Experiment No 1

Aim:

To study Operating system Commands, System Calls and Shell scripts w.r.t. linux

Part-1 To demonstrate various Linux Commands and System Calls (For eg: (mkdir, chdir, cat, ls, chown, chmod, chgrp, ps etc. system calls: open, read, write, close, getpid, setpid, getuid, getgid, getegid, geteuid, etc.))

Part-2 To write shell scripts to do the following:

a. Display OS version, release number, kernel version
b. Display top 10 processes in descending order
c. Display processes with highest memory usage.
d. Display current logged in user and log name.
c. Display current shell, home directory, operating system type, current path setting, current working directory.

Objectives:

To conduct a comprehensive exploration of Linux commands and system calls, including but not limited to (mkdir, chdir, cat, ls, chown, chmod, chgrp, ps, etc.), and delve into system calls such as open, read, write, close, getpid, setpid, getuid, getgid, getegid, geteuid, with the aim of gaining a profound understanding of their functionalities and applications within the Linux operating system.

Outcomes:

Students are able to understand and demonstrate various linux commands, System Calls and Shell scripts w.r.t. linux.

Theory:

Linux is a UNIX variant for the IBM PC (Intel 80386) architecture, a 32-bit processor in Intel's range of PC-compatible CPUs. Linux is a full-featured UNIX system. The basic Linux system provides a standardized environment for applications and user programming, but it does not impose any standard method of managing all available functionality. A Linux distribution includes all the standard components of the Linux system, as well as a set of administrative tools to assist the initial installation and subsequent upgrades of Linux, as well as managing the installation and removal of other packages on the system. A modern distribution includes tools for managing file systems, creating and managing user accounts, network administration, Web browsers, word editors, etc..

Text editors in Linux

Linux is just as well suited for word processing as any other operating system. There are several excellent word processing programs for Linux like AbiWord, KWord, part of the Koffice suite and the OpenOffice.org as well as StarOfficesuite's word processor.

Why use a text editor?

A text editor is just like a word processor without a lot of features. All operating systems come with a basic text editor. Linux comes with several. The main use of a text editor is for writing something in plain text with no formatting so that another program can read it. Based on the information it gets from that file, the program will run one way or another.

The text editor "vi"

The most popular text editor for Linux is called 'vi'. This is a program that comes from UNIX. There is a more recent version called 'vim' which means 'vi improved'. The problem with 'vi' or 'vim' is that a lot of people don't like it. You have to remember a lot of key combinations to do stuff that other text editors will do for you more easily.

Following linux commands are used frequently,

Sr. No	Command	Command Description
1.	`cd`	This command is used to 'change directory'.
2.	`ls`	Typing 'ls' will list the contents of a directory with just information about file names.
3.	`cp`	'cp' is used for copying files from one place to another, or for making a duplicate of one file under a different name.
4.	`mv`	'mv' is used for moving files from one place to another. It cuts the file from one place and pastes it to another.
5.	`mkdir`	This command is used for making or creating directories.
6.	`rmdir`	This is the opposite of 'mkdir'- which is used to delete the directories.
7.	`rm`	This command is used for removing or deleting files.
8.	`grep`	The grep command allows us to search one file or multiple files for lines that contain a pattern.
9.	`"pipes" in Linux`	To use the pipe command, we don't type: pipe. We press the '|' key.
10.	`who`	This is used to find out who's working on our system.
11.	`whoami`	It is a little program that tells us who we are, i.e. it prints effective user id.
12.	`pwd`	(print working directory)The pwd command displays the full pathname of the current directory.

13.	`cat`	The cat command reads one or more files and prints them to standard output.
14.	`wc`	This command will give us the number of lines, words and letters (characters) in a file and in that order.
15.	`ps`	It gives us a list of the processes running on our system.
16.	`chmod`	The chmod command changes the access mode of one file or multiple files.
17.	`Date`	print or set system date & time
18.	`cal`	displays a calendar and the date
19.	`echo`	display a text
20.	`bc`	An arbitrary precision calculator language.
21.	`man`	an interface to the on-line reference manuals.
22.	`uptime`	Tell how long the system has been running.
23.	`uname`	print system information
24.	`hostname`	show or set the system's host name
25.	`head`	output the first part of files
26.	`tail`	output the last part of files
27.	`sort`	sort lines of text files
28.	`chown`	change file owner and group

A system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. This may include hardware-related services (for example, accessing a hard disk drive), creation and execution of new processes, and communication with integral kernel services such as process scheduling. System calls provide an essential interface between a process and the operating system. To understand system calls, first one needs to understand the difference between kernel mode and user mode of a CPU. Every modern operating system supports these two modes.

When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a system call.
When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a context switch. Then the kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from kernel mode back to user mode.
Generally, system calls are made by the user level programs in the following situations:

1. Creating, opening, closing and deleting files in the file system.
2. Creating and managing new processes.
3. Creating a connection in the network, sending and receiving packets.
4. Requesting access to a hardware device, like a mouse or a printer.

Various system calls:

1. open
2. read
3. write
4. close
5. getpid
6. getuid
7. getgid
8. getegid
9. geteuid

Shell Script

Shell accepts commands and executes them. If there is a sequence of commands , then we can store this sequence of commands to a text file and tell the shell to execute this text file instead of entering the commands. This is known as a ***shell script***.

- Shell script can take input from the user, file and output them on screen.
- Useful to create our own commands.
- Save lots of time.
- To automate some tasks of day to day life.
- The System Administration part can also be automated etc.

The following commands in the shell script are provided as,

a. Display OS version, release number, kernel version

Sr.No	Command	Command Description
1.	lsb_release -a	To display linux standard base modules details such as distributor id, description details, release version and codename.
2.	lsb_release -d	To display linux standard base module description
3.	cat /etc/os-release	To display the details of current distribution
4.	hostnamectl	It provides an appropriate API that may be used to alter the hostname of a Linux system and its associated settings. The tool also enables changing the hostname of a machine without physically accessing and altering the /etc/hostname file.
To display all information		
1.	uname -a or uname - -all	This command is used to display linux information.

2.	uname -s or uname --kernel -name	To print the kernel name
3.	uname -r or uname --kernel-release	To print the Linux kernel release
4.	uname -v or uname --kernel-version	To print the kernel version
5.	uname -m or uname --machine	To print the machine hardware name
6.	uname -p or uname --hardware-platform	To print the processor type or "unknown"
7.	uname -o or uname --operating_system	To print the operating system

b. Display top 10 processes in descending order

Sr.No	Command	Command Description
1.	top -b -o +%MEM | head -n 22	top: To view the process information through RAM and CPU Usage. -b: Runs top in batch mode. -o: To specify fields for sorting processes. +: To sort the process in descending order. head: To display the first few lines of the file. -n: To specify the number of lines to be displayed

c. Display processes with highest memory usage.

Sr.No	Command	Command Description
1.	ps -eo pid,ppid,cmd,%mem,%cpu - -sort=-%mem | head	This command will display processes with highest memory usage.

d. Display current logged in user and log name.

Sr.No	Command	Command Description
1.	$ echo -e "Currently Logged: $nouser user(s)"	This command will display the current logged in user.

2.	$ echo -e "User name: $USER (Login name: $LOGNAME)"	This command will display the current logged user name.

e. Display current shell, home directory, operating system type, current path setting, current working directory.

Sr.No	Command	Command Description
1.	$ echo -e "Current Shell: $SHELL"	This command displays the current shell.
2.	$ echo -e "Home Directory: $HOME"	This command displays the home directory.
3.	$ echo -e "Your O/s Type: $OSTYPE"	This command displays the operating system type.
4.	$ echo -e "PATH: $PATH"	This command displays the path.
5.	$ echo -e "Current working directory: `pwd`"	This command displays the current working directory.

Conclusion:

The shell acts as an interface between the user and the kernel. Thus a shell is used to execute the commands. The shell scripts containing above commands are created & executed. Also the process will try to access the kernel using a system call.

Experiment No 2

Aim:

To study Linux- API and Process
Part-1 Implement any one basic commands of linux like ls, cp, mv and others using kernel APIs.
Part-2 To create a child process in Linux using the fork system call. From the child process obtain the process ID of both child and parent by using getpid and getppid system calls.

Objectives:

To explore the concept of process creation in Linux through the use of the fork system call, and to analyze the parent-child relationship established during the process creation. Specifically, the objective is to create a child process, obtain and display the process IDs of both the child and parent processes using getpid and getppid system calls, thereby gaining insight into the hierarchical structure of processes in the Linux environment.

Outcomes:

Students are able to understand and study linux API and Process.

Theory:

In Linux, basic commands like ls, cp, mv, and others are typically implemented as user-space programs that interact with the kernel through system calls and library functions. The kernel provides a set of APIs (Application Programming Interfaces) that allow user-space programs to communicate with and request services from the operating system. The below table discussed about basic commands using kernel APIs:

Sr.No	Command	Command Description
1.	ls (List Files and Directories)	The ls command can be implemented using the opendir, readdir, and closedir system calls for directory manipulation. The program may use the stat system call to gather information about each file and directory in a directory.
2.	cp (Copy Files)	The cp command involves opening the source file with the open system call and the destination file with another open call. Reading from the source file using read and writing to the destination file using write. The close system

		call is used to close both the source and destination files.
3.	mv (Move or Rename Files)	The mv command involves similar steps to cp but also includes the removal of the source file after a successful copy, which can be achieved using the unlink system call. Alternatively, renaming a file within the same filesystem can be done using the rename system call.
6.	ps (Process Status)	The ps command can be implemented by reading information about processes from the /proc filesystem, which provides a view of the system's process information.

The method by which a process makes a kernel service request is via a system call. It is broadly divided by its usage i.e. file system, process, scheduling, interprocess communication, memory.

fork() is a kernel system call which creates a process. When a fork is requested by a process, the OS performs the following functions,

1. It provides new processes a slot in the process table.
2. It assigns a unique process ID to the child process.
3. It creates a copy of the parent's process image, excluding any shared memory.
4. It increments counters for any files owned by the parent, which reflects that an additional process also owns those files.
5. It assigns the child process to the Ready to Run state.
6. It returns the ID number of the child to the parent process, and a 0 value to the child process.

The above functions are performed by the parent process in the kernel mode. As part of the dispatcher process, once the kernel has performed these functions, it can do one of the following:

Stay in the parent process- At the point of the parent's fork call, control returns to user mode.

Transfer control to the child process- The child process starts executing code at the same point as the parent, specifically the return from the fork call.

Transfer control to another process- Both the parent and the child are in the Ready to Run state.

When a process creates a new process, there are two below possibilities for execution,
1. The parent continues to execute concurrently with its children.

2. The parent waits until some or all of its children have terminated.

Also there are two address-space possibilities for the new process,
1. The child process has the same program and data as the parent process).
2. The child process has a new program loaded into it.

Generally, system calls are made by the user level programs in the following situations:

1. Creating, opening, closing and deleting files in the file system.
2. Creating and managing new processes.
3. Creating a connection in the network, sending and receiving packets.
4. Requesting access to a hardware device, like a mouse or a printer.

Following are the linux system calls,

1. getpid: Get process identification.
2. getppid: Get parent process identification.
3. setuid: Set user identity of the current process

Conclusion:

In this experiment, the Linux API and Process is studied.

Experiment No 3

Aim:

To implement the following process scheduling algorithms and evaluate their performance (Non- preemptive scheduling algorithm).
1. FCFS 2. SJF

Objectives:

1. To implement First Come First Served (FCFS) Process Non-Preemptive Scheduling Algorithm.
2. To implement Shortest Job First (SJF) Process Non-Preemptive Scheduling Algorithm.

Outcomes:

Students are able to implement Process Non-Preemptive Scheduling Algorithms such as FCFS & SJF.

Theory:

Process is a program in execution. The processes require CPU for its execution. Thus, the CPU must be allocated to each process for a particular time unit. There are two types of scheduling algorithms - Non-preemptive scheduling algorithms & preemptive scheduling algorithms. In a non-preemptive scheduling algorithm, once the resources are allocated to a process, the process holds the CPU till it gets terminated or reaches a waiting state. It is used when a process terminates, or a process switches from running to the waiting state. Preemptive Scheduling is a scheduling method where the tasks are assigned with their priorities. This helps to run a task with a higher priority before another lower priority task, even if the lower priority task is still running.

The non-preemptive scheduling algorithms are listed as follows,

- First Come First Served (FCFS) Process Non-Preemptive Scheduling Algorithm
- Shortest Job First (SJF) Process Non-Preemptive Scheduling Algorithm

The preemptive scheduling algorithms are listed as follows,

- Shortest Job First (SJF) Process Preemptive Scheduling Algorithm
- Round-Robin Process Preemptive Scheduling Algorithm
- Priority Process Preemptive Scheduling Algorithm

Following are the important terms for solving scheduling algorithms.
Burst time is the total time taken by the process for its execution on the CPU.
Arrival time is the time when a process enters into the ready state and is ready for its execution.

Criterias to evaluate scheduling algorithm

- CPU Utilization: Keep CPU utilization as high as possible. (What is utilization, by the way?).
- Throughput: number of processes completed per unit time.
- Turnaround Time: amount of time to execute a particular process
- Waiting Time: Amount of time spent ready to run but not running.

- Response Time: Time between submission of requests and first response to the request.

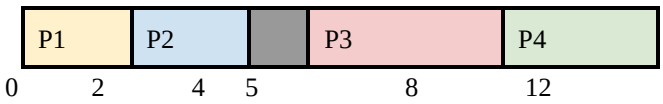
1. First Come First Served (FCFS) Process Scheduling Algorithm:

The first come, first served (FCFS) scheduling algorithm schedules jobs based on their arrival time. The CPU will be assigned to the job that is first present in the ready queue. The earlier the job arrives, the sooner it will receive the CPU. FCFS scheduling may result in starvation if the initial process's burst time is the longest of all jobs.

Consider the below table which contains process information,

Process No	Arrival Time	Burst Time
P1	0	2
P2	1	2
P3	5	3
P4	6	4

The gantt chart is given as,



Following criterias are evaluated,

Turn Around Time = Completion Time - Arrival Time
Waiting Time = Turn Around Time - Execution Time
Response Time = First Time process getting CPU - Arrival Time

Process No	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time	Response Time
P1	0	2	2	2	0	0
P2	1	2	4	3	1	1
P3	5	3	8	3	0	0
P4	6	4	12	6	2	2

Average Turnaround Time = 3.5 time unit
Average Waiting Time = 0.75 time unit
Average Response Time = 0.75 time unit

Following are the advantages and disadvantages for SJF algorithm,
Advantages:

- Simple
- Easy to understand

Disadvantages:
- Long waiting Time
- Lower CPU Utilization.

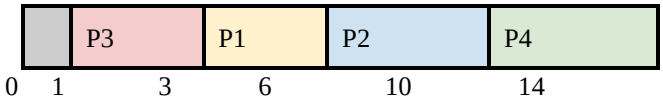
2. Shortest Job First (SJF) Process Management Algorithm:

This algorithm is associated with the length of the next CPU burst for each process. When the CPU becomes available, it is allocated to the process with the shortest next CPU burst. If two processes' next CPU bursts are the same, FCFS scheduling is utilised to break the tie. The scheduling of a process is determined by the length of its next CPU burst rather than its total length.

Consider the below table which contains process information,

Process No	Arrival Time	Burst Time
P1	1	3
P2	2	4
P3	1	2
P4	4	4

The gantt chart is given as,



Following criterias are evaluated,

Turn Around Time = Completion Time - Arrival Time
Waiting Time = Turn Around Time - Execution Time
Response Time = First Time process getting CPU - Arrival Time

Process No	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time	Response Time
P1	1	3	6	5	2	2
P2	2	4	10	8	4	4
P3	1	2	3	2	0	0
P4	4	4	14	10	6	6

Average Turnaround Time = 6.25 time unit

Average Waiting Time = 3 time unit
Average Response Time = 3 time unit

Following are the advantages and disadvantages for SJF algorithm,

Advantages:

- Reduces average wait time.
- In terms of average turnaround time, this is optimal.

Disadvantages:

- We can't use this approach for CPU scheduling in the short term since we can't foresee how long the upcoming CPU burst will be.
- Cause lengthy turnaround times or starvation.

Conclusion:

Thus, FCFS & SJF has been implemented successfully. Also, the Avg. waiting time & Avg. turnaround time of SJF is better than FCFS.

Experiment No 4

Aim:

To implement the following process scheduling algorithms and evaluate their performance (Preemptive scheduling algorithm).
1. RR 2. Priority

Objectives:

1. To implement Round Robin (RR) Process Non-Preemptive Scheduling Algorithm.
2. To implement Priority Process Non-Preemptive Scheduling Algorithm.

Outcomes:

Students are able to implement Process Non-Preemptive Scheduling Algorithms such as RR & Priority.

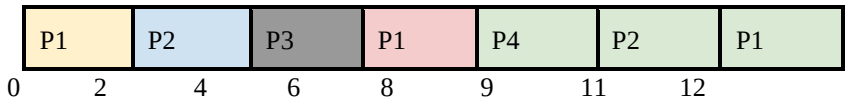
Theory:

1. Round Robin (RR) Process Management Algorithm:

The round-robin (RR) scheduling technique is intended for use in timesharing systems. It is similar to FCFS scheduling, except it includes preemption to allow the system to move between processes. A time quantum or time slice is defined as a small unit of time. A time quantum is typically between 10 and 100 milliseconds long. The ready queue is treated as a circular queue. The CPU scheduler cycles over the ready queue, allocating the CPU to each process for up to one time quantum. To implement RR scheduling, new processes are added to the ready queue's tail. The CPU scheduler selects the first process from the ready queue, starts a timer for 1 time quantum, then dispatches the process. Then one of two things will happen. A CPU burst of less than one time quantum is possible for the procedure. In this instance, the process will freely release the CPU. After that, the scheduler will move on to the next process in the ready queue. If the current running process's CPU burst is longer than one time quantum, the timer will stop and the operating system will be interrupted. A context switch is executed, and the process is placed to the end of the ready queue. The next process in the ready queue will be chosen by the CPU scheduler.
Consider the below table which contains process information,

Process No	Arrival Time	Burst Time
P1	0	5
P2	1	4
P3	2	2
P4	4	1

The gantt chart is given as,



Following criterias are evaluated,

Turn Around Time = Completion Time - Arrival Time
Waiting Time = Turn Around Time - Execution Time
Response Time = First Time process getting CPU - Arrival Time

Process No	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time	Response Time
P1	0	5	12	12	7	0
P2	1	4	11	10	6	1
P3	2	2	6	4	2	2
P4	4	1	9	5	4	4

Average Turnaround Time = 7.75 time unit
Average Waiting Time = 4.75 time unit
Average Response Time = 1.75 time unit

Following are the advantages and disadvantages for RR algorithm,
Advantages:

- No starvation

Disadvantages:

- Increases overhead
- Lower CPU efficiency.

2. Priority Process Management Algorithm:

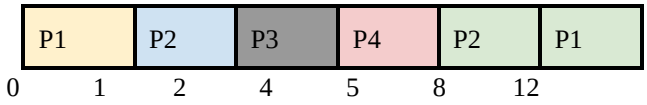
Priority scheduling can be preemptive or non-preemptive. In the preemptive priority scheduling algorithm, when a process arrives in the ready queue, its priority is compared to the priority of the process that is currently running. If the priority of the newly arrived process is higher than the priority of the presently running process, a preemptive priority scheduling algorithm will preempt the CPU.

Consider the below table which contains process information,

Process No	Arrival Time	Burst Time	Priority
P1	0	5	10 (Low)
P2	1	4	20
P3	2	2	30

P4	4	1	40 (High)

The gantt chart is given as,



Following criterias are evaluated,

Turn Around Time = Completion Time - Arrival Time
Waiting Time = Turn Around Time - Execution Time
Response Time = First Time process getting CPU - Arrival Time

Process No	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	5	12	12	7
P2	1	4	8	7	3
P3	2	2	4	2	0
P4	4	1	5	1	0

Average Turnaround Time = 5.5 time unit
Average Waiting Time = 2.5 time unit

The disadvantages for Priority algorithm is given below,

- Lower priority process may starve.

Conclusion:

Thus, RR & Priority scheduling algorithms has been implemented successfully.

Experiment No 5

Aim:

Write a C program to implement solution of Producer consumer problem through Semaphore

Objectives:

1. To understand the Producer-Consumer problem.
2. To overcome the Producer-Consumer problem using semaphore.

Outcomes:

Students are able to implement Producer-Consumer using semaphore.

Theory:

Producer Consumer Problem

In computing, the Producer-Consumer Problem, also known as Bounded Buffer problem is a classic example of a multi-process synchronisation problem. The problem describes two processes, the producer and the consumer, which share a common, fixed size buffer using a queue.

- The producer's job is to generate data, put it into a buffer, and start again.
- At the same time, the consumer is consuming the data i.e. removing it from the buffer, one piece at a time.

Problem: To make sure that the producer won't try to add data into the buffer if it is completely occupied and the consumer will not try to remove data from an empty buffer. Only one thread can manipulate the buffer queue at a time (Mutual Exclusion).

Solution: The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

An adequate solution could result in a deadlock where both the processes are waiting to be awakened.

- Semaphore fullBuffers; //consumer's constraint
- Semaphore emptyBuffers; //producer's constraint
- Semaphore mutex; // mutual exclusion

Algorithm:

```
Semaphore fullBuffers("full",0)                    //initially no message
Semaphore emptyBuffers("empty",BUFF_SIZE); //the size of the round buffer
Semaphore mutex("ring",1)                          //no one accesses the ring buffer
```

```
Producer() {
        emptyBuffers.P();           //check if there's slot for more messages
        mutex.P();                  //make sure no one else is accessing the ring buffer
        put one message in the ring buffer
        mutex.V();                  //ok for others to use the ring buffer
        fullBuffers.V(); //tell consumers there's a new message in the ring buffer
}
Consumer() {
        fullBuffers.P();    //check if there's a message in the ring buffer
        mutex.P();                  //make sure no one else is accessing the ring buffer
        take one message out
        mutex.V();                  //others' turn to use the ring buffer
        emptyBuffers.V();           //tell producer regarding more messages are needed
}
```

Conclusion:

Thus, the solution for Producer-Consumer problem has been implemented successfully.

Experiment No 6

Aim:

Write a program to demonstrate the concept of deadlock avoidance through Banker's Algorithm.

Objectives:

To implement Deadlock Avoidance using Banker's Algorithm.

Outcomes:

Students are able to implement Deadlock Avoidance using Banker's Algorithm.

Theory:

Deadlock Avoidance

The multiple processes may compete for a limited number of resources in a multiprogramming environment. When a process requests resources, it enters a waiting state if the resources are not accessible at the time. Because the resources it has requested are held by other waiting processes, a waiting process may never be able to change state again. This is referred to as a deadlock.

Deadlock can be handled using the following methods,

- A protocol for preventing or avoiding deadlocks, assuring that the system will never become blocked.
- Allow the system to enter a deadlock, detect it, and recover from it.
- Ignore the issue entirely and pretend that system deadlocks never occur.

To minimise deadlocks, the operating system must be provided with additional information on which resources a process will request and use during its lifespan. With this additional information, the operating system can determine whether or not to wait for each request. The system must assess the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process to determine if the current request can be satisfied or must be delayed.

Banker's Algorithm

When a new process is added to the system, it must indicate how many instances of each resource type it will require. This quantity cannot exceed the total number of system resources. When a user demands a set of resources, the system must decide if allocating these resources would keep the system safe. If it does, the resources are allocated; otherwise, the process must wait for another process to release sufficient resources.

To implement the banker's algorithm, several data structures must be maintained. The state of the resource-allocation system is encoded in these data structures. The following data structures are required, where n denotes the number of processes in the system and m denotes the number of resource types:

- Available. A vector of length m indicates the number of available resources of each type. If Available[j] equals k, then k instances of resource type Rj are available.

- Max. An n × m matrix defines the maximum demand of each process. If Max[i][j] equals k, then process Pi may request at most k instances of resource type Rj .
- Allocation. An n × m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i][j] equals k, then process Pi is currently allocated k instances of resource type Rj .
- Need. An n × m matrix indicates the remaining resource need of each process. If Need[i][j] equals k, then process Pi may need k more instances of resource type Rj to complete its task. Note that Need[i][j] equals Max[i][j] − Allocation[i][j].

These data structures vary over time in both size and value.

Safety Algorithm

Below algorithm is used to find whether or not a system is in a safe state. The algorithm is described as follows:

1. Let Work and Finish be vectors of length m and n, respectively. Initialize Work = Available and Finish[i] = false for i = 0, 1, ..., n − 1.
2. Find an index 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.

This algorithm may require an order of m × n2 operations to determine whether a state is safe.

Resource-Request Algorithm

The below algorithm determines whether requests can be safely granted.

Let Requesti be the request vector for process Pi . If Requesti [j] == k, then process Pi wants k instances of resource type Rj . When a request for resources is made by process Pi , the following actions are taken:

1. If Requesti ≤ Needi , go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If Requesti ≤ Available, go to step 3. Otherwise, Pi must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

   Available = Available–Requesti ;
   Allocationi = Allocationi + Requesti ;
   Needi = Needi –Requesti ;

If the resulting resource-allocation state is safe, the transaction is completed, and process Pi is allocated its resources. However, if the new state is unsafe, then Pi must wait for Requesti , and the old resource-allocation state is restored.

Consider the following snapshot of a system.

Process	**Allocation**				**Maximum Requirement**				**Available Resource**			
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2	1	5	2	0

P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

With reference to Banker's Algorithm
1) Find need matrix.
2) Is the system in a safe state?

Solution,
To find need matrix,

Need Matrix = Maximum Requirement - Allocation

Safe State? Need <= Available Resource

P0 : **0,0,0,0 <= 1,5,2,0** ; The condition is True. Thus, P0 is executed.

New Availability = Availability + Allocation = **1,5,2,0 + 0,0,1,2 = 1,5,3,2**
Safe State? Need <= Available Resource

P1 : **0,7,5,0 <= 1,5,3,2** ; The condition is False. Thus, P1 is not executed.
Safe State? Need <= Available Resource

P2 : **1,0,0,2 <= 1,5,3,2** ; The condition is True. Thus, P2 is executed.

New Availability = Availability + Allocation = **1,5,3,2 + 1,3,5,4 = 2,8,8,6**
Safe State? Need <= Available Resource

P3 : **0,0,2,0 <= 2,8,8,6** ; The condition is True. Thus, P3 is executed.

New Availability = Availability + Allocation = **2,8,8,6 + 0,6,3,2 = 2,14,11,8**
Safe State? Need <= Available Resource

P4 : **0,6,4,2 <= 2,14,11,8** ; The condition is True. Thus, P4 is executed.

New Availability = Availability + Allocation = **2,14,11,8 + 0,0,1,4 = 2,14,12,12**
Safe State? Need <= Available Resource

P1 : **0,7,5,0 <= 2,14,12,12** ; The condition is True. Thus, P1 is executed.

New Availability = Availability + Allocation = **2,14,12,12 + 1,0,0,0 = 3,14,12,12**

Process	**Allocation**				**Maximum Requirement**				**Remaining Need**				**Available Resource**			
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2	**0**	**0**	**0**	**0**	~~1~~	~~5~~	~~2~~	~~0~~

P1	1	0	0	0	1	7	5	0	**0**	**7**	**5**	**0**	~~1~~	~~5~~	~~3~~	~~2~~
P2	1	3	5	4	2	3	5	6	**1**	**0**	**0**	**2**	~~2~~	~~8~~	~~8~~	~~6~~
P3	0	6	3	2	0	6	5	2	**0**	**0**	**2**	**0**	~~2~~	~~14~~	~~11~~	~~8~~
P4	0	0	1	4	0	6	5	6	**0**	**6**	**4**	**2**	~~2~~	~~14~~	~~12~~	~~12~~
													3	14	12	12

Safe Sequence: P0, P2, P3, P4, P1

Conclusion:

Thus, we have successfully deadlock avoidance using banker's algorithm.

Experiment No 7

Aim:

Write a program to demonstrate the concept of dynamic partitioning strategies i.e. Best Fit, First Fit, Worst-Fit etc. and evaluate their performance.

Objectives:

1. To understand dynamic partitioning strategies.
2. To understand internal and external fragmentation.

Outcomes:

Students are able to implement page placement algorithms such as best-fit, first-fit and worst-fit.

Theory:

Dynamic Partitioning

With dynamic partitioning, the partitions are of variable length and number. When a process is brought into main memory, it is allocated exactly as much memory as it requires and no more. If there are multiple free blocks of memory with adequate size when it's time to load or swap a process into main memory, the operating system must choose which free block to allocate.

There are three placement algorithms,

1. Best-fit
2. First-fit
3. Worst-fit

However, all are constrained to selecting from among free main memory blocks that are as least as big as the process being introduced.

Best-fit allocates the process to the smallest hole that is big enough. First-fit selects the first accessible block that is big enough after scanning memory. Worst-fit allocates the process to the largest hole that is big enough. Additionally, Next-fit starts to select the next available block that is large enough after scanning memory from the location of the most recent placement.

Consider there are 4 processes. Each of size 212K, 417K, 112K and 426K. The given memory partitions are 100K, 500K, 200K, 300K and 600K. We will place the processes in the given memory partitions using First-Fit, Best-Fit and Worst-Fit Algorithms. This will conclude which algorithm makes the most efficient use of memory?

1. Best-Fit:
In Best-Fit Algorithm, processes are allocated to the smallest hole that is big enough.

Partition Size	Memory	Fragmentation
100K		
500K	417K	500-417 = 83
200K	112K	200-112 = 88
300K	212K	300-212 = 88
600K	426K	600-426 = 174

Memory utilized = Memory utilized by P1, P2, P3, P4 i.e. 212K, 417K, 112K and 426K.

Memory Utilization = (Memory utilized) / (Total Memory)
= (417 + 112 + 212 + 426) / (100 + 500 + 200 + 300 + 600)
= 1167 / 1700
= 0.686 unit.

2. First-Fit:
In the First-Fit algorithm, processes are allocated to the first hole that is big enough.

Partition Size	Memory	Fragmentation
100K		
500K	212K	500-212 = 288
200K	112K	200-112 = 88
300K		
600K	417K	600-417 = 183

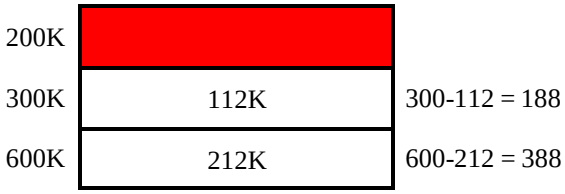
Memory utilized = Memory utilized by P1, P2, P3 i.e. 212K, 417K and 112K.

Memory Utilization = (Memory utilized) / (Total Memory)
= (212 + 112 + 417) / (100 + 500 + 200 + 300 + 600)
= 741 / 1700
= 0.436 unit.

3. Worst-fit:

In Worst-Fit Algorithm, processes are allocated to the largest hole that is big enough.

Partition Size	Memory	Fragmentation
100K		
500K	417K	500-417 = 83



Memory utilized = Memory utilized by P1, P2, P3 i.e. 212K, 417K and 112K.

Memory Utilization = (Memory utilized) / (Total Memory)
= (212 + 112 + 417) / (100 + 500 + 200 + 300 + 600)
= 741 / 1700
= 0.436 unit.

External Fragmentation- The process for which memory is not allocated from given partition.
First-fit : P4 i.e. 426K
Best-fit : No process
Worst-fit : P4 i.e. 426K

Internal Fragmentation- The process which occupies the memory from partition, but still wastes the memory partition.
First-fit: memory is wasted due to P1, P2, P3
Thus, Internal Fragmentation = 288 + 183 + 88 = 559K

Best-fit: memory is wasted due to P1, P2, P3, P4
Thus, Internal Fragmentation = 83 + 88 + 88 + 174= 433K

Worst-fit: memory is wasted due to P1, P2, P3
Thus, Internal Fragmentation = 288 + 183 + 88 = 559K

Therefore, Best-fit algorithm makes the most efficient use of memory.

Conclusion:

Thus, we have successfully implemented various dynamic partitioning strategies i.e. Best Fit, First Fit, Worst-Fit.

Experiment No 8

Aim:

Write a program in C demonstrating the concept of page replacement policies for handling page faults.

1. FIFO
2. LRU
3. Optimal

Objectives:

To understand and implement various page replacement algorithms.

Outcomes:

Students are able to implement page replacement algorithms such as FIFO, LRU and Optimal.

Theory:

1. First In First Out (FIFO) Algorithm:

The first-in-first-out (FIFO) policy uses a process's page frames as a circular buffer, and pages are discarded in a round-robin fashion. All that is needed is a pointer that circles through the process's page frames. As a result, this is one of the simplest page replacement strategies to apply. Aside from its simplicity, the logic behind this option is that one is replacing the page that has been in memory the longest: A page retrieved from memory a long time ago may have now become obsolete. This argument is frequently incorrect, because there are many parts of a program or data that are heavily used over the life of a program. Those pages will be paged in and out repeatedly. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced, the page in the front of the queue is selected for removal.

2. Least Recently Used (LRU) Algorithm:

LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. In this algorithm, those pages are selected which have not been used for the longest period of time.

3. Optimal Least Frequently Used (LFU) ALGORITHM:

The optimal policy selects for replacement that page for which the time to the next reference is the longest. It can be shown that this policy results in the fewest number of page faults In this algorithm, we select the page that will not be used for the longest period of time.

Consider the following page reference sequence: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2. In the below solutions, Frame numbers are represented as F1, F2 & F3. Hits are represented as "H" & Miss i.e. Page Fault is represented as "M".Total string references are 12.

Page Fault Rate or Page Miss Rate is evaluated as,

- Fault Rate = number of Miss i.e. faults / number of page references) * 100

Page Hit Rate is evaluated as,

- Page Hit Rate = number of Hits / number of page references * 100

1. Using FIFO Page Replacement Algorithm,

Consider the number of frame as "Three" for entering the above Page References,

Page No	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Reference String	**2**	**3**	**2**	**1**	**5**	**2**	**4**	**5**	**3**	**2**	**5**	**2**
F1	**2**	**2**	**2**	**2**	**5**	**5**	**5**	**5**	**3**	**3**	**3**	**3**
F2		**3**	**3**	**3**	**3**	**2**	**2**	**2**	**2**	**2**	**5**	**5**
F3				**1**	**1**	**1**	**4**	**4**	**4**	**4**	**4**	**2**
Hit/Miss	**H**	**H**	**H**	**H**	**F**	**F**	**F**	**H**	**F**	**H**	**F**	**F**

So, the Number of Page Fault occurs using "Three frames" are 6 & the number of Hits are 6.

Page Fault Rate or Page Miss Rate is evaluated as,

- Fault Rate = (6/12)% = 50%

Page Hit Rate is evaluated as,

- Page Hit Rate = (6/12)% = 50%

2. Using Least Recently Used (LRU) Page Replacement Algorithm,

Consider the number of frame as "Three" for entering the above Page References,

Page No	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Reference String	**2**	**3**	**2**	**1**	**5**	**2**	**4**	**5**	**3**	**2**	**5**	**2**
F1	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**3**	**3**	**3**	**3**
F2		**3**	**3**	**3**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
F3				**1**	**1**	**1**	**4**	**4**	**4**	**2**	**2**	**2**
Hit/Miss	**H**	**H**	**H**	**H**	**F**	**H**	**F**	**H**	**F**	**F**	**H**	**H**

So, the Number of Page Fault occurs using "Three frames" are 4 & the number of Hits are 8.

Page Fault Rate or Page Miss Rate is evaluated as,

- Fault Rate = (4/12)% = 33.3%

Page Hit Rate is evaluated as,

- Page Hit Rate = (8/12)% = 66.6%

3. Using Optimal Page Replacement Algorithm,

Consider the number of frame as "Three" for entering the above Page References,

Page No	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Reference String	**2**	**3**	**2**	**1**	**5**	**2**	**4**	**5**	**3**	**2**	**5**	**2**
F1	**2**	**2**	**2**	**2**	**2**	**2**	**4**	**4**	**4**	**2**	**2**	**2**
F2		**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
F3				**1**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
Hit/Miss	**H**	**H**	**H**	**H**	**F**	**H**	**F**	**H**	**H**	**F**	**H**	**H**

So, the Number of Page Fault occurs using "Three frames" are 15 & the number of Hits are 5.

Page Fault Rate or Page Miss Rate is evaluated as,

- Fault Rate = (3/12)% = 25%

Page Hit Rate is evaluated as,

- Page Hit Rate = (9/12)% = 75%

Conclusion:

Hence we have successfully implemented Page Replacement algorithms such as FIFO, LRU and Optimal.

Experiment No 9

Aim:

Part-1 Write a C program to simulate File allocation strategies typically sequential, indexed and linked files.
Part-2 Write a C program to simulate file organization of multi-level directory structure.

Objectives:

To understand and implement various file allocation strategies.

Outcomes:

Students are able to implement File allocation strategies such as sequential, indexed and linked files.

Theory:

File Allocation Strategies

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Sequential Allocation
- Linked Allocation
- Indexed Allocation

Sequential File Allocation

In this allocation strategy, each file occupies a set of contiguously blocks on the disk. This strategy is best suited. For sequential files, the file allocation table consists of a single entry for each file. It shows the filenames, starting block of the file and size of the file. The main problem with this strategy is, it is difficult to find the contiguous free blocks in the disk and some free blocks could happen between two files. Below figure 10.1 represents an example of sequential file allocation strategy.

Indexed File allocation

Indexed allocation supports both sequential and direct access files. The file indexes are not physically stored as a part of the file allocation table. Whenever the file size increases, we can easily add some more blocks to the index. In this strategy, the file allocation table contains a single entry for each file. The entry consists of one index block, the index blocks having the pointers to the other blocks. No external fragmentation. Below figure 10.2 represents an example of indexed file allocation strategy.

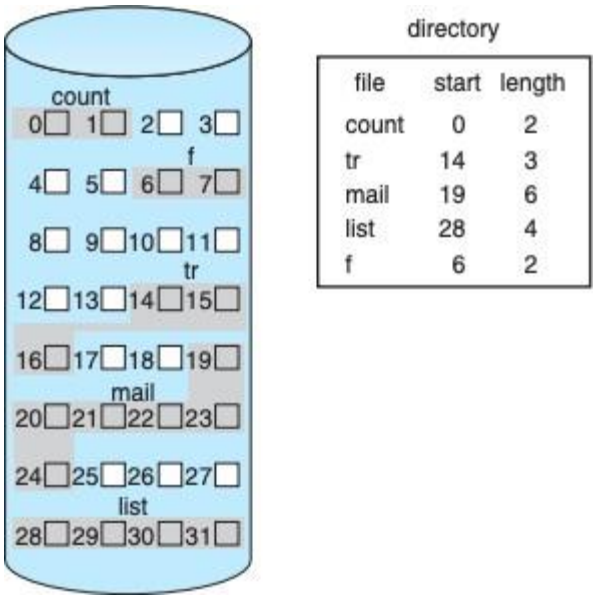


Fig 10.1: Sequential File Allocation Strategy (*Example*)

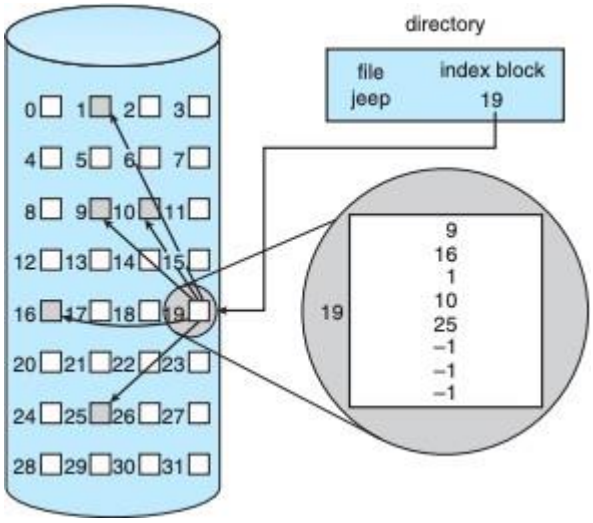


Fig 10.1: Indexed File Allocation Strategy (*Example*)

Linked File allocation

It is easy to allocate the files because allocation is on an individual block basis. Each block contains a pointer to the next free block in the chain. Here also the file allocation table consists of a single entry for each file. Using this strategy any free block can be added to a chain very easily. There is a link between one block to another block, that's why it is said to be linked allocation. We can avoid the external fragmentation. Below figure 10.3 represents an example of linked file allocation strategy.

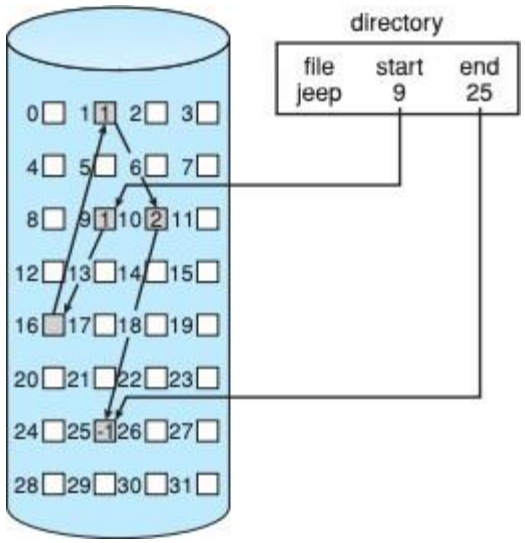
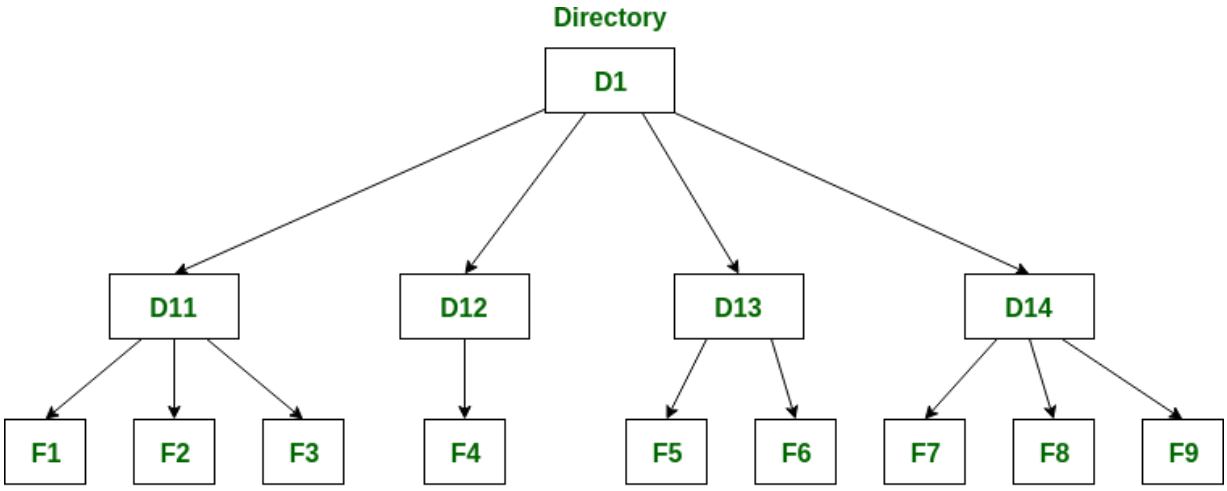


Fig 10.1: Linked File Allocation Strategy (*Example*)

A **directory** is a container that is used to contain folders and files. It organizes files and folders in a hierarchical manner.

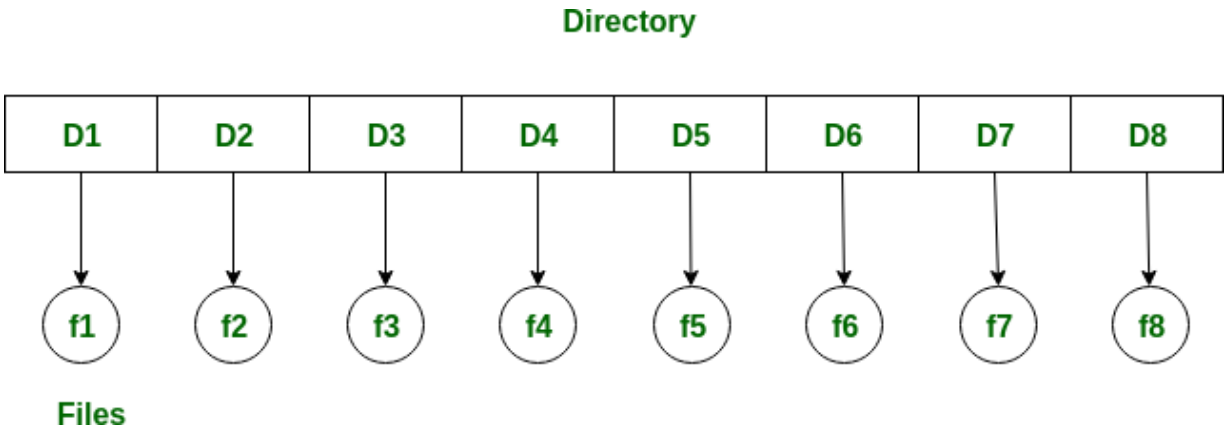


Following are the logical structures of a directory, each providing a solution to the problem faced in previous type of directory structure.

Single-level directory:

The single-level directory is the **simplest directory structure**. In it, all files are contained in the same directory which makes it easy to support and understand.

A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have a **unique name**. If two users call their dataset test, then the unique name rule violated.



Advantages:

- Since it is a single directory, so its implementation is very easy.
- If the files are smaller in size, searching will become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.
- **Logical Organization**: Directory structures help to logically organize files and directories in a hierarchical structure. This provides an easy way to navigate and manage files, making it easier for users to access the data they need.
- **Increased Efficiency:** Directory structures can increase the efficiency of the file system by reducing the time required to search for files. This is because directory structures are optimized for fast file access, allowing users to quickly locate the file they need.
- **Improved Security**: Directory structures can provide better security for files by allowing access to be restricted at the directory level. This helps to prevent unauthorized access to sensitive data and ensures that important files are protected.
- **Facilitates Backup and Recovery**: Directory structures make it easier to backup and recover files in the event of a system failure or data loss. By storing related files in the same directory, it is easier to locate and backup all the files that need to be protected.
- **Scalability:** Directory structures are scalable, making it easy to add new directories and files as needed. This helps to accommodate growth in the system and makes it easier to manage large amounts of data.

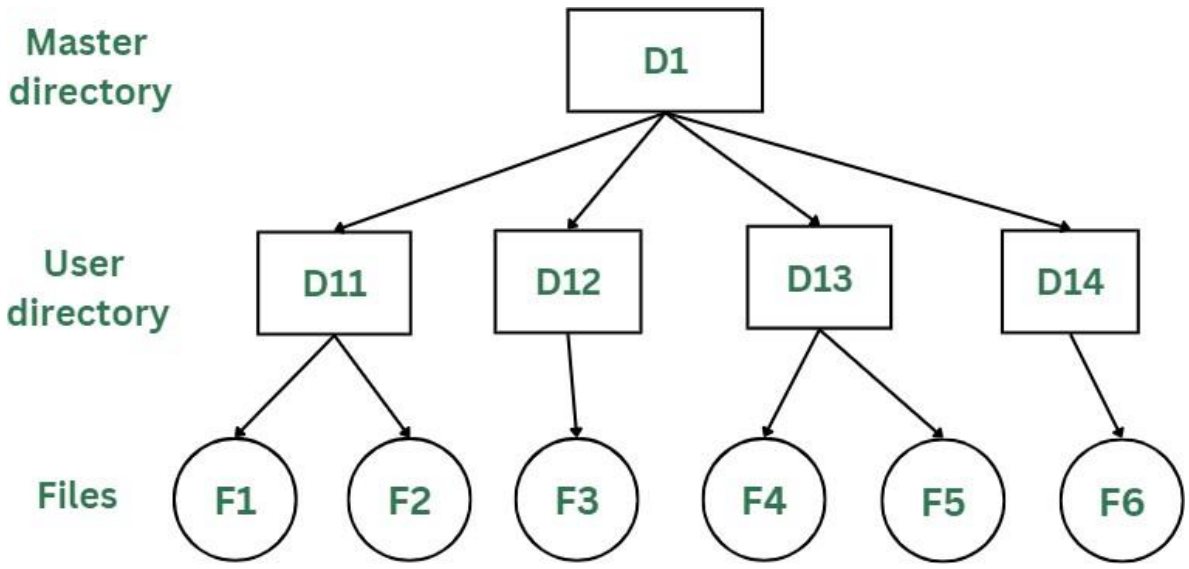
Disadvantages:

- There may chance of name collision because two files can have the same name.
- Searching will become time taking if the directory is large.
- This can not group the same type of files together.

Two-level directory:

As we have seen, a single level directory often leads to confusion of files names among different users. The solution to this problem is to create a **separate directory for each user**.

In the two-level directory structure, each user has their own **user files directory (UFD).** The UFDs have similar structures, but each lists only the files of a single user. System's **master file directory (MFD*)*** is searched whenever a new user id is created.



Two-Levels Directory Structure

Advantages:

- The main advantage is there can be more than two files with same name, and would be very helpful if there are multiple users.
- A security would be there which would prevent user to access other user's files.
- Searching of the files becomes very easy in this directory structure.

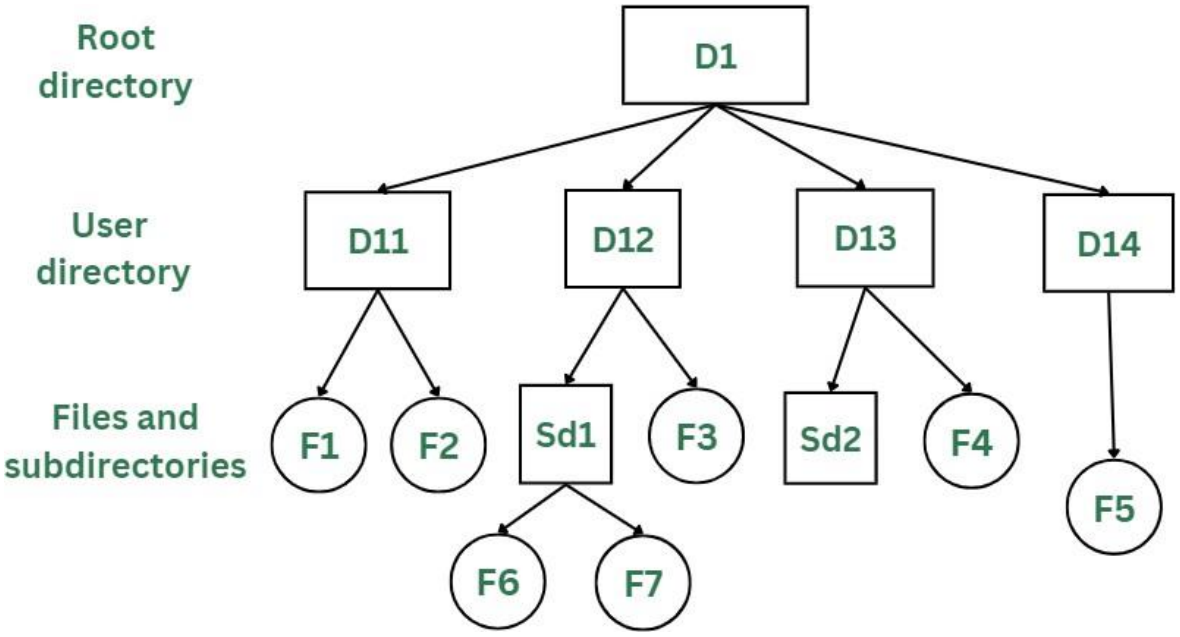
Disadvantages:

- As there is advantage of security, there is also disadvantage that the user cannot share the file with the other users.
- Unlike the advantage users can create their own files, users don't have the ability to create subdirectories.
- Scalability is not possible because one use can't group the same types of files together.

3) Tree Structure/ Hierarchical Structure:

Tree directory structure of operating system is most commonly used in our **personal computers**. User can create files and subdirectories too, which was a disadvantage in the previous directory structures.

This directory structure resembles a real tree upside down, where the **root directory** is at the peak. This root contains all the directories for each user. The users can create subdirectories and even store files in their directory. A user do not have access to the root directory data and cannot modify it. And, even in this directory the user do not have access to other user's directories. The structure of tree directory is given below which shows how there are files and subdirectories in each user's directory.



Tree/Hierarchical Directory Structure

Advantages:

- This directory structure allows subdirectories inside a directory.
- The searching is easier.
- File sorting of important and unimportant becomes easier.
- This directory is more scalable than the other two directory structures explained.

Disadvantages:

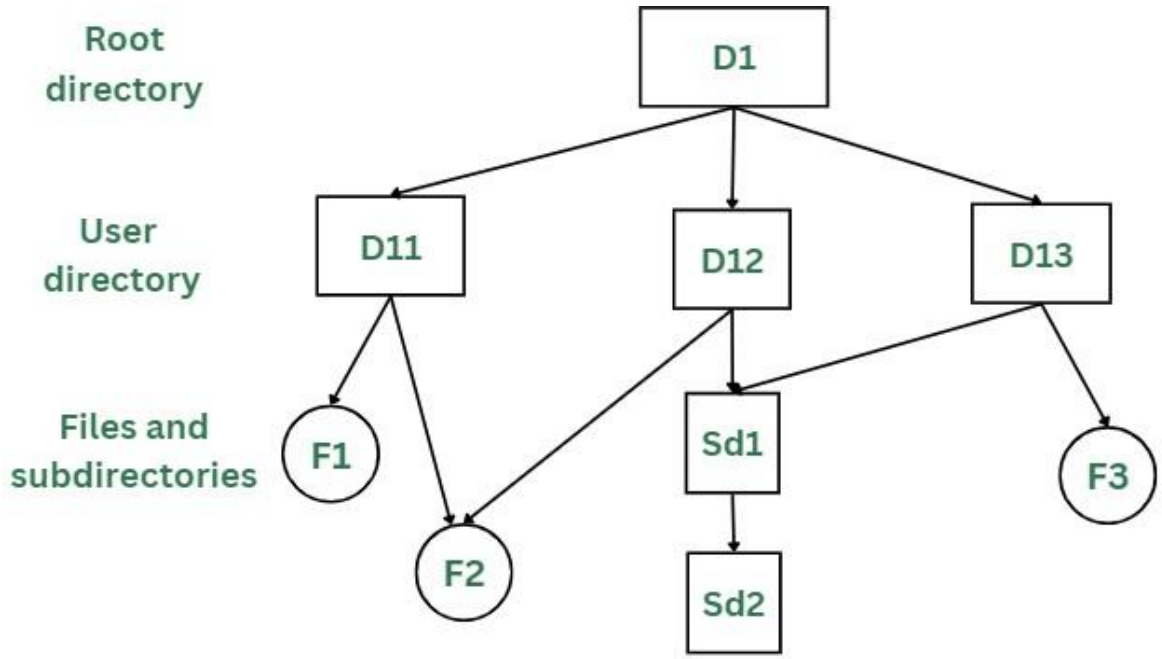
- As the user isn't allowed to access other user's directory, this prevents the file sharing among users.
- As the user has the capability to make subdirectories, if the number of subdirectories increase the searching may become complicated.
- Users cannot modify the root directory data.
- If files do not fit in one, they might have to be fit into other directories.

4) Acyclic Graph Structure:

As we have seen the above three directory structures, where none of them have the capability to access one file from multiple directories. The file or the subdirectory could be accessed through the directory it was present in, but not from the other directory.

This problem is solved in acyclic graph directory structure, where a file in one directory can be accessed from multiple directories. In this way, the files could be shared in between the users. It is designed in a way that multiple directories point to a particular directory or file with the help of links.

In the below figure, this explanation can be nicely observed, where a file is shared between multiple users. If any user makes a change, it would be reflected to both the users.



Acyclic Graph Structure

Advantages:

- Sharing of files and directories is allowed between multiple users.
- Searching becomes too easy.
- Flexibility is increased as file sharing and editing access is there for multiple users.

Disadvantages:

- Because of the complex structure it has, it is difficult to implement this directory structure.
- The user must be very cautious to edit or even deletion of file as the file is accessed by multiple users.
- If we need to delete the file, then we need to delete all the references of the file inorder to delete it permanently.

Conclusion:

Thus, the file allocation strategies such as sequential, indexed and linked files are implemented successfully.

Experiment No 10

Aim:

Write a program in C to do disk scheduling - FCFS, SCAN, C-SCAN

Objectives:

To understand and implement various disk scheduling algorithms.

Outcomes:

Students are able to implement disk scheduling algorithms such as FCFS, SCAN and C-SCAN.

Theory:

Disk scheduling is done by operating systems to schedule I/O requests arriving for disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by disk controller. Thus other I/O requests need to wait in waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of computer system and thus need to be accessed in an efficient manner.

There are many Disk Scheduling Algorithms but before discussing them let's have a quick look at some of the important terms:

- **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.
- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
- **Disk Access Time:** Disk Access Time is:
    - Disk Access Time = Seek Time + Rotational Latency +Transfer Time
- **Disk Response Time:** Response Time is the average of time spent by a request waiting to perform its I/O operation. *Average Response time* is the response time of the all requests. *Variance Response Time* is measure of how individual request are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

Disk Scheduling Algorithms

1. **FCFS:** FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

2. **SCAN:** In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works like an elevator and hence also known as **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

3. **CSCAN**: In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

   These situations are avoided in *CSCAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Conclusion:

Hence we have successfully implemented Disk scheduling algorithms like FCFS, SCAN and CSCAN.