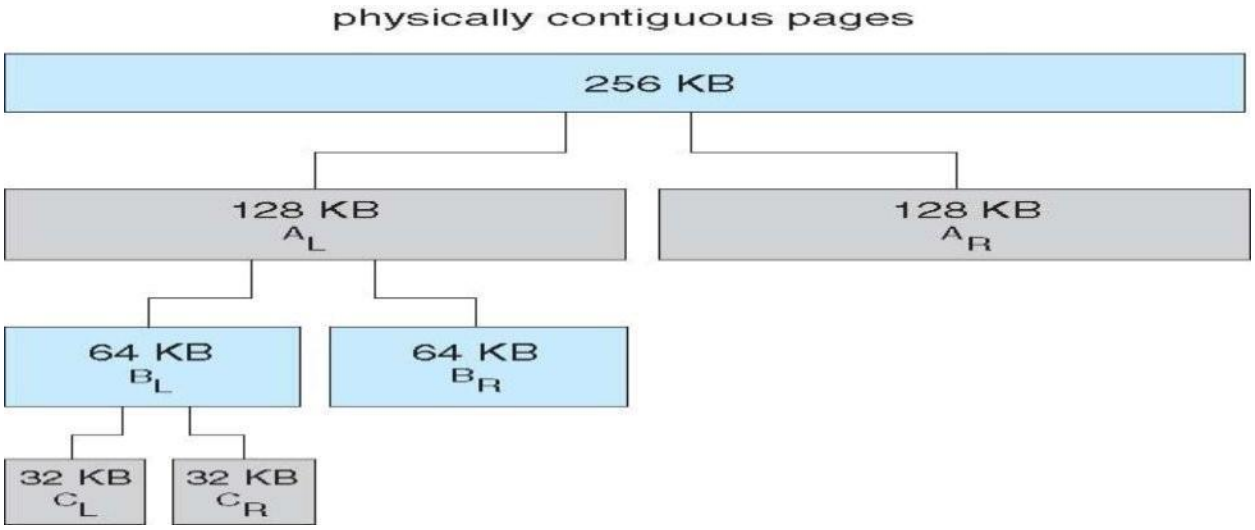
Allocating Kernel Memory

- Treated differently from user memory
- Often allocated from a free-memory pool
  - Kernel requests memory for structures of varyingsizes
  - Some kernel memory needs to be contiguous
    - I.e. for device I/O

Buddy System

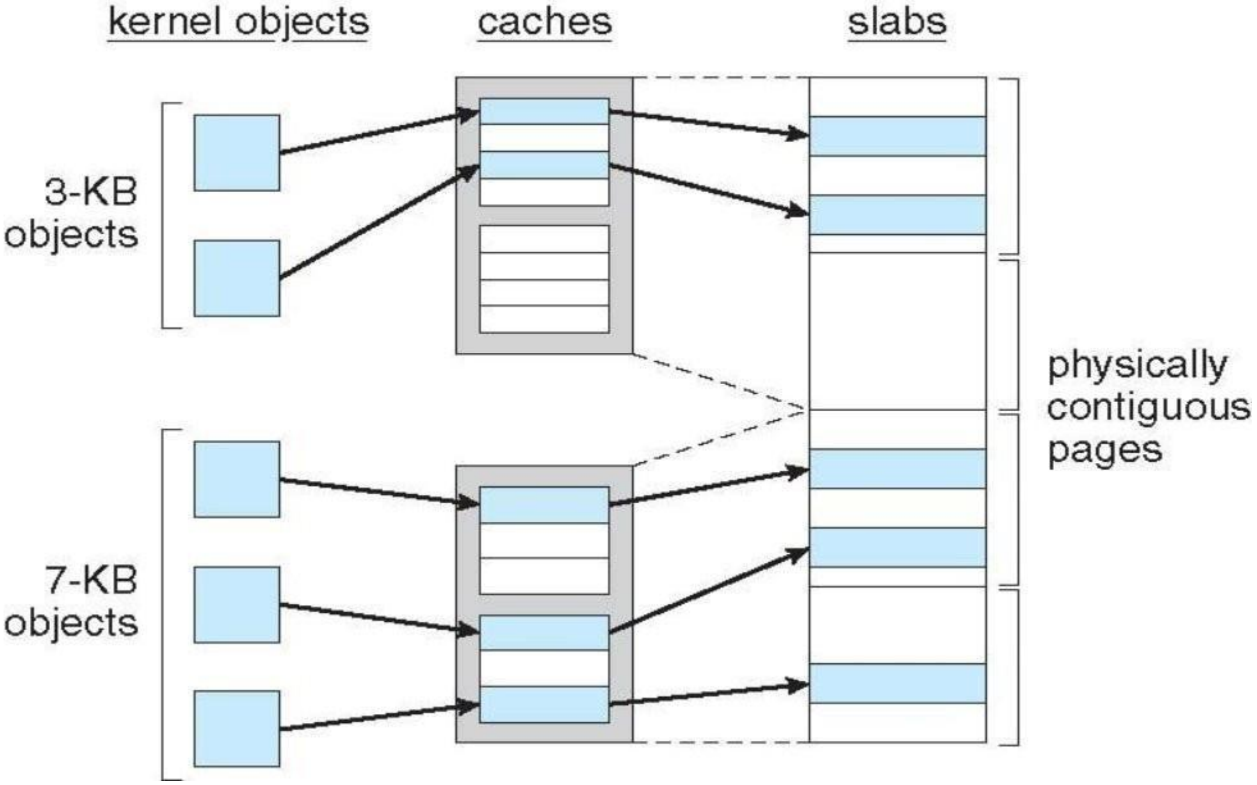
- Allocates memory from fixed-size segment consisting of physically-contiguouspages
- Memory allocated using **power-of-2allocator**
  - Satisfies requests in units sized as power of2
  - Request rounded up to next highest power of 2
  - When smaller allocation needed than is available, current chunk split into two buddies of next-lower power of 2
    - Continue until appropriate sized chunk available
- For example, assume 256KB chunk available, kernel requests21KB
  - Split into $A_{Land}A_r$ of 128KBeach
    - One further divided into B_L and B_R of 64KB
      - One further into C_L and C_R of 32KB each – one used to satisfy request
- Advantage – quickly coalesce unused chunks into largerchunk

- Disadvantage -fragmentation



Slab Allocator

- Alternate strategy
- **Slab** is one or more physically contiguouspages
- **Cache** consists of one or moreslabs
- Single cache for each unique kernel data structure
  - Each cache filled with **objects** – instantiations of the datastructure
  - When cache created, filled with objects marked as**free**
- When structures stored, objects marked as**used**
- If slab is full of used objects, next object allocated from emptyslab
  - If no empty slabs, new slab allocated
  - Benefits include no fragmentation, fast memory request satisfaction



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 diskdrives
  - P_1 and P_2 each hold one disk drive and each needs anotherone
  - Example
  - semaphores *A* and *B*, initialized to 1 P_0P_1

wait(A); wait(B) wait (B); wait(A)

System Model

- Resource types R_1, R_2, . . .,

R_m*CPU cycles, memory space,*

I/Odevices

- Each resource type R_i has $W_i$instances.

- Each process utilizes a resource asfollows:

- **request**

- ○ **use**
- ○ **release**

DeadlockCharacterization

Deadlock can arise if four conditions hold simultaneously.

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

∑ **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 itstask

∑ **Circular wait:** there exists a set {P_0, P_1, …, P_n} 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 heldby

P_2, …,P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

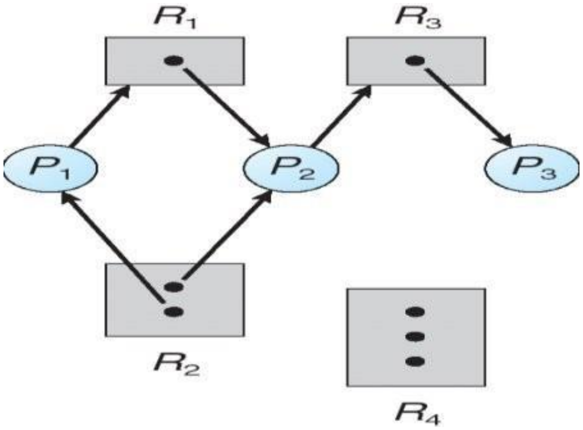
Resource-Allocation Graph

A set of vertices *V* and a set of edges *E*.

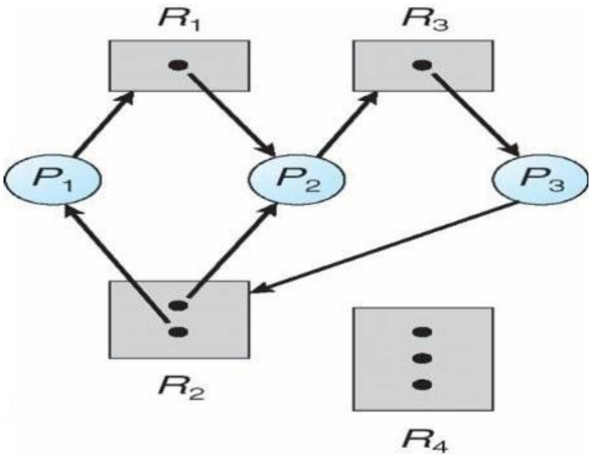
∑ V is partitioned into two types:

- ○ $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in thesystem
- ○ $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in thesystem
- ○ **request edge** – directed edge $P_i \emptyset R_j$

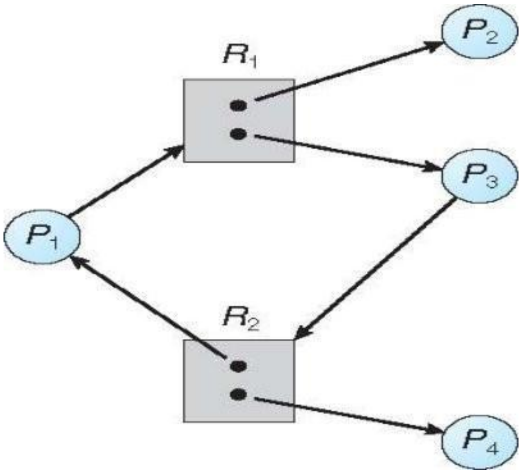
∑ **assignment edge** – directed edge$R_j \emptyset P_i$



Resource Allocation Graph With A Deadlock



Graph With A Cycle But No Deadlock



- If graph contains no cycles ⇒ nodeadlock
- If graph contains a cycle⇒
  - if 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 deadlockstate
- Allow the system to enter a deadlock state and thenrecover
- Ignore 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 fornonsharable 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 hasnone

  - 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 arereleased

  - 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 ofenumeration

Deadlock Avoidance

Requires that the system has some additional *a priori* informationavailable

- Simplest and most useful model requires that each process declare the *maximumnumber* 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 safestate

- System is in **safe state** if there exists a sequence $<P_1, P_2, \ldots, P_n>$ of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can besatisfied by currently available resources + resources held by all the P_j, with $j<I$

- That is:

  - If P_i resource needs are not immediately available, then $P_i$can wait until all $P_j$have finished

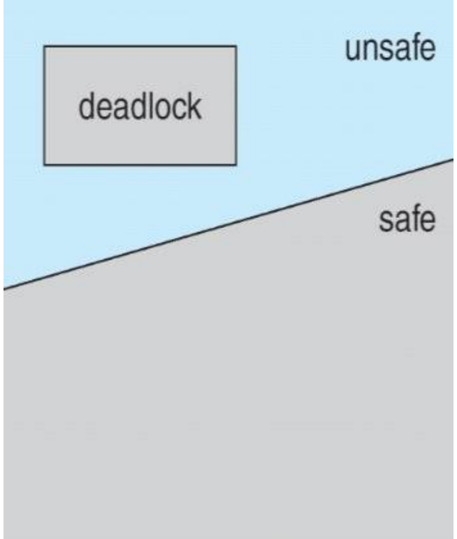
  - When $P_j$is finished, $P_i$can obtain needed resources, execute, return allocated resources, and terminate

- When $P_i$terminates, P_{i+1} can obtain its needed resources, and soon

∑ If a system is in safe state ⇒ nodeadlocks

∑ If a system is in unsafe state ⇒ possibility ofdeadlock

Avoidance ⇒ ensure that a system will never enter an unsafe 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

Resource-Allocation Graph Scheme

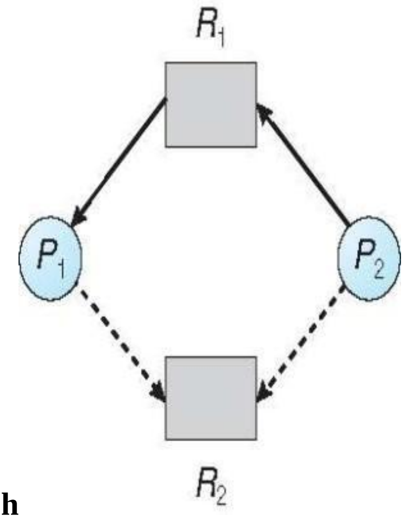
∑ **Claim edge**P_i∅$R_j$indicated that process $P_j$may request resource R_j; represented by a dashed line

∑ Claim edge converts to request edge when a process requests aresource

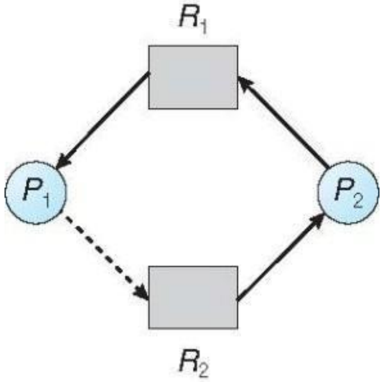
∑ Request edge converted to an assignment edge when the resource is allocated to the process

∑ When a resource is released by a process, assignment edge reconverts to a claimedge

∑ Resources must be claimed *a priori* in thesystem



Unsafe State In Resource-Allocation Graph



Banker's Algorithm

∑ Multiple instances

∑ Each process must a priori claim maximum use

∑ When a process requests a resource it may have towait

∑ When a process gets all its resources it must return them in a finite amount of time

Let *n* = number of processes, and *m* = number of resourcestypes.

∑ **Available***:* Vector of length *m*. If available [*j*] = *k*, there are *k* instances of resourcetype $R_j$available

∑ **Max***: n x m* matrix. If *Max* [*i*,*j*] = *k*, then process $P_i$may request at most *k* instances of resource typeR_j

∑ **Allocation***: n* x *m* matrix. If Allocation[*i*,*j*] = *k* then $P_i$is currently allocated *k* instances ofR_j

∑ **Need***: n* x *m* matrix. If *Need*[*i*,*j*] = *k*, then $P_i$may need *k* more instances of $R_j$to complete its task

Need [*i*,*j]* = *Max*[*i*,*j*] – *Allocation* [*i*,*j*]

SafetyAlgorithm

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:

Work = Available

Finish [i] = false fori = 0, 1, …,n- 1

2. Find an isuch that both:

(a) Finish [i] =false

(b) Need$_i$≤Work

If no such iexists, go to step 4

3. Work = Work +
Allocation$_i$Finish[i] =true
go to step 2

4.IfFinish [i] == true for all i, then the system is in a safestate

Resource-Request Algorithm for Process *P$_i$*

Request = request vector for process *P$_i$*. If *Request$_i$*[*j*] = *k* then process *P$_i$*wants *k* instances of resource type*R$_j$*

1. If *Request$_i$*≤*Need$_i$*go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim

2. If *Request$_i$*≤*Available*, go to step 3. Otherwise *P$_i$*must wait, since resources are not available

3. Pretend to allocate requested resources to *P$_i$*by modifying the state as follows:

Available = Available – Request;

Allocation$_i$= Allocation$_i$+ Request$_i$;

Need$_i$=Need$_i$– Request$_i$;

- *If safe ⇒the resources are allocated toPi*
- *If unsafe ⇒Pi must wait, and the old resource-allocation state isrestored*

Example of Banker's Algorithm

∑ 5 processes *P*$_0$ through *P*$_4$;

3 resource types:

A (10 instances), *B* (5instances), and *C* (7 instances)

Snapshot at time*T*$_0$:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

∑ The content of the matrix *Need* is defined to be *Max – Allocation*

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

∑ The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria

P_1 Request (1,0,2)

∑ Check that Request ≤ Available (that is, (1,0,2) ≤ (3,3,2) ⇒true

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	30 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

∑ Executing safety algorithm shows that sequence < P_1, P_3, P_4, P_0, P_2> satisfies safety requirement

- Can request for (3,3,0) by P_4 begranted?
- Can request for (0,2,0) by P_0 begranted?

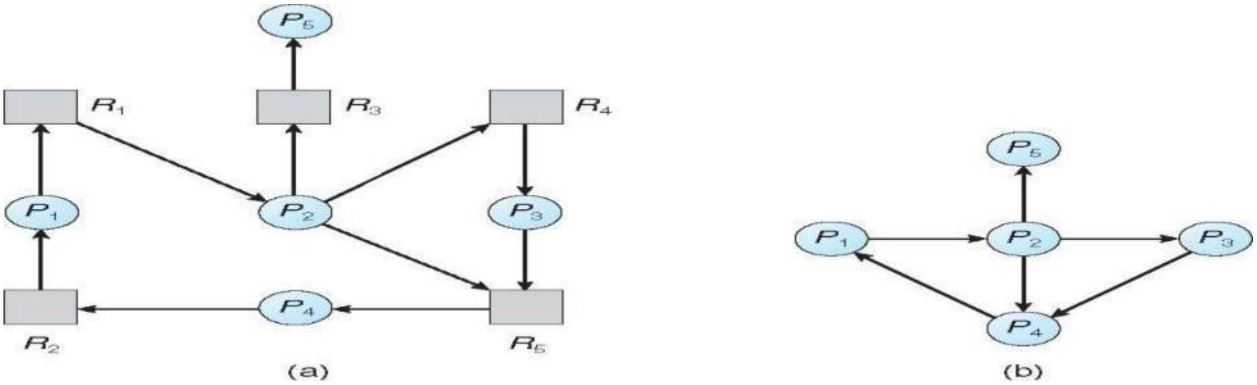
Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

Single Instance of Each Resource Type

- Maintain *wait-for* graph
  - Nodes are processes
  - $P_i \varnothing P_j$if$P_i$is waitingforP_j
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph

Resource-Allocation Graph and Wait-for Graph



Resource-AllocationGraph

Corresponding wait-for graph

Several Instances of a Resource Type

- **Available***:* A vector of length *m* indicates the number of available resources of each type.
- **Allocation***:* An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.
- **Request***:* An *n* x *m* matrix indicates the current request of each process. If *Request*[*i*][*j*] = *k*, then process $P_i$is requesting *k* more instances of resource type.R_j.

Detection Algorithm

Let Work and Finish be vectors of length m and n, respectively Initialize:

(a) Work =Available

(b) For i = 1,2, …, n, if $Allocation_i$π 0, then
Finish[i] = false; otherwise, Finish[i] = true

2. Find an index isuch that both:

(a) Finish[i] == false

(b) $Request_i \leq Work$

If no such i exists, go to step 4

3. *Work* = *Work* + $Allocation_i$
Finish[*i*] =*true*
go to step 2

4. If *Finish*[*i*] == false, for some *i*, $1 \leq i \leq n$, then the system is in deadlock state. Moreover,if *Finish*[*i*] == *false*, then $P_i$is deadlocked

Recovery from Deadlock:

Process Termination

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle iseliminated
- In which order should we choose to abort?
    - Priority of the process
    - How long process has computed, and how much longer tocompletion
    - Resources the process has used
    - Resources process needs to complete
    - How many processes will need to beterminated
    - Is process interactive or batch?

Resource Preemption

- Selecting a victim – minimizecost
- Rollback – return to some safe state, restart process for thatstate
- Starvation – same process may always be picked as victim, include number of rollback in cost factor

UNIT-4

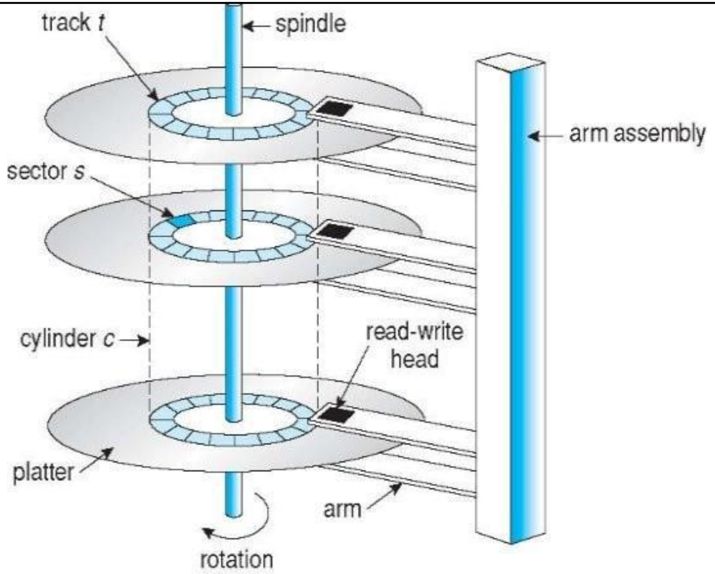
Secondary-Storage Systems, File-System Interface and Implementation

Overview of Secondary Storage Structure

- Magnetic disks provide bulk of secondary storage of moderncomputers
  - Drives rotate at 60 to 250 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 disksurface
    - 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,SAS, Firewire
  - Host controller in computer uses bus to talk to disk controller built into drive or storage array

Magnetic Disks

- Platters range from .85” to 14” (historically)
  - Commonly 3.5”, 2.5”, and 1.8”
- Range from 30GB to 3TB per drive
- Performance
  - Transfer Rate – theoretical – 6Gb/sec
  - Effective Transfer Rate – real –1Gb/sec
  - Seek time from 3ms to 12ms – 9ms common for desktop drives
  - Average seek time measured or calculated based on 1/3 of tracks
  - Latency based on spindle speed
    - 1/(RPM * 60)
  - Average latency = ½ latency



Magnetic Tape

- Was early secondary-storage medium
  - Evolved from open spools to cartridges
- Relatively permanent and holds large quantities of data
- Access time slow
- Random access ~1000 times slower thandisk
- 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
  - 140MB/sec and greater
- 200GB to 1.5TB typical storage
- Common technologies are LTO-{3,4,5} and T10000

Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of **logical blocks**, where the logical block is the smallest unit of transfer
- The 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 outermostcylinder
  - 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 toinnermost
  - Logical to physical address should be easy

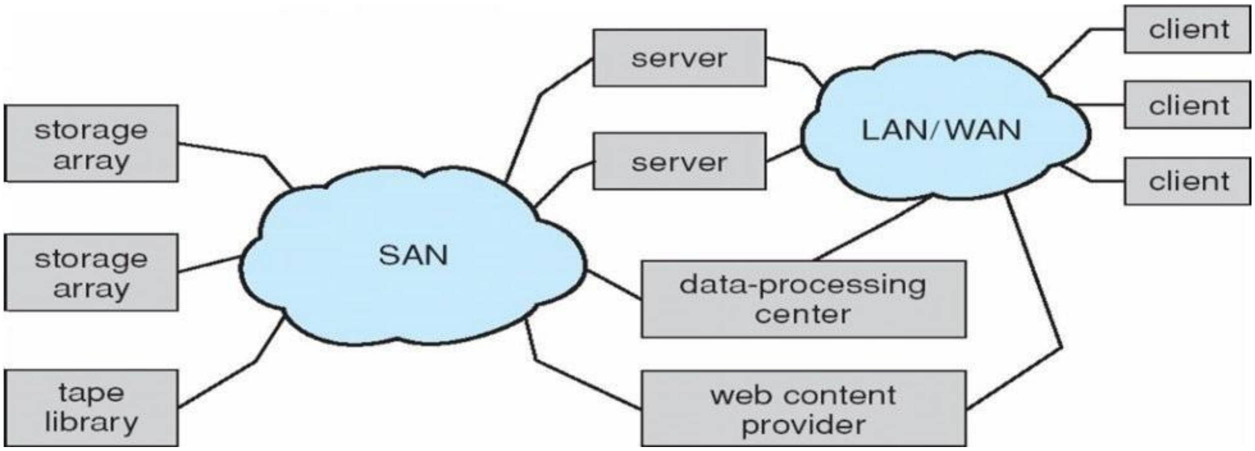
        - Except for bad sectors
        - Non-constant # of sectors per track via constant angular velocity

Disk Attachment

- Host-attached storage accessed through I/O ports talking to I/Obusses
- SCSI itself is a bus, up to 16 devices on one cable, **SCSI initiator** requests operationand **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**storagearea networks (SAN**s**)** in which many hosts attach to many storageunits
- I/O directed to bus ID, device ID, logical unit(LUN)

Storage Area Network

- Common in large storage environments
- Multiple hosts attached to multiple storage arrays – flexible



- SAN is one or more storage arrays
    - Connected to one or more Fibre Channelswitches
- Hosts also attach to theswitches
- Storage made available via **LUN Masking** from specific arrays to specificservers
- Easy to add or remove storage, add new host and allocate it storage
    - Over low-latency Fibre Channel fabric
    - Why have separate storage networks and communications networks?
    - Consider iSCSI, FCOE

Network-Attached Storage

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

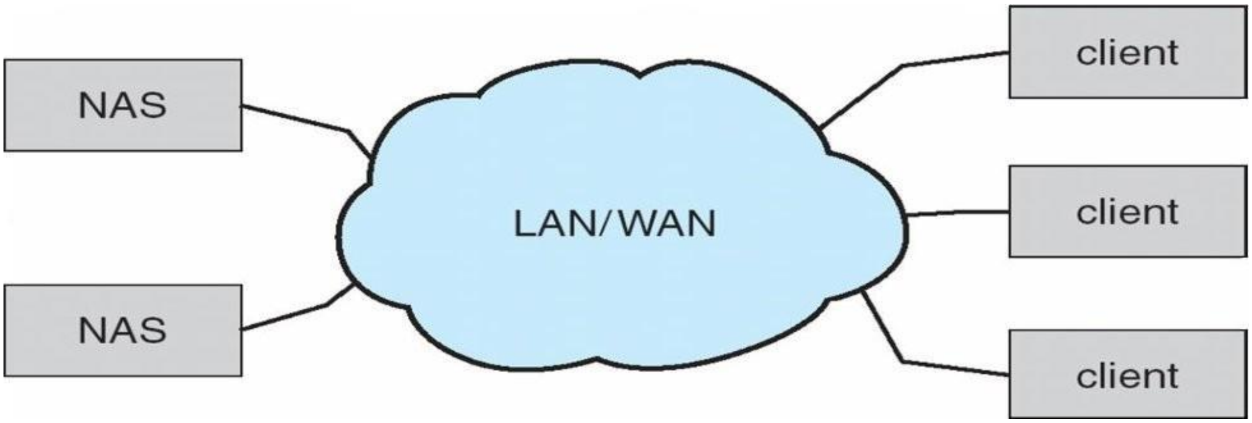
 o Remotely attaching to file systems

∑ NFS and CIFS are common protocols

∑ Implemented via remote procedure calls (RPCs) between host and storage over typically TCP or UDP on IP network

∑ **iSCSI** protocol uses IP network to carry the SCSIprotocol

 o Remotely attaching to devices (blocks)



Disk Scheduling

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

∑ Minimize seek time

∑ Seek time ♠ seekdistance

∑ 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 lasttransfer

∑ There are many sources of disk I/O request

∑ OS

∑ System processes

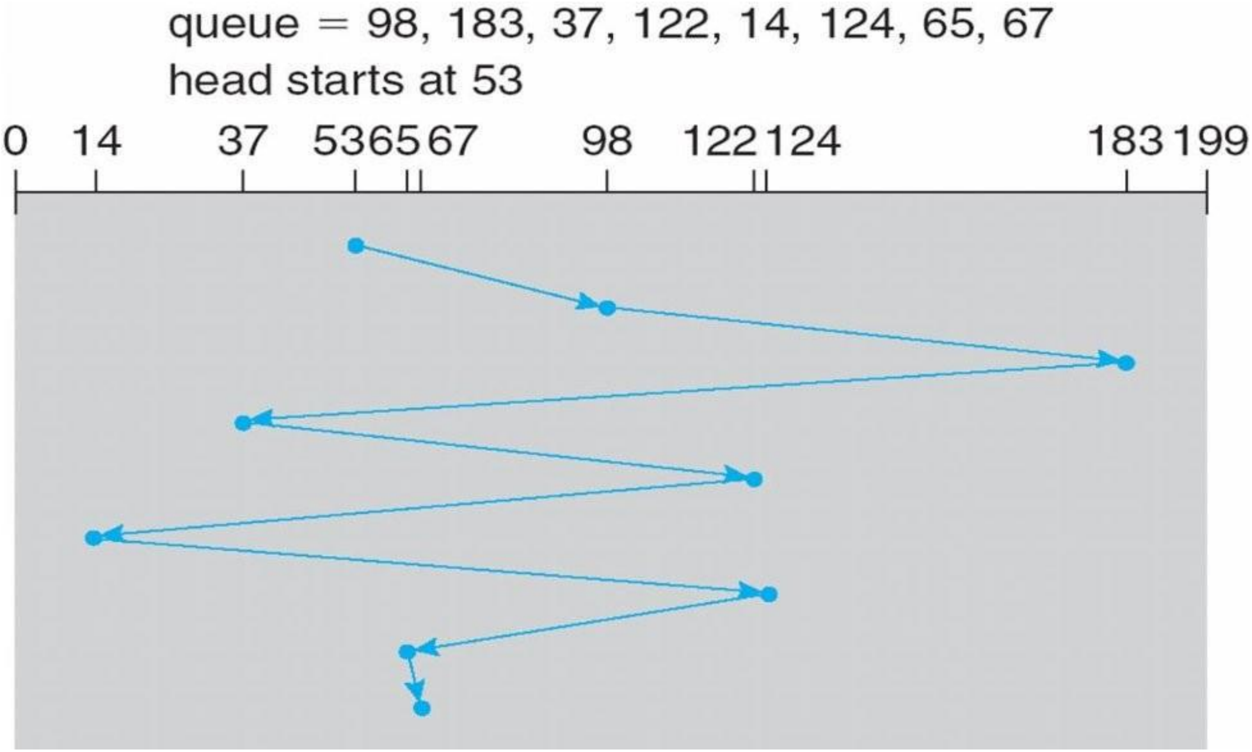
∑ Users processes

∑ I/O request includes input or output mode, disk address, memory address, number of sectors to transfer

∑ OS maintains queue of requests, per disk or device

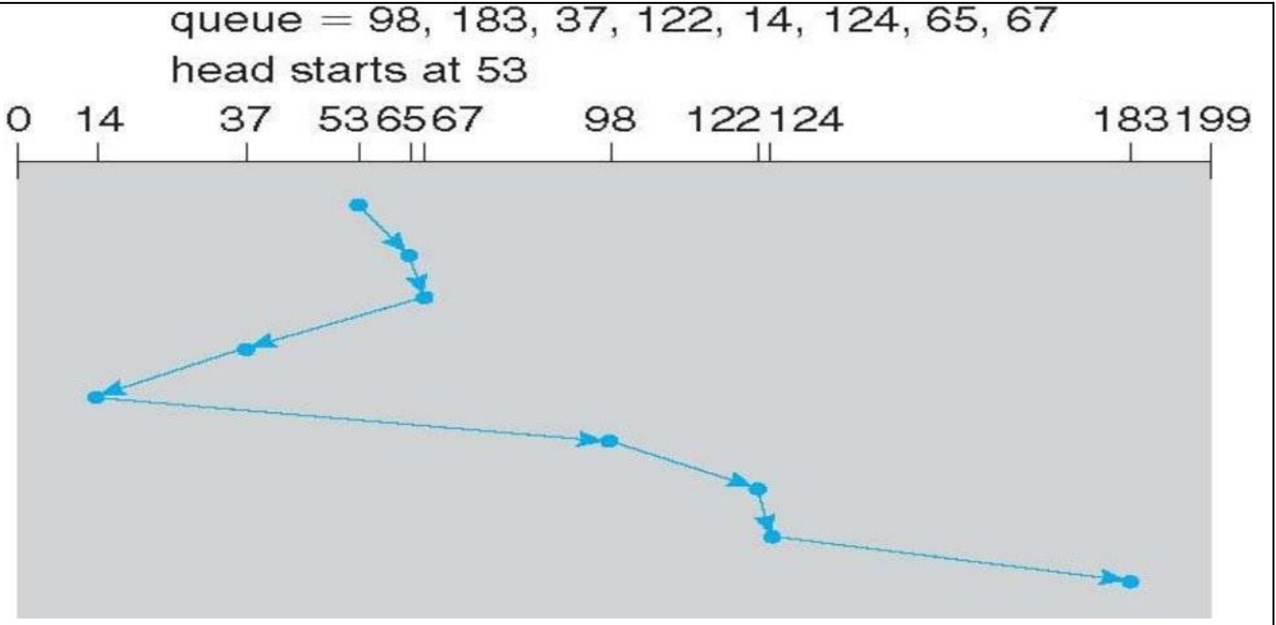
∑ Idle disk can immediately work on I/O request, busy disk means work mustqueue

- Optimization algorithms only make sense when a queueexists
- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- Several algorithms exist to schedule the servicing of disk I/Orequests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (0-199)
- 98, 183, 37, 122, 14, 124, 65, 67
- Head pointer 53



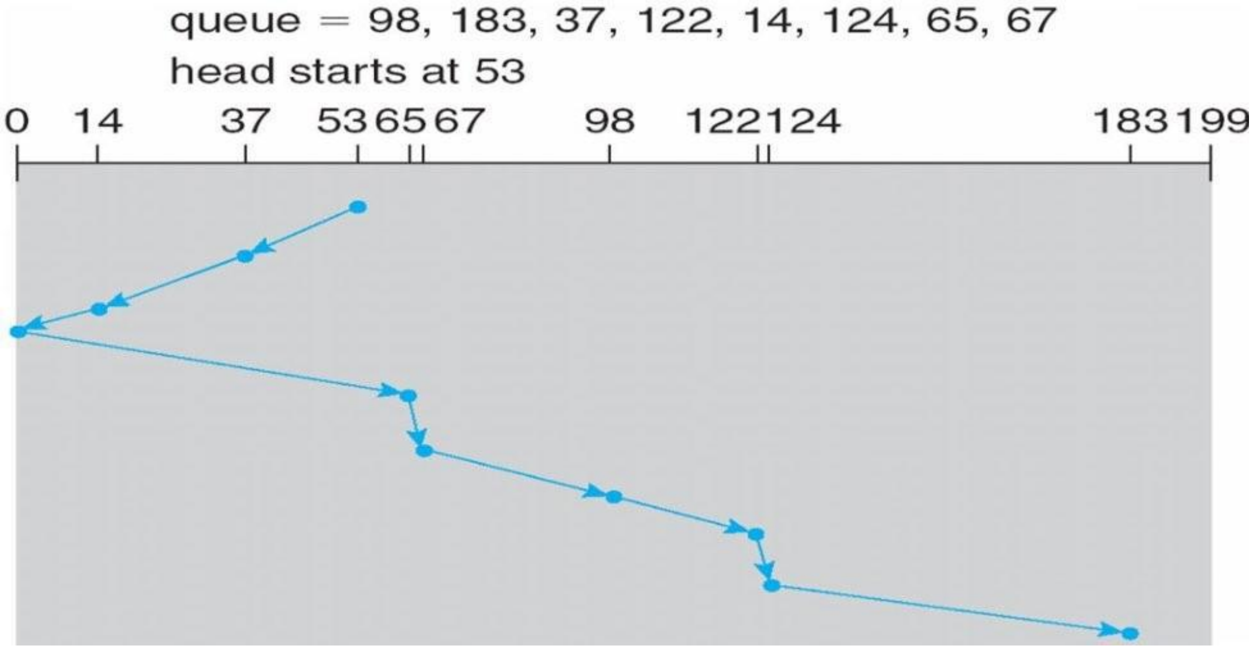
SSTF

- Shortest Seek Time First 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 somerequests
- Illustration shows total head movement of 236cylinders



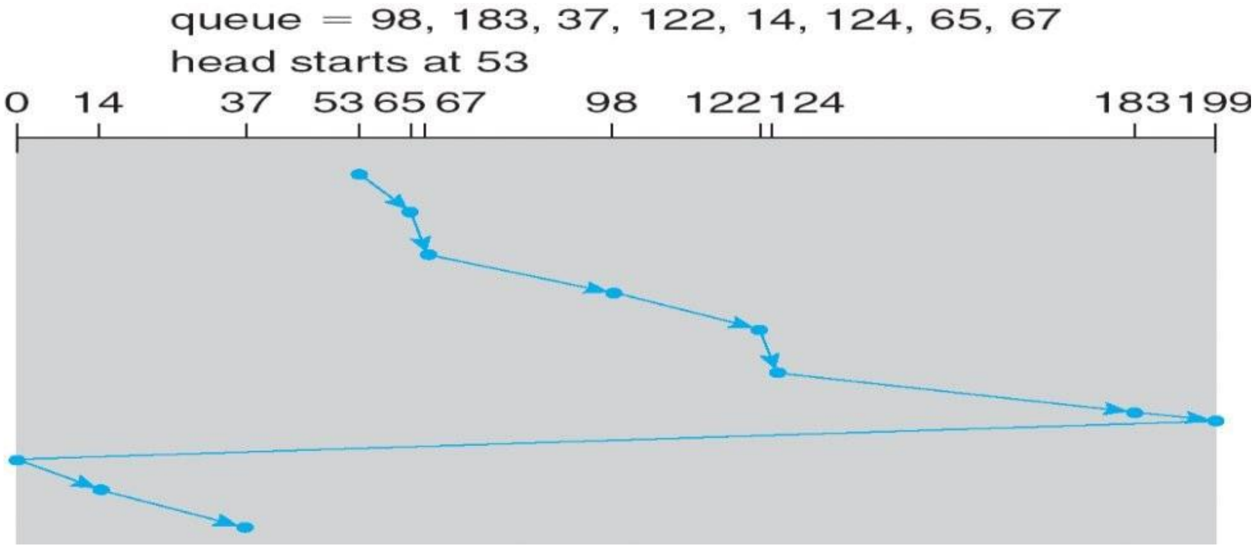
SCAN

- 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.
- **SCAN algorithm** Sometimes called the **elevatoralgorithm**
- Illustration shows total head movement of 208cylinders
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest



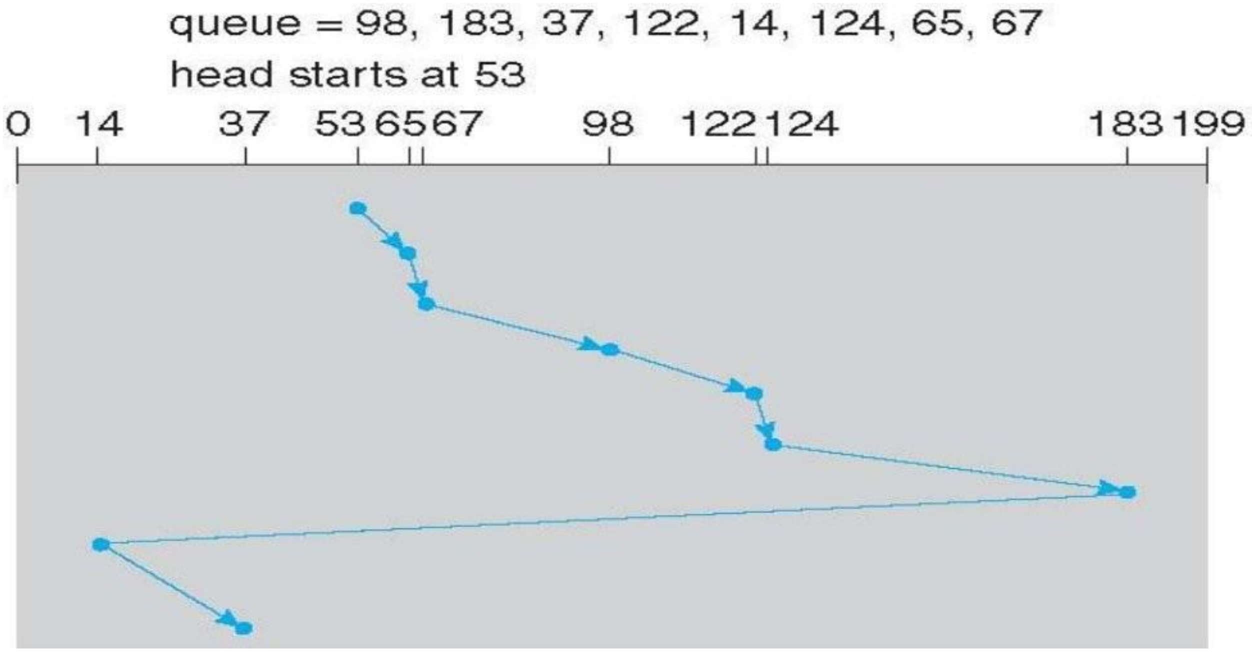
C-SCAN

- Provides a more uniform wait time thanSCAN
- The head moves from one end of the disk to the other, servicing requests as itgoes
  - When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the returntrip
  - Treats the cylinders as a circular list that wraps around from the last cylinder to the first one
- Total number of cylinders?



C-LOOK

- LOOK a version of SCAN, C-LOOK a version ofC-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 thedisk

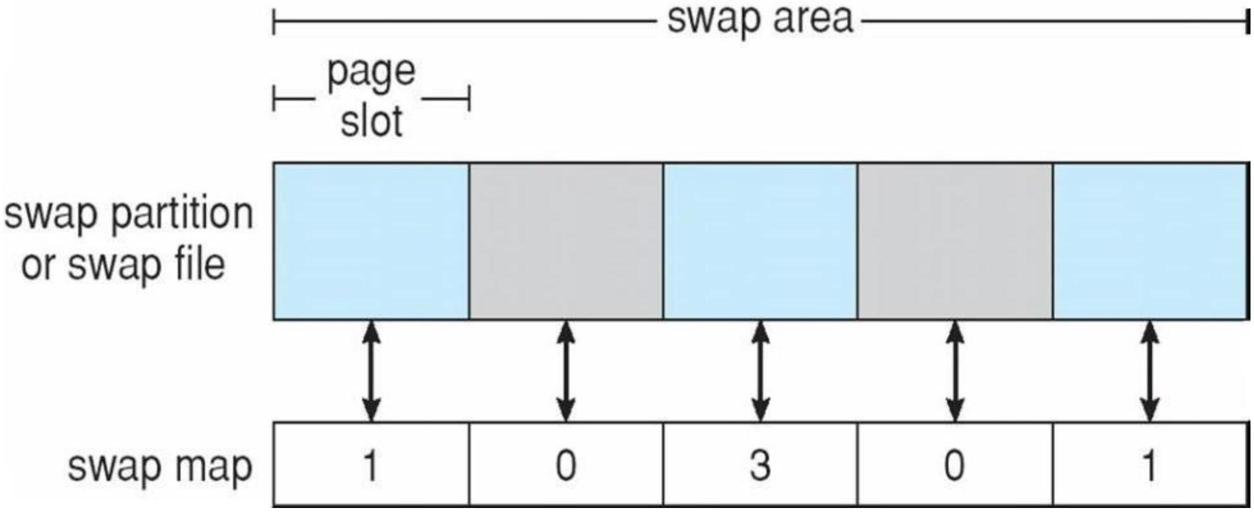


Disk Management

- **Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
  - Each sector can hold header information, plus data, plus error correction code (**ECC**)
  - Usually 512 bytes of data but can be selectable
  - 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, each treated as a logical disk
  - **Logical formatting** or "making a filesystem"
  - To increase efficiency most file systems group blocks into**clusters**
    - DiskI/Odoneinblocks
    - FileI/Odoneinclusters
    - Boot block initializes system
  - The bootstrap is stored in ROM
  - **Bootstrap loader** program stored in boot blocks of bootpartition
  - Methods such as **sector sparing** used to handle bad blocks

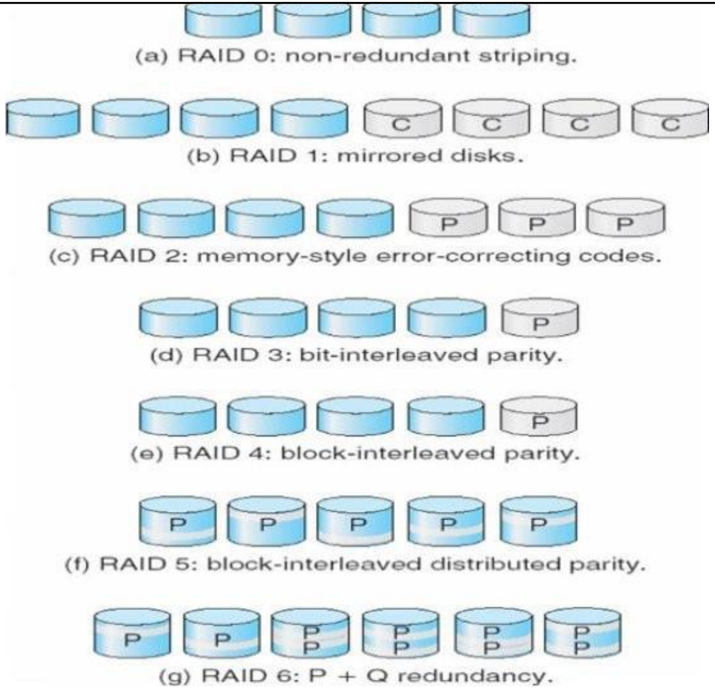
Swap-Space Management

- Swap-space — Virtual memory uses disk space as an extension of mainmemory
  - Less common now due to memory capacity increases
- Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition(raw)
- Swap-space management
  - 4.3BSD allocates swap space when process starts; holds text segment (the program) and data segment
  - Kernel uses **swap maps** to track swap-spaceuse
  - Solaris 2 allocates swap space only when a dirty page is forced out of physical memory, not when the virtual memory page is firstcreated
    - File data written to swap space until write to file system requested
    - Other dirty pages go to swap space due to no other home
    - Text segment pages thrown out and reread from the file system as needed



RAID Structure

- ∑ RAID – multiple disk drives provides reliability viaredundancy
- ∑ Increases the **mean time tofailure**
- ∑ Frequently combined with **NVRAM** to improve writeperformance
- ∑ RAID is arranged into six different levels
- ∑ Several improvements in disk-use techniques involve the use of multiple disks working cooperatively
- ∑ Disk **striping** uses a group of disks as one storageunit
- ∑ RAID schemes improve performance and improve the reliability of the storage system by storing redundant data
  - n **Mirroring** or **shadowing** (**RAID 1**) keeps duplicate of eachdisk
  - n Striped mirrors (**RAID 1+0**) or mirrored stripes (**RAID 0+1**) provides high performance and high reliability
  - n **Block interleaved parity** (**RAID 4, 5, 6**) uses much lessredundancy
- ∑ RAID within a storage array can still fail if the array fails, so automatic **replication** of the data between arrays is common
- ∑ Frequently, a small number of **hot-spare** disks are left unallocated, automatically replacing a failed disk and having data rebuilt onto them



(a) RAID 0: non-redundant striping.

(b) RAID 1: mirrored disks.

(c) RAID 2: memory-style error-correcting codes.

(d) RAID 3: bit-interleaved parity.

(e) RAID 4: block-interleaved parity.

(f) RAID 5: block-interleaved distributed parity.

(g) RAID 6: P + Q redundancy.

File-System Interface

File Concept

- Contiguous logical address space
- Types:
  - Data
  - numeric
  - character
  - binary
  - Program

File Structure

- None - sequence of words,bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file

- Can simulate last two with first method by inserting appropriate controlcharacters
- Who decides:
    - Operating system
    - Program

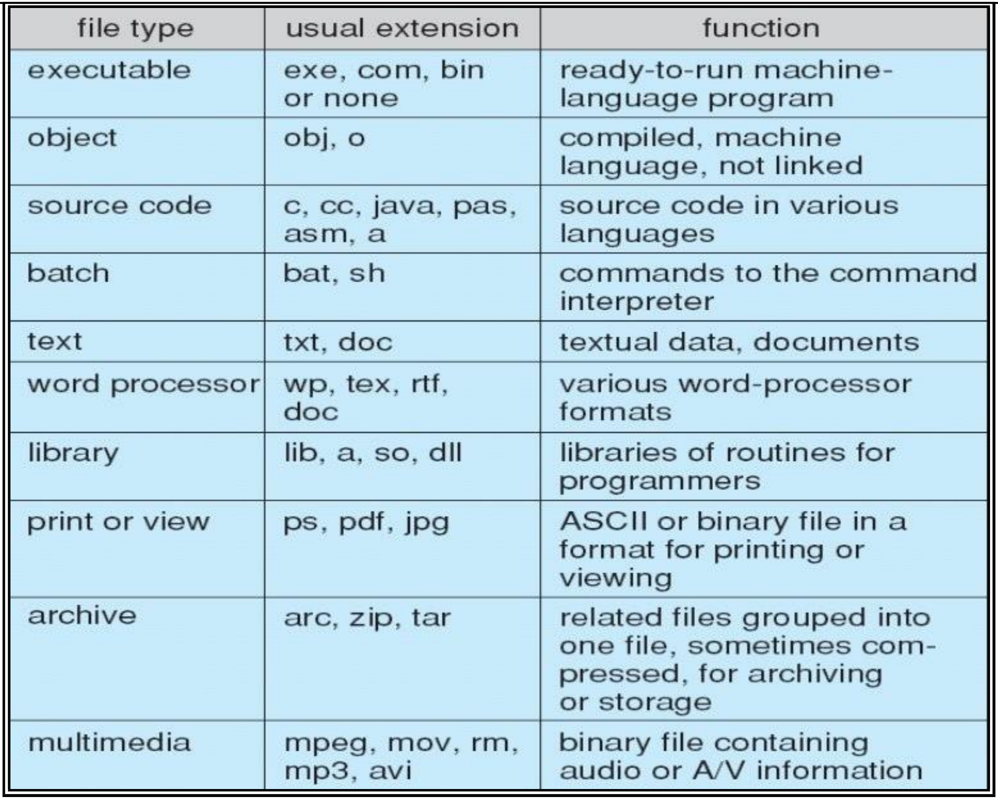
File Attributes

- **Name** – only information kept in human-readableform
- **Identifier** – unique tag (number) identifies file within filesystem
- **Type** – needed for systems that support differenttypes
- **Location** – pointer to file location on device
- **Size** – current filesize
- **Protection** – controls who can do reading, writing,executing
- **Time, date, and user identification** – data for protection, security, and usagemonitoring
- Information about files are kept in the directory structure, which is maintained on thedisk

File Operations

- File is an **abstract datatype**
- **Create**
- **Write**
- **Read**
- **Reposition within file**
- **Delete**
- **Truncate**
- *Open(F_i)* – search the directory structure on disk for entry *F_i*, and move the content of entry to memory
- *Close (F_i)* – move the content of entry *F_i*in memory to directory structure ondisk

File Types – Name, Extension



file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Access Methods

∑ **Sequential Access**

read next

write next

reset

no read after last write

(rewrite)

∑ **DirectAccess**

read*n*wr

ite*n*

position to *n*

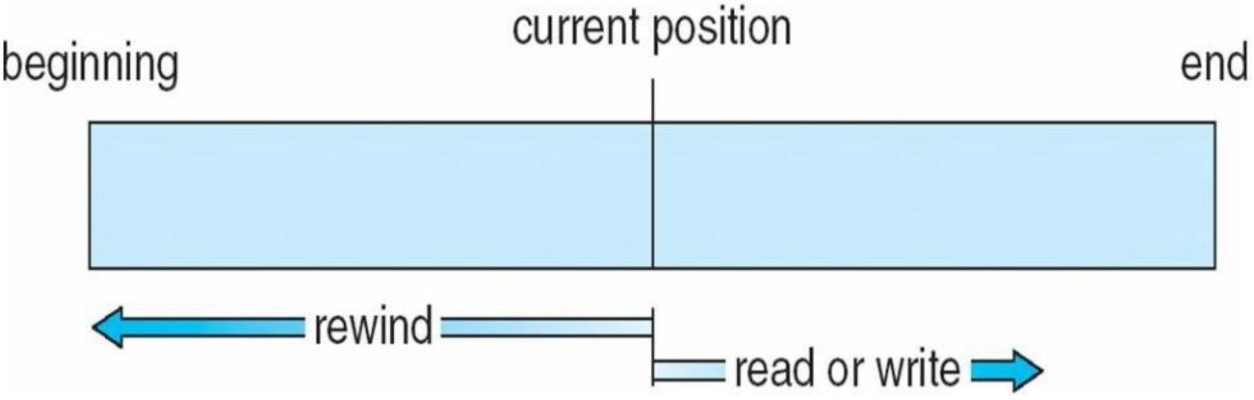
read next

write next

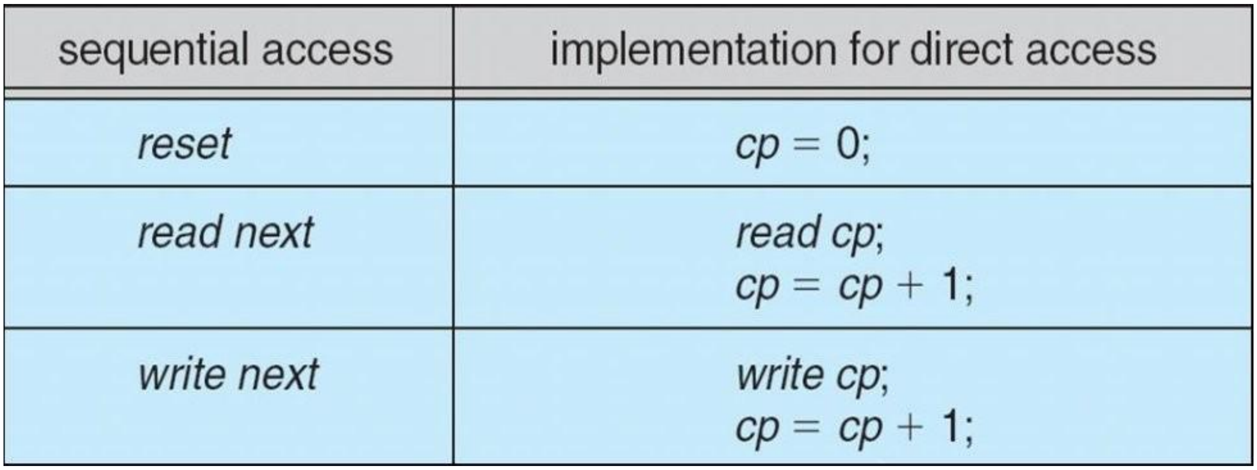
rewrite*n*

n = relative block number

Sequential-access File

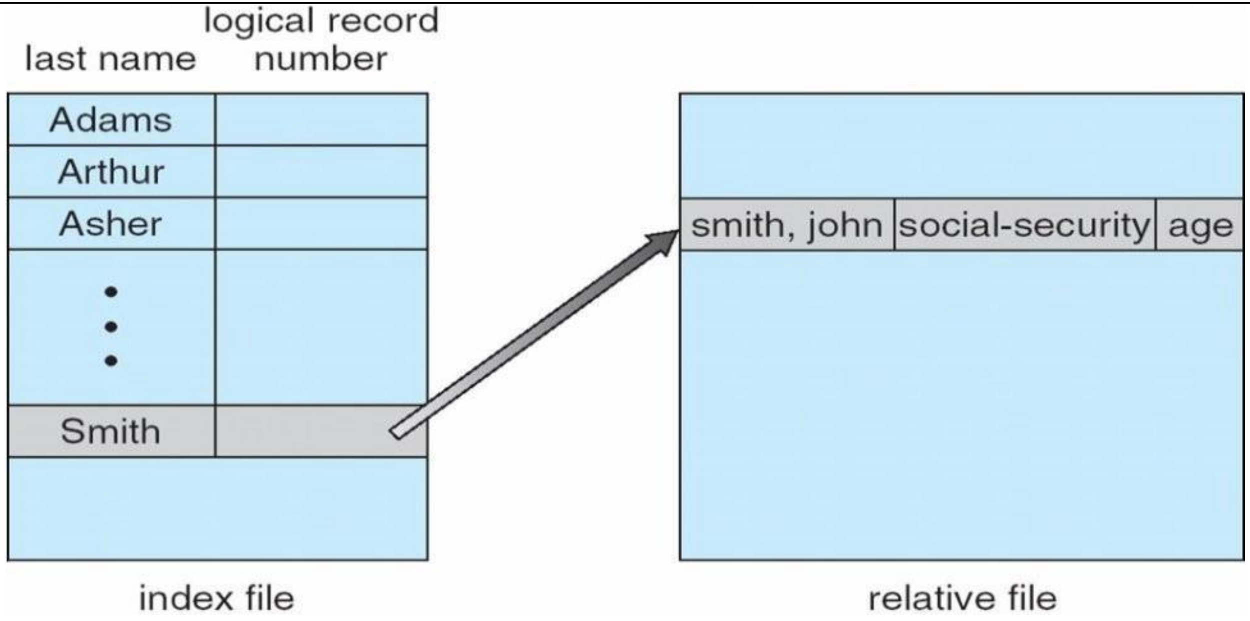


Simulation of Sequential Access on Direct-access File



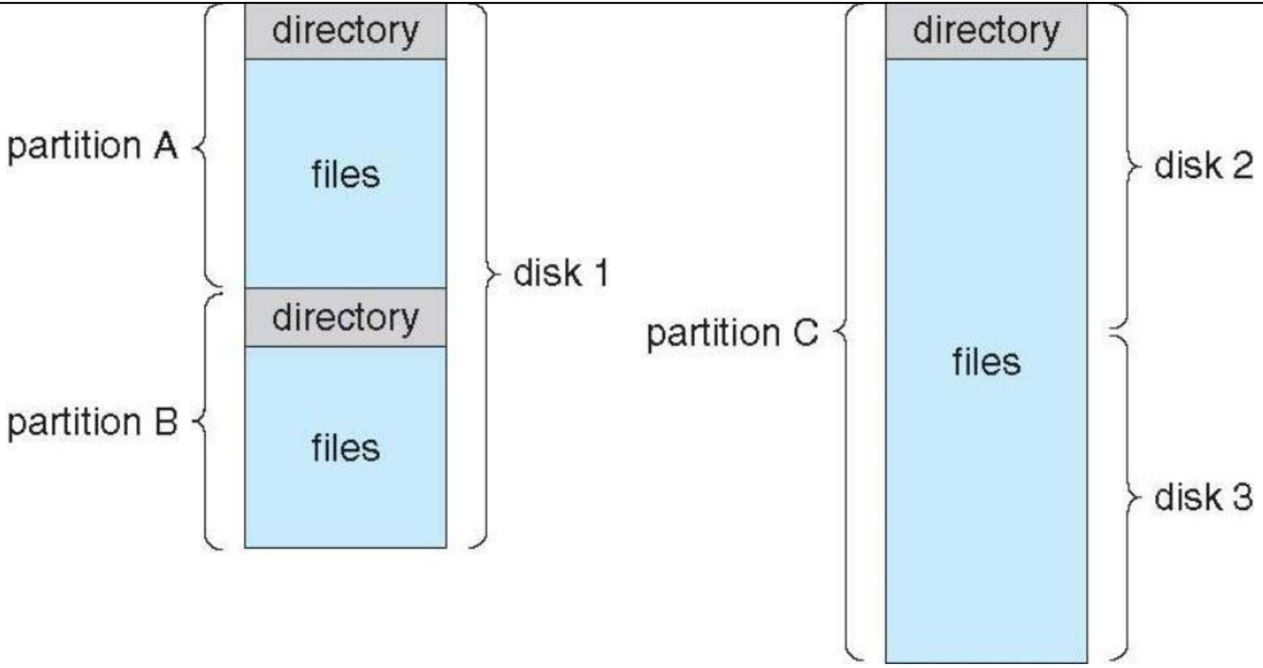
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Example of Index and Relative Files



Directory Structure

- A collection of nodes containing information about all files
- Disk can be subdivided into**partitions**
- Disks or partitions can be **RAID** protected againstfailure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a filesystem
- Partitions also known as minidisks, slices
- Entity containing file system known as a**volume**
- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents**
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system orcomputer

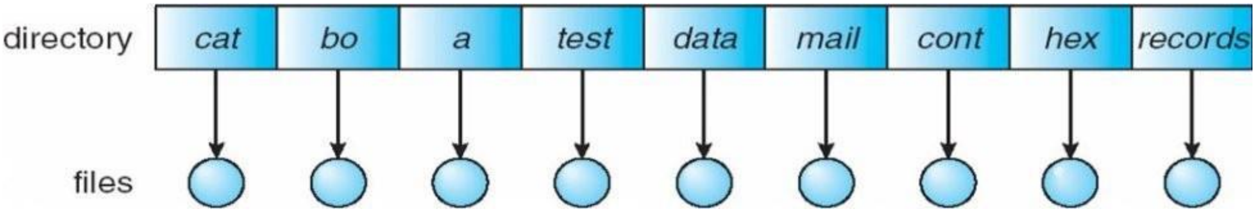


Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

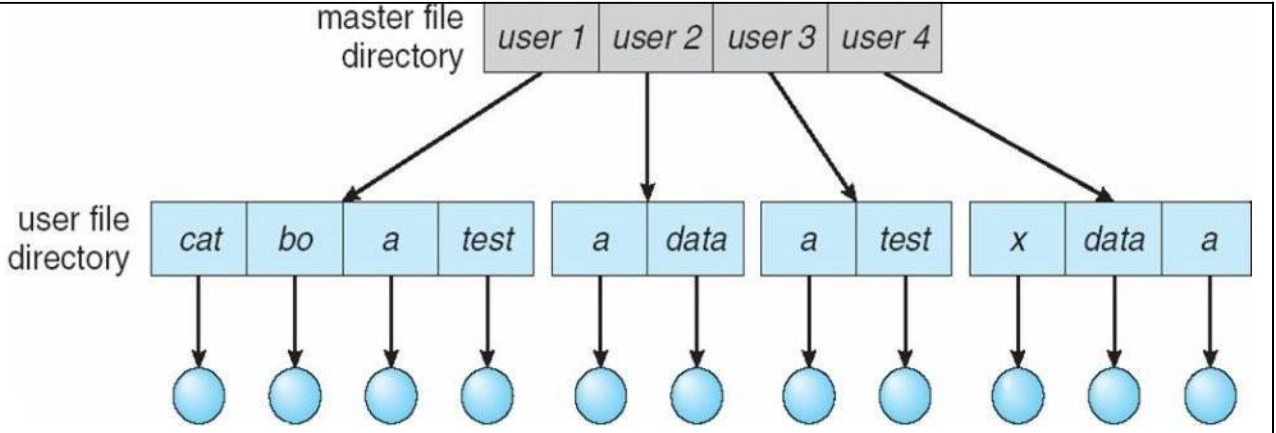
Single-Level Directory

- A single directory for all users



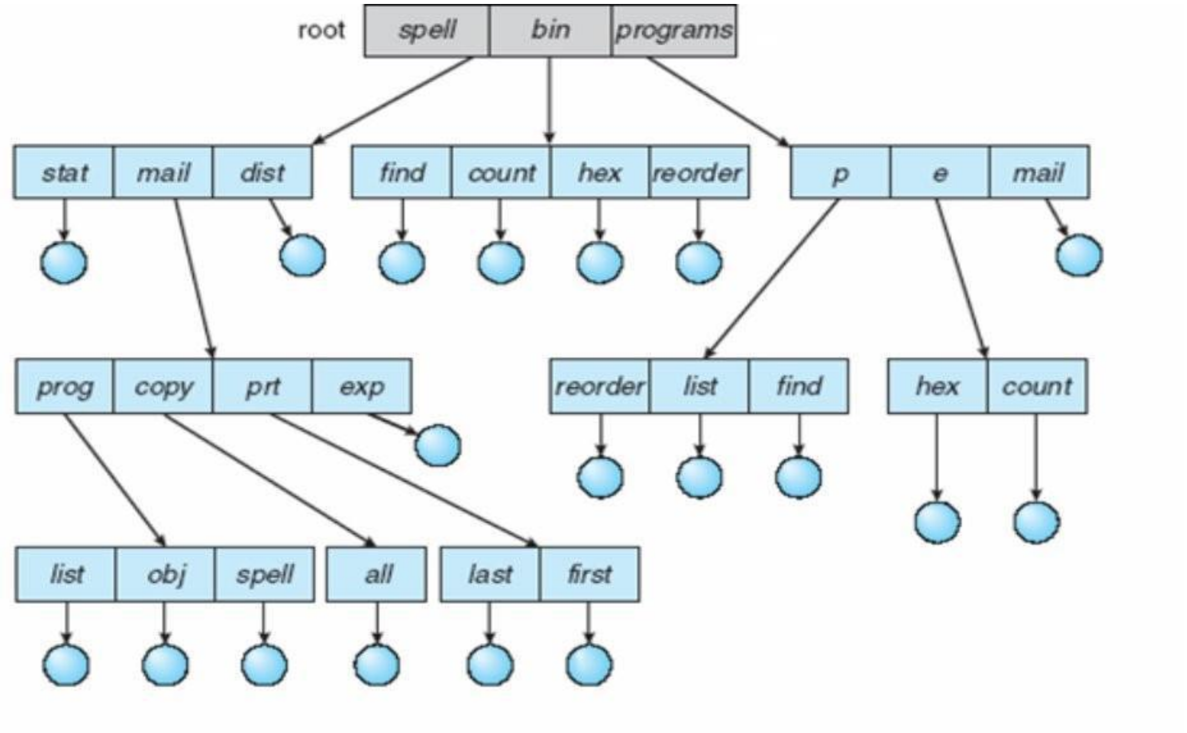
Two-Level Directory

- Separate directory for each user



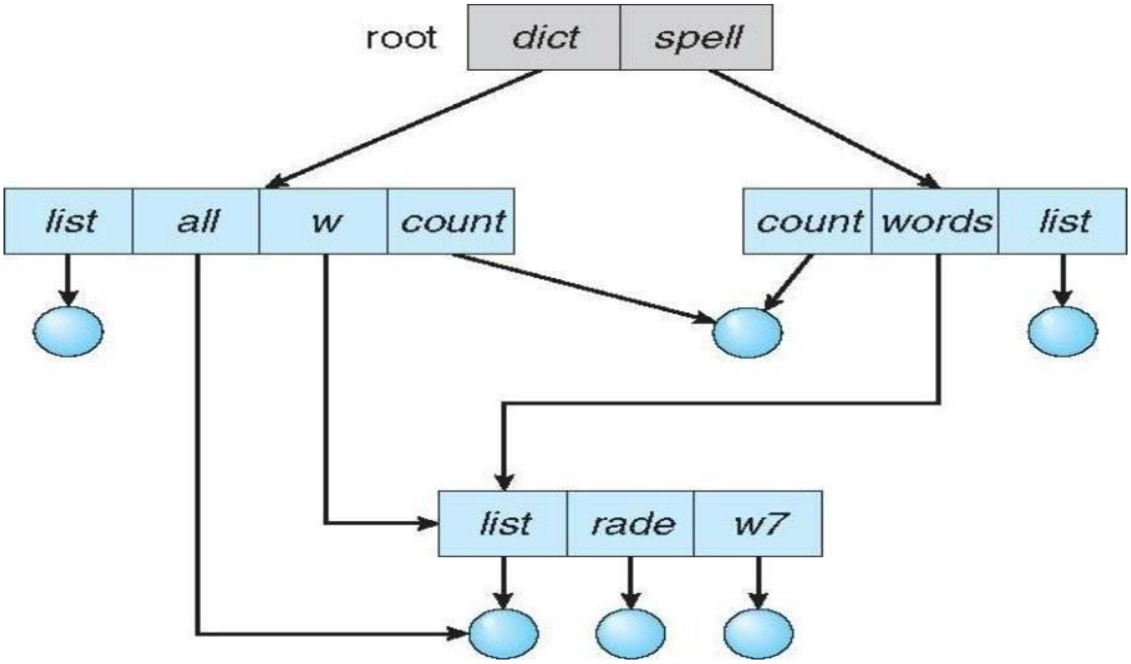
- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - **cd /spell/mail/prog**
  - **type list**

Acyclic-Graph Directories

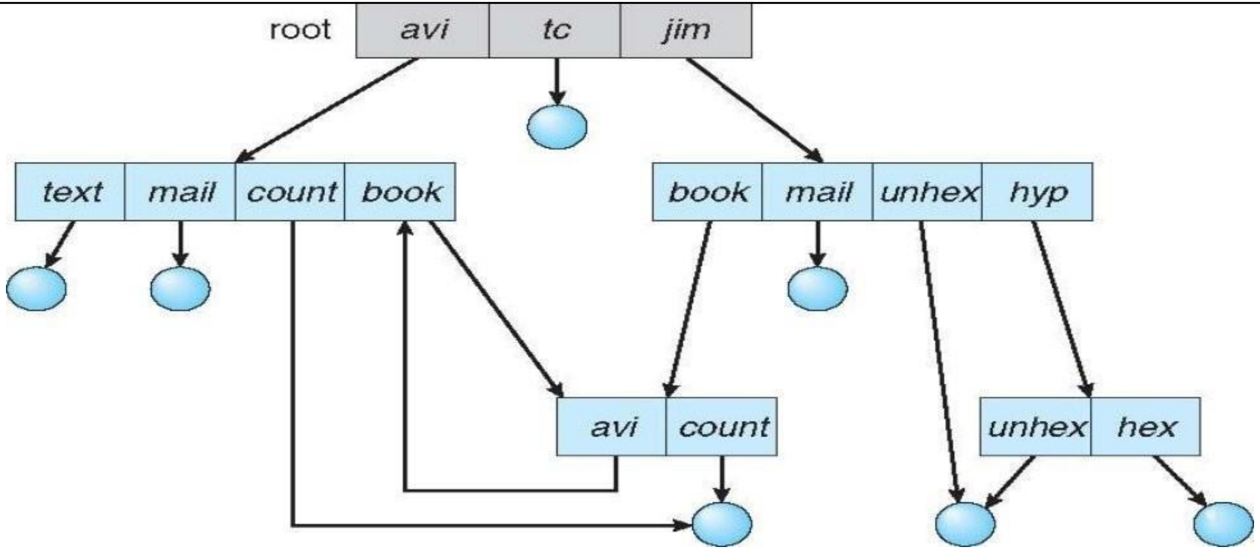


- ∑ Two different names (aliasing)
- ∑ If *dict*deletes *list*⇒ danglingpointer

Solutions:

- o Backpointers, so we can delete all pointers
  Variable size records a problem
- o Backpointers using a daisy chain organization
- o Entry-hold-count solution
- o New directory entrytype
- o **Link** – another name (pointer) to an existingfile
- o **Resolve the link** – follow pointer to locate thefile

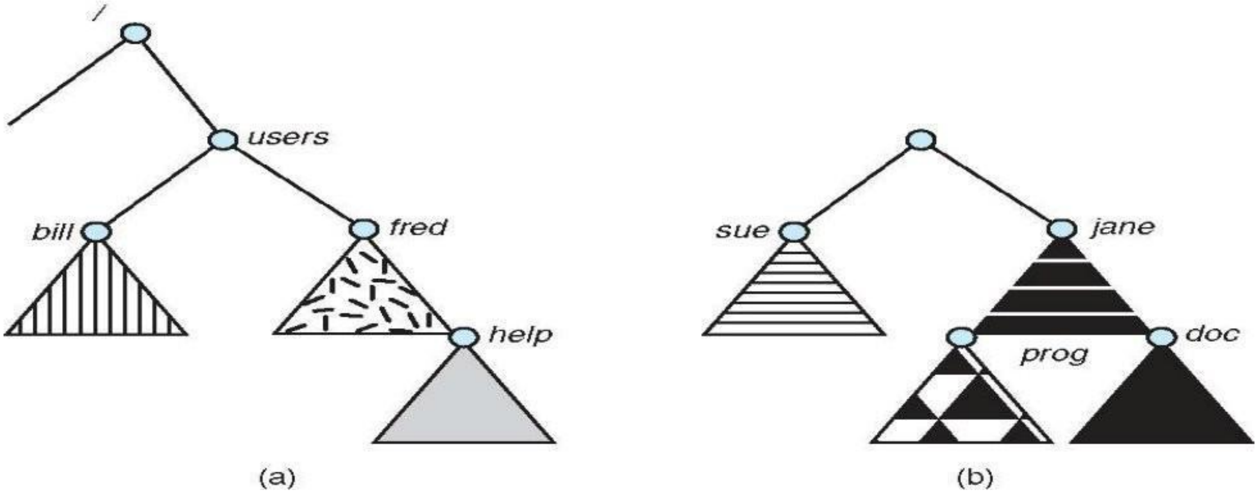
General Graph Directory



File System Mounting

- ∑ A file system must be **mounted** before it can beaccessed
- ∑ A unmounted file system (i.e., Fig. 11-11(b)) is mounted at a **mountpoint**

(a) Existing (b) Unmounted Partition



File Sharing

- ∑ Sharing of files on multi-user systems isdesirable
- ∑ Sharing may be done through a **protection**scheme
- ∑ On distributed systems, files may be shared across anetwork
- ∑ Network File System (NFS) is a common distributed file-sharingmethod

File Sharing – Multiple Users

- ∑ **User IDs** identify users, allowing permissions and protections to beper-user
- ∑ **Group IDs** allow users to be in groups, permitting group accessrights

Remote File Systems

- ∑ Uses networking to allow file system access betweensystems
  - o Manually via programs like FTP
  - o Automatically, seamlessly using **distributed filesystems**
  - o Semi automatically via the **world wideweb**
- ∑ **Client-server** model allows clients to mount remote file systems fromservers
  - o Server can serve multiple clients
  - o Client and user-on-client identification is insecure or complicated
  - o **NFS** is standard UNIX client-server file sharingprotocol
  - o **CIFS** is standard Windowsprotocol
  - o Standard operating system file calls are translated into remotecalls
- ∑ Distributed Information Systems **(distributed naming services)** such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

Failure Modes

- ∑ Remote file systems add new failure modes, due to network failure, serverfailure
- ∑ Recovery from failure can involve state information about status of each remoterequest
- ∑ Stateless protocols such as NFS include all information in each request, allowing easy recovery but less security

Consistency Semantics

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

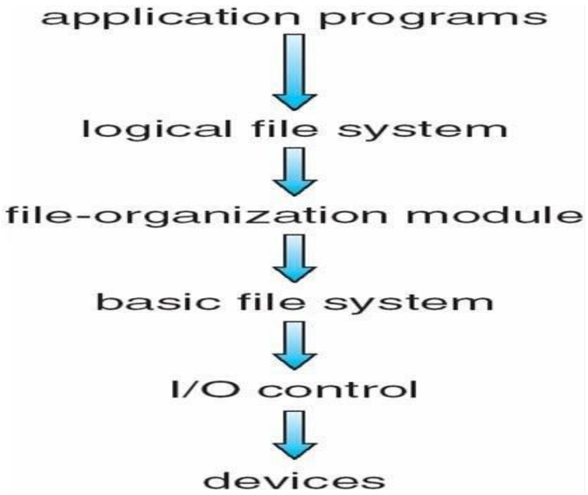
- ∑ Similar to Ch 7 process synchronization algorithms
  - 4TendtobelesscomplexduetodiskI/Oandnetworklatency(forremote filesystems
- ∑ Andrew File System (AFS) implemented complex remote file sharingsemantics
- ∑ Unix file system (UFS) implements:
  - 4Writestoanopenfilevisibleimmediatelytootherusersofthesameopen file
  - 4 Sharing file pointer to allow multiple users to read and write concurrently
- ∑ AFS has session semantics
  - 4 Writes only visible to sessions starting after the file is closed

File System Implementation\

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- **File system** resides on secondary storage(disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and randomaccess
  - I/O transfers performed in **blocks** of **sectors** (usually 512bytes)
- **File control block** – storage structure consisting of information about afile
- **Device driver** controls the physicaldevice
- File system organized into layers

Layered File System

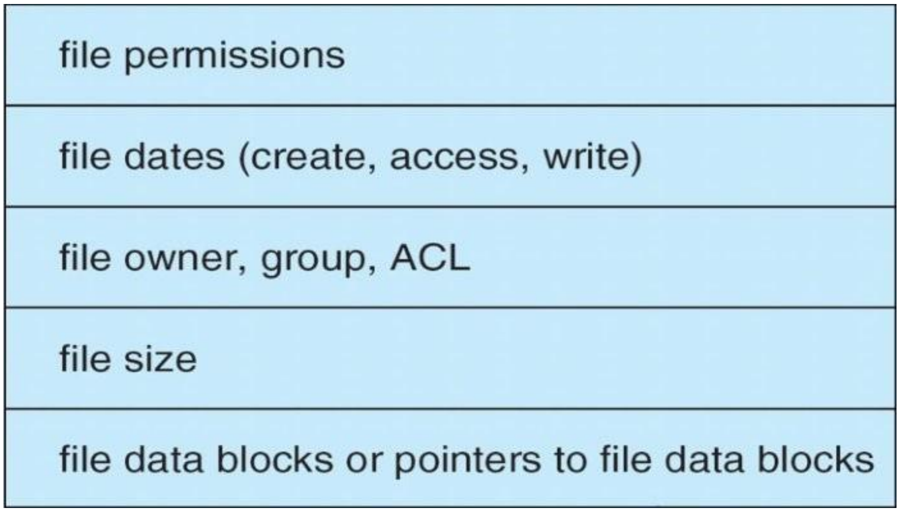


File-System Implementation

- We have system calls at the API level, but how do we implement theirfunctions?
  - On-disk and in-memory structures
- **Boot control block** contains info needed by system to boot OS from thatvolume
  - Needed if volume contains OS, usually first block ofvolume
- **Volume control block (superblock, master file table)** contains volumedetails
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files

  - Names and inode numbers, master file table
- Per-file **File Control Block (FCB)** contains many details about thefile
  - Inode number, permissions, size, dates
  - NFTS stores into in master file table using relational DBstructures

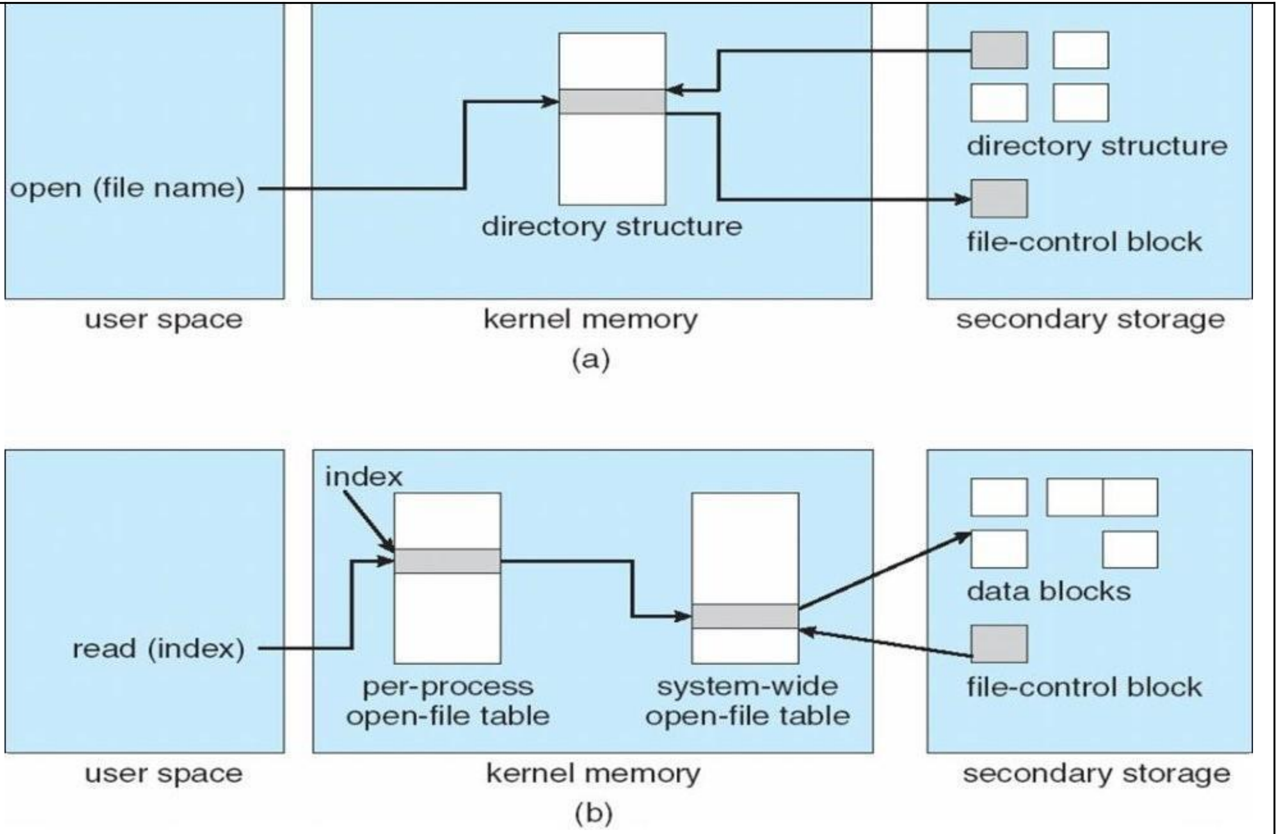
A Typical File Control Block



file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

In-Memory File System Structures

- Mount table storing file system mounts, mount points, file systemtypes
- The following figure illustrates the necessary file system structures provided by the operating systems
- Figure 12-3(a) refers to opening a file
- Figure 12-3(b) refers to reading a file
- Plus buffers hold data blocks from secondary storage
- Open returns a file handle for subsequent use
- Data from read eventually copied to specified user process memoryaddress



Virtual File Systems

∑ Virtual File Systems (VFS) on Unix provide an object-oriented way of implementing file systems

∑ VFS allows the same system call interface (the API) to be used for different types of file systems

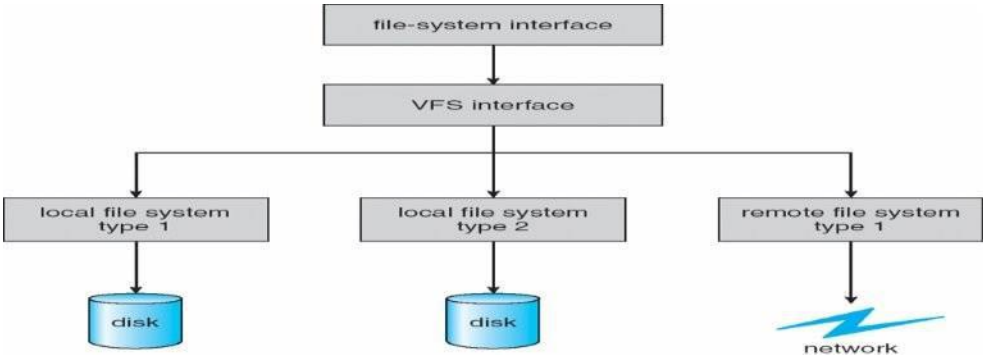
  o Separates file-system generic operations from implementation details

  o Implementation can be one of many file systems types, or network filesystem

    4 Implements vnodes which hold inodes or network file details

  o Then dispatches operation to appropriate file system implementationroutines

∑ The API is to the VFS interface, rather than any specific type of filesystem

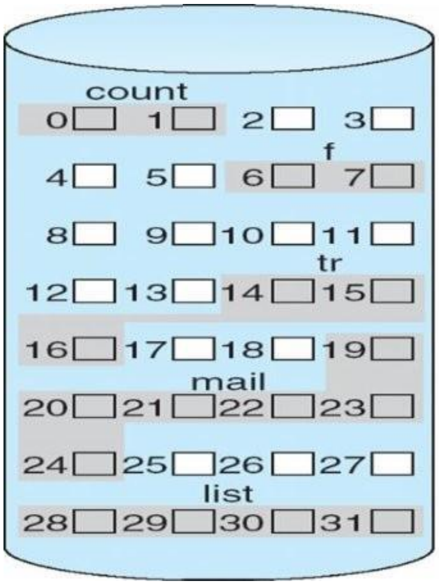


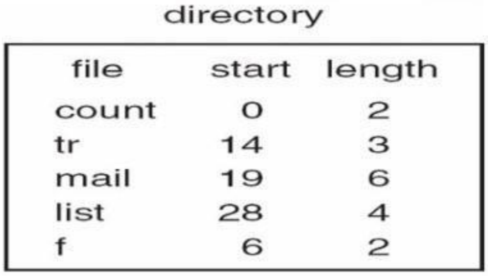
Directory Implementation

- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash datastructure
  - Decreases directory search time
  - **Collisions** – situations where two file names hash to the samelocation
  - Only good if entries are fixed size, or use chained-overflow method

Allocation Methods – Contiguous

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguousblocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or**on-line**



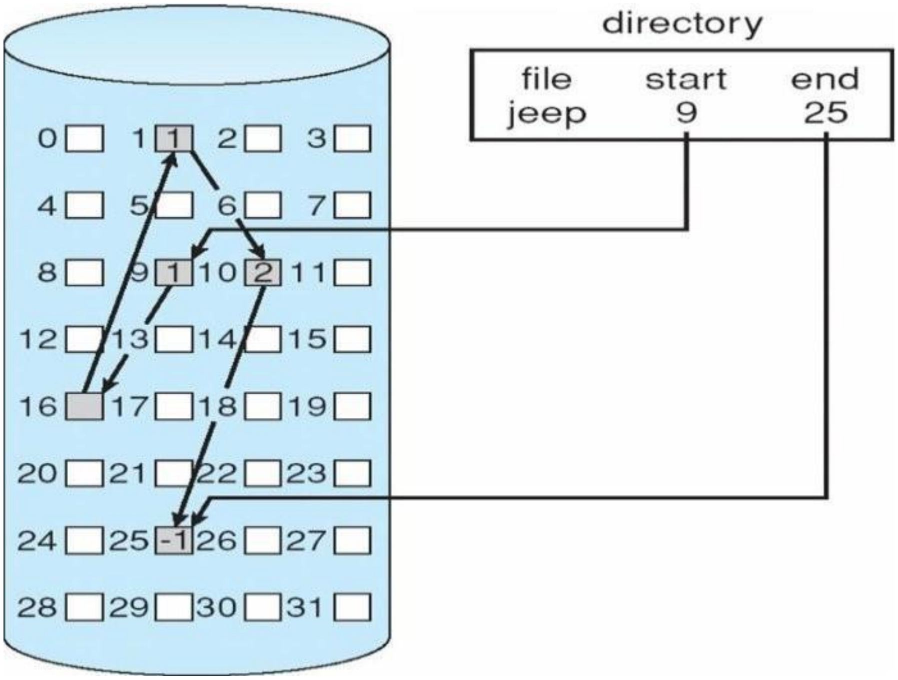


directory

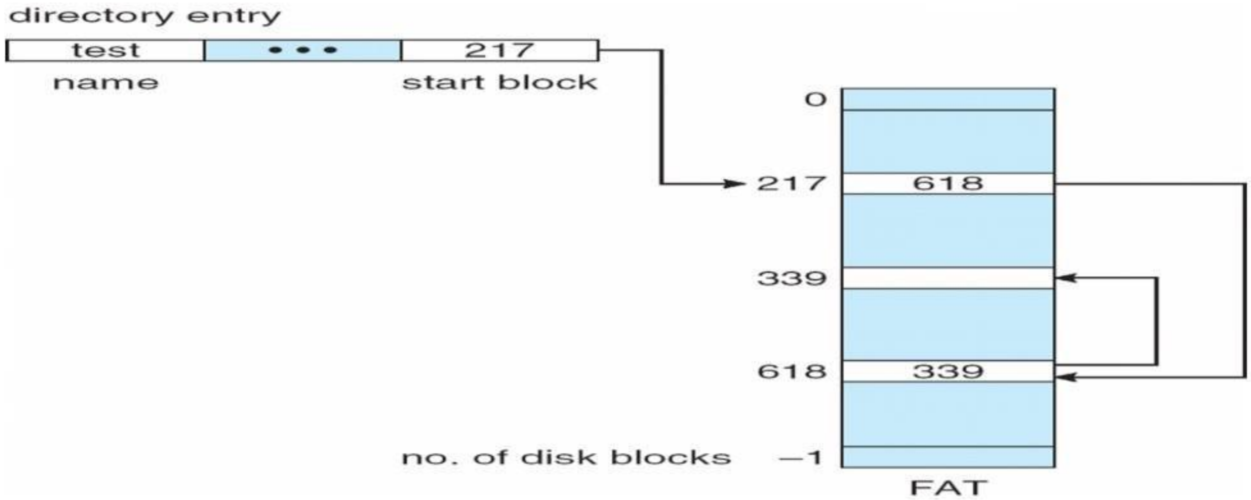
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Linked

- **Linked allocation** – each file a linked list of blocks
    - File ends at nil pointer
    - No external fragmentation
    - Each block contains pointer to next block
    - No compaction, external fragmentation
    - Free space management system called when new blockneeded
    - Improve efficiency by clustering blocks into groups but increases internal fragmentation
    - Reliability can be a problem
    - Locating a block can take many I/Os and diskseeks
- FAT (File Allocation Table) variation
    - Beginning of volume has table, indexed by block number
    - Much like a linked list, but faster on disk and cacheable
    - New block allocation simple

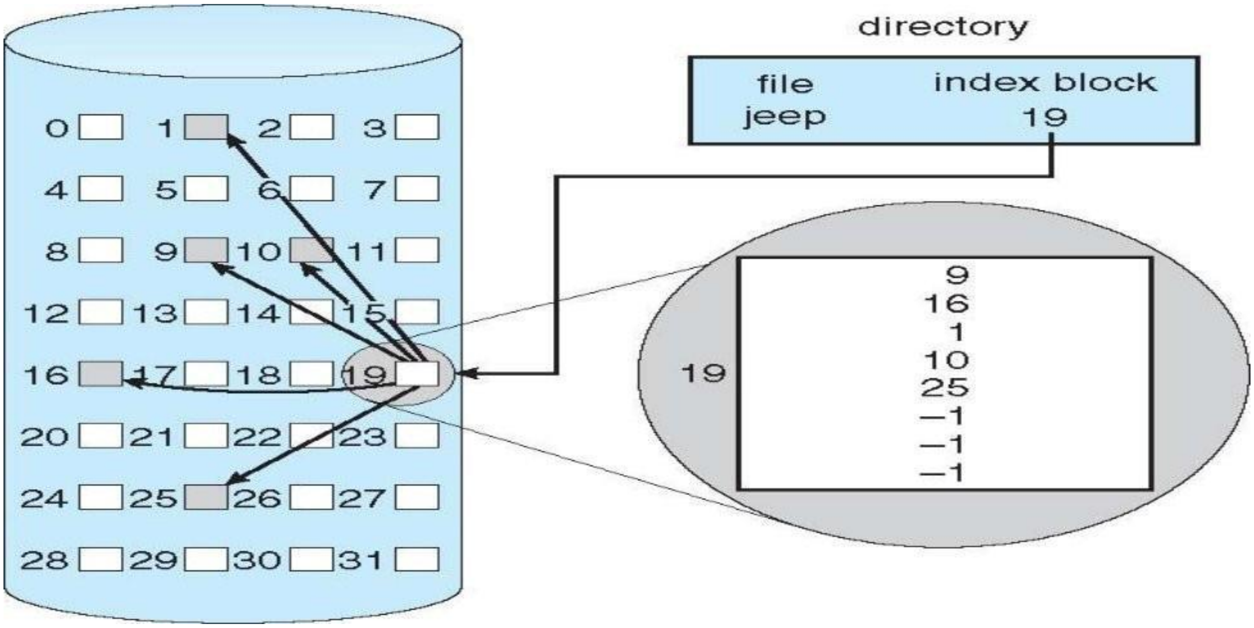


File-Allocation Table



Indexed

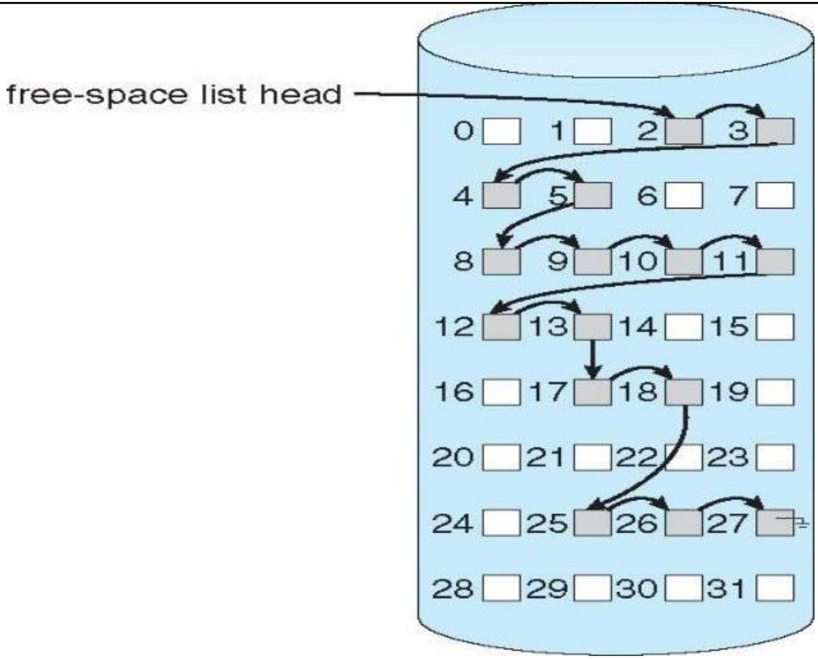
- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its datablocks



Free-Space Management

- File system maintains **free-space list** to track availableblocks/clusters
- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocksrecorded)

Linked Free Space List on Disk



Grouping

- Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like thisone).

Counting

- Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering.
- Keep address of first free block and count of following freeblocks.
- Free space list then has entries containing addresses andcounts.

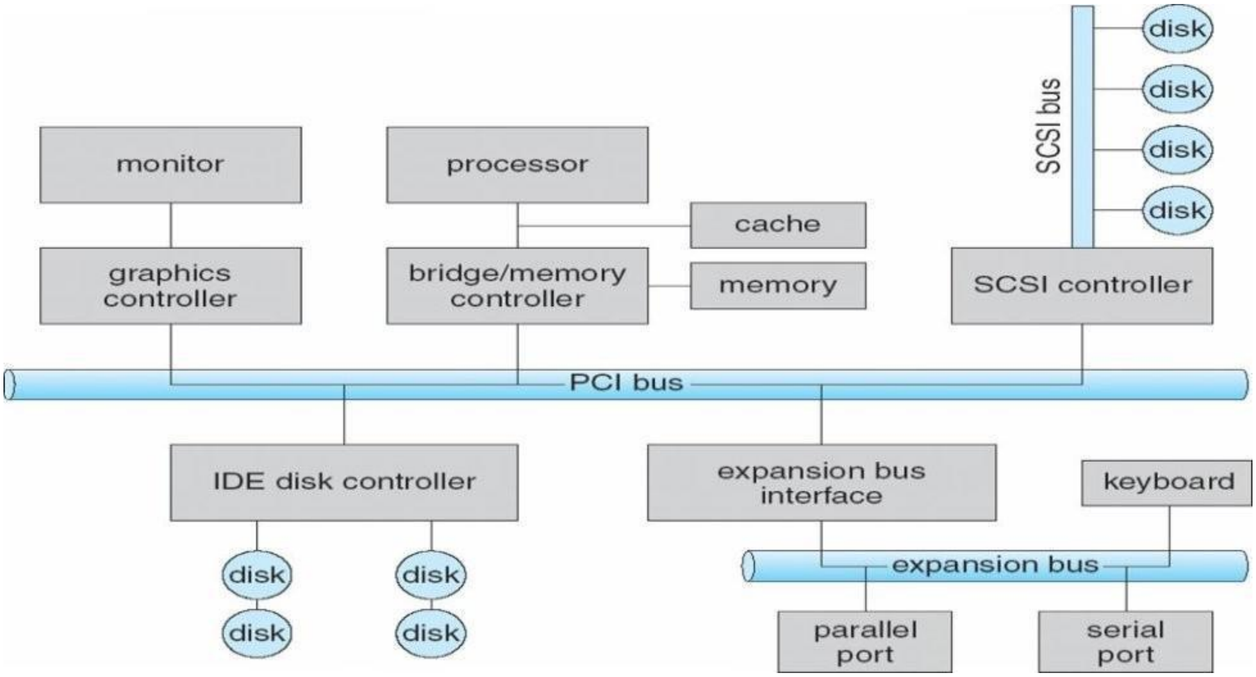
UNIT-5

I/O Systems, Protection, Security

I/O Hardware

- Incredible variety of I/O devices
    - Storage
    - Transmission
    - Human-interface
- Common concepts – signals from I/O devices interface withcomputer
    - Port – connection point fordevice
    - Bus - daisy chain or shared directaccess
    - Controller (host adapter) – electronics that operate port, bus,device
        - Sometimes integrated
        - Sometimes separate circuit board (host adapter)
        - Contains processor, microcode, private memory, bus controller, etc
            - Some talk to per-device controller with bus controller, microcode, memory, etc

A Typical PC Bus Structure



- I/O instructions control devices

- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
  - Data-in register, data-out register, status register, control register
  - Typically 1-4 bytes, or FIFObuffer
- Devices have addresses, used by
  - Direct I/Oinstructions
  - Memory-mappedI/O
    - Device data and command registers mapped to processor address space
    - Especially for large address spaces (graphics)

Polling

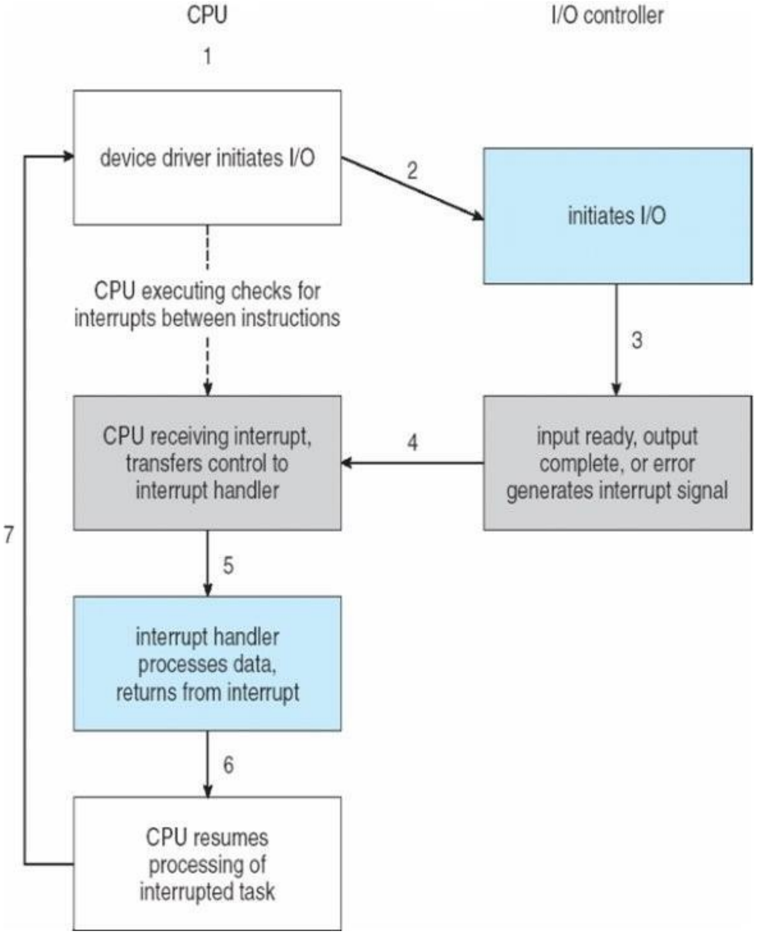
- For each byte ofI/O
  1. Read busy bit from status register until0
  2. Host sets read or write bit and if write copies data into data-outregister
  3. Host sets command-readybit
  4. Controller sets busy bit, executestransfer
  5. Controller clears busy bit, error bit, command-ready bit when transferdone
  6. Step 1 is busy-wait cycle to wait for I/O fromdevice
  7. Reasonable if device is fast
  8. But inefficient if deviceslow
  9. CPU switches to othertasks?
     - But if miss a cycle data overwritten / lost

Interrupts

- Polling can happen in 3 instructioncycles
  - Read status, logical-and to extract status bit, branch if notzero
  - How to be more efficient if non-zeroinfrequently?
- CPU **Interrupt-request line** triggered by I/O device
  - Checked by processor after each instruction
- **Interrupt handler** receives interrupts
  - **Maskable** to ignore or delay someinterrupts
- Interrupt vector to dispatch interrupt to correct handler

- Context switch at start andend
- Based on priority
- Some **non maskable**
- Interrupt chaining if more than one device at same interruptnumber

Interrupt-Driven I/O Cycle


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
4
CPU receiving interrupt, transfers control to interrupt handler
input ready, output complete, or error generates interrupt signal
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

∑ Interrupt mechanism also used for exceptions

- Terminate process, crash system due to hardwareerror
- Page fault executes when memory accesserror

∑ System call executes via trap to trigger kernel to executerequest

∑ Multi-CPU systems can process interruptsconcurrently

- If operating system designed to handleit
- Used for time-sensitive processing, frequent, must befast