CBCS SCHEME PAPER SET BY Dr. GNANESWARI 2022-23

USN 1CR22MC098

22MCA12

First Semester MCA Degree Examination, Jan./Feb. 2023

Operating System Concepts

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

VTU-14-06-2023 01:08:01pm CR - CR - CR

		Module – 1	M	L	C
Q.1	a.	What is an operating system? Explain the various services of the operating system.	10	L1	CO1
	b.	What is a system call? Explain the different types of system calls.	10	L1	CO1
		OR			
Q.2	a.	Explain simple, layered and microkernel structures of the operating system.	10	L1	CO1
	b.	What are Virtual Machines? Explain the implementation of virtual machines.	10	L1	CO1
		Module – 2			
Q.3	a.	What is a process? Explain the five state process model with a neat diagram.	10	L1	CO1
	b.	Consider the following processes, which have arrived at the ready queue with the burst and the arrival time given in milliseconds as shown below: (see table below) Draw the Gantt chart and calculate the average waiting time using the following scheduling algorithms: (i) FCFS (ii) SJF (Preemptive) (iii) RR (Q = 4)	10	L3	CO5
		OR			
Q.4	a.	What is a process control block? Explain the use of PCB in context switching.	10	L2	CO1
	b.	What are user threads and kernel threads? Explain the various multithreading models.	10	L2	CO1
		Module – 3			
Q.5	a.	What is a critical section problem? Illustrate Peterson's two process solution for a critical section problem.	10	L2	CO2
	b.	What are Semaphores? Explain the producer-consumer problem and give a solution using semaphores.	10	L3	CO4
		OR			

Table for Q.3 b:

Process	Burst Time	Arrival Time
P1	8	0
P2	4	1
P3	9	2
P4	5	3

Q.6	a.	What is a deadlock? What are the necessary conditions for a deadlock to occur?	10	L2	CO2
	b.	Consider the following snapshot of a system: (see table below) Answer the following questions using Bankers Algorithm: (i) Construct a need matrix. (ii) Is the system in a safe state? If yes, what is the safe sequence? (iii) If process P_1 makes a request (1, 0, 2), can the request be granted?	10	L3	CO5

	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			

Module – 4

Q.7	a.	Write a C program to simulate the multiprogramming with variable member of tasks (MVT) memory management technique. Given the size of the memory, size of OS, number of processes and size of each process, calculate the external fragmentation.	10	L3	CO5
	b.	What is Paging? Explain the paging hardware with a neat diagram.	10	L2	CO5

OR

Q.8	a.	What is demand paging? Explain how demand paging can be implemented.	10	L2	CO5
	b.	Consider the following reference string:7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 How many page faults would occur in case of the following page replacement algorithms: (i) Optimal (ii) LRU? Assuming 3 frames. Note : Initially all frames are empty.	10	L3	CO3

Module – 5

Q.9	a.	What are the various access methods used for accessing files?	10	L2	CO5
	b.	Explain the various directory structures with neat diagrams.	10	L2	CO5

OR

Q.10	a.	Write a note on different file allocation methods.	10	L2	CO5
	b.	Show how free space management is done using: (i) Bit vector (ii) Linked list (iii) Grouping (iv) Counting	10	L3	CO5

* * * * *

1. A. Operating System Services

- One set of operating-system services provides functions that are helpful to the user
- Communications – Processes may exchange information, on the same computer or between computers over a network.
- Communications may be via shared memory or through message passing (packets moved by the OS)
- Error detection – OS needs to be constantly aware of possible errors may occur in the CPU and memory hardware, in I/O devices, in user program
- For each type of error, OS should take the appropriate action to ensure correct and consistent computing.
- Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system.

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
- Resource allocation - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
- Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code
- Accounting - To keep track of which users use how much and what kinds of computer resources
- Protection and security - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other.
- Protection involves ensuring that all access to system resources is controlled.
- Security of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts.
- If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

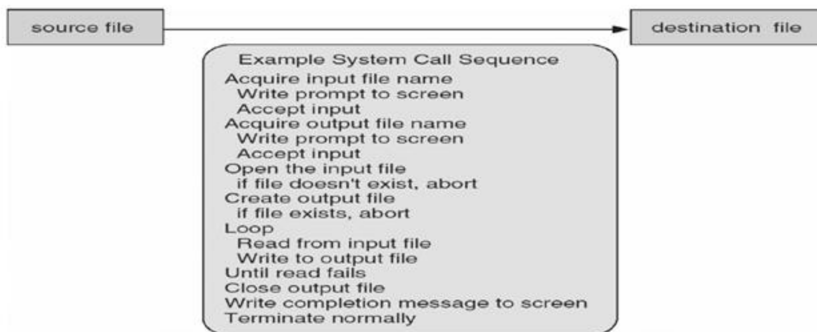
1.b

SystemCalls

- Programminginterfacetotheservicesprovided bytheOS
- Typicallywritteninahigh-level language(CorC++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather thandirect system call usenThree most common APIs are Win32 API for Windows, POSIX API forPOSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and JavaAPIfor theJava virtualmachine(JVM)

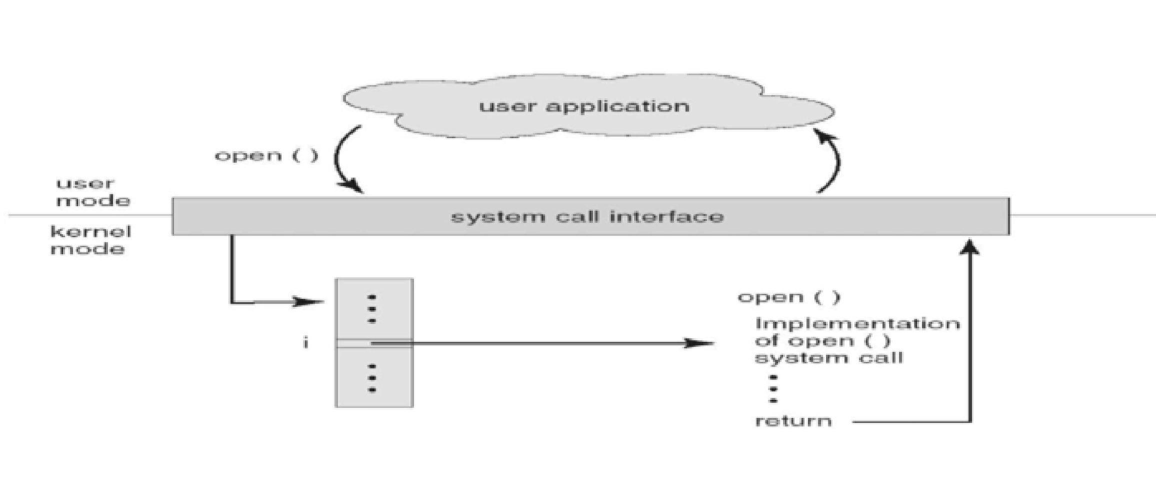
- Why use APIs rather than system calls?(Note that the system-call names used throughout this textaregeneric)

Exampleof SystemCalls



SystemCallImplementation

- Typically,anumber associatedwitheachsystemcall
- System-callinterfacemaintainsatableindexedaccordingtothesenumbers
- Thesystemcall interfaceinvokesintendedsystemcallinOSkernel andreturnsstatusofthesystemcalland anyreturnvalues
- Thecallerneed knownothingabouthowthesystemcallisimplemented
- Justneeds toobeyAPIand understandwhatOSwilldoas aresultcall
- Most details of OS interface hidden from programmer by API Managed by run-time supportlibrary(setoffunctionsbuiltinto librariesincluded with compiler)



1. 2.a. modernoperatingsystemsare**interruptdriven.**Iftherearenoprocessestoexecute,noI/Odevicesto service, and no users to whom to respond, an operating system will sit quietly, waiting forsomethingtohappen.Eventsarealmostalwayssignaledbytheoccurrenceofaninterruptoratrap
2. A **trap (or** an **exception)** is a software-generated interrupt causedeitherbyan erroror bya specificrequestfroma userprogram thatanoperating-systemserviceisperformed.
3. Theinterrupt-drivennatureofanoperatingsystemdefinesthatsystem'sgeneralstructure.Foreachtype of interrupt, separate segments of code in the operating system determine what action shouldbetaken.Aninterruptserviceroutineisprovidedthat isresponsiblefordealingwiththeinterrupt.

4. The operating system and the users share the hardware and software resources of the computersystem, we need to make sure that an error in a user program could cause problems only for theoneprogramthatwasrunning.Withsharing,manyprocessescouldbeadverselyaffectedbyabuginoneprogram.Forexample,ifaprocessgetsstuckinaninfiniteloop,thisloopcouldpreventthecorrect operationof manyotherprocesses.
5. Without protection against these sorts of errors, either the computer must execute only oneprocess atatime oralloutput mustbe suspect.

Dual-ModeOperation

Dual-mode operation allows OS to protect itself and other systemcomponents**Usermode**and **kernelmode**

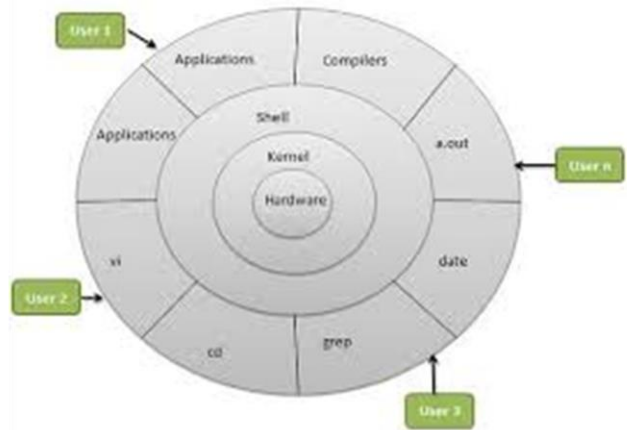
Mode bit provided by hardware Provides ability to distinguish when system is running user codeorkernelcodeSomeinstructionsdesignatedas**privileged**,onlyexecutableinkernelmodeSystemcall changes mode to kernel, returnfromcallresetsittouser

Transitionfrom User toKernelMode

- Timertopreventinfiniteloop/ process hoggingresourcesSetinterruptafterspecificperiod
- Operatingsystemdecrements counter
- Whencounter zerogenerateaninterrupt
- Set upbeforeschedulingprocesstoregaincontrol orterminateprogramthatexceedsallottedtime

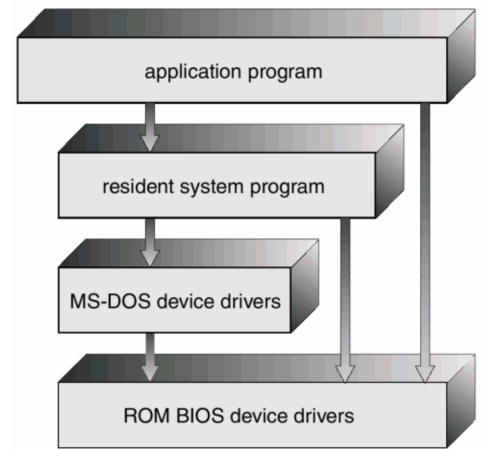
1. Simple Structure:

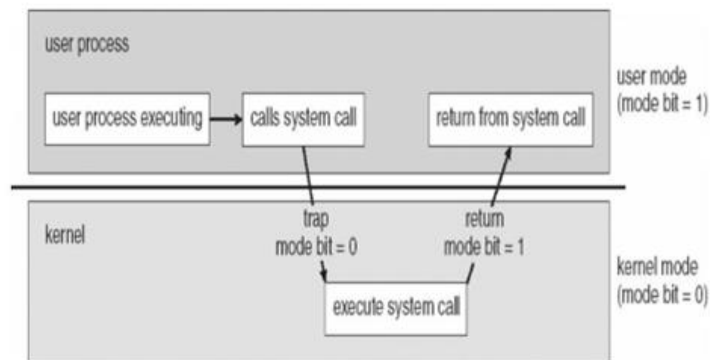
- **Shell:** It acts as a command interpreter, a program that acts as an interface b/w kernel and the user.
- **Kernel:** It is the core of the OS, It controls the resources.



2. Layered Structure:

- In the layered approach, the OS is divided into a number of layers (levels), each built on top of lower layers.
- The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- Each layer uses functions (operations) and services of only lower-level layers.

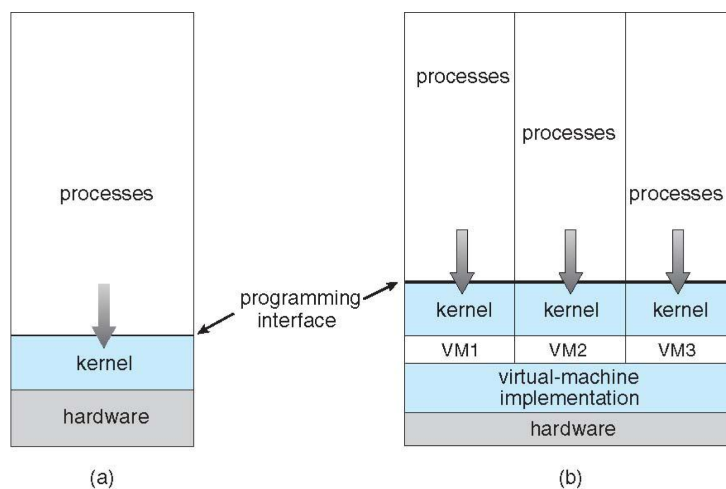




2.b

Virtual Machines

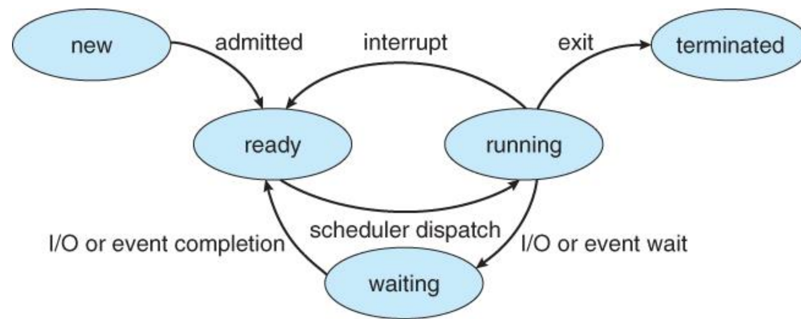
- The fundamental idea behind a virtual machine is to abstract the hardware of a single computer into several environments.
- Thus creating an illusion that each separate execution environment is running its own private computer by using CPU scheduling and virtual memory techniques.
- Each process is provided with a virtual copy of the underlying computer.
- Eg: VM Virtual Box, VMware



Advantages:

- **Consolidation:** Multiple OS can run in isolation in the same server eliminating the need to dedicate 1 machine for 1 application.
- **Sharing:** Hardware can be shared.
- **Network:** A network of virtual machines can send information across each other.
- **Development Flexibility:** A virtual machine can host different OS in the same system, allowing the user to test his program in various versions.
- **Security:** Crashing in one virtual machine will not bring down the whole system.

.



ProcessState

Asaprocessexecutes,itchanges**state.**Thestateofaprocessisdefinedinpartbythecurrentactivityofthat process. Each processmaybe in one ofthefollowingstates:

- **New.**The process is beingcreated.
- **Running.**Instructionsarebeingexecuted.
- **Waiting.** The process is waiting for some event to occur (such as anI/Ocompletionorreceptionof asignal).
- **Ready.**The processiswaitingto beassignedtoa processor.
- **Terminated.**Theprocesshas finishedexecution.

Thesenamesarearbitrary,andtheyvaryacrossoperatingsystems.Certainoperatingsystemsalsomorefinelydelineateprocessstates.Itisimportanttorealize thatonlyone process can be *running*on any processor at any instant.

3. B.

Open with Google Docs

(i) FCFS

P1	P2	P3	P4

0 8 12 21 26

$$\text{Avg. WT} = \frac{0+(8-1)+(12-2)+(21-3)}{4} = \underline{\underline{8.75}}$$

$$\text{Avg. TAT} = \frac{(8-0)+(12-1)+(21-2)+(26-3)}{4} = \underline{\underline{15.25}}$$

(ii) SJF (Preemptive)

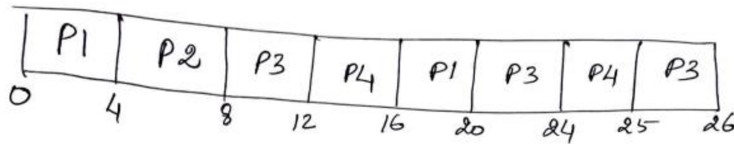
P1	P2	P4	P1	P3

0 1 5 10 17 26

$$\text{Avg. WT} = \frac{(10-1)+(1-1)+(17-2)+(5-3)}{4} = \underline{\underline{6.5}}$$

$$\text{Avg. TAT} = \frac{(17-0)+(5-1)+(26-2)+(10-3)}{4} = \underline{\underline{13}}$$

1) Round Robin (Q=4)



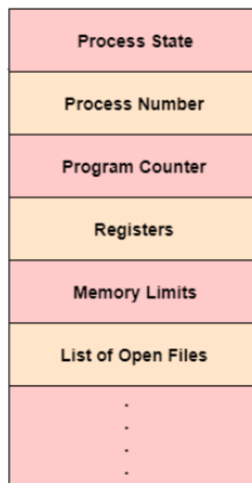
$$\text{Avg. WT} = \frac{(16-4)+(4-1)+(25-10)+(24-7)}{4} = 11.75$$

$$\text{Avg. TAT} = \frac{(20-4)+(8-1)+(26-8)+(25-4)}{4} = 18$$

4. a.

Each process is represented in the operating system by a **process control block (PCB)**—also called a *taskcontrol block.*

Processstate. Thestatemaybenew, ready,running,andwaiting,halted,andso on.



Process Control Block (PCB)

Program counter-The counter indicates the address of the next instruction to be executed for thisprocess.
• CPU **registers-** The registers vary in number and type, depending on the computer architecture. Theyinclude accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-codeinformation.

CPU-schedulinginformation-
Thisinformationincludesaprocesspriority,pointerstoschedulingqueues,andanyotherschedulingparameters.

Memory-management information- This information may include such information as the value of thebase and limit registers, the page tables, or the segment tables, depending on the memory system used bythe operatingsystem

Accounting information-This information includes the amount of CPU and real time used, time limits,account members,job or processnumbers, and so on.

I/O status information-This information includes the list of I/O devices allocated to the process, a list ofopen files,and soon.

4.b.

Threads

- A thread is a basic unit of CPU utilization.
- It consists of a thread ID, program counter, a stack, and a set of registers.
- Traditional processes have a single thread of control. It is also called as **heavyweight process**. There is one program counter, and one sequence of instructions that can be carried out at any given time.
- A multi-threaded application have multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code, data, and certain structures such as open files. Such process are called **as lightweight process.**

Advantages

Responsiveness – may allow continued execution if part of process is blocked, especially important for user interfaces

Resource Sharing – threads share resources of process, easier than shared memory or message passing

Economy – cheaper than process creation, thread switching lower overhead than context switching

Scalability – process can take advantage of multiprocessor architectures

Types of Threads

1) **User Threads** are above the kernel and are managed without kernel support.
2) **Kernel Threads**-managed by the operating system itself

There are 3 types of relationships b/w user threads and kernel threads.

User level threads

Each thread is represented by a PC, registers, stack and a small control block, all stored in the user process address space.

Thread management done by user-level threads library.

Three primary thread libraries: POSIX Pthreads ,Win32 threads ,Java threads

User-level threads implement in user-level libraries, rather than via systems calls, so

thread switching does not need to call operating system and to cause interrupt to the kernel.

Fast and Efficient: Thread switching is not much more expensive than a procedure call.

Simple Management: Creating a thread, switching between threads and synchronization between threads can

all be done without intervention of the kernel.

Kernel level threads

Instead of thread table in each process, the kernel has a thread table that keeps track of all threads in the system.

The kernel-level threads are slow and inefficient. For instance, threads operations are hundreds of times slower than that of user-level threads.

Kernel must manage and schedule threads as well as processes. It requires a full thread control block (TCB) for each thread to maintain information about threads.

As a result there is significant overhead and increased in kernel complexity.

Multithreaded Processes

Many-to-One Model

Many user-level threads are mapped to single kernel thread.

Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time.

Thread management is done by thread library in user space.

If one of the thread makes a blocking system call then the entire process will be blocked.

Few systems currently use this model

Examples:

Solaris Green Threads

GNU Portable Threads

One-to-One Model

Each user-level thread maps to one kernel thread

Creating a user-level thread creates a kernel thread

Allows multiple threads to run in parallel on multiprocessors.

Number of threads per process sometimes restricted due to overhead

Examples

Windows

Linux

Solaris 9 and later

Many-to-Many Model

Allows many user level threads to be mapped to many kernel threads

Allows the operating system to create a sufficient number of kernel threads

Example:

Solaris prior to version 9

Windows with the *ThreadFiber* package

Multicore Programming

A recent trend in computer architecture is to produce chips with multiple cores, or CPUs on a single chip.

A multi-threaded application running on a traditional single-core chip, would have to execute the threads one after another.

On a multi-core chip, the threads could be spread across the available cores, allowing true parallel processing.

Thread Library

A thread library provides the programmer with an API for creating and managing thread.

There are two primary ways of implementing thread library, which are as follows −

- The first approach is to provide a library entirely in user space with kernel support. All code and data structures for the library exist in a local function call in user space and not in a system call.
- The second approach is to implement a kernel level library supported directly by the operating system. In this case the code and data structures for the library exist in kernel space.

Invoking a function in the API for the library typically results in a system call to the kernel.

The main thread libraries which are used are given below −

- **POSIX threads** − Pthreads, the threads extension of the POSIX standard, may be provided as either a user level or a kernel level library.
- **WIN 32 thread** − The windows thread library is a kernel level library available on windows systems.
- **JAVA thread** − The JAVA thread API allows threads to be created and managed directly as JAVA programs.

Threading Issues

Fork() and exec() system calls

Cancellation

Signal handling

Thread pools

Thread specific data

The fork( ) and exec( ) System Calls

The fork( ) system call is used to create a separate, duplicate process.

When a thread program calls fork( ), The new process can be a copy of the parent, with all the threads

The new process is a copy of the single thread only (that invoked the process)

If the thread invokes the exec( ) system call, the program specified in the parameter to exec( ) will be executed by the thread created.

Cancellation

Terminating the thread before it has completed its task is called thread cancellation.

The thread to be cancelled is called target thread.

Example : Multiple threads required in loading a webpage is suddenly cancelled, if the browser window is closed.

Threads that are no longer needed may be cancelled in one of two ways:

1. Asynchronous Cancellation - cancels the thread immediately.

2. Deferred Cancellation - the target thread periodically check whether it has to terminate, thus gives an opportunity to the thread, to terminate itself in an orderly fashion.

Signal Handling

Signals are software interrupts sent to a process to indicate that an important event has occurred .

Eg. SIGKILL 9 If a process gets this signal it must quit immediately.

All signals follow same path-

1) A signal is generated by the occurrence of a particular event.

2) A generated signal is delivered to a process.

3) Once delivered, the signal must be handled.

A signal can be invoked in 2 ways :

Synchronous signal delivered to the same program.

Eg illegal memory access, divide by zero error.

Asynchronous signal is sent to another program.

5.a.

Critical-Section Problem

• **Critical-section** is a segment-of-code in which a process may be
→ changing common variables
→ updating a table or
→ writing a file.
• Each process has a critical-section in which the shared-data is accessed.
• General structure of a typical process has following (Figure 2.12):

1) Entry-section

● Requests permission to enter the critical-section.

2) Critical-section

● Mutually exclusive in time i.e. no other process can execute in its critical-section.

3) Exit-section

● Follows the critical-section.

4) Remainder-section

Figure 2.12 General structure of a typical process

• Problem statement:

―Ensure that when one process is executing in its critical-section, no other process is to be allowed to execute in its critical-section‖.

• A solution to the problem must satisfy the following 3 requirements:

1) Mutual Exclusion:

● No more than one process can be in critical-section at a given time.

2) Progress:

● When no process is in the critical section, any process that requests entry into the critical section must be permitted without any delay..

3) Bounded Waiting (No starvation):

● There is an upper bound on the number of times a process enters the critical section, while another is waiting.

• Two approaches used to handle critical-sections:

1) Preemptive Kernels

● Allows a process to be preempted while it is running in kernel-mode.

● More suitable for real-time proframming

2) Non-preemptive Kernels

● Does not allow a process running in kernel-mode to be preempted as it is free from race conditions on kernel data structures, as only one process is active in the kernel at a time.

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

Figure 6.1 General structure of a typical process P_i.

Peterson's Solution

******Detailed understanding: https://nptel.ac.in/courses/106106144/26******

• This is a classic **software-based solution** to the critical-section problem.

• This is limited to 2 processes.

• The 2 processes alternate execution between

→ critical-sections and

→ remainder-sections.

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

remainder section

} while (TRUE);

• The 2 processes (say i & j)share two globally defined variables:
'turn' – indicates whose turn it is to enter its critical-section.
(i.e., if turn==i, then process Pi is allowed to execute in its critical-section).
'flag' – indicates if a process is ready to enter its critical-section.
(i.e. if flag[i]=true, then Pi is ready to enter its critical-section).
● The following code shows the structure of ***process Pi*** in Peterson's solution:

UNLOCK LOCK

• To enter the critical-section,
→ firstly, process Pi sets flag[i] to be true and
→ then sets turn to the value j.
• If both processes try to enter at the same time, turn will be set to both i and j at roughly the same time.
• The final value of turn determines which of the 2 processes is allowed to enter its critical-section first.
• To prove that this solution is correct, we show that:
1) Mutual-exclusion is preserved:
• Observation1: Pi enters the CS only if flag[j]== false or turn ==i.
• Observation2: If both processes can be executing in their CSs at the same time, then
flag[i]==flag[j]==true.
These two observations imply that Pi and Pj could not have successfully executed their *while* statements at about the same time, since the value of turn can be either i or j but cannot be both.
Hence, the process which sets 'turn' first will execute and Mutual Exclusion is preserved.
2) The progress requirement & The bounded-waiting requirement is met:
• The process which executes while statement first (say Pi), doesn't change the value of
turn. So other process (Say Pj) will enter the CS (Progress) after at most one entry
(Bounded Waiting)

5.b

Semaphores

******Detailed understanding:* ***https://nptel.ac.in/courses/106106144/30********

• A semaphore is a synchronization-tool.
• It used to control access to shared-variables so that only one process may at any point
in time change the value of the shared-variable.
• A semaphore(S) is an integer-variable that is accessed only through 2 atomic-operations:
1) wait() and
2) signal().
• wait() is termed P ("to test or decrement") signal() is termed V ("to increment").

Definition of wait(): Definition of signal():

• When one process modifies the semaphore-value, no other process can simultaneously
modify that same semaphore-value.

Semaphore Usage:

a) Binary Semaphore

● The value of a semaphore can range only between 0 and 1.
● On some systems, binary semaphores are known as **mutex locks**, as they are locks

that provide mutual-exclusion.
● Used for two processes
b) Counting Semaphore:
● The value of a semaphore can ranges over an unrestricted domain
● Used for multiple processes

```
wait (s) {
        while s <= 0  // no-op
          s = s -1
     }
signal (s) {
       s = s+1
    }
```

```
Semaphore mutex;    //  initialized to 1
do {
    wait (mutex);
       // Critical Section
    signal (mutex);
             // remainder section
    } while (TRUE);
```

Examples of Semaphore Usage:

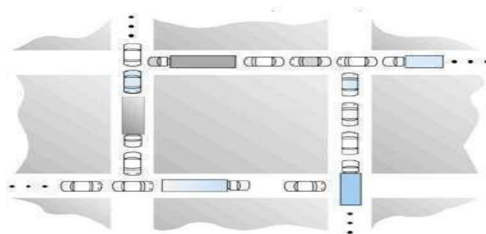
1) Solution for Critical-section Problem using Binary Semaphores
• Binary semaphores can be used to solve the critical-section problem for multiple processes.
• The _n‘ processes share a semaphore mutex initialized to 1
Mutual-exclusion implementation with semaphores
2) Use of Counting Semaphores
• Counting semaphores can be used to control access to a given resource consisting of a finite number o£ instances.
• The semaphore is initialized to the number of resources available.
• Each process that wishes to use a resource performs a wait() operation on the semaphore (thereby decrementing the count).
• When a process releases a resource, it performs a signal() operation (incrementing the count).
• When the count for the semaphore goes to 0, all resources are being used.
• After that, processes that wish to use a resource will block until the count becomes greater than 0.
3) Solving Synchronization Problems
• Semaphores can also be used to solve synchronization problems.
• For example, consider 2 concurrently running-processes:
● P1 with a statement S1 and P2 with a with a statement S2
• Suppose we require that S2 be executed only after S1 has completed.
• We can implement this scheme readily
→ by letting P1 and P2 share a common semaphore synch initialized to 0, and

→ by inserting following statements in process P1:
S1;
Signal(synch);
→ by inserting following statements in process P2:
Wait(synch);
S2;
● Because synch is initialized to 0, P2 will execute S2 only after P1 has invoked signal (synch), which is after statement S1 has been executed.

6.a.

Deadlocks

- Deadlock is a situation where a set of processes are blocked because each process is
  → holding a resource and
  → waiting for another resource held by some other process.
- Real life example:
  When 2 trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other.
- Similar situation occurs in operating systems when there are two or more processes hold some resources and wait for resources held by other(s).



- Here is an example of a situation where deadlock can occur (Figure 3.1).

Figure 3.1 Deadlock Situation

Deadlock Characterization

- In a deadlock, processes never finish executing, and system resources are tied up,preventing other jobs from starting.

1) Necessary Conditions

- There are four conditions that are necessary to achieve deadlock:

i) Mutual Exclusion

At least one resource must be held in a non-sharable mode.

i.e., If one process holds a non-sharable resource and if any other process requests this resource, then the requesting-process must wait for the resource to be released.

ii) Hold and Wait

□A process must be simultaneously
→ holding at least one resource and
→ waiting to acquire additional resources held by the other process.

iii) No Preemption

Resources cannot be preempted.

A resource can be released voluntarily by the process holding it.

iv) Circular Wait

A set of processes { P0, P1, P2, . . ., PN } mustexist

P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is heldby P2 ….and PN is waiting for a resource held by P0.

6.b

Bankers Algorithm

No. of Proceses – 4 (P1,P2,P3,P4)

No. of Resources – 3 (A, B, C)

Total available resources:

A	B	C
9	3	6

Problem:

Process	Max	Allotted	Need	Available	Completed
	A B C	A B C	A B C	A B C	
P1	3 2 2	1 0 0	2 2 2	1 1 2	P2
P2	6 1 3	5 1 1	1 0 2	6 2 3	P1
P3	3 1 4	2 1 1	1 0 3	7 2 3	P3
P4	4 2 2	0 0 2	4 2 0	9 3 4	P4
				9 3 6	

Safe Sequence : P2 -> P1 -> P3 -> P4

Remaining:

Process	Need	Available	Remaining	Completed
	A B C	A B C	A B C	
P1	2 2 2	1 1 2	0 1 0	P2
P2	1 0 2	6 2 3	4 0 1	P1
P3	1 0 3	7 2 3	6 2 0	P3
P4	4 2 0	9 3 4	5 1 4	P4
		9 3 6		

iii) Yes

7. a.

PROGRAM: a

```
#include<stdio.h>
#include<conio.h>
main()
{
int i,m,n,tot,s[20];
clrscr();
printf("Enter total memory size:");
scanf("%d",&tot);
printf("Enter no. of processes:");
scanf("%d",&n);
printf("Enter memory for OS:");
scanf("%d",&m);
for(i=0;i<n;i++)
{
printf("Enter size of process %d:",i+1);
scanf("%d",&s[i]);
}
tot=tot-m;
for(i=0;i<n;i++)
{
if(tot>=s[i])
{
printf("Allocate memory to process %d\n",i+1);
tot=tot-s[i];
}
else
printf("process p%d is blocked\n",i+1);
}
printf("External Fragmentation is=%d",tot);
getch();
}
```

OUTPUT:

```
Enter total memory size : 50
Enter no.of pages : 4
Enter memory for OS :10
Enter size of page : 10
Enter size of page : 9
Enter size of page : 9
Enter size of page : 10
External Fragmentation is = 2
```

7. b

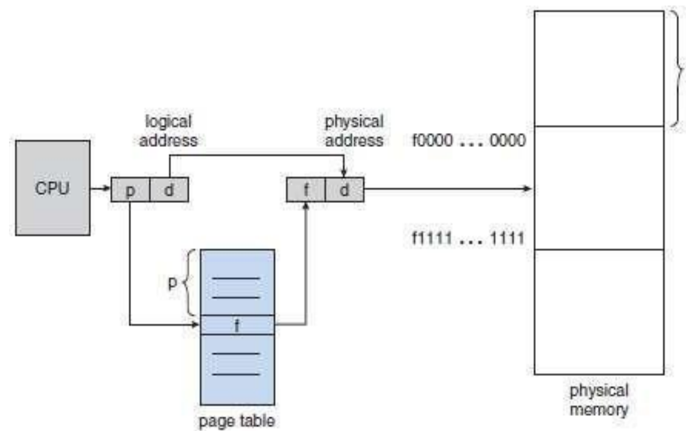
- Paging is a memory-management scheme.
- This permits the physical-address space of a process to be non-contiguous.
- This also solves the considerable problem of fitting memory-chunks of varying sizesonto the backing-store.
- Traditionally: Support for paging has been handled by hardware.Recent designs: The hardware & OS are closely integrated.

1) **Basic Method**

□ Physical-memory is broken into fixed-sized blocks calledframes (Figure 3.16). Logical-memory is broken into same-sized blocks called **pages.**

□When a process is to be executed, its pages are loaded into any availablememory-frames from the backing-store.

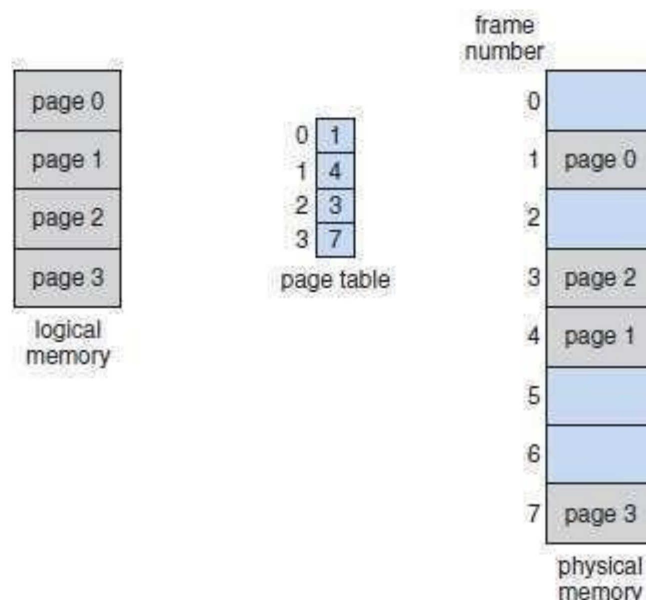
□The backing-store is divided into fixed-sized blocks that are of the same size as



the memory-frames.

Figure 3.16 Paging hardware

- The page-table contains the base-address of each page in physical-memory.
- Address generated by CPU is divided into 2 parts (Figure 3.17):
  1) **Page-number(p)** is used as an index to the page-table and
  2) **Offset (d)** is combined with the base-address to definethe physical-address. This physical-address is sent to



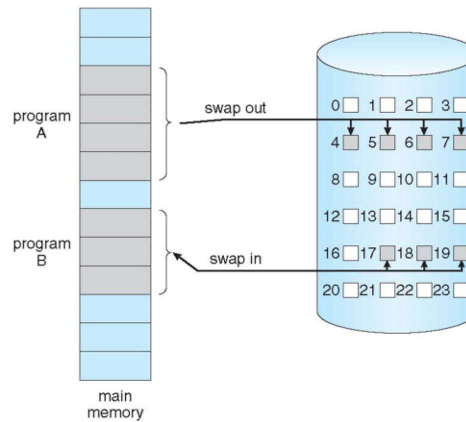
the memory-unit.

Figure 3.17 Paging model of logical andphysical-memory

□The page-size (like the frame size) is defined by the hardware (Figure 3.18).

□If the size of the logical-address space is 2^m, and a page-size is 2^n addressing-units

- Entire process cannot be loaded into memory at once
- Page is loaded only when it is needed
- Page is needed ⇒ reference to it
  - not-in-memory ⇒ bring to memory
- **Lazy swapper** – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**



8.b

(i) LRU with 3 frames:

Frames	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
No. of Page faults	√	√	√	√		√		√	√	√	√			√		√		√		

No of page faults=12

(ii) FIFO with 3 frames:

Frames	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1	7	0	1	2	2	3	0	4	2	3	0	0	0	1	2	2	2	7	0	1
2		7	0	1	1	2	3	0	4	2	3	3	3	0	1	1	1	2	7	0
3			7	0	0	1	2	3	0	4	2	2	2	3	0	0	0	1	2	7
No. of Page faults	√	√	√	√		√	√	√	√	√	√			√	√			√	√	√

No of page faults=15

(iii) Optimal with 3 frames:

Frames	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
No. of Page faults	√	√	√	√		√		√			√			√				√		

No of page faults=9

Conclusion: The optimal page replacement algorithm is most efficient among three algorithms, as it has lowest page faults i.e. 9.

9.a

Sequential Access

- This is based on a tape model of a file.
- This works both on
  - → sequential-access devices and
  - → random-access devices.
- Info. in the file is processed in order (Figure 4.15).

For ex: editors and compilers

- Reading and writing are the 2 main operations on the file.
- File-operations:

  1) read next

  ➢ This is used to

  → read the next portion of the file and

  → advance a file-pointer, which tracks the I/O location.

  2) write next

  ➢ This is used to

  → append to the end of the file and

  → advance to the new end of file.

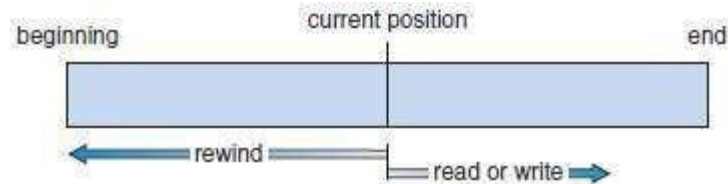


Figure 4.15 Sequential-access file

Direct Access (Relative Access)

- This is based on a disk model of a file (since disks allow random access to any file-block).
- A file is made up of fixed length logical records**.**
- Programs can read and write records rapidly in no particular order.
- Disadvantages:

  3) Useful for immediate access to large amounts of info.

  4) Databases are often of this type.

- File-operations include a relative block-number as parameter.
- The **relative block-number** is an index relative to the beginning of the file.
- File-operations (Figure 4.16):

  1) read n

  2) write n

  where n is the block-number

- Use of relative block-numbers:

  → allows OS to decide where the file should be placed and

  → helps to prevent user from accessing portions of file-system that may not be part of his file.

sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp ; cp = cp + 1;
write_next	write cp; cp = cp + 1;

Figure 4.16 Simulation of sequential access on a direct-access file

Other Access Methods

- These methods generally involve constructing a **file-index**.
- The index contains pointers to the various blocks (like an index in the back of a book).
- To find a record in the file(Figure 4.17):
    5) First, search the index and
    6) Then, use the pointer to
        → access the file directly and
        → find the desired record.
- Problem: With large files, the index-file itself may become too large to be kept in memory.

Solution: Create an index for the index-file. (The primary index-file may contain pointers to secondary index-files, which would point to the actual data-items).

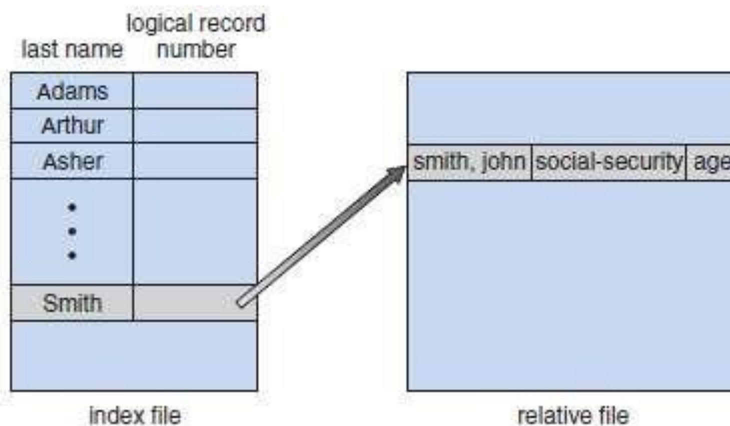


Figure 4.17 Example of index and relative files

9.b

4.9.1 Single Level Directory

- All files are contained in the same directory (Figure 4.19).
- Disadvantages (Limitations):
    1) Naming problem: All files must have unique names.
    2) Grouping problem: Difficult to remember names of all files, as number of files increases.

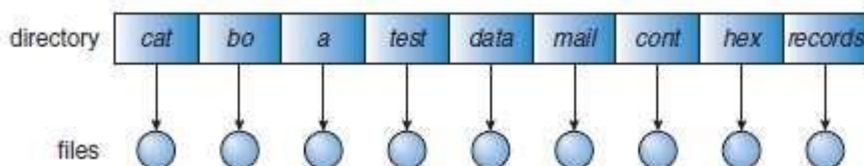


Figure 4.19 Single-level directory

4.9.2 Two Level Directory

- A separate directory for each user.
- Each user has his own UFD (user file directory).
- The UFDs have similar structures.
- Each UFD lists only the files of a single user.
- When a user job starts, the system's MFD is searched (MFD=master file directory).
- The MFD is indexed by user-name.
- Each entry in MFD points to the UFD for that user (Figure 4.20).

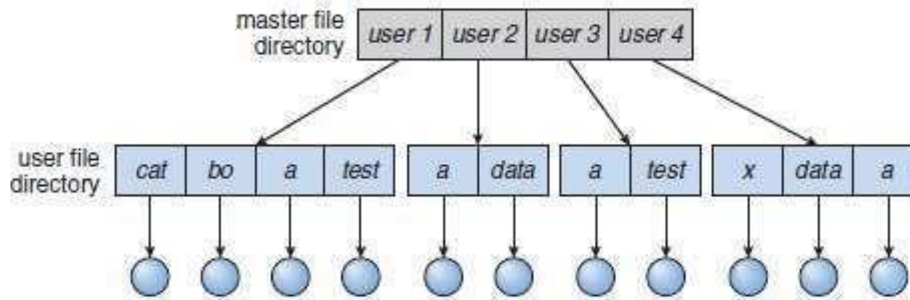


Figure 4.20 Two-level directory-structure

- To create a file for a user,
  - the OS searches only that user's UFD to determine whether another file of that name exists.
- To delete a file,
  - the OS limits its search to the local UFD. (Thus, it cannot accidentally delete another user's file that has the same name).
- Advantages:
  1) No filename-collision among different users.
  2) Efficient searching.
- Disadvantage:
  1) Users are isolated from one another and can"t cooperate on the same task.

4.9.3 Tree Structured Directories

- Users can create their own subdirectories and organize files (Figure 4.21).
- A tree is the most common directory-structure.
- The tree has a root directory.
- Every file in the system has a unique path-name.

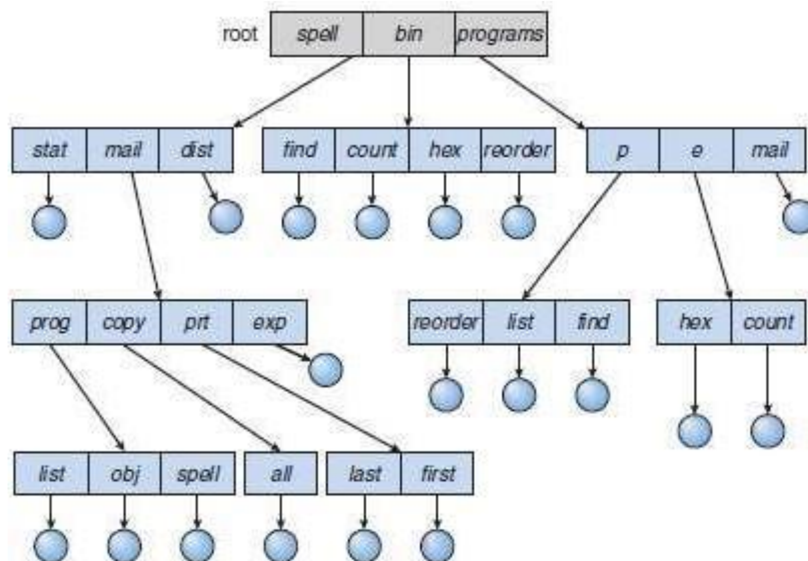


Figure 4.21 Tree-structured directory-structure

- A directory contains a set of files (or subdirectories).
- A directory is simply another file, but it is treated in a special way.
- In each directory-entry, one bit defines as
  - file (0) or
  - subdirectory (1).
- Path-names can be of 2 types:
- Two types of path-names:
  1) **Absolute path-name** begins at the root.
  2) **Relative path-name** defines a path from the current directory.
- How to delete directory?
  1) To delete an **empty directory**:
     → Just delete the directory.
  2) To delete a **non-empty directory**:
     → First, delete all files in the directory.
     → If any subdirectories exist, this procedure must be applied recursively to them.
- Advantage:
  1) Users can be allowed to access the files of other users.
- Disadvantages:
  1) A path to a file can be longer than a path in a two-level directory.
  2) Prohibits the sharing of files (or directories).

10.a

4.16 Allocation Methods

- The direct-access nature of disks allows us flexibility in the implementation of files.
- In almost every case, many files are stored on the same disk.
- Main problem:
  How to allocate space to the files so that
  → disk-space is utilized effectively and
  → files can be accessed quickly.

- Three methods of allocating disk-space:
    1) Contiguous
    2) Linked and
    3) Indexed
- Each method has advantages and disadvantages.
- Some systems support all three (Data General's RDOS for its Nova line of computers).

4.16.1 Contiguous Allocation

- Each file occupies a set of contiguous-blocks on the disk (Figure 4.30).
- Disk addresses define a linear ordering on the disk.
- The number of disk seeks required for accessing contiguously allocated files is minimal.
- Both sequential and direct access can be supported.
- Problems:
    1) Finding space for a new file
    ➢ External fragmentation can occur.
    2) Determining how much space is needed for a file.
    ➢ If you allocate too little space, it can't be extended.
        Two solutions:
        i) The user-program can be terminated with an appropriate error-message. The user must then allocate more space and run the program again.
        ii) Find a larger hole,
            copy the contents of the file to the new space and
                release the previous space.
- To minimize these drawbacks:
    1) A contiguous chunk of space can be allocated initially and
    2) Then when that amount is not large enough, another chunk of contiguous space (known as an „extent") is added.

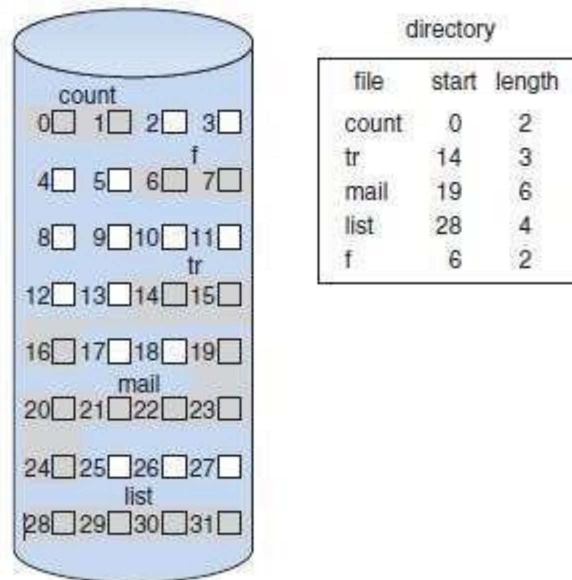


Figure 4.30 Contiguous allocation of disk-space

4.16.2 Linked Allocation

- Each file is a linked-list of disk-blocks.
- The disk-blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the file (Figure 4.31).
- To create a new file, just create a new entry in the directory (each directory-entry has a pointer to the disk-block of the file).

  1) A **write** to the file causes a free block to be found. This new block is then written to and linked to the eof (end of file).

  2) A **read** to the file causes moving the pointers from block to block.

- Advantages:

  1) No external fragmentation, and any free block on the free-space list can be used to satisfy a request.

  2) The size of the file doesn't need to be declared on creation.

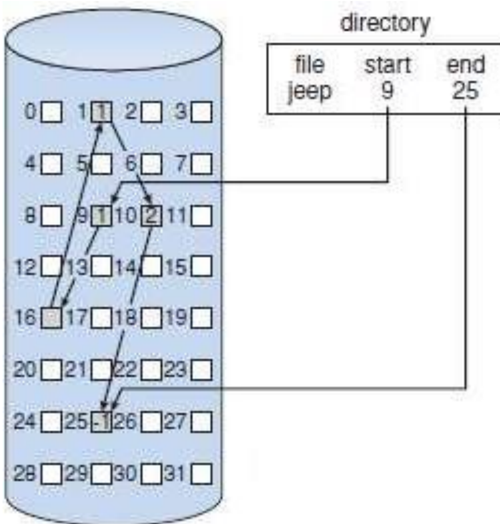
  3) Not necessary to compact disk-space.

- Disadvantages:

  1) Can be used effectively only for sequential-access files.

  2) Space required for the pointers.
     Solution: Collect blocks into multiples (called „clusters") & allocate clusters rather than blocks.

  3) Reliability: Problem occurs if a pointer is lost(or damaged). Partial solutions: i) Use doubly linked-lists.
     ii) Store file name and relative block-number in each block.



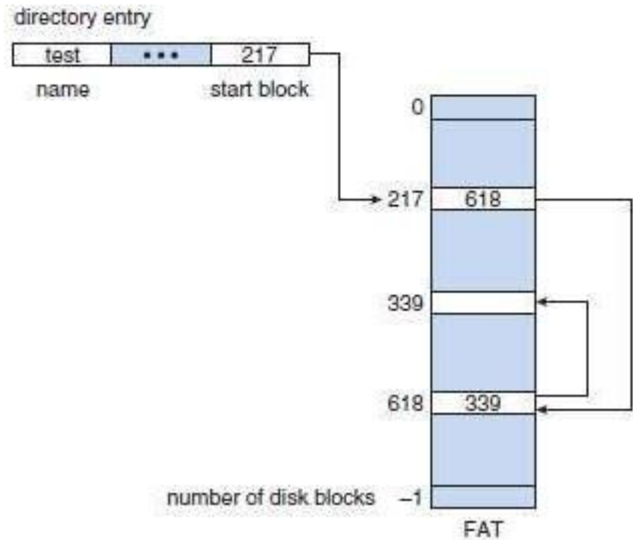


Figure 4.31 Linked allocation of disk-space

Figure 4.32 File-allocation table

- FAT is a variation on linked allocation (FAT=File Allocation Table).
- A section of disk at the beginning of each partition is set aside to contain the table (Figure 4.32).
- The table

  → has one entry for each disk-block and

→ is indexed by block-number.

- The directory-entry contains the block-number of the first block in the file.
- The table entry indexed by that block-number then contains the block-number of the next block in the file.
- This chain continues until the last block, which has a special end-of-file value as the table entry.
- Advantages:
  1) Cache can be used to reduce the no. of disk head seeks.
  2) Improved access time, since the disk head can find the location of any block by reading the info in the FAT.

10.b

FREE-SPACE MANAGEMENT

- FILE SYSTEM MAINTAINS **FREE-SPACE LIST** TO TRACK AVAILABLE BLOCKS/CLUSTERS
  - (USING TERM "BLOCK" FOR SIMPLICITY)
- **BIT VECTOR** OR **BIT MAP** (*N* BLOCKS)

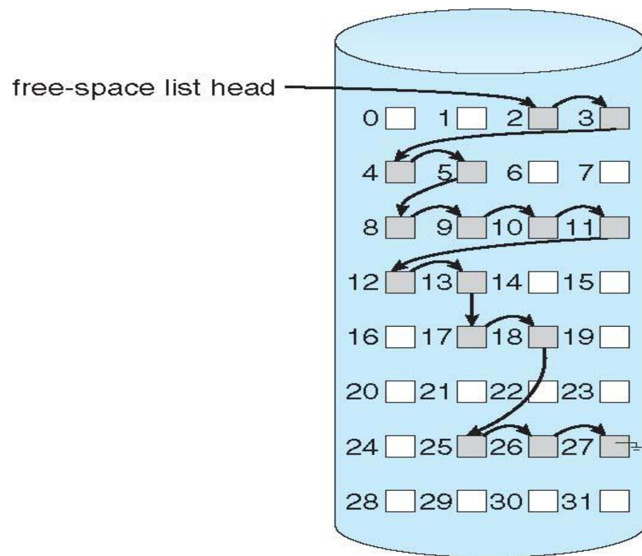
Block number calculation

(number of bits per word) *
(number of 0-value words) +
offset of first 1 bit

bit[*i*] =
1 ⇒ block[*i*] free
0 ⇒ block[*i*] occupied

CPUs have instructions to return offset within word of first "1" bit

- All the free spaces are linked together using pointers.
- A pointer pointing to the first free block is stored in the disk.
- It connects to a free block and so on.



- **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 this one)
- **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 free blocks
    - Free space list then has entries containing addresses and counts