Operating System – 17MCA24
INTERNAL TEST –III Answer Key

1 (a) Explain process management and various process schedule algorithm in Linux operating system — 10

Process Management

A process is the basic context within which all user-requested activity is serviced within the OS. To be compatible with other UNIX systems, Linux must use a process model similar to those of other versions of UNIX.

8.11.1 The Fork/Exec Process Model

UNIX process management separates the creation of processes and the running of a new program into two distinct operations. The ***fork*** system call creates a new process. A new program is run after a call to ***exec***. Under UNIX, a process encompasses all the information that the operating system must maintain t track the context of a single execution of a single program.

Under Linux, process properties fall into three groups:

- **Process Identity:** The process identity consists of mainly following items:
- **rocess ID (PID)**: The unique identifier for the process; used to specify processes to the OS when an application makes a system call to signal, modify, or wait for another process.
- **Credentials:** Each process must have an associated user ID and one or more group IDs that determine the process's rights to access system resources and files.
- **Personality:** Not traditionally found on UNIX systems, but under Linux each process has an associated personality identifier that can slightly modify the semantics of certain system calls. Used primarily by emulation libraries to request that system calls be compatible with certain specific flavors of UNIX.
- **Process environment:** The process's environment is inherited from its parent, and is composed of two null-terminated vectors:
  o The **argument vector** lists the command-line arguments used to invoke the running program; conventionally starts with the name of the program itself
  o The **environment vector** is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values.
  Passing environment variables among processes and inheriting variables by a process's children are flexible means of passing information to components of the user-mode system software. The environment-variable mechanism provides a customization of the operating system that can be set on a per-process basis, rather than being configured for the system as a whole.

Context: It is the (constantly changing) state of a running program at any point in
time. There are various parts:

Scheduling context is the most important part of the process context; it is the information that the scheduler needs to suspend and restart the process.

Accounting: The kernel maintains accounting information about the resources currently being consumed by each process, and the total resources consumed by the process in its lifetime so far.

o **File table** is an array of pointers to kernel file structures. When making file I/O system calls, processes refer to files by their index into this table.

o **File System Context:** Whereas the file table lists the existing open files, the file-system context applies to requests to open new files. The current root and default directories to be used for new file searches are stored here.
o **Signal-handler table** defines the routine in the processs's address space to be called when specific signals arrive.
o **Virtual-memory context** of a process describes the full contents of its private address space.

Processes and Threads

Linux uses the same internal representation for processes and threads; a thread is simply a new process that happens to share the same address space as its parent. A distinction is only made when a new thread is created by the **clone** system call:

- **fork** creates a new process with its own entirely new process context
- **clone** creates a new process with its own identity, but that is allowed to share the data structures of its parent

Using **clone** gives an application fine-grained control over exactly what is shared between
two threads.

SCHEDULING

The job of allocating CPU time to different tasks within an OS. While scheduling is normally thought of as the running and interrupting of processes, in Linux, scheduling also includes the running of the various kernel tasks. Running kernel tasks encompasses both tasks that are requested by a running process and tasks that execute internally on behalf of a device driver.
Various aspects of scheduling in Linux are discussed here.

- **Kernel Synchronization:** A request for kernel-mode execution can occur in two ways:

o A running program may request an operating system service, either explicitly via a system call, or implicitly, for example, when a page fault occurs.
o A device driver may deliver a hardware interrupt that causes the CPU to start executing a kernel-defined handler for that interrupt.
Kernel synchronization requires a framework that will allow the kernel's critical sections to run without interruption by another critical section.
Linux uses two techniques to protect critical sections:
o Normal kernel code is non-preemptable : when a time interrupt is received while a process is executing a kernel system service routine, the kernel's **need_resched** flag is set so that the scheduler will run once the system all has completed and control is about to be returned to user mode.
o The second technique applies to critical sections that occur in an interrupt service routines. By using the processor's interrupt control hardware to disable interrupts during a critical section, the kernel guarantees that it can proceed without the risk of concurrent access of shared data structures.

- **Process Scheduling:** Linux uses two process-scheduling algorithms:

o A time-sharing algorithm for fair preemptive scheduling between multiple processes
o A real-time algorithm for tasks where absolute priorities are more important than fairness
A process's scheduling class defines which algorithm to apply. For time-sharing processes, Linux uses a prioritized, credit based algorithm.

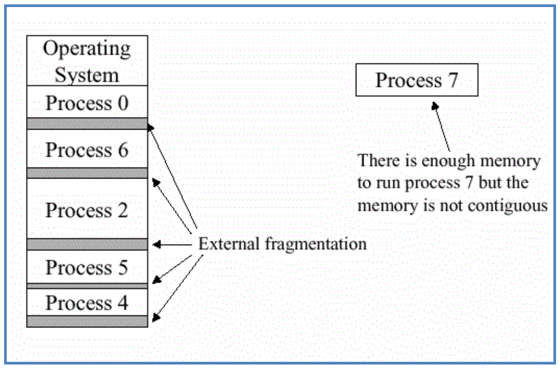
	Symmetric Multiprocessing: Linux 2.0 was the first Linux kernel to support SMP hardware; separate processes or threads can execute in parallel on separate processors. To preserve the kernel's non-preemptible synchronization requirements, SMP imposes the restriction, via a single kernel spinlock, that only one processor at a time may execute kernel-mode code.	
2 (a)	Explain hardware solution for critical section problem There are many software algorithms for enforcing mutual exclusion. But they have high processing overhead and the risk of logical errors. Hence, we will here consider few hardware approaches. **4.3.1 Interrupt Disabling** In a single-processor system, concurrent processes cannot be overlapped, but can be interleaved. Moreover, a process will continue to execute until it is interrupted. Hence, to guarantee mutual exclusion, a process has to be prevented from being interrupted. This capability can be provided in the form of primitives defined by OS kernel for disabling and enabling interrupts. So, a process should not be interrupted till it completes its critical section. But, by doing so, the efficiency of OS will slow-down, as OS cannot switch between the processes. Moreover, this approach does not guarantee mutual exclusion in case of multi-processor systems. Normally, access to a memory location excludes any other access to that same location. Hence, processor designers have proposed several machine instructions that carry out two actions atomically (that is, an instruction is a single step and not interrupted), such as reading and writing or reading and testing, of a single memory location with one instruction fetch cycle. During execution of the instruction, access to the memory location is blocked for any other instruction referencing that location. Two most commonly implemented instructions are explained below: **Compare and Swap Instruction:** It compares the contents of a memory location to a given value. If they are same, it modifies the contents of that memory location to a given new value. This is done as a single atomic operation. The atomicity guarantees that the new value is calculated based on up-to-date information; if the value had been updated by another process in the meantime, the write would fail. The result of the operation must indicate whether it performed the substitution; this can be done either with a simple boolean response, or by returning the value read from the memory location. **Exchange Instruction:** The instruction exchanges the contents of a register with that of a memory location. The use of a special machine instruction to enforce mutual exclusion has a number of advantages: It is applicable to any number of processes on either a single processor or multiple processors sharing main memory. It is simple and therefore easy to verify. It can be used to support multiple critical sections; each critical section can be defined by its own variable. There are some serious disadvantages as well: **Busy waiting is employed:** Thus, while a process is waiting for access to a critical section, it continues to consume processor time. **Starvation is possible:** When a process leaves a critical section and more than one process is waiting, the selection of a waiting process is arbitrary. Thus, some process could indefinitely be denied access. **Deadlock is possible:** Consider the following scenario on a single-processor system. Process P1 executes the special instruction and enters its critical section. P1 is then interrupted to give the processor to P2, which has higher priority. If P2 now attempts to use the same resource as P1, it will be denied access because of the mutual exclusion mechanism. Thus, it will go into a busy waiting loop. However, P1 will never be dispatched because it is of lower priority than another ready process, P2.	5

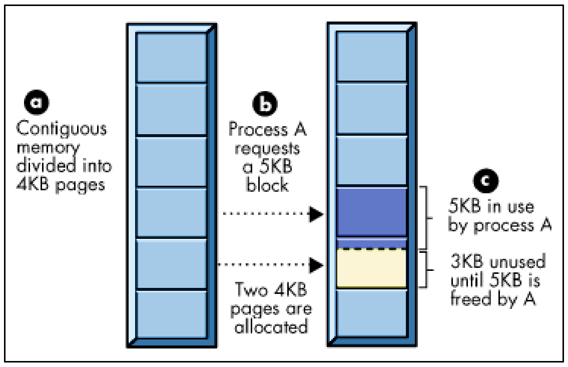
2.b) Explain fragmentation and its types with neat diagram

External Fragmentation – when total memory space exists to satisfy a request, but the available spaces are not contiguous.

- Storage is fragmented into a large number of small holes caused by first-fit and best-fit strategies for memory allocation plus variable-partition scheme

Internal Fragmentation – when allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used




a
Contiguous memory divided into 4KB pages
b
Process A requests a 5KB block
Two 4KB pages are allocated
c
5KB in use by process A
3KB unused until 5KB is freed by A

External Fragmentation Example — Internal Fragmentation Example

(b) What are the different types of scheduler? — 5

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler :

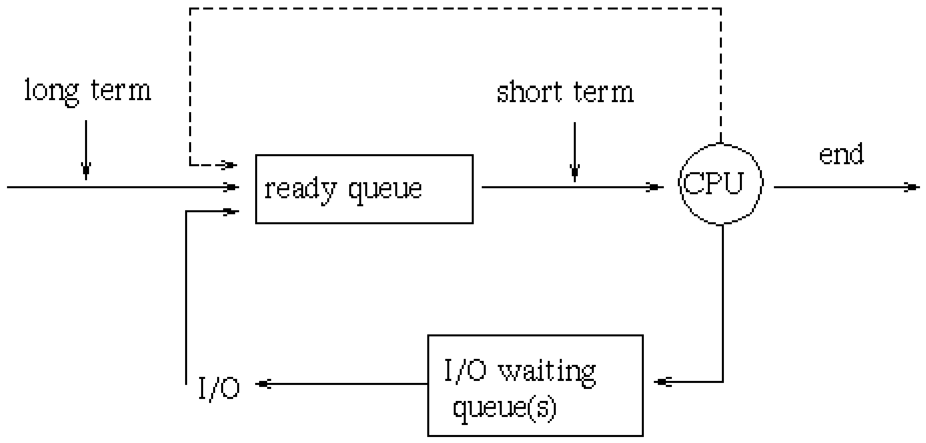
It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

Short Term Scheduler :

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of *ready state to running state* of the process. *CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.*

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next.

Short-term schedulers are *faster* than long-term schedulers.



Medium Term Scheduler :

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

3 (a) What is Critical Section and Mutual Exclusion 5

Critical Section

Suppose two processes require an access to single non-sharable resource like printer. During the course of execution, each process will be sending commands to the I/O device, receiving status information, sending data, and/or receiving data. We will refer to such a resource as a **critical resource**, and the portion of the program that uses it a **critical section** of the program.

It is important that only one program at a time be allowed in its critical section. We cannot simply rely on the OS to understand and enforce this restriction. For example, in case of the printer, we want only one process to have control of the printer till it finishes printing the entire document. Otherwise, lines from competing processes will be interleaved.

Mutual Exclusion

while one process executes the shared variable, all other processes desiring to do so at the same time moment should be kept waiting; when that process has finished executing the shared variable, one of the processes waiting; while that process has finished executing the shared variable, one of the processes waiting to do so should be allowed to proceed. In this fashion, each process executing the shared data (variables) excludes all others from doing so simultaneously. This is called Mutual Exclusion.

(b) What is symmetric multiprocessor system? What are its benefits? 5

Traditionally, the computer has been viewed as a sequential machine. That is, a processor executes instructions one at a time in a sequence and each instruction is a sequence of operations. But, as computer technology has evolved, parallel processing got importance.
One of the popular approaches for providing parallelism is symmetric multiprocessors (SMPs), where processors are replicated.

3.8.1 SMP Architecture
Single instruction single data (SISD) stream: A single processor executes a single instruction stream to operate on data stored in a single memory.
- **Single instruction multiple data (SIMD) stream:** Each instruction is executed on a different set of data by the different processors.
- **Multiple instruction single data (MISD) stream:** A sequence of data is transmitted to a set of processors, each of execute a different instruction sequence.
- **Multiple instruction multiple data (MIMD) stream**: A set of processors simultaneously execute different instruction sequences on different data sets. Based on the communication among processors, MIMD can be further divided. If every processor has a dedicated memory, then each processing element is a self-contained computer. Communication among the computers is either via fixed path or via some network. Such a system is known as a ***cluster***. If the processors share a common memory, it is known as ***shared-memory multiprocessor.*** This again can be further divided into ***master/slave*** architecture and ***SMP.***

The master/slave architecture has disadvantages:
- A failure of the master brings down the whole system
- As master has to do all scheduling and process management, the performance may slow down.

But, in SMP, the kernel can execute on any processor and it allows portions of kernel to execute in parallel. Here, each processor does self-scheduling from the pool of available process or threads.

3.8.2 SMP Organization
In SMP, there are multiple processors, each of which contains its own control unit, arithmetic-logic unit and registers as shown in Figure 3.14. Each processor has access to a shared main memory and the I/O devices through a shared bus.

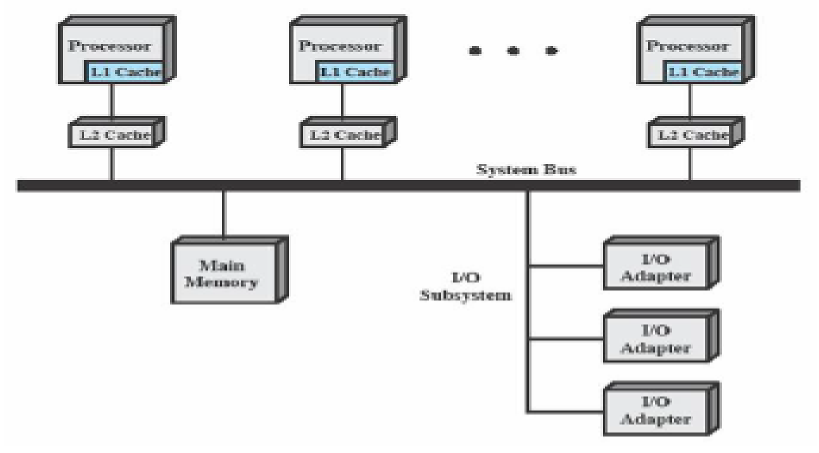


Figure 3.14 SMP Organization

4 (a) What is system call? What are the types of system call? Give examples of each. 6

The system call provides an interface to the operating system services.
Application developers often do not have direct access to the system calls, but can access them through an application programming interface (API). The functions that are included in the API

invoke the actual system calls. By using the API, certain benefits can be gained:

- Portability: as long a system supports an API, any program using that API can compile and run.
- Ease of Use: using the API can be significantly easier then using the actual system call.

.2. Types of System Calls

There are 5 different categories of system calls:

process control, file manipulation, device manipulation, information maintenance and communication.

1.12.2.1. Process Control

A running program needs to be able to stop execution either normally or abnormally. When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger.

1.12.2.2. File Management

Some common system calls are *create*, *delete*, *read*, *write*, *reposition*, or *close*. Also, there is a need to determine the file attributes – *get* and *set* file attribute. Many times the OS provides an API to make these system calls.

1.12.2.3. Device Management

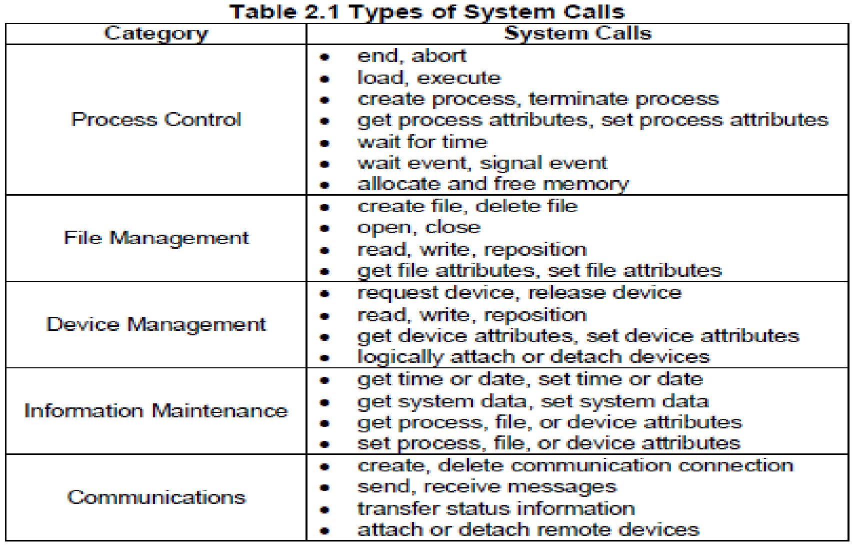
Process usually require several resources to execute, if these resources are available, they will be granted and control returned to the user process. These resources are also thought of as devices. Some are physical, such as a video card, and others are abstract, such as a file.
User programs *request* the device, and when finished they *release* the device. Similar to files, we can *read*, *write*, and *reposition* the device.

1.12.2.4. Information Management

Some system calls exist purely for transferring information between the user program and the operating system. An example of this is *time*, or *date*.
The OS also keeps information about all its processes and provides system calls to report this information.

1.12.2.5. Communication

Table 2.1 Types of System Calls



Category	System Calls
Process Control	• end, abort • load, execute • create process, terminate process • get process attributes, set process attributes • wait for time • wait event, signal event • allocate and free memory
File Management	• create file, delete file • open, close • read, write, reposition • get file attributes, 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 or date • get system data, set system data • get process, file, or device attributes • set process, file, or device attributes
Communications	• create, delete communication connection • send, receive messages • transfer status information • attach or detach remote devices

There are two models of interprocess communication, the message-passing model and the shared memory model.

- Message-passing uses a common mailbox to pass messages between processes.
- Shared memory use certain system calls to create and gain access to create and gain

		access to regions of memory owned by other processes. The two processes exchange information by reading and writing in the shared data.	
	(b)	What is a PCB? What are its contents A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below − **Identifier:** A unique identifier associated with this process, to distinguish it from all other processes. **State:** If the process is currently executing, it is in the running state. **Priority:** Priority level relative to other processes. **Program counter:** The address of the next instruction in the program to be executed. **Memory pointers:** Includes pointers to the program code and data associated with this process, plus any memory blocks shared with other processes. **Context data:** These are data that are present in registers in the processor while the process is executing. **I/O status information:** Includes outstanding I/O requests, I/O devices (e.g., disk drives) assigned to this process, a list of files in use by the process, and so on. **Accounting information:** May include the amount of processor time and clock time used, time limits, account numbers, and so on. The above information is stored in a data structure known as ***process control block***	4
5	(a)	Write and explain Banker's algorithm for deadlock avoidance The resource-allocation graph algorithm is not applicable when there are multiple instances for each resource. The banker's algorithm addresses this situation, but it is less efficient. The name was chosen because this algorithm could be used in a banking system to ensure that the bank never allocates its available cash such that it can no longer satisfy the needs of all its customers. When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise, the process must wait until some other process releases enough resources. Data structure for Banker's algorithms is as below – Let n be the number of processes in the system and m be the number of resource types. ***Available**:* Vector of length *m* indicating number of available resources. If Available[*j*] = *k*, there are *k* instances of resource type *Rj* available. ***Max**:* An *n x m* matrix defines the maximum demand of each process. If *Max* [*i*,*j*] = *k*, then process *Pi* may request at most *k* instances of resource type *Rj*. ***Allocation**:* An *n x m* matrix defines the number of resources currently allocated to each process. If Allocation[*i, j*] = *k* then *Pi* is currently allocated *k* instances of *Rj*. ***Need**:* An *n x m* matrix indicates remaining resource need of each process. If *Need*[*i*,*j*] = *k*, then *Pi* may need *k* more instances of *Rj* to complete its task. Note that, *Need* [*i*,*j] = Max*[*i*,*j*] – *Allocation* [*i*,*j*]. The Banker's algorithm has two parts: 1. **Safety Algorithm:** It is for finding out whether a system is in safe state or not. The steps are as given below – 1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize:	5

Work = *Available*
Finish [*i*] = *false* for *i* = 1, 2, 3, …, *n*.
2. Find an *i* such that both:
(a) *Finish* [*i*] = *false*
(b) *Needi* □ *Work*
If no such *i* exists, go to step 4.
3. *Work* = *Work* + *Allocationi*
Finish[*i*] = *true*
go to step 2.
4. If *Finish* [*i*] == true for all *i*, then the system is in a safe state.

Resource – Request Algorithm:
Let *Requesti* be the request vector for process *Pi*. If
Requesti [*j*] = *k* then process *Pi* wants *k* instances of resource type *Rj*.
1. If *Requesti* □ *Needi* go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If *Requesti* □ *Available*, go to step 3. Otherwise *Pi* must wait, since resources are not available.
3. Pretend to allocate requested resources to *Pi* by modifying the state as follows:
Available = Available - Requesti;
Allocationi = Allocationi + Requesti;
Needi = Needi – Requesti;
If the resulting resource allocation is safe, then the transaction is complete and the process Pi is allocated its resources. If the new state is unsafe, then Pi must wait for *Requesti* , and the old resource-allocation state is restored

(b) Write short note about Inter process communication in Linux — 5

Like UNIX, Linux informs processes that an event has occurred via signals. There is a limited number of signals, and they cannot carry information: Only the fact that a signal occurred is available to a process. The Linux kernel does not use signals to communicate with processes with are running in kernel mode, rather, communication within the kernel is accomplished via scheduling states and wait-queue structures.

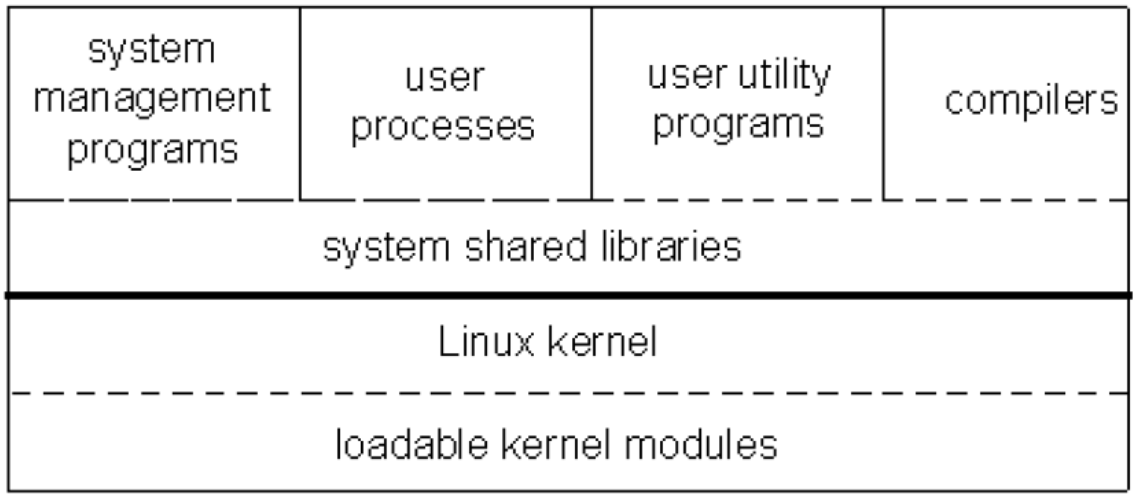
Passing of Data among Processes: The pipe mechanism allows a child process to inherit a communication channel to its parent; data written to one end of the pipe can be read by the other. Shared memory offers an extremely fast way of communicating; any data written by one process to a shared memory region can be read immediately by any other process that has mapped that region into its address space. To obtain synchronization, however, shared memory must be used in conjunction with another inter process communication mechanism.

Explain the components of Linux OS
Ans: The Linux system is composed of three main bodies of code, in line with most traditional UNIX implementations:
1. Kernel. The kernel is responsible for maintaining all the important abstractions of the operating system, including such things as virtual memory and processes.
2. System libraries. The system libraries define a standard set of functions through which applications can interact with the kernel. These functions implement much of the operating-system functionality

that does not needthe full privileges of kernel code.

3. System utilities. The system utilities are programs that perform individual, specialized management tasks. Some system utilities may be invoked just once to initialize and configure some aspect of the system; others—
known as *daemons* in UNIX terminology—may run permanently, handling such tasks as responding to incoming network connections, accepting logon requests from terminals, and updating log files.

Figure 21.1 illustrates the various components that make up a full Linux system. The most important distinction here is between the kernel and everything else. All the kernel code executes in the processor's privileged mode with full access to all the physical resources of the computer. Linux refers to this privileged mode as **kernel mode.**



6 (a) With a neat diagram explain the steps in handling page fault 6

In Demand paging , we need to distinguish the pages which are in the memory and pages which are there on the disk. For this purpose, the valid – invalid bit is used. When this bit is set to *valid,* it indicates the page is in the memory. Whereas, the value of bit as *invalid* indicates page is on the disk.

If the process tries to access a page which is not in the memory (means, it is on the disk), ***page fault*** occurs. The paging hardware notices the *invalid* bit in the page table and cause a trap to the OS. This trap is the result of the failure of OS to bring the desired page into memory. This error has to be corrected. The procedure for handling this page fault is as shown in Figure 6.16. The steps are explained below:

1. We check an internal table (usually kept with the process control block) for this process, to determine whether the reference was a valid or invalid memory access.

2. If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page into memory, it is brought now.

3. We find a free frame.

4. We schedule a disk operation to read the desired page into the newly allocated frame.

5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

6. We restart the instruction that was interrupted by the illegal address trap. The process can now access the page as though it had always been in memory.

It is important to realize that, because we save the state (registers, condition code, instruction counter etc.) of the interrupted process when the page fault occurs, we can restart the process in exactly the same place and state In an extreme situation, a process may starts executing with no page in the memory. So, each time an instruction has to be executed, page fault occurs and the required page needs to be brought into the memory. This situation is called as ***pure demand paging***.

That is, no page is brought into the memory until it is required.

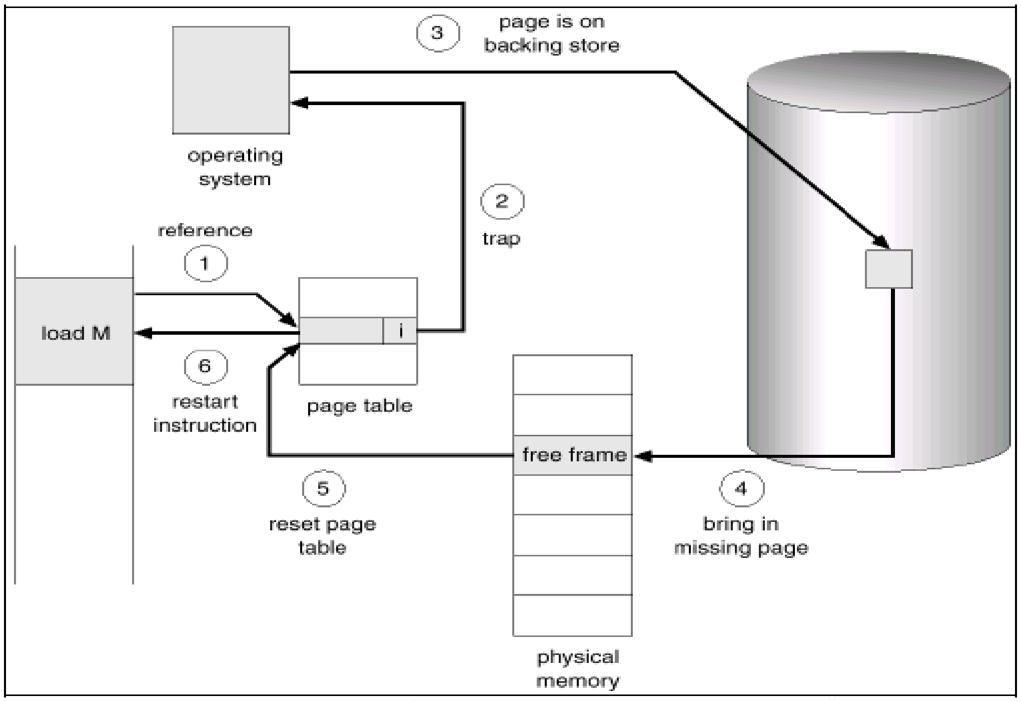


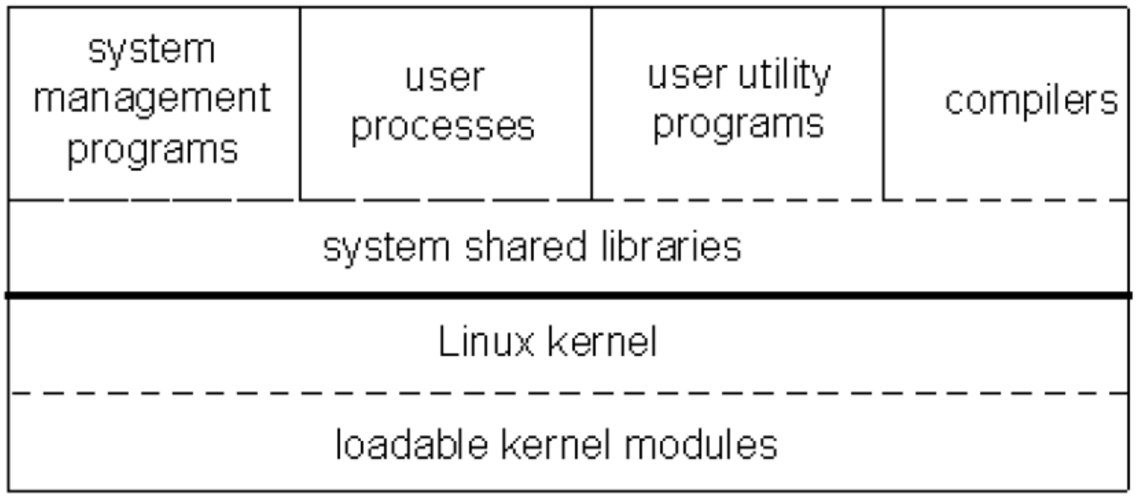
Figure 6.16 Steps in handling page fault

(b) Explain the components of Linux OS — 4

Ans: The Linux system is composed of three main bodies of code, in line with most traditional UNIX implementations:

1. Kernel. The kernel is responsible for maintaining all the important abstractions of the operating system, including such things as virtual memory and processes.

2. System libraries. The system libraries define a standard set of functions through which applications can interact with the kernel. These functions implement much of the operating-system functionality

	that does not needthe full privileges of kernel code. **3. System utilities.** The system utilities are programs that perform individual, specialized management tasks. Some system utilities may be invoked just once to initialize and configure some aspect of the system; others— known as *daemons* in UNIX terminology—may run permanently, handling such tasks as responding to incoming network connections, accepting logon requests from terminals, and updating log files. Figure 21.1 illustrates the various components that make up a full Linux system. The most important distinction here is between the kernel and everything else. All the kernel code executes in the processor's privileged mode with full access to all the physical resources of the computer. Linux refers to this privileged mode as **kernel mode.**  system management programs user processes user utility programs compilers system shared libraries Linux kernel loadable kernel modules `	
7	Mention the different page table structure explain in brief There are three common techniques for structuring a page table. They are: Hierarchical Paging - Break up the logical address space into multiple page tables. A simple technique is a two-level page table. Hashed Page	10

A logical address (on 32-bit machine with 1K page size) is divided into:

- a page number consisting of 22 bits
- a page offset consisting of 10 bits

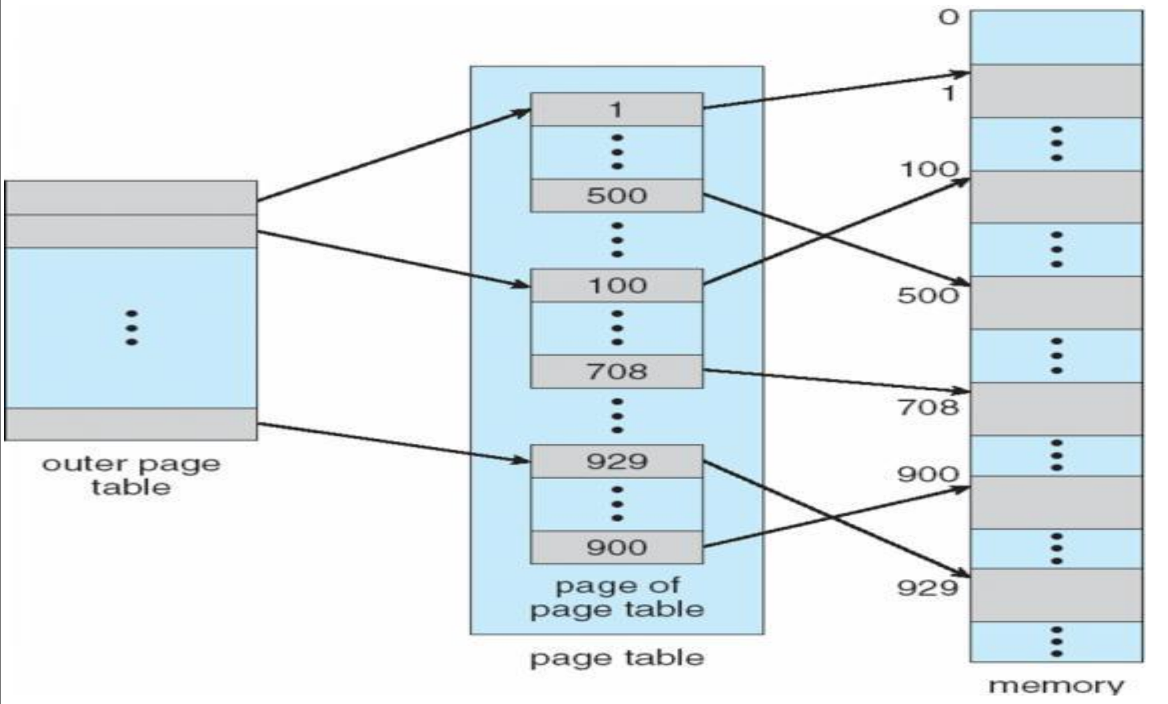
Since the page table is paged, the page number is further divided into:

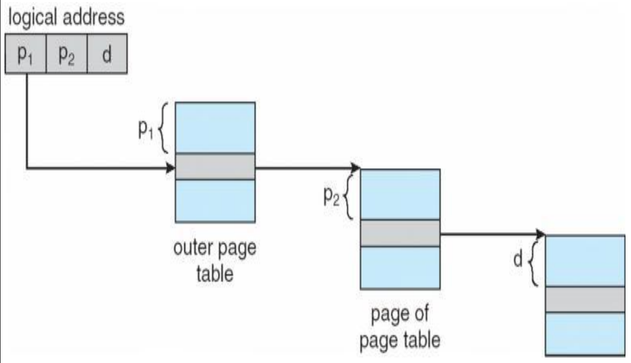
- a 12-bit page number
- a 10-bit page offset

Thus, a logical address is as follows:

page number		page offset
p_i	p_2	d
12	10	10

where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

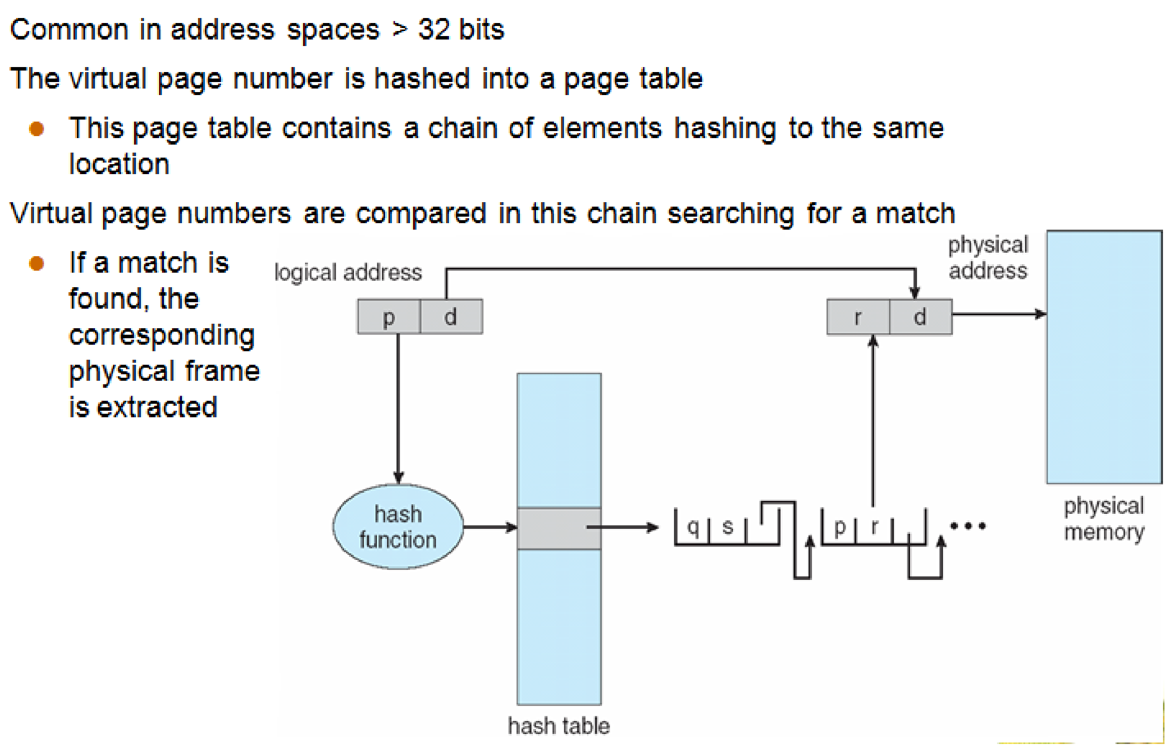




outer page	inner page	offset
p_1	p_2	d
42	10	12

2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d

Hashed Page Table –



Common in address spaces > 32 bits

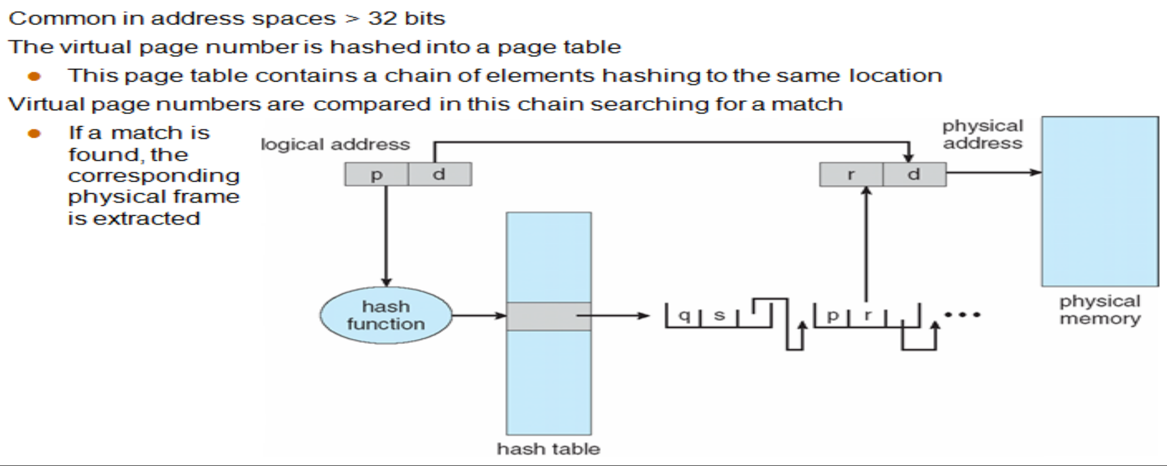
The virtual page number is hashed into a page table

- This page table contains a chain of elements hashing to the same location

Virtual page numbers are compared in this chain searching for a match

- If a match is found, the corresponding physical frame is extracted

Inverted Page Table –



Common in address spaces > 32 bits

The virtual page number is hashed into a page table

- This page table contains a chain of elements hashing to the same location

Virtual page numbers are compared in this chain searching for a match

- If a match is found, the corresponding physical frame is extracted

8 (a) Define Security problem and system threat 10

We have discussed earlier that each file/directory can be secured by controlling access to them. But, total security cannot be achieved by this. Security violations of the system canbe categorized as intentional (malicious) or accidental. It is easier to protect against accidental misuse. Malicious access has following forms:

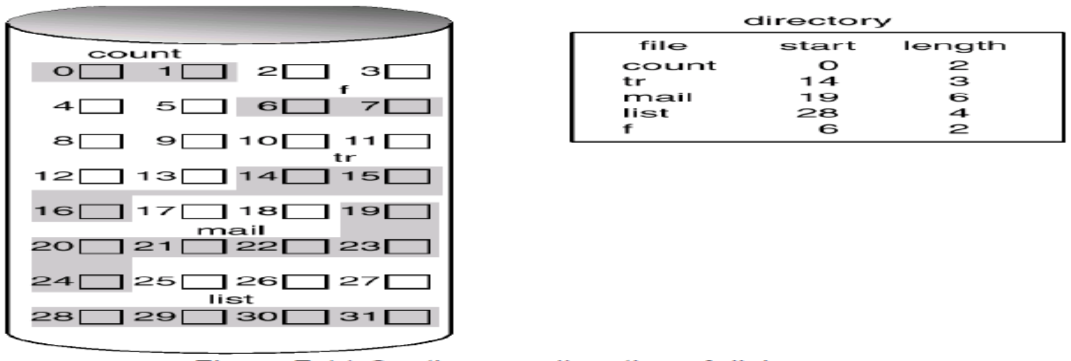
- Unauthorized reading of data
- Unauthorized modification of data
- Unauthorized destruction of data
- Preventing legitimate use of the system

To protect the system, we must take security measures at four levels:

1. **Physical:** The site or sites containing the computer systems must be physically

	secured against armed or surreptitious entry by intruders. 2. **Human:** Users must be screened carefully to reduce the chance of authorizing a user who then gives access to an intruder. 3. **Network:** Data being transferred via network is prone to attack. 4. **Operating System:** The system must protect itself security breaches. **SYSTEM THREATS** Most OS allows processes to spawn (creating child process) other processes. In such a situation, it is possible that OS resources and user files are misused. The methods for misusing are discussed here: **Worms:** A worm is a process that uses the spawn mechanism to attack system performance. It creates multiple copies of itself using system resources and locking all other processes from using the system. **Viruses:** It is a code embedded within a legitimate program. They spread into other programs and mess-up the system – modifying/destroying files, causing system crash etc. **Denial of Service:** Here, the hacker does not gain any information, but disables legitimate user from accessing system.	
(b)	Explain the role of Operating system to prevent The major security problem for OS is authentication. The protection system depends on an ability identify the programs and processes currently executing. Generally authentication is based on one or more of three items: User possession (a key or a card) User knowledge (identifier or password) User attribute (fingerprint, retina, signature etc) **Passwords** Passwords are used to protect the data in the computer, when there are no complete protection schemes. User is asked to provide the username and password during the access. If the password matches with the one stored in the system, access is allowed. **8.5.2 Password Vulnerabilities** Though passwords are popularly used because of its easy usage and understanding, it is vulnerable to threats. They can be guessed, accidentally exposed, illegally transferred from one to other etc. To avoid these various possibilities can be tried: keeping non-guessable passwords keep changing the passwords frequently keeping encrypted passwords using One – Time Passwords (OTP) using biometrics authentication	
9	Explain the following with respect to the file system: i) Contiguous allocation ii) Linked allocation iii) Indexed allocation The direct-access nature of disks allows us flexibility in the implementation of files. In almost every case, many files will be stored on the same disk. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly. Three major methods of allocating disk space are: contiguous, linked, and indexed. **Contiguous Allocation** In contiguous allocation, files are assigned to contiguous areas of secondary storage. A	10

user specifies in advance the size of the area needed to hold a file to be created. If the desired amount of contiguous space is not available, the file cannot be created. A contiguous allocation of disk space is shown in Figure 7.11.



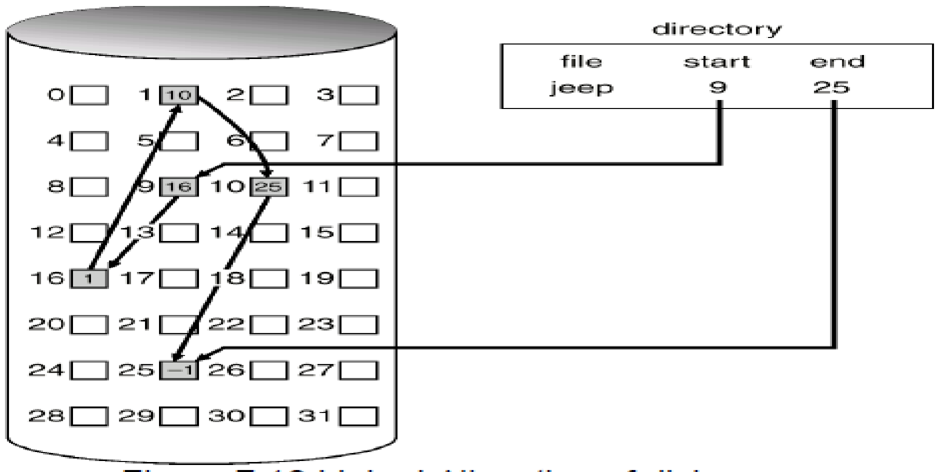
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Figure 7.11 Contiguous allocation of disk space

One advantage of contiguous allocation is that all successive records of a file are normally physically adjacent to each other. This increases the accessing speed of records. It means that if records are scattered through the disk it is accessing will be slower. For sequential access the file system remembers the disk address of the last block and when necessary reads the next block. For direct access to block B of a file that starts at location L, we can immediately access block L+B. Thus contiguous allocation supports both sequential and direct accessing. The disadvantage of contiguous allocation algorithm is, it suffers from external fragmentation. As files are allocated and deleted, the free disk space is broken into little pieces. Depending on the total amount of disk storage and the average file size, external
fragmentation may be a minor or a major problem.

Linked Allocation

Linked allocation solves all problems of contiguous allocation. With 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 as shown in Figure 7.12.



file	start	end
jeep	9	25

Figure 7.12 Linked Allocation of disk space

Linked allocation solves the problem of external fragmentation, which was present in contiguous allocation. But, still it has a disadvantage: Though it can be effectively used for sequential-access files, to find ith file, we need to start from the first location. That is, random-access is not possible.

Indexed Allocation

This method allows direct access of files and hence solves the problem faced in linked allocation. Each file has its own index block, which is an array of disk-block addresses. The ith entry in the index block points to the ith block of the file. The directory contains the address of the index block as shown in Figure 7.13.

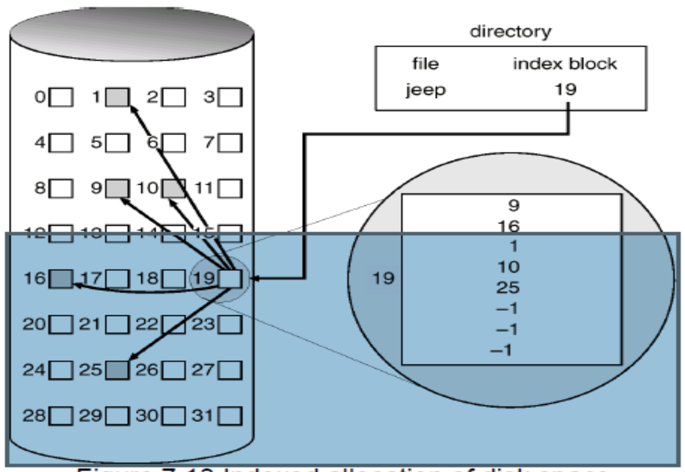


Figure 7.13 Indexed allocation of disk space

10. Describe the methods used for implementing directory structures 10

The file systems of computers can be very large. To manage all these data, we need to organize them. The disks are split into one or more partitions. Typically, each disk on a system contains at least one partition, which is a low-level structure in which files and directories reside. Sometimes, partitions are used to provide several separate areas within one disk, each treated as a separate storage device, whereas other systems allow partitions to be larger than a disk to group disks into one logical structure. Each partition contains information about files within it. This information is kept in a device directory or volume table of contents. The device directory records information-such as name, location, size, and type-for all files on that partition. Figure 7.3 shows file-system organization.

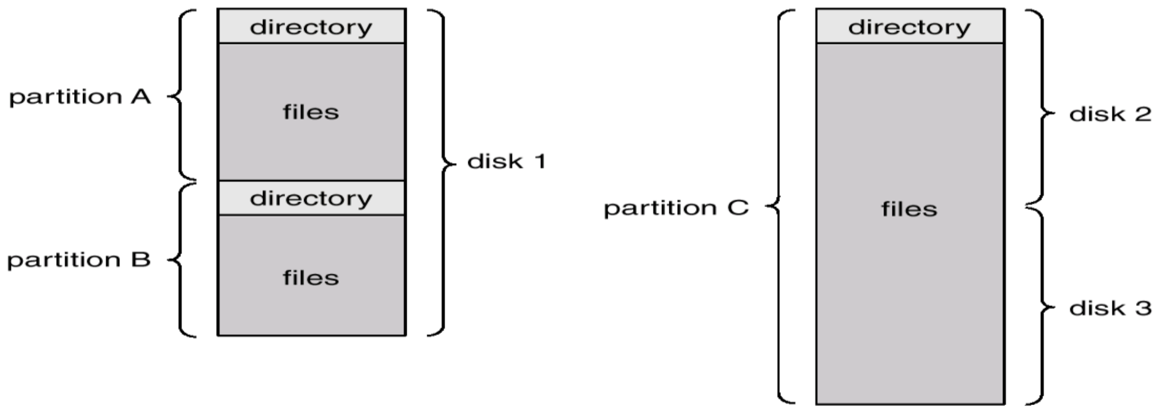


Figure 7.3 A typical file-system organization

Various operations that are to be performed on a directory:

 Search for a file: We need to be able to search a directory structure to find the entry for

a particular file.
 Create a file: New files need to be created and added to the directory.
 Delete a file: When a file is no longer needed, we want to remove it from the directory.
 List a directory: We need to be able to list the files in a directory, and the contents of
the directory entry for each file in the list.
 Rename a file: As the name of a file represents its contents to its users, the name must be changeable when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.
 Traverse the file system: Files may need to be copied into another storage device like tape, pen-drive etc. and the disk space of that file released for reuse by another file.

7.3.1 Single – Level Directory

The simplest directory structure is the single-level directory. All files are contained in the same directory, which is easy to support and understand. It is shown in Figure 7.4. There are certain limitations for single-level directory:
 As all files will be in one directory, each file should have a unique name.
 If there are more users on the same system, each one of them should remember
their file names – which is difficult.
 It is not possible to group the related files together.

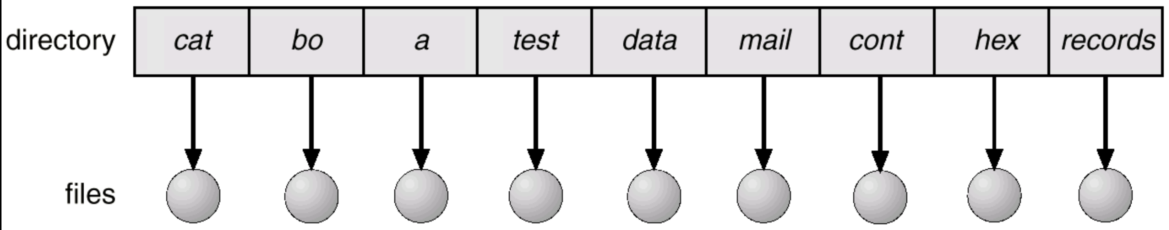


Figure 7.4 Single – level directory structure

7.3.2 Two – Level Directory

A solution to the problems faced with single-level directory is to create a separate directory for each user. In the two-level directory structure, each user has his/her own user file directory (UFD). Each UFD has a similar structure, but lists only the files of a single user.
When a user job starts or a user logs in, the system's master file directory (MFD) is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user. Two – level directory is shown in Figure 7.5.
When a user refers to a particular file, only his own UFD is searched. Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.

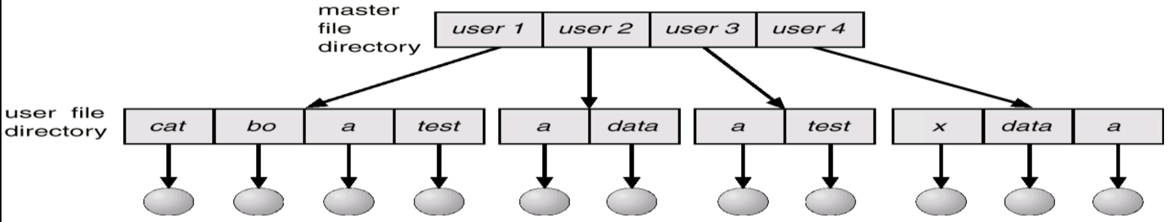


Figure 7.5 Two – level directory structure

7.3.3 Tree – Structured Directories

The generalization of two-level directory is a tree structure. Here, users can have any number of directories and sub-directories and files inside each directory as shown in Figure 7.6. Tree – structure is a most common directory structure applied in almost all OS.

Here, every file has a path name. Path names can be of two types: ***absolute*** and ***relative.*** An absolute path name is a sequence of directory names starting from root to file name. A relative path name defines a path from current directory. For example, in the tree-structured file system of Figure 7.6, if the current directory is *root/spell/mail*, then the relative path name *prt/first* refers to the same file as does the absolute path name *root/spell/mail/prt/first.*

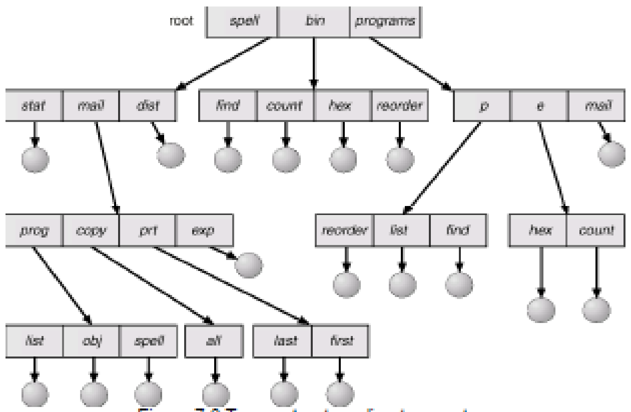


Figure 7.6 Tree – structure directory system

7.3.4 Acyclic – Graph Directories

A tree structure prohibits the sharing of files or directories. An acyclic graph allows directories to have shared subdirectories and files (Figure 7.7). The same file or subdirectory may be in two different directories. This is useful when more than one programmers are working on a same project and need to share the files.

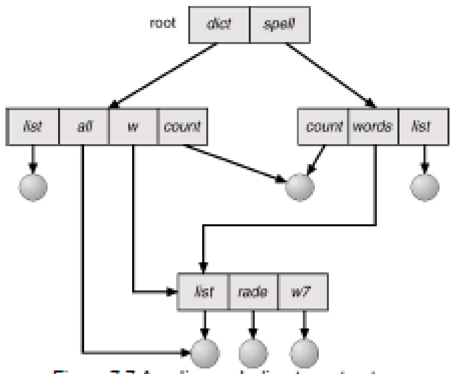


Figure 7.7 Acyclic graph directory structure