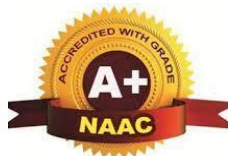




You Choose, We Do It
St. JOSEPH'S COLLEGE OF ENGINEERING
(An Autonomous Institution)
St. Joseph's Group of Institutions
OMR, CHENNAI - 119



DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE STAFF MANUAL

Prepared by

Name: P KUMARI DEEPIKA

Year/Semester: II/III

Sub. Code & Name: CS25308 -OPERATING SYSTEMS
(Lab Integrated)



You Choose, We Do It
St. JOSEPH'S COLLEGE OF ENGINEERING
(An Autonomous Institution)
St. Joseph's Group of Institutions
OMR, CHENNAI - 119



CS25308 -OPERATING SYSTEMS(Lab Integrated)
(LABORATORY MANUAL)

(R-2025)

III SEMESTER

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Prepared by:
P KUMARI DEEPIKA
AP/ADS



You Choose, We Do It
St. JOSEPH'S COLLEGE OF ENGINEERING
(An Autonomous Institution)
St. Joseph's Group of Institutions
OMR, CHENNAI - 119



COLLEGE VISION

To become a premier institution and centre of professional education with moral ethics in engineering and technology to produce socio-friendly technocrats.

COLLEGE MISSION

- To provide quality skill-based professional education for all deserving candidates belonging to all sectors of society.
- To adopt innovative strategies, add-on courses, and establishes the state-of-the-art infrastructure for effective teaching learning processes and research activities.
- To attract and nurture the excellent and committed faculty members and the technical workforce.
- To organize healthy interactions with the stakeholders for the students' overall personality development.
- To inculcate the social responsibilities among the prospective engineers.

VISION AND MISSION OF THE DEPARTMENT

VISION

Our vision is to become a global hub of innovation and knowledge, empowering students to shape a better future through excellence in education and research. We aspire to be a beacon of academic and ethical leadership, inspiring positive change in society.

MISSION

At the forefront of developing technologies, The Department of Artificial Intelligence and Data Science fosters the next generation of innovators. We empower students to not just analyse data, but to wield it as a tool for revolutionising healthcare, predicting global trends, and even pushing the boundaries of creative expression. We are a collaborative hub where renowned faculty mentors, and students engage in cutting - edge research alongside industry leaders. This network extends far beyond graduation, providing invaluable connections and propelling careers in fields like machine learning, data science, and AI research. Investing in our program is an investment in a future shaped by intelligence and innovation. Your child will graduate not just as a technically proficient professional, but as a leader equipped to tackle the challenges and opportunities of the data-driven world.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1: Graduates to exploit the knowledge of basic science, mathematics, statistics, and data science to build systems that require management and analysis of large volumes of real time data.

PEO2: To enrich the graduates with technical and professional skills to apply the concept of Artificial Intelligence to develop elegant solutions for the complex problems in various domains.

PEO3: To enable the graduates to think logically, pursue lifelong learning, and pioneering research in the field of Artificial Intelligence and Data Science to create disruptive and sustainable solutions for the real world issues.

PEO4: To inculcate ethical attitude, social responsibilities, and soft skills to work as a team to solve social, business and environmental problems.

Programme Outcomes(POs):

1. **PO1:** Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and specialization to solve complex engineering problems.
2. **PO2:** Problem Analysis: Identify, formulate, and analyze complex engineering problems using first principles of mathematics and natural sciences.
3. **PO3:** Design / Development of Solutions: Design system components or processes that meet specified needs considering public health, safety, and environment.
4. **PO4:** Conduct Investigations of Complex Problems: Use research methods including experiments, analysis, interpretation of data, and synthesis of information to draw valid conclusions.
5. **PO5:** Engineering Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitation to solve complex engineering problems.
6. **PO6:** The Engineer and The World: Analyse and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment.
7. **PO7:** Ethics: Apply ethical principles and commit to professional ethics, human values, diversity and inclusion, adhere to national and international laws.
8. **PO8:** Individual and collaborative Team Work: Function effectively as an individual and as a member or leader in diverse/multi-disciplinary teams.
9. **PO9:** Communication: Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language and learning differences.
10. **PO10:** Project Management and Finance: Apply knowledge and understanding of engineering management principles and economic decision-making apply these to once own work, as a member and leader in a team, and to manage projects and in multi-disciplinary environments.
11. **PO11:** Life-long Learning: Recognize the need for, and have the preparation and ability for i) Independent and lifelong- learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change.

(COMMON TO CSE, ADS, AML, CSBS, CYBER, M.TECH CSE)

OBJECTIVES

Introduce the fundamental concepts and design principles of operating systems.

- Explain process management, CPU scheduling, synchronization, and deadlock handling.
- Describe memory and file management mechanisms.
- Provide practical exposure to system-level programming and simulation of OS concepts.
- Enable students to relate theory to practice through Linux commands, shell scripting, and algorithmic simulation.

UNIT I INTRODUCTION TO OPERATING SYSTEMS

Basic Concepts: Operating system objectives and functions, evolution of OS-OS Services, System Calls, Types of OS (Batch, Time Sharing, Distributed, Real-Time) -Operating System Structure – Layered, Monolithic, Microkernel, Modular.

Case Study: Linux System Structure.

Lab Integration:

Practice basic Linux/UNIX commands for file, process, and user management. Simple shell scripting exercises.

UNIT II PROCESS MANAGEMENT

Processes: Concept, States, Process Control Block (PCB). Process Scheduling: Scheduling criteria and algorithms – FCFS, SJF, RR, Priority. Threads: Multithreading models. Interprocess Communication (IPC).

Lab Integration:

Implement CPU Scheduling algorithms in C/Python. Implement Interprocess Communication using pipes/message queues/shared memory.

UNIT III PROCESS SYNCHRONIZATION AND DEADLOCK

Critical Section Problem, Semaphores, Monitors. Classical Synchronization Problems: Producer-Consumer, Reader-Writer, Dining Philosophers. Deadlock: Characterization, Prevention, Avoidance (Banker's Algorithm), Detection, and Recovery.

Lab Integration:

Simulate process synchronization using semaphores or monitors. Implement deadlock detection and avoidance algorithms.

UNIT IV MEMORY MANAGEMENT

Memory Partitioning, Paging, Segmentation. Virtual Memory: Demand Paging, Page Replacement Algorithms – FIFO, LRU, Optimal. Allocation Methods – Contiguous, Non-contiguous, Fragmentation.

Lab Integration:

Simulate memory allocation techniques (First Fit, Best Fit, Worst Fit). Implement page replacement algorithms.

UNIT V FILE SYSTEMS AND I/O MANAGEMENT

File Concepts, Access Methods, Directory Structure, File Allocation (Contiguous, Linked Indexed). File Protection and Access Control-Disk Scheduling: FCFS, SSTF, SCAN, C SCAN.I/O Hardware and Device Drivers overview.

Lab Integration:

Simulate file allocation techniques and disk scheduling algorithms. Mini project: Build a simple file system or process scheduler simulation.

TOTAL PERIODS: 75**TEXT BOOKS**

1. Abraham Silberschatz, Peter B. Galvin, Greg Gagne — *Operating System Concepts*, 10th Edition, Wiley.
2. William Stallings — *Operating Systems: Internals and Design Principles*, Pearson, 9th Edition.

REFERENCE BOOKS

1. Andrew S. Tanenbaum — *Modern Operating Systems*, Pearson.
2. D.M. Dhamdhare — *Operating Systems: A Concept-Based Approach*, McGraw Hill.
3. Achyut S. Godbole — *Operating Systems*, Tata McGraw Hill.

COURSE OUTCOMES

Upon completion of the course, students will be able to

CO1 Explain the fundamental structure, functions, and services of modern operating systems.

CO2 Implement and analyze process management and CPU scheduling mechanisms.

CO3 Simulate process synchronization and deadlock handling algorithms.

CO4 Demonstrate memory management and page replacement techniques through programming.

CO5 Design and simulate file system and I/O management algorithms.

MAPPING OF COs WITH POs AND PSOs

COs	PROGRAM OUTCOMES (POs)												PROGRAM SPECIFIC
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1
CO1	3	2	1	2	2	-	1	-	-	1	2	-	3
CO2	3	3	2	3	3	-	-	-	1	1	2	-	3
CO3	3	3	2	3	3	-	-	-	-	1	2	-	3
CO4	3	2	2	3	3	-	-	-	-	1	2	-	3
CO5	3	3	3	3	3	-	-	1	2	2	3	-	3

Ex No: 1a

BASICS OF UNIX COMMANDS

Date :

AIM

To study and execute the basics of UNIX commands

1. pwd

Display the full path of the current working directory

```
user@sys246:~$ pwd
/home/user
```

2. ls

This command displays the list of files in current working directory
\$ls -l list the files in long format

\$ls -t list in order of last modification time

\$ls -d Lists directory instead of contents

\$ls -u Lists in order of last access

```
user@sys246:~$ ls
f1.c f2.c f3.txt f4.txt CSEal
```

3. cd

This command is used to change from working directory to any other directory specified
\$cd directoryname

```
user@sys107:~$ cd Desktop
```

```
user@sys107:~/Desktop$
```

4. cd..

This command is used to come out of the current working directory.
\$cd..

```
user@sys107:~/Desktop $ cd
```

```
user@sys107:~$
```

5. mkdir

This command helps us to make a directory

\$mkdir directoryname

```
user@sys246:~$ mkdir cse.
```

6. rmdir

This command is used to remove a directory specified in command line.
\$rmdir directoryname

```
user@sys246:~$ rmdir cse
```

```
user@sys246:~$ ls
```

```
f1.c f2.c f3.txt f4.txt f4.c f5.c
```

7. cat

This command helps us to list the contents of the file we specify.

\$cat[option][file]

cat>filename - This is used to create a new file

cat>>filename - This is used to append the contents of the file

```
user@sys246:~$ cat f4.c
```

```
India is good country
```

8. cp

This command helps us to create duplicate copies of ordinary files. \$cp source destination
 user@sys246:~\$ cp f1.c f5.c
 user@sys246:~\$ ls
 f1.c f2.c f3.txt f4.txt f5.c CSEal

9. mv

This command is used to move or rename the files.
 \$mv source destination
 user@sys246:~\$ mv f1.c f4.c
 user@sys246:~\$ ls
 f2.c f3.txt f4.txt f5.c f4.c CSEal
 user@sys246:~\$

10. ln

This command is to establish an additional filename for the same ordinary file. \$ln firstName secondName
 user@sys246:~\$ ls
 f1.c f2.c f4.c f5.c f3.txt f4.txt IT
 user@sys246:~\$ ln -t CSE f5.c
 user@sys246:~\$ cd CSE
 user@sys246:~\$ ls
 F5.c

11. rm

This command is used to delete one or more files from the directory. \$rm [option] filename
 \$rm -i : Asks the user if he wants to delete the file mentioned.
 \$rm -r : Recursively delete the entire contents of directory as well as the directory itself user@sys246:~\$ rm f3.txt
 user@sys246:~\$ ls
 f2.c f4.txt f5.c f4.c CSEal
 user@sys246:~\$

PROCESS AND STATUS INFORMATION COMMANDS:**1. who**

This command gives the details of who all have logged into the UNIX system currently. \$who
 user@sys246:~\$ who
 guest-lpdjsy tty7 2023-03-07 08:56 (:0)
 user tty8 2023-03-07 09:30 (:1)

2. who am i

This command tells us as to when we had logged in and the system's name for the connection being used. \$whoami
 user@sys246:~\$ whoami
 user

3. date

This command displays the current date in different formats.
 +%D mm/dd/yy +%w Day of the week
 +%H hr-00 to 23 +%a Abbr Weekday

```
+%M min-00 to 59 +%h Abbr Month
+%S sec-00 to 59 +%r Time in AM/PM
user@sys246:~$ date
Tue Mar 7 09:45:00 IST 2023
user@sys246:~$ date +%m
03
```

4. echo

This command will display the text typed from keyboard.

```
$echo
```

```
user@sys246:~$ echo SJCE
```

```
SJCE
```

TEXT RELATED COMMANDS**1. head**

Displays the initial part of the text file. By default, it displays the first 10 lines of a file. user@sys246:~\$ head -1 f3.txt

```
Hi this is Unix Lab
```

2. tail

Displays the last part of the text file. By default, it displays the last 10 lines of a file. user@sys246:~\$ tail -1 f3.txt

```
I am studying B.E CSE in SJCE.
```

3. wc

This command is used to count the no. of lines, words, characters in a file.

```
$wc[-lwc]filename
```

```
user@sys246:~$ wc f1.txt
```

```
8 26 150 f4.txt
```

4. find

The find command is used to locate files in a directory and in a subdirectory.

```
user@sys246:~$ find
```

```
.
```

```
./1.c
```

```
./asd.bak
```

```
./f1.txt
```

```
./file_copy.c
```

```
./f2.c
```

```
./a.out
```

FILE PERMISSION COMMANDS**1. chmod**

Change the file/directory permission mode \$chmod777 file 1. Give full permission to owner, groups, and others \$chmod o-w file 1. Remove write permission for others.

```
user@sys246:~$ chmod777 welcome.txt.
```

USEFUL COMMANDS:**1. exit**

Ends your work on the UNIX system.

```
user@sys246:~$ exit
```

2. clear

Clears the screen

```
user@sys246:~$ clear
```

3. Ctrl-z

Pauses the currently running program

```
user@sys246:~$ pause
```

4. man COMMAND

Looks up the UNIX command in online manual pages.

```
user@sys246:~$ man
```

What manual page do you want?

```
user@sys246:~$ man ls
```

NAME

ls - list directory contents

SYNOPSIS

ls [OPTION]... [FILE]...

DESCRIPTION

List information about the FILES (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.

5. Ctrl-c

Stops the program currently running

6. history

Lists all the commands typed so far.

```
user@sys246:~$ history
```

```
10 mkdir os
```

```
11 ls
```

```
12 who
```

```
13 who am i
```

```
14 date
```

```
15 whoami
```

```
16 echo
```

```
17 echo hello
```

7. more FILE

Display the contents of FILE, pausing after each screenful. There are several keys that control the output once a screenful has been printed. <enter> will advance the output one line at a time. <spacebar> will advance the output by another full screenful. “q” will quit and return you to the UNIX prompt.

```
user@sys246:~$ more message1
```

```
Hello
```

```
Welcome
```

8. less FILE

“less” is a program similar to “more”, but which allows backward movement in the file as well as forward movement.

```
user@sys246:~$ less message1
```

```
Hello
```

```
Message
```

```
message1(end).
```

RESULT

Thus, the programs for basics of UNIX commands have been implemented and the output was verified successfully.

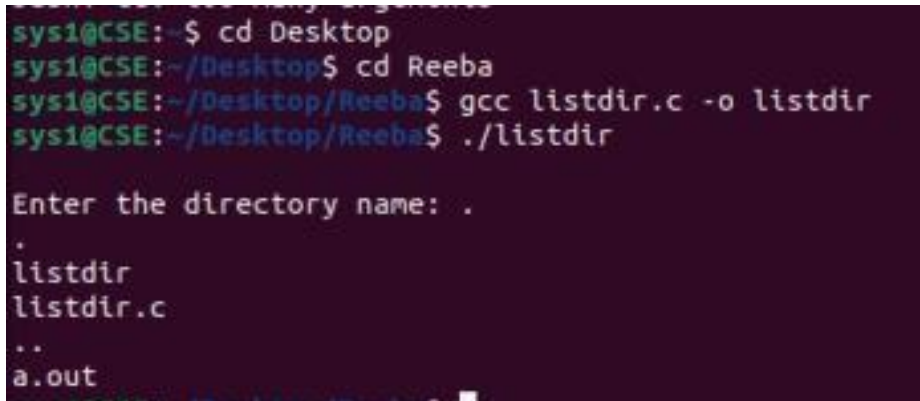
AIM:

To write a „C“ program for implementing **ls**, **cp**, **grep**, **cat** command

// ls command**ALGORITHM:**

1. Start
2. Input directory name to opendir and readdir system call to open and read contents from directory.
3. Use exit system call if directory not exists.
4. Close directory after reading using closedir()
5. Stop.

```
#include <stdio.h>
#include <dirent.h>
#include <stdlib.h>
int main()
{
    char buff[100];
    DIR *dirp;
    struct dirent *dptr;
    printf("\nEnter the directory name: ");
    scanf("%99s", buff); // Prevent buffer overflow
    dirp = opendir(buff);
    if (dirp == NULL)
    {
        printf("The given directory does not exist\n");
        exit(1);
    }
    while ((dptr = readdir(dirp)) != NULL)
    {
        printf("%s\n", dptr->d_name);
    }
    closedir(dirp);
    return 0;
}
```

OUTPUT

```
sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc listdir.c -o listdir
sys1@CSE:~/Desktop/Reeba$ ./listdir

Enter the directory name: .
.
listdir
listdir.c
..
a.out
```

//cp command**ALGORITHM:**

Step 1: Get the name of the source and destination file from the command line.

Step 2: Open the source file in read-only mode using the open system call. If the file could not be open, print an error message.

Step 3: Create the destination file using creat system call. If the file could not be open, print an error message. **Step 4:** Read from the source file using the read system call and store the content in the buffer.

Step 5: From the buffer write to the destination file using the write system call.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main(int argc, char *argv[])
{
    int f1, f2, n;
    char buf[512];
    if (argc != 3)
    {
        printf("Usage: %s source_file destination_file\n", argv[0]);
        exit(1);
    }
    if ((f1 = open(argv[1], O_RDONLY)) < 0)
    {
        printf("Can't open %s\n", argv[1]);
        exit(1);
    }
    if ((f2 = creat(argv[2], 0644)) <
0) {

    printf("Can't create %s\n", argv[2]);

    exit(1);

    }
    while ((n = read(f1, buf, sizeof(buf))) >
0) {

    write(f2, buf, n);
    }
    close(f1);
    close(f2);
    printf("File copied successfully\n");

    return 0;

    }
user@sys246:~$ gcc pg4.c
user@sys246:~$ ./a.outsamp.txt s1.txt
user@sys246:~$ cat samp.txt
hello welcome to Chennai.
```

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc copy.c -o copy
sys1@CSE:~/Desktop/Reeba$ ./copy src.txt dest.txt
File copied successfully
sys1@CSE:~/Desktop/Reeba$ cat src.txt
hello Reeba
sys1@CSE:~/Desktop/Reeba$ cat dest.txt
hello Reeba
sys1@CSE:~/Desktop/Reeba$

```

// SIMULATION OF GREP COMMAND

ALGORITHM:

Step 1: Include the header files essential to simulate CAT command. **Step 2:** Create a file with some contents in it.

Step 3: Give two choices

- (i) To print the similar pattern word.
- (ii) To print the entire line of the specified pattern.

Step 4: Define the two cases using Switch cases.

Step 5: Print the lines of similar pattern depending upon the choice of selection. **Step 6:** Close the file and exit.

PROGRAM:

```

#include <stdio.h>
#include <string.h>
#define bs 1024
int main()
{
    FILE *f1;
    char file1[bs], str[bs], pat[bs];
    printf("Enter the file name: ");
    scanf("%s", file1);
    printf("Enter the pattern to be searched: ");
    scanf("%s", pat);
    f1 = fopen(file1, "r");
    if (f1 == NULL)
    {
        printf("Cannot open file\n");
        return 1;
    }
    while (fscanf(f1, "%s", str) != EOF)
    {
        if (strstr(str, pat) != NULL)
        {
            printf("Pattern found in word: %s\n", str);
        }
    }
    fclose(f1); return 0; }

```

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc pattern.c -o pattern
sys1@CSE:~/Desktop/Reeba$ ./pattern
Enter the file name: test.txt
Enter the pattern to be searched: ls
Pattern found in word: this
Pattern found in word: ls
Pattern found in word: this
Pattern found in word: ls
sys1@CSE:~/Desktop/Reeba$

```

// SIMULATION OF CAT COMMAND

ALGORITHM:

Step 1: Include the header files essential to simulate CAT command.

Step 2: Define Buffer size of 1024 which can be used as the standard size for file and buffer. **Step 3:** Create a source file with some contents in it.

Step 4: Open the source file in READ mode.

Step 5: Print the contents of the file that was opened in READ mode.

Step 6: Close the file and exit.

PROGRAM:

```

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#define bs 1024
int main()
{
    int f;
    char file1[bs], buf[bs];
    int n;
    printf("Enter the filename:\n");
    scanf("%s", file1);
    f = open(file1, O_RDONLY);
    if (f < 0)
    {
        printf("File does not exist\n");
        return 1; }
    while ((n = read(f, buf, bs)) > 0)
    {
        write(1, buf, n); // write to stdout }
    close(f);
    return 0; }

```

OUTPUT

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc display.c -o display
sys1@CSE:~/Desktop/Reeba$ ./display
Enter the filename:
test.txt
hello world
this is my first lab
this is os lab
sys1@CSE:~/Desktop/Reeba$

```

RESULT

Thus, the programs for simulation of UNIX commands have been implemented and the output was verified successfully.

Ex No: 2a

SHELL PROGRAMMING-LARGEST OF N NUMBERS

Date:

AIM:

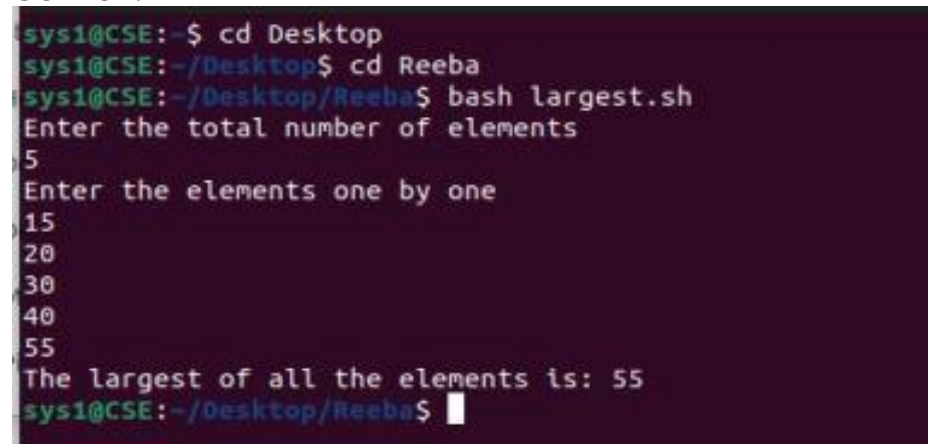
To write a shell program to find largest of n numbers.

ALGORITHM:

1. Input the total number of elements (n) .
2. Let i=0
3. Read the elements one by one in an array a[i] .
4. Initialize large=0 .
5. Using a for loop, do the following for all elements in the array:
Let temp=a[i] ie. Assign the array element to temp.
If temp greater than large, then reassign large=temp
6. large displays the largest of all elements.
7. Stop.

PROGRAM:

```
#!/bin/bash
echo "Enter the total number of elements"
read n
echo "Enter the elements one by one"
for ((i=0; i<n; i++))
do
    read a[i]
done
large=${a[0]}
for ((i=1; i<n; i++))
do
    if (( a[i] > large ))
    then
        large=${a[i]}
    fi
done
echo "The largest of all the elements is: $large"
```

OUTPUT:

```
sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ bash largest.sh
Enter the total number of elements
5
Enter the elements one by one
15
20
30
40
55
The largest of all the elements is: 55
sys1@CSE:~/Desktop/Reeba$
```

RESULT

Thus, the program to find Largest of N numbers been implemented and the output was verified successfully.

AIM:

To write a shell program using case...esac to execute unix commands.

ALGORITHM:

1. Get the choice from the user.
2. List the choices for date command, list of users logged in and name of the home directory.
3. Date command displays today's date.
4. Who command displays the users who have logged in.
5. \$Home command is used to display the home directory.
6. Depending upon the choice, any one of the above command is executed using case...esac.
7. Stop.

PROGRAM:

```
while true
do
    echo "Enter your choice:"
    echo "Enter w to quit"
    echo "Menu"
    echo "1.Today's date"
    echo "2.List of users logged in"
    echo "3.Name of the home directory"
    read choice
    case $choice in
        1) date;;
        2) who;;
        3) echo "Home directory is $HOME";;
        w)break;;
        *)echo "Invalid choice"
    esac
done
```

OUTPUT:

```
sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ bash large.sh
Enter your choice:
Enter w to quit
Menu
1.Today's date
2.List of users logged in
3.Name of the home directory
1
Friday 19 December 2025 10:49:12 AM IST
Enter your choice:
Enter w to quit
Menu
1.Today's date
2.List of users logged in
3.Name of the home directory
2
sys1      tty2      2025-12-19 09:12 (tty2)
Enter your choice:
Enter w to quit
Menu
1.Today's date
2.List of users logged in
3.Name of the home directory
3
Home directory is /home/sys1
Enter your choice:
Enter w to quit
Menu
1.Today's date
2.List of users logged in
3.Name of the home directory
w
sys1@CSE:~/Desktop/Reeba$
```

RESULT:

Thus, the program for implementation of UNIX commands using case has been implemented and the output was verified successfully.

AIM:

To write a „C“ program to implement FCFS Scheduling.

ALGORITHM:

1. Start.
2. Include the header files for simulating FCFS scheme.
3. Declare the variables to calculate the essential aspects of FCFS.
4. Initially get the process time, arrival time and burst time.
5. Swap all the processes in such a way that they are arranged in FCFS order.
6. Prepare the gantt chart for each process so that their waiting time, turn around time is calculated.
7. Calculate the average waiting time and average turn around time and display the result.
8. Stop

// FCFS SCHEDULING ALGORITHM

PROGRAM:

```
#include<stdio.h>
void main()
{
    int i,n,br[10],wt[10],wat[10],tr[10],ttr=0,twt=0,twat=0;
    float awat,atur;
    printf("Enter no. of process:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the burst time for process %d: ",(i+1));
        scanf("%d",&br[i]);
    }
    printf("\nPROCESS\tBURST TIME\tWAITING TIME\tTURN AROUND TIME\n");
    wt[0]=0;
    for(i=0;i<n;i++)
    {
        wt[i+1]=wt[i]+br[i];
        tr[i]=wt[i]+br[i];
        printf("\n%d\t%d\t%d\t%d\n",i+1,br[i],wt[i],tr[i]);
    }
    for(i=0;i<n;i++)
    {
        ttr=ttr+tr[i];
        twat=twat+wt[i]; //wat i changed to wt
    }
    printf("\nGAANT CHART\n\n");
    for(i=0;i<n;i++)
    {
        printf("----- ");
    }
    printf("\n");
    for(i=0;i<n;i++)
    {
        printf(" P%d |",i+1);
    }
    printf("\n");
}
```

```

for(i=0;i<n;i++)
{
    printf(" ----- ");
}
printf("\n");
for(i=0;i<=n;i++)
{
    printf("%d\t",wt[i]);
}
awat=(float)twat/n;
atur=(float)ttr/n;
printf("\n\nTotal waiting time :%d",twat);
printf("\n\nTotal turn around time :%d",ttr);
printf("\n\nAverage waiting time :%.2f",awat);
printf("\n\nAverage turn around time :%.2f\n",atur); }

```

OUTPUT

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc fcfs.c -o fcfs
sys1@CSE:~/Desktop/Reeba$ ./fcfs
Enter no. of process :5
Enter the burst time for process 1: 3
Enter the burst time for process 2: 1
Enter the burst time for process 3: 4
Enter the burst time for process 4: 6
Enter the burst time for process 5: 8

PROCESS BURST TIME          WAITING TIME          TURN AROUND TIME
1              3              0              3
2              1              3              4
3              4              4              8
4              6              8              14
5              8              14             22

GAANT CHART
-----
P1 | P2 | P3 | P4 | P5 |
-----
0   3   4   8  14  22

Total waiting time :29
Total turn around time :51

Average waiting time :5.80
Average turn around time :10.20
sys1@CSE:~/Desktop/Reeba$

```

RESULT:

Thus, the program for FCFS scheduling has been implemented and the output was verified successfully.

Ex No: 3B

SJF SCHEDULING

Date :

AIM:

To write a „C“ program to implement SJF Scheduling.

ALGORITHM:

1. Include the header files for simulating SJF scheme.
2. Declare the variables to calculate the essential aspects of SJF.
3. Initially get the process time and burst time.
4. Swap all the processes in such a way that they are arranged in shortest job first (SJF) order.

5. Prepare the gantt chart for each process so that their waiting time, turn around time is calculated.

6. Calculate the average waiting time and average turn around time and display the result. // SJF

SCHEDULING ALGORITHM**PROGRAM:**

```
#include<stdio.h>
void main()
{
    int i,j,k,n,p[10],br[10],wt[10],tr[10],ttr=0,twt=0,t;
    float awat,atur;
    printf("\nEnter no. of process :");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter the burst time for process %d: ",(i+1));
        scanf("%d",&br[i]);
        p[i]=i+1;
    }
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(br[i]>br[j])
            {
                t=br[i];
                br[i]=br[j];
                br[j]=t;
                t=p[i];
                p[i]=p[j];
                p[j]=t;
            }
        }
    }
    printf("\nPROCESS\tBURST TIME \t WAITING TIME \t TURN AROUND TIME\n");
    wt[0]=0;
    for(i=0;i<n;i++)
    {
        wt[i+1]=wt[i]+br[i];
        tr[i]=wt[i]+br[i];
    }
    for(i=0;i<n;i++)
    {
        ttr=ttr+tr[i];
        twt=twt+wt[i];
    }
    for(i=0;i<n;i++) { printf("\n%d\t%d\t%d\t%d\n",p[i],br[i],wt[i],tr[i]); }
    printf("\nGAANT CHART\n\n");
    for(i=0;i<n;i++)
```

```

{
    printf(" ----- ");
}
printf("\n");
for(i=0;i<n;i++)
{
    printf(" P%d |",p[i]);
}
printf("\n");
for(i=0;i<n;i++)
{
    printf(" ----- ");
}
printf("\n");
for(i=0;i<=n;i++)
{
    printf("%d\t",wt[i]);
}
awat=(float)tw/n;
atur=(float)ttr/n;
printf("\nTotal waiting time :%d\n",tw);
printf("\nTotal turnaround time :%d\n",ttr);
printf("\nAverage waiting time :%.2f\n",awat);
printf("\nAverage turn around time :%.2f\n",atur);
}

```

OUTPUT

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc sjf.c -o sjf
sys1@CSE:~/Desktop/Reeba$ ./sjf

Enter no. of process :5
Enter the burst time for process 1: 4
Enter the burst time for process 2: 1
Enter the burst time for process 3: 7
Enter the burst time for process 4: 9
Enter the burst time for process 5: 3

PROCESS BURST TIME      WAITING TIME      TURN AROUND TIME
2          1             0             1
5          3             1             4
1          4             4             8
3          7             8             15
4          9             15            24

GAANT CHART
-----
P2 | P5 | P1 | P3 | P4 |
0  1  4  8  15  24
Total waiting time :28
Total turnaround time :52
Average waiting time :5.60
Average turn around time :10.40
sys1@CSE:~/Desktop/Reeba$

```

RESULT:

Thus, the program for SJF scheduling has been implemented and the output was verified successfully.

Ex No: 3C

PRIORITY SCHEDULING

Date :

AIM:

To write a „C“ program to implement PRIORITY Scheduling.

ALGORITHM:

1. Start
2. Include the header files for simulating priority scheme.
3. Declare the variables to calculate the essential aspects of priority scheme.
4. Initially get the process time, arrival time and burst time along with their priorities.
5. Execute the process by referring to their priority values.
6. Prepare the gantt chart for each process so that their waiting time, turn around time is calculated.
7. Calculate the average waiting time and average turn around time and display the result. 8. Stop.

// PRIORITY SCHEDULING ALGORITHM

PROGRAM:

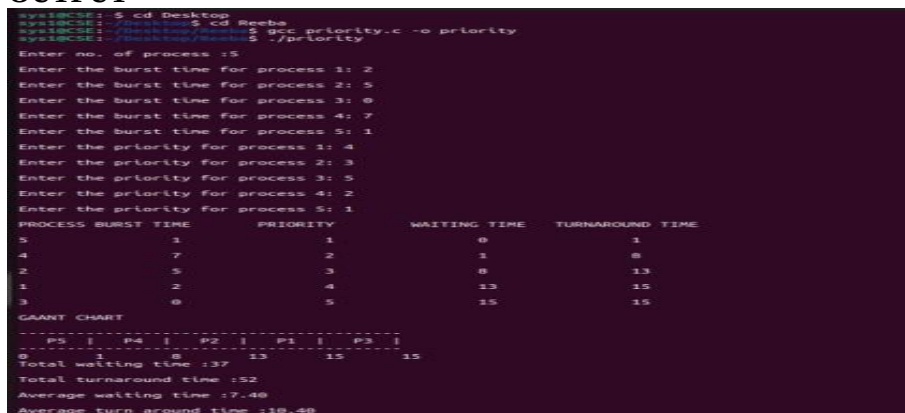
```
#include<stdio.h>
void main()
{
    int i,j,k,n,p[10],pri[10],bur[10],wat[10], tur[10],ttur=0,twat=0,t;
    float awat,atur;
    printf("\nEnter no. of process :");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter the burst time for process %d: ",(i+1));
        scanf("%d",&bur[i]);
        p[i]=i+1;
    }
    for(i=0;i<n;i++)
    {
        printf("\nEnter the priority for process %d: ",(i+1));
        scanf("%d",&pri[i]);
    }
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(pri[i]>pri[j])
            {
                t=bur[i];
                bur[i]=bur[j];
                bur[j]=t;
                t=p[i];
                p[i]=p[j];
                p[j]=t;
                t=pri[i];
                pri[i]=pri[j];
                pri[j]=t;
            }
        }
    }
    printf("\nPROCESS\tBURST TIME \t PRIORITY \t WAITING TIME \tTURNAROUND TIME\n");
    wat[0]=0;
```

```

for(i=0;i<n;i++)
{
    wat[i+1]=wat[i]+bur[i];
    tur[i]=wat[i]+bur[i]; }
for(i=0;i<n;i++)
{
    ttur=ttur+tur[i];
    twat=twat+wat[i]; }
for(i=0;i<n;i++)
{
    printf("\n%d\t%d\t%d\t%d\t%d\t%d\n",p[i],bur[i],pri[i],wat[i],tur[i]); }
printf("\nGAANT CHART\n\n");
for(i=0;i<n;i++)
{
    printf("-----"); }
printf("\n");
for(i=0;i<n;i++)
{
    printf(" P%d |",p[i]); }
printf("\n");
for(i=0;i<n;i++)
{
    printf("-----"); }
printf("\n");
for(i=0;i<=n;i++)
{
    printf("%d\t",wat[i]); }
awat=(float)twat/n;
atur=(float)ttur/n;
printf("\nTotal waiting time :%d\n",twat);
printf("\nTotal turnaround time :%d\n",ttur);
printf("\nAverage waiting time :%.2f\n",awat);
printf("\nAverage turn around time :%.2f\n",atur);
}

```

OUTPUT



RESULT:

Thus, the program for priority scheduling has been implemented and the output was verified successfully.

AIM:

To write a „C“ program to implement ROUND ROBIN Scheduling.

ALGORITHM:

1. Include the header files for simulating ROUND ROBIN scheduling scheme.
 2. Declare the variables to calculate the essential aspects of ROUND ROBIN.
 3. Initially get the time slice along with process time and burst time.
 4. Start from the 1st process and execute it for the given time slice and move to 2nd process and so on till all process gets executed for each of the time slice allotted.
 5. Prepare the gantt chart for each process getting executed within the time slice in a round robin fashion.
 6. Calculate the waiting time and turn around time along with their averages display the result. //
- ROUND ROBIN SCHEDULING ALGORITHM

PROGRAM:

```
#include<stdio.h>
void main()
{
    int i,x=-1,k[10],m=0,n,t,s=0;
    int a[50],temp,b[50],p[10],bur[10],bur1[10];
    int wat[10],tur[10],ttur=0,twat=0,tres=0,j=0;
    float awat,atur,ares;
    printf("Enter no. of process:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the burst time for process %d: ",(i+1));
        scanf("%d",&bur[i]);
        bur1[i]=bur[i];
    }
    printf("\nEnter the slicing time :");
    scanf("%d",&t);
    for(i=0;i<n;i++)
    {
        b[i]=bur[i]/t;
        if((bur[i]%t)!=0)
            b[i]=b[i]+1;
        m=b[i]+m;
    }
    printf("\nGAANT CHART\n\n");
    for(i=0;i<m;i++)
    {
        printf("----- ");
    }
    printf("\n");
    a[0]=0;
    while(j<m)
    {
        if(x==n-1)
            x=0;
        else
            x++;
        if(bur[x]>=t)
        {
            bur[x]=bur[x]-t;
```

```

        a[j+1]=a[j]+t;
        if(b[x]==1)
        {
            p[s]=x;
            k[s]=a[j+1];
            s++;
        }
        j++;
        b[x]=b[x]-1;
        printf(" P%d |",x+1);
    }
    else if(bur[x]!=0)
    {
        a[j+1]=a[j]+bur[x];
        bur[x]=0;
        if(b[x]==1)
        {
            p[s]=x;
            k[s]=a[j+1];
            s++;
        }
        j++;
        b[x]=b[x]-1;
        printf(" P%d |",x+1);
    }
}
printf("\n");
for(i=0;i<m;i++)
    printf(" ----- ");
printf("\n");
for(j=0;j<=m;j++)
    printf("%d\t",a[j]);
for(i=0;i<n;i++)
{
    for(j=i+1;j<n;j++)
    {
        if(p[i]>p[j])
        {
            temp=p[i]; p[i]=p[j];
            p[j]=temp; temp=k[i];
            k[i]=k[j]; k[j]=temp;
        }
    }
}
for(i=0;i<n;i++)
{
    wat[i]=k[i]-bur1[i];
    tur[i]=k[i];
}
printf("\nPROCESS\tBURST TIME \t WAITING TIME \t TURN AROUND\n");
printf("TIME\t"); printf("RESPONSE TIME\n");
for(i=0;i<n;i++)
{
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\n",p[i]+1,bur1[i],wat[i],

```

```

        tur[i],a[i]);
    }
    for(i=0;i<n;i++)
    {
        ttur=ttur+tur[i];
        twat=twat+wat[i];
        tres=tres+a[i];
    }
    awat=(float)twat/n;
    atur=(float)ttur/n;
    ares=(float)tres/n;
    printf("\nTotal waiting time:%d",twat);
    printf("\nTotal turnaround time:%d",ttur);
    // printf("\nTotal response time:%d",tres);
    printf("\nAverage waiting time :%.2f",awat);
    printf("\nAverage turn around time :%.2f",atur);
    //printf("\nAverage response time:%.2f",ares);
}

```

OUTPUT

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc roundrobin.c -o roundrobin
sys1@CSE:~/Desktop/Reeba$ ./roundrobin
Enter no. of process :5
Enter the burst time for process 1: 3
Enter the burst time for process 2: 5
Enter the burst time for process 3: 2
Enter the burst time for process 4: 8
Enter the burst time for process 5: 10

Enter the slicing time :5

GAANT CHART

-----
P1 | P2 | P3 | P4 | P5 | P4 | P5 |
-----
0   3   8   10  15  20  23  28
PROCESS BURST TIME      WAITING TIME      TURN AROUND TIME
1           3           0           3
2           5           3           8
3           2           8          10
4           8          15          23
5          10          18          28

Total waiting time:44
Total turnaround time:72
Average waiting time :8.80
sys1@CSE:~/Desktop/Reeba$ 

```

RESULT:

Thus, the program for Round Robin scheduling has been implemented and the output was verified successfully.

AIM:

To write a program to implement producer consumer problem using semaphore

ALGORITHM:

1. Declare the buffer size variable.
2. Initialize empty, full and the value of mutex.
3. For case 1 , produce the item till the buffer is full after performing the wait operation
Of variables empty and mutex.
4. Then perform signal of full.
5. For case 2, consume the item till the buffer is empty. Perform the wait and signal operation..
6. print the contents of the buffer.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define SIZE 3

struct process {
    char a[5];
} buffer[SIZE];

int mutex = 1, full = 0, empty = SIZE;

/* Function prototypes */
int wait(int);
int signal(int);

int main() {
    int ch, i;
    char item[5];

    printf("\n\tProducer Consumer Problem\n");

    while (1) {
        printf("\n1. Producer");
        printf("\n2. Consumer");
        printf("\n3. Buffer Contents");
        printf("\n4. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1: /* Producer */
                if (empty == 0) {
                    printf("\nBuffer is Full\n");
                    break;
                }

                empty = wait(empty);
                mutex = wait(mutex);
```

```
printf("Enter item to produce: ");
scanf("%s", item);
strcpy(buffer[full].a, item);
full++;
printf("Item Produced Successfully\n");
mutex = signal(mutex);
break;
case 2: /* Consumer */
    if (full == 0) {
        printf("\nBuffer is Empty\n");
        break;
    }
    full = wait(full);
    mutex = wait(mutex);
    printf("Consumed item: %s\n", buffer[0].a);
    /* Shift buffer */
    for (i = 0; i < full; i++)
        strcpy(buffer[i].a, buffer[i + 1].a);
    empty = signal(empty);
    mutex = signal(mutex);
    break;
case 3: /* Display */
    if (full == 0) {
        printf("\nBuffer is Empty\n");
    } else {
        printf("\nBuffer Contents:\n");
        for (i = 0; i < full; i++)
            printf("%s\n", buffer[i].a);
    }
    break;
case 4:
    exit(0);
default:
    printf("\nInvalid Choice\n");
}
}

/* Semaphore operations */
int wait(int s) {
    return --s;
}

int signal(int s) {
    return ++s;
}
```

OUTPUT

```
sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Beeba
sys1@CSE:~/Desktop/Beeba$ gcc prodcon.c -o prodcon
sys1@CSE:~/Desktop/Beeba$ ./prodcon

      Producer Consumer Problem

1. Producer
2. Consumer
3. Buffer Contents
4. Exit
Enter your choice: 1
Enter item to produce: Blue
Item Produced Successfully

1. Producer
2. Consumer
3. Buffer Contents
4. Exit
Enter your choice: 1
Enter item to produce: Green
Item Produced Successfully

1. Producer
2. Consumer
3. Buffer Contents
4. Exit
Enter your choice: 3

Buffer Contents:
Blue
Green

1. Producer
2. Consumer
3. Buffer Contents
4. Exit
Enter your choice: 2
Consumed item: Blue

1. Producer
2. Consumer
3. Buffer Contents
4. Exit
Enter your choice: 2
Consumed item: Green

1. Producer
2. Consumer
3. Buffer Contents
4. Exit
Enter your choice: 3

Buffer is Empty
```

RESULT:

Thus, the program of producer consumer problem using semaphores has been implemented and the output was verified successfully.

Ex No: 5 INTER PROCESS COMMUNICATION USING SHARED MEMORY Date :**AIM:**

To write a program to implement inter process communication using shared memory

ALGORITHM:

1. Create and shared memory using shmget.
2. Attach pointer to a memory using shmat.
3. Create child process to write data to memory.
4. Print the shared memory data using server process.
5. Stop the execution.

PROGRAM:

```
include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <unistd.h> // for fork()
#include <stdio.h>
#include <stdlib.h>

#define SHMSZ 27

int main() {
    char c;
    intshm, childpid;
    key_t key;
    char *shm, *s;

    key = 5679;

    // Create shared memory
    if ((shm = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);
    }

    // Attach shared memory
    if ((shm = shmat(shm, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }

    s = shm;

    // Create child process
    if ((childpid = fork()) < 0) {
        perror("Fork Failed");
        exit(1);
    }
    else if (childpid == 0) { // Child process
        for (c = 'a'; c <= 'z'; c++)
            *s++ = c;
```

```

*s = '\0'; // Null terminator

    exit(0); // Child exits }
else { // Parent process
    wait(NULL); // Wait for child to finish

    // Print content from shared memory
    for (s = shm; *s != '\0'; s++)
        putchar(*s);
    putchar('\n');

    // Detach and remove shared memory
    shmdt(shm);
    shmctl(shmid, IPC_RMID, NULL);
}

return 0;
}

```

OUTPUT

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc ipc.c -o ipc
sys1@CSE:~/Desktop/Reeba$ ./ipc
abcdefghijklmnopqrstuvwxyz
sys1@CSE:~/Desktop/Reeba$ ipcs

----- Message Queues -----
key          msqid      owner          perms         used-bytes   messages

----- Shared Memory Segments -----
key          shmid       owner          perms         bytes        nattch       status

----- Semaphore Arrays -----
key          semid       owner          perms         nsems

sys1@CSE:~/Desktop/Reeba$

```

RESULT:

Thus, the program for interprocess communication using shared memory been implemented and the output was verified successfully.

Ex.No:6A DEAD LOCK AVOIDANCE USING BANKERS ALGORITHM Date :

AIM: To implement deadlock avoidance by using Banker's Algorithm.

Data structures

- n-Number of process, m-number of resource types.
- Available: Available[j]=k, k – instance of resource type R_j is available.
- Max: If max[i, j]=k, P_i may request at most k instances resource R_j.
- Allocation: If Allocation [i, j]=k, P_i allocated to k instances of resource R_j.
- Need: If Need[I, j]=k, P_i may need k more instances of resource type R_j,
Need[I, j]=Max[I, j]-Allocation[I, j];

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both
 - Finish[i] =False
 - Need<=Work
 If no such I exists go to step 4.
3. work=work+Allocation, Finish[i] =True;
4. if Finish[1]=True for all I, then the system is in safe state.

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety or not if we allow the request.
9. Stop the program.

PROGRAM

```
#include <stdio.h>
int main() {
    int allocation[10][5], max[10][5], need[10][5], available[5], flag[10],
    sq[10]; int n, r, i, j, k, count, count1 = 0;
    printf("\nInput the number of processes running (<10): ");
    scanf("%d", &n);
    printf("\nInput the number of resources (<5): ");
    scanf("%d", &r);
    for (i = 0; i < 10; i++)
        flag[i] = 0;
    printf("\nInput the allocation matrix for the processes (row-wise):\n");
    for (i = 0; i < n; i++) {
        printf("\nProcess %d\n", i);
        for (j = 0; j < r; j++) {
            printf("Resource %d: ", j);
            scanf("%d", &allocation[i][j]);
        }
    }
    printf("\nInput the maximum matrix for the processes:\n");
    for (i = 0; i < n; i++) {
```

```

printf("\nProcess %d\n", i);
for (j = 0; j < r; j++) {
    printf("Resource %d: ", j);
    scanf("%d", &max[i][j]);
}
}
printf("\nInput the number of available instances of each resource:\n");
for (i = 0; i < r; i++) {
    printf("Resource %d: ", i);
    scanf("%d", &available[i]);
}
// Calculate need matrix
printf("\nThe need matrix is:\n");
for (i = 0; i < n; i++) {
    for (j = 0; j < r; j++) {
        need[i][j] = max[i][j] - allocation[i][j];
        printf("\t%d", need[i][j]);
    }
    printf("\n");
}
// Banker's Algorithm
do {
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            if (flag[i] == 0) {
                count = 0;
                for (j = 0; j < r; j++) {
                    if (available[j] >= need[i][j])
                        count++;
                }
                if (count == r) {
                    count1++;
                    flag[i] = 1;
                    sq[count1 - 1] = i;
                    for (j = 0; j < r; j++)
                        available[j] += allocation[i][j];
                    break;
                }
            }
        }
    }
    if (count1 != n && k == n - 1) {
        printf("\n-----IT'S AN UNSAFE STATE-----\n");
        return 0;
    }
} while (count1 != n);
printf("\n*****IT'S A SAFE STATE*****\n");
printf("The safe sequence is: ");
for (i = 0; i < n; i++)
    printf("P%d ", sq[i]);
printf("\nThe available resources after execution: ");
for (i = 0; i < r; i++)
    printf("%d ", available[i]);
printf("\n");

```

```

return 0;
}

```

OUTPUT

```

sysigCSE:~$ cd Desktop
sysigCSE:~/Desktop$ cd Recba
sysigCSE:~/Desktop/Recba$ gcc deadlock.c -o deadlock
sysigCSE:~/Desktop/Recba$ ./deadlock

Input the number of processes running (<10): 3
Input the number of resources (<5): 3
Input the allocation matrix for the processes (row-wise):

Process 0
Resource 0: 1
Resource 1: 2
Resource 2: 1

Process 1
Resource 0: 3
Resource 1: 2
Resource 2: 3

Process 2
Resource 0: 1
Resource 1: 2
Resource 2: 3

Input the maximum matrix for the processes:

Process 0
Resource 0: 2
Resource 1: 3
Resource 2: 2

Process 1
Resource 0: 4
Resource 1: 3
Resource 2: 4

Process 2
Resource 0: 2
Resource 1: 3
Resource 2: 4

Input the number of available instances of each resource:
Resource 0: 2
Resource 1: 1
Resource 2: 2

The need matrix is:
    1    1    1
    1    1    1
    1    1    1

*****IT'S A SAFE STATE*****
The safe sequence is: P0 P1 P2
The available resources after execution: 7 7 9
sysigCSE:~/Desktop/Recba$

```

RESULT

Thus, the program for deadlock avoidance by using Banker's Algorithm has been implemented and the output was verified successfully.

AIM: To implement deadlock detection by using Banker's Algorithm.

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the allocation and request matrix value.
4. Get the available instances for each resources
5. Check whether its possible to allocate using below condition.
6. If $Need \leq Work$ update work as $work = work + allocation$
7. Else proceed with next process.
8. If it is possible to allocate then the system is in safe state.
9. Else system is not in safe state.
10. Stop the program.

PROGRAM

```
#include <stdio.h>
int main() {
    int alloc[10][10], req[10][10], ins[10], avail[10];
    int tp, tr, i, j;
    int tmp[10] = {0}, finish[10] = {0};
    int done, flag;
    printf("Enter total no of processes:");
    scanf("%d", &tp);
    printf("Enter total no of resources:");
    scanf("%d", &tr);
    printf("\nAllocation Matrix:\n");
    for(i = 0; i < tp; i++) {
        for(j = 0; j < tr; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }
    printf("\nRequest Matrix:\n");
    for(i = 0; i < tp; i++) {
        for(j = 0; j < tr; j++) {
            scanf("%d", &req[i][j]);
        }
    }
    printf("\nResource Instance Vector:\n");
    for(i = 0; i < tr; i++) {
        scanf("%d", &ins[i]);
    }
    /* Calculate Availability Vector */
    for(j = 0; j < tr; j++) {
        tmp[j] = 0;
        for(i = 0; i < tp; i++) {
            tmp[j] += alloc[i][j];
        }
        avail[j] = ins[j] - tmp[j];
    }
    printf("\nCalculated Availability Vector:\n");
```

```

for(j = 0; j < tr; j++) {
    printf("%d ", avail[j]);
}
/* Deadlock Detection */
done = 0;
while(done == 0) {
    flag = 0;
    for(i = 0; i < tp; i++) {
        if(finish[i] == 0) {
            int canFinish = 1;
            for(j = 0; j < tr; j++) {
                if(req[i][j] > avail[j]) {
                    canFinish = 0;
                    break; } }
            if(canFinish) {
                printf("\nProcess %d is transferred to running state and assumed finished", i + 1);
                for(j = 0; j < tr; j++) {
                    avail[j] += alloc[i][j];
                }
                finish[i] = 1;
                flag = 1; } } }
    if(flag == 0)
        break; }
/* Deadlock Check */
int deadlock = 0;
for(i = 0; i < tp; i++) {
    if(finish[i] == 0) {
        printf("\nOops! Deadlock detected. Causing process: %d", i + 1);
        deadlock = 1;
    } }
if(!deadlock) {
    printf("\n\nHurray! Deadlock not detected :-)");
}
return 0; }

```

OUTPUT

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc banker.c -o banker
sys1@CSE:~/Desktop/Reeba$ ./banker
Enter total no of processes:3
Enter total no of resources:3

Allocation Matrix:
1 2 1
3 2 3
1 2 3

Request Matrix:
1 0 1
0 0 0
1 0 1

Resource Instance Vector:
10 8 10

Calculated Availability Vector:
5 2 3
Process 1 is transferred to running state and assumed finished
Process 2 is transferred to running state and assumed finished
Process 3 is transferred to running state and assumed finished

```

RESULT

Thus, the program for deadlock detection by using Banker's Algorithm has been implemented and the output was verified successfully.

AIM: To implement memory allocation methods first fit, best fit & worst fit for fixed partitions.

FIRST FIT ALGORITHM:

1. Get no. of Processes and no. of blocks.
2. After that get the size of each block and process requests.
3. Now allocate processes
if(block size $\geq$ process size)
//allocate the process
else
//move on to next block
4. Display the processes with the blocks that are allocated to a respective process.
5. Stop.

PROGRAM

```
#include <stdio.h>
#define MAX 10
int main()
{
    int bsize[MAX], psize[MAX], bno, pno;
    int flags[MAX], allocation[MAX];
    int i, j;
    for(i = 0; i < MAX; i++)
    {
        flags[i] = 0;
        allocation[i] = -1;
    }
    printf("Enter no. of blocks: ");
    scanf("%d", &bno);
    printf("Enter size of each block:\n");
    for(i = 0; i < bno; i++)
        scanf("%d", &bsize[i]);
    printf("Enter no. of processes: ");
    scanf("%d", &pno);
    printf("Enter size of each process:\n");
    for(i = 0; i < pno; i++)
        scanf("%d", &psize[i]);
    // First Fit Allocation
    for(i = 0; i < pno; i++)
    {
        for(j = 0; j < bno; j++)
        {
            if(flags[j] == 0 && bsize[j]  $\geq$  psize[i])
            {
                allocation[j] = i;
                flags[j] = 1;
                break; } } }
    printf("\nBlock No\tBlock Size\tProcess No\tProcess Size");
    for(i = 0; i < bno; i++)
```

```

{
    printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
    if(flags[i] == 1)
        printf("%d\t\t\t%d", allocation[i]+1, psize[allocation[i]]);
    else
        printf("Not Allocated");
}
return 0;
}

```

OUTPUT

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc firstfit.c -o firstfit
sys1@CSE:~/Desktop/Reeba$ ./firstfit
Enter no. of blocks: 4
Enter size of each block:
20
30
40
50
Enter no. of processes: 4
Enter size of each process:
10
20
15
25

Block No      Block Size    Process No    Process Size
1             20           1             10
2             30           2             20
3             40           3             15
sys1@CSE:~/Desktop/Reeba$

```

BEST FIT ALGORITHM

1. Get no. of Processes and no. of blocks.
2. After that get the size of each block and process requests.
3. Then select the best memory block that can be allocated using the above definition.
4. Display the processes with the blocks that are allocated to a respective process.
5. Value of Fragmentation is optional to display to keep track of wasted memory.
6. Stop.

PROGRAM

```

#include<stdio.h>
void main()
{
    int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
    static int barray[20],parray[20];
    printf("\n\t\t\tMemory Management Scheme - Best Fit");
}

```

```

printf("\nEnter the number of blocks:");
scanf("%d",&nb);

printf("Enter the number of processes:");
scanf("%d",&np);

printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
    printf("Block no.%d:",i);
    scanf("%d",&b[i]);
}
printf("\nEnter the size of the processes :-\n");
for(i=1;i<=np;i++)
{
    printf("Process no.%d:",i);
    scanf("%d",&p[i]);
}
for(i=1;i<=np;i++)
{
    for(j=1;j<=nb;j++)
    {
        if(barray[j]!=1)
        {
            temp=b[j]-p[i];
            if(temp>=0)
            if(lowest>temp)
            {
                parray[i]=j;
                lowest=temp;
            }
        }
    }

    fragment[i]=lowest;
    barray[parray[i]]=1;
    lowest=10000;
}

printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment"
); for(i=1;i<=np && parray[i]!=0;i++)
    printf("\n%d\t%d\t%d\t%d\t%d",i,p[i],parray[i],b[parray[i]],fragment[i
]); }

```

OUTPUT

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc bestfit.c -o bestfit
sys1@CSE:~/Desktop/Reeba$ ./bestfit

                          Memory Management Scheme - Best Fit
Enter the number of blocks:3
Enter the number of processes:3

Enter the size of the blocks:-
Block no.1:30
Block no.2:20
Block no.3:40

Enter the size of the processes :-
Process no.1:15
Process no.2:25
Process no.3:35



| Process_no | Process_size | Block_no | Block_size | Fragment |
|------------|--------------|----------|------------|----------|
| 1          | 15           | 2        | 20         | 5        |
| 2          | 25           | 1        | 30         | 5        |


sys1@CSE:~/Desktop/Reeba$

```

WORST FIT ALGORITHM

1. Start
2. Input memory blocks and processes with sizes.
3. Initialize all memory blocks as free.
4. Arrange the partition in descending order.
5. Start picking each process and allocate to block only if process memory size is less than block size.
6. Else leave that process and keep checking other process.
7. Allocate process to blocks and calculate the fragment in each block.
8. Leave process unallocated if not satisfying above conditions.
9. Stop

PROGRAM

```

#include <stdio.h>

int main()
{
    int i, j, temp;
    int f[10], fp[10];
    int no, p[10], part[10];
    int pno, pr[10], prmem[10];

    printf("\n*****");

    printf("\n IMPLEMENTATION OF WORST-FIT ALGORITHM");
    printf("\n*****");
}

```

```
printf("\n Enter the number of partitions");
scanf("%d", &no);

for(i = 1; i <= no; i++)
{
    p[i] = i;
    printf("Enter the memory for partition %d:\t", i);
    scanf("%d", &part[i]);
}

/* Sort partitions in descending order */
for(i = 1; i <= no; i++)
{
    for(j = i + 1; j <= no; j++)
    {
        if(part[i] < part[j])
        {
            temp = part[i];
            part[i] = part[j];
            part[j] = temp;

            temp = p[i];
            p[i] = p[j];
            p[j] = temp;
        }
    }
}

printf("\nFree memory");
for(i = 1; i <= no; i++)
{
    printf("\n partition %d:\t %d", p[i], part[i]);
    f[i] = 0;
}

printf("\n Enter the number of process");
scanf("%d", &pno);

for(i = 1; i <= pno; i++)
{
    pr[i] = i;
    printf("Enter the size for process %d:\t", i);
    scanf("%d", &prmem[i]);
    fp[i] = 0;
}

printf("\n-");
printf("\nPROCESS\t PARTITION\t FREE_MEMORY");
printf("\n-");

j = 1;

for(i = 1; i <= no && j <= pno; i++)
{
```

```

        if(part[i] >= prmem[j])

        {
            part[i] = part[i] - prmem[j];

            f[i] = 1;
            fp[j] = 1;

            printf("%d \t %d \t %d\n", pr[j], p[i], part[i]);
            j++;
        }
    }

    for(i = 1; i <= no; i++)
    {
        if(f[i] == 0)
        {
            printf(" \t %d \t %d\n", p[i], part[i]);
        }
    }

    printf("The following process is not allocated:");
    for(i = 1; i <= pno; i++)
    {
        if(fp[i] == 0)
            printf(" %d", pr[i]);
    }
    return 0;
}

```

OUTPUT

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc worstfit.c -o worstfit
sys1@CSE:~/Desktop/Reeba$ ./worstfit

*****
IMPLEMENTATION OF WORST-FIT ALGORITHM
*****
Enter the number of partitions4
Enter the memory for partition 1:          35
Enter the memory for partition 2:          25
Enter the memory for partition 3:          50
Enter the memory for partition 4:          60

Free memory
partition 4:          60
partition 3:          50
partition 1:          35
partition 2:          25
Enter the number of process4
Enter the size for process 1:          25
Enter the size for process 2:          15
Enter the size for process 3:          20
Enter the size for process 4:          30

-----
PROCESS|          PARTITION|          FREE_MEMORY|
-----
1         4             35
2         3             35
3         1             15
          2             25

```

RESULT:

Thus, the program for memory allocation methods has been implemented and the output was verified successfully.

AIM

To write a c program to implement FIFO page replacement algorithm

ALGORITHM

1. Start the process
2. Declare the size with respect to page length
3. Check the need of replacement from the page to memory
4. Check the need of replacement from old page to new page in memory
5. Form a queue to hold all pages
6. Insert the page require memory into the queue
7. Check for bad replacement and page fault
8. Get the number of processes to be inserted
9. Display the values
10. Stop the process

PROGRAM:

```
#include<stdio.h>
int main()
{
    int i,j,n,a[50],frame[10],no,k,avail,count=0;
    printf("\n ENTER THE NUMBER OF PAGES:\n");
    scanf("%d",&n);
    printf("\n ENTER THE PAGE NUMBER :\n");
    for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
    printf("\n ENTER THE NUMBER OF FRAMES :");
    scanf("%d",&no);
    for(i=0;i<no;i++)
        frame[i]= -1;
    j=0;
    printf("\trefstring\t page frames\n");
    for(i=1;i<=n;i++)
    {
        printf("%d\t\t",a[i]);
        avail=0;
        for(k=0;k<no;k++)
            if(frame[k]==a[i])
                avail=1;
        if (avail==0)
        {
            frame[j]=a[i];
            j=(j+1)%no;
            count++;
            for(k=0;k<no;k++)
                printf("%d\t",frame[k]);
        }

        printf("\n"); }
}
```

```
printf("Page Fault Is %d",count);  
return 0; }
```

OUTPUT

```
sys1@CSE:~$ cd Desktop  
sys1@CSE:~/Desktop$ cd Reeba  
sys1@CSE:~/Desktop/Reeba$ gcc fifo.c -o fifo  
sys1@CSE:~/Desktop/Reeba$ ./fifo  
  
ENTER THE NUMBER OF PAGES:  
8  
  
ENTER THE PAGE NUMBER :  
1 3 2 1 4 5 3 2  
  
ENTER THE NUMBER OF FRAMES :3  
    ref string      page frames  
1          1        -1        -1  
3          1         3        -1  
2          1         3         2  
1  
4          4         3         2  
5          4         5         2  
3          4         5         3  
2          2         5         3  
Page Fault Is 7sys1@CSE:~/Desktop/Reeba$
```

RESULT

Thus, the program for FIFO page replacement algorithm has been implemented and the output was verified successfully.

AIM:

To implement LRU page replacement technique.

ALGORITHM:

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

PROGRAM

```
#include<stdio.h>
int main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];

printf("Enter no of pages:");
scanf("%d",&n);
printf("Enter the reference string:");
for(i=0;i<n;i++)
scanf("%d",&p[i]);
printf("Enter no of frames:");
scanf("%d",&f);
q[k]=p[k];
printf("\n\t%d\n",q[k]);
c++;
k++;
for(i=1;i<n;i++)
{
    c1=0;
    for(j=0;j<f;j++)
    {
        if(p[i]!=q[j])
            c1++;
    }
    if(c1==f)
    {
        c++;
        if(k<f) {
        }
        else
        {
        }
        q[k]=p[i];
    }
}
```

```

k++;
for(j=0;j<k;j++)
)
printf("\t%d",q[j]);
printf("\n"); for(r=0;r<f;r++)
{ c2[r]=0;
if(q[r]!=p[j]) c2[r]++;
else
    for(j=i-1;j<n;j- break;
    -)          }
    {
    for(r=0;r<f;r++)
    b[r]=c2[r];
    for(r=0;r<f;r++)
    {
        for(j=r;j<f;j++)
        {
            if(b[r]<b[j])
            {
                t=b[r];
                b[r]=b[j];
                b[j]=t;
            }
        }
    }
    for(r=0;r<f;r++)
    {
        if(c2[r]==b[0])
        q[r]=p[i];
        printf("\t%d",q[r]);
    }
    printf("\n");
} } }

printf("\nThe no of page faults is %d",c);

return 0;
}

```

OUTPUT

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc lru.c -o lru
sys1@CSE:~/Desktop/Reeba$ ./lru
Enter no of pages:8
Enter the reference string:2 3 4 3 2 4 1 5
Enter no of frames:2

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

The no of page faults is 7sys1@CSE:~/Desktop/Reeba$

```

RESULT

Thus, the program for LRU page replacement algorithm has been implemented and the output was verified successfully.

Ex No:8C**OPTIMAL PAGE REPLACEMENT ALGORITHM****Date:****AIM:**

To implement optimal page replacement technique.

ALGORITHM:

Here we select the page that will not be used for the longest period of time. 1. Start Program

2. Read Number Of Pages And Frames

3. Read Each Page Value

4. Search For Page In The Frames

5. If Not Available Allocate Free Frame

6. If No Frames Is Free Repalce The Page With The Page That Is Not Used In Next N Pages(N Is Number Of Frames)

7. Print Page Number Of Page Faults

8. Stop process.

PROGRAM

```
#include<stdio.h>
int findmax(int*);
int n;
int main()
{
intseq[30],fr[5],pos[5],find,flag,max,i,j,m,k,t,s,pf=0;
int count=1,p=0;
float pfr;
printf("enter max limit of the sequence:");
scanf("%d",&max);
printf("enter the sequence:");
for(i=0;i<max;i++)
scanf("%d",&seq[i]);
printf("enter the no of frames:");
scanf("%d",&n);
fr[0]=seq[0];
pf++;
printf("%d\t",fr[0]);
i=1;
while(count<n)
{
flag=1;
p++;
for(j=0;j<i;j++)
{
if(seq[i]==seq[j])
flag=0;

```

```

    }
    if(flag!=0)
    {
        fr[count]=seq[i];
        printf("%d\t",fr[count] );
        count++;
        pf++;
    }
    i++;}
    printf("\n");
    for(i=p;i<max;i++)
    {
        flag=1;
        for(j=0;j<n;j++)
        {
            if(seq[i]==fr[j])
            flag=0;
        }
        if(flag!=0)
        {
            for(j=0;j<n;j++)
            {
                m=fr[j];
                for(k=i;k<max;k++)
                ) {
                    if(seq[k]==m)
                    {
                        pos[j]=k;
                        break;}
                    else
                        pos[j]=-1;
                }
            }
            for(k=0;k<n;k++)
            {
                if(pos[k]==-1)
                flag=0;
            }
            if(flag!=0)
            s=findmax(pos);
            if(flag==0)
            {
                for(k=0;k<n;k++)
                {
                    if(pos[k]==-1)
                    {

                        s=k; break;          }}}
            pf++;
            fr[s]=seq[i];

```

```

for(k=0;k<n;k++)
printf("%d\t",fr[k]);
printf("\n");
}}
pfr=(float)pf/(float)max;
printf("\n theno of page faults are:%d",pf);
printf("\n page fault rate:%f",pfr); }
int findmax(int a[])
{
int max,i,k=0;
max=a[0];
for(i=0;i<n;i++)
{
if(max<a[i])
{
max=a[i];
k=i;
}}
return k;
}

```

OUTPUT

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc optimal.c -o optimal
sys1@CSE:~/Desktop/Reeba$ ./optimal
enter max limit of the sequence:15
enter the sequence:1 3 5 8 4 2 0 6 9 7 5 8 4 2 1
enter the no of frames:4
1      3      5      8
1      4      5      8
2      4      5      8
0      4      5      8
6      4      5      8
9      4      5      8
7      4      5      8
2      4      5      8
1      4      5      8

theno of page faults are:12
page fault rate:0.800000sys1@CSE:~/Desktop/Reeba$

```

RESULT

Thus, the program for Optimal page replacement algorithm has been implemented and the output was verified successfully.

AIM:

To write a C program to implement contiguous file allocation techniques.

ALGORITHM:

Step 1: Start.

Step 2: Memory before allocation is displayed.

Step 3: The file Id is entered by the user.

Step 4: the number of blocks memory required and the starting address of the file is given by the user..

Step 5: If starting address and length of block is greater than 20 print error message. Else , the memory address is replaced by the id.

Display memory after allocation.

Step 6: Stop.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
int a[20],num=0;
int fid[10],length[10],start[10];
void filedescrptor();
void display();
void filedescrptor()
{
    int i;
    printf("\n file id \t starting address \t length \n");
    for(i=0;i<num;i++)
        printf("%d\t\t %d\t\t %d\n",fid[i],start[i],length[i]);
}
void display()
{
    int i;
    for(i=0;i<20;i++)
        printf("%2d",i);
    printf("\n");
    for(i=0;i<20;i++)
        printf("%2d", a[i]);
}
void main()
{
    int i,n,k,temp,st,l,id,flag=0,cho;
    for(i=0;i<20;i++)
        a[i]=0;
    printf("\n memory before allocation:\n");
    display();
    while(1)
    {
        printf("\n enter the file id:");
        scanf("%d",&id);
        printf("\n enter the number of blocks the file occupies:");
        scanf("%d", &l);
        fid[num]=id;
        length[num]=l;

        l:printf("\n enter the starting address:");
```

```

scanf("%d", &st);
flag=0;
if((st+1)>20)
{
printf("\n sorry the given memory goes out of space:");
goto l; }
for(i=st;i<(st+1);i++)
{
if(a[i]!=0)
{
flag=1;
break;
}}
if(flag==0)
{
start[num]=st;
for(i=st;i<(st+1);i++)
a[i]=id;
}
else
{
printf("\n sorry the given blocks are already occupied: Enter new starting address");
goto l;
}
flag=0;
num++;
filedescriptor();
printf("\n memory after allocation \n");
display();
printf("\n want to continue? \n1.yes \n2.no");
scanf("%d",&cho);
if(cho==2)
exit(0);
}}

```

OUTPUT:

```

sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc contiguous.c -o contiguous
sys1@CSE:~/Desktop/Reeba$ ./contiguous

memory before allocation:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
enter the file id:15

enter the number of blocks the file occupies:5
enter the starting address:8

file id      starting address      length
15           8                   5

memory after allocation
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 0 0 0 0 0 0 0 15 0 0 0 0 0 0 0 0 0 0 0
want to continue?
1.yes
2.no

```

RESULT

Thus, the program for contiguous file allocations strategies has been implemented and the output was verified successfully.

AIM:

To write a C program to implementation linked list file allocation techniques.

ALGORITHM:

1. Start.
2. Get the number of blocks already allocated and its block numbers as input from user. 3. Add that block numbers to a list as allocated block number list and mark that block numbers are allocated.
4. To perform allocation .Get the starting block number and length of the input file 5. From the starting block check whether each block is in allocated block number list 6. If so Print block is already allocated. Else allocate file to the block number.
7. Continue the loop if the processes want to be repeated.
8. Stop.

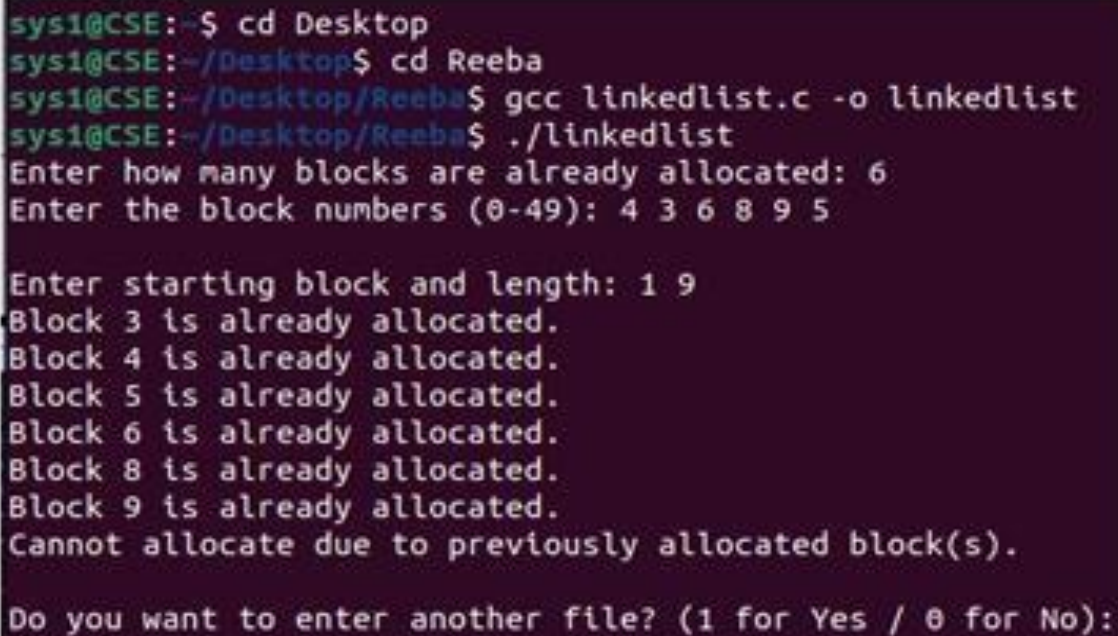
PROGRAM

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

int main()
{
    int f[50], i, j, p, a, st, len, c;
    int flag;

    // Initialize all blocks as free
    for(i = 0; i < 50; i++)
        f[i] = 0;
    printf("Enter how many blocks are already allocated: ");
    scanf("%d", &p);
    printf("Enter the block numbers (0-49): ");
    for(i = 0; i < p; i++)
    {
        scanf("%d", &a);
        f[a] = 1;
    }
    do
    {
        printf("\nEnter starting block and length: ");
        scanf("%d %d", &st, &len);
        flag = 0;
        for(i = st; i < st + len; i++)
        {
            if(f[i] == 1)
            {
                printf("Block %d is already allocated.\n", i);
                flag = 1;
            }
        }
    }
    if(flag == 0)
    {
        printf("File allocated at blocks:\n");
        for(i = st; i < st + len; i++)
        {
```

```
f[i] = 1;
printf("%d -> %d\n", i, f[i]);
}
}
else
{
    printf("Cannot allocate due to previously allocated block(s).\n");
}
printf("\nDo you want to enter another file? (1 for Yes / 0 for No): ");
scanf("%d", &c);
} while(c == 1);
return 0;
}
```

OUTPUT:

```
sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc linkedlist.c -o linkedlist
sys1@CSE:~/Desktop/Reeba$ ./linkedlist
Enter how many blocks are already allocated: 6
Enter the block numbers (0-49): 4 3 6 8 9 5

Enter starting block and length: 1 9
Block 3 is already allocated.
Block 4 is already allocated.
Block 5 is already allocated.
Block 6 is already allocated.
Block 8 is already allocated.
Block 9 is already allocated.
Cannot allocate due to previously allocated block(s).

Do you want to enter another file? (1 for Yes / 0 for No):
```

RESULT

Thus, the program for Linked list file allocations strategies has been implemented and the output was verified successfully

AIM:

To write a C program to implementation indexed file allocation techniques.

ALGORITHM:

1. Start.
2. Get the index block number and number of files in the index block as input from user.
3. Get the file numbers (i.e referred block numbers holding file) as input .
4. Check whether that the input block number is already allocated if so print block allocated a. Else increase the count and allocate the file.
5. Continue the loop to enter another index block.
6. Stop

PROGRAM

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

```
int main()
```

```
{
```

```
    int f[50], index[50];
```

```
    int indblk, n, i, j, c;
```

```
    int count;
```

```
    // Initialize disk blocks
```

```
    for(i = 0; i < 50; i++)
```

```
        f[i] = 0;
```

```
    do
```

```
    {
```

```
        count = 0;
```

```
        printf("Enter index block (0-49): ");
```

```
        scanf("%d", &indblk);
```

```
        if(f[indblk] == 1)
```

```
        {
```

```
            printf("Index block already allocated\n");
```

```
        }
```

```
        else
```

```
        {
```

```
            printf("Enter number of files on index: ");
```

```
            scanf("%d", &n);
```

```
            printf("Enter file blocks (0-49): ");
```

```
            for(i = 0; i < n; i++)
```

```
            {
```

```
                scanf("%d", &index[i]);
```

```
                if(f[index[i]] == 0)
```

```
                    count++;
```

```
            }
```

```
            if(count == n)
```

```
            {
```

```
                printf("\nAllocated");
```

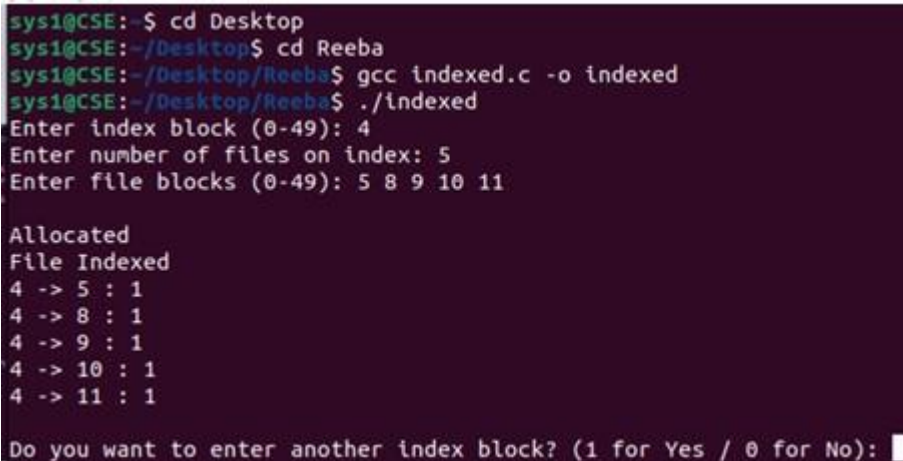
```
printf("\nFile Indexed\n");

f[indblk] = 1;
for(j = 0; j < n; j++)
{
    f[index[j]] = 1;
    printf("%d -> %d : %d\n", indblk, index[j], f[index[j]]);
}
}
else
{
    printf("\nFile block already allocated\n");
}
}

printf("\nDo you want to enter another index block? (1 for Yes / 0 for No): ");
scanf("%d", &c);

} while(c == 1);

return 0;
}
```

OUTPUT:

```
sys1@CSE:~$ cd Desktop
sys1@CSE:~/Desktop$ cd Reeba
sys1@CSE:~/Desktop/Reeba$ gcc indexed.c -o indexed
sys1@CSE:~/Desktop/Reeba$ ./indexed
Enter index block (0-49): 4
Enter number of files on index: 5
Enter file blocks (0-49): 5 8 9 10 11

Allocated
File Indexed
4 -> 5 : 1
4 -> 8 : 1
4 -> 9 : 1
4 -> 10 : 1
4 -> 11 : 1

Do you want to enter another index block? (1 for Yes / 0 for No):
```

RESULT:

Thus, the program for Indexed file allocations strategies has been implemented and the output was verified successfully

AIM

Write a C program to simulate disk scheduling algorithms. a) FCFS b) SCAN c) C-SCAN

a.)FCFS ALGORITHM

- **Start the program.**
- **Input** the number of processes n.
- **Input** the burst time (BT) for each process.
- Assume the processes arrive in the order they are entered (FCFS order).
- Set the waiting time of the first process:

$WT[0] = 0$

- Calculate the waiting time for the remaining processes:

$WT[i] = WT[i-1] + BT[i-1]$

- Calculate the turnaround time for each process:

$TAT[i] = WT[i] + BT[i]$

- Calculate the average waiting time:

Average WT = (Sum of all WT) / n

- Calculate the average turnaround time:

Average TAT = (Sum of all TAT) / n

- Display the process number, burst time, waiting time, and turnaround time.
- Display the average waiting time and average turnaround time.
- **Stop** the program.

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int t[20], n, i, tohm[20], tot = 0;
    float avhm;

    printf("Enter the number of tracks: ");
    scanf("%d", &n);

    printf("Enter the initial head position: ");
    scanf("%d", &t[0]);

    printf("Enter the tracks to be traversed:\n");
    for(i = 1; i <= n; i++)
        scanf("%d", &t[i]);

    for(i = 0; i < n; i++)
    {
        tohm[i] = abs(t[i + 1] - t[i]);
        tot += tohm[i];
    }
```

```

avhm = (float)tot / n;

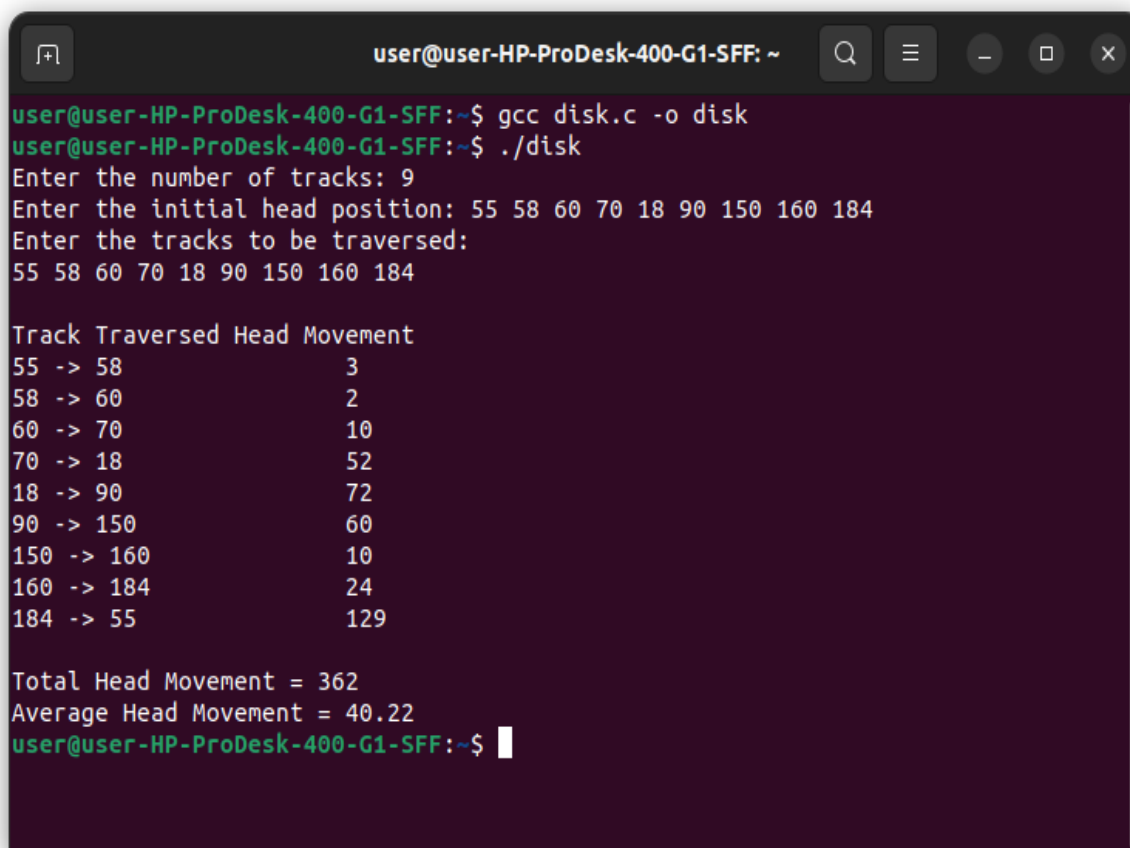
printf("\nTrack Traversed\tHead Movement\n");
for(i = 0; i < n; i++)
    printf("%d -> %d\t\t%d\n", t[i], t[i + 1], tohm[i]);

printf("\nTotal Head Movement = %d", tot);
printf("\nAverage Head Movement = %.2f\n", avhm);

return 0;
}

```

OUTPUT



```

user@user-HP-ProDesk-400-G1-SFF: ~
user@user-HP-ProDesk-400-G1-SFF:~$ gcc disk.c -o disk
user@user-HP-ProDesk-400-G1-SFF:~$ ./disk
Enter the number of tracks: 9
Enter the initial head position: 55 58 60 70 18 90 150 160 184
Enter the tracks to be traversed:
55 58 60 70 18 90 150 160 184

Track Traversed Head Movement
55 -> 58          3
58 -> 60          2
60 -> 70         10
70 -> 18         52
18 -> 90         72
90 -> 150        60
150 -> 160       10
160 -> 184       24
184 -> 55       129

Total Head Movement = 362
Average Head Movement = 40.22
user@user-HP-ProDesk-400-G1-SFF:~$

```

b.)SCAN

SCHEDULING ALGORITHM

Algorithm

1. Read the following inputs:
 - Number of disk requests (n)
 - Request queue
 - Initial head position (head)
 - Disk size (maximum track number)
 - Initial direction (left or right)

2. Sort the request queue in ascending order.
3. Divide the requests into two groups:
 - Requests less than the current head position (left)
 - Requests greater than the current head position (right)
4. Add the end track of the disk to the side toward which the head is initially moving:
 - If moving left, add track 0.
 - If moving right, add the maximum track number.
5. Move the disk head in the chosen direction:
 - Service all requests encountered.
 - Calculate head movement for each service.
6. When the end track is reached:
 - Reverse the direction.
 - Continue servicing the remaining requests.
7. Compute:
 - Total head movement
 - Average head movement (if required)
8. Display the seek sequence and total head movement.

PROGRAM

```
#include <stdio.h>
int main()
{
    int i, j, n;
    int d[20];
    int disk;      // Head position
    int temp, max;
    int dloc = -1;
    int sum;

    printf("Enter number of disk requests: ");
    scanf("%d", &n);

    printf("Enter initial head position: ");
    scanf("%d", &disk);

    printf("Enter disk queue elements:\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &d[i]);
    }

    /* Insert head position into array */
    d[n] = disk;
    n++;

    /* Sort disk queue */
    for (i = 0; i < n - 1; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (d[i] > d[j])
```

```
{
    temp = d[i];
    d[i] = d[j];
    d[j] = temp;
}
}

max = d[n - 1];

/* Find head position in sorted array */
for (i = 0; i < n; i++)
{
    if (d[i] == disk)
    {
        dloc = i;
        break;
    }
}

printf("\nSCAN Service Order:\n");

/* Move towards 0 */
for (i = dloc; i >= 0; i--)
{
    printf("%d", d[i]);

    if (i > 0)
        printf(" --> ");
}

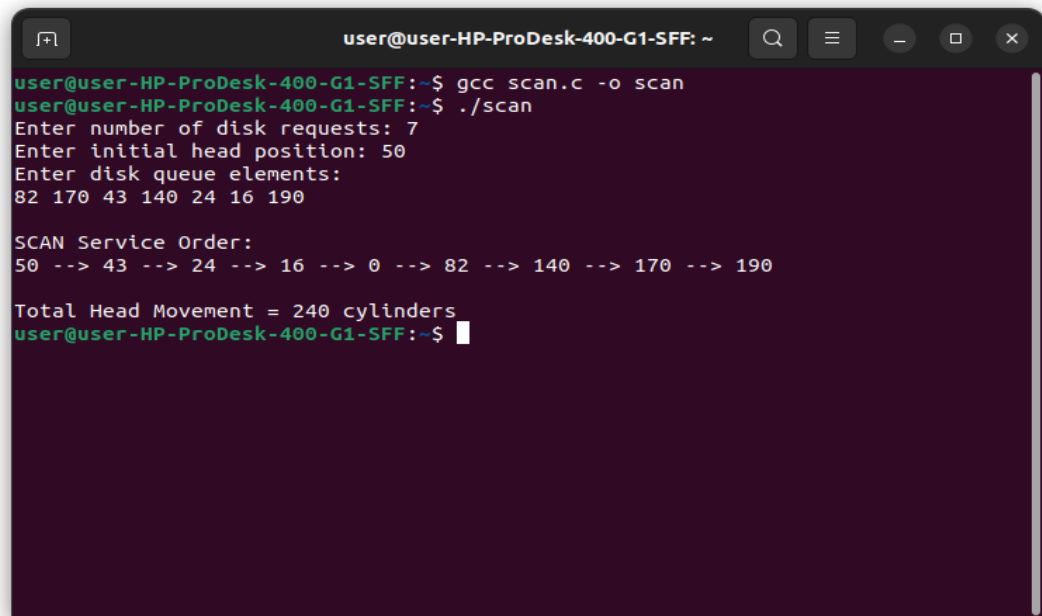
printf(" --> 0");

/* Reverse direction */
for (i = dloc + 1; i < n; i++)
{
    printf(" --> %d", d[i]);
}

/* Total head movement */
sum = disk + max;

printf("\n\nTotal Head Movement = %d cylinders\n", sum);

return 0;
}
```



```
user@user-HP-ProDesk-400-G1-SFF: ~  
user@user-HP-ProDesk-400-G1-SFF:~$ gcc scan.c -o scan  
user@user-HP-ProDesk-400-G1-SFF:~$ ./scan  
Enter number of disk requests: 7  
Enter initial head position: 50  
Enter disk queue elements:  
82 170 43 140 24 16 190  
  
SCAN Service Order:  
50 --> 43 --> 24 --> 16 --> 0 --> 82 --> 140 --> 170 --> 190  
  
Total Head Movement = 240 cylinders  
user@user-HP-ProDesk-400-G1-SFF:~$
```

C.)C-SCAN SCHEDULING ALGORITHM

ALGORITHM:

Step 1: Start.

Step 2: Read the number of disk requests n.

Step 3: Read the disk request queue and the initial head position.

Step 4: Sort all disk requests in ascending order.

Step 5: Find the position of the head in the sorted queue.

Step 6: Service all requests with cylinder numbers greater than the current head position, moving toward the higher end of the disk.

Step 7: Continue moving the head to the maximum cylinder number (disk end).

Step 8: Jump from the maximum cylinder to cylinder 0 without servicing any requests during the jump.

Step 9: Continue servicing the remaining requests from the beginning of the disk up to the requests that are less than the initial head position.

Step 10: Calculate the total head movement:

- Movement from current head position to the last cylinder.
- Jump from last cylinder to first cylinder.
- Movement from first cylinder to the last pending request.

Step 11: Display the order of service and total head movement.

Step 12: Stop.

PROGRAM:

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

```
#include <stdlib.h>
```

```
int main()
```

```
{
```

```
    int queue[50], queue1[50], queue2[50];
```

```
    int head, max, n;
```

```
    int i, j, temp;
```

```
    int temp1 = 0, temp2 = 0;
```

```
int seek = 0, diff;
float avg;

printf("Enter maximum disk range: ");
scanf("%d", &max);

printf("Enter initial head position: ");
scanf("%d", &head);

printf("Enter number of requests: ");
scanf("%d", &n);

printf("Enter disk queue requests:\n");
for(i = 0; i < n; i++)
{
    scanf("%d", &temp);

    if(temp >= head)
        queue1[temp1++] = temp;
    else
        queue2[temp2++] = temp;
}

/* Sort queue1 */
for(i = 0; i < temp1 - 1; i++)
{
    for(j = i + 1; j < temp1; j++)
    {
        if(queue1[i] > queue1[j])
        {
            temp = queue1[i];
            queue1[i] = queue1[j];
            queue1[j] = temp;
        }
    }
}

/* Sort queue2 */
for(i = 0; i < temp2 - 1; i++)
{
    for(j = i + 1; j < temp2; j++)
    {
        if(queue2[i] > queue2[j])
        {
            temp = queue2[i];
            queue2[i] = queue2[j];
            queue2[j] = temp;
        }
    }
}

int k = 0;

queue[k++] = head;
```

```
for(i = 0; i < temp1; i++)
    queue[k++] = queue1[i];

queue[k++] = max;

queue[k++] = 0;

for(i = 0; i < temp2; i++)
    queue[k++] = queue2[i];

printf("\nC-SCAN Disk Scheduling Order:\n");

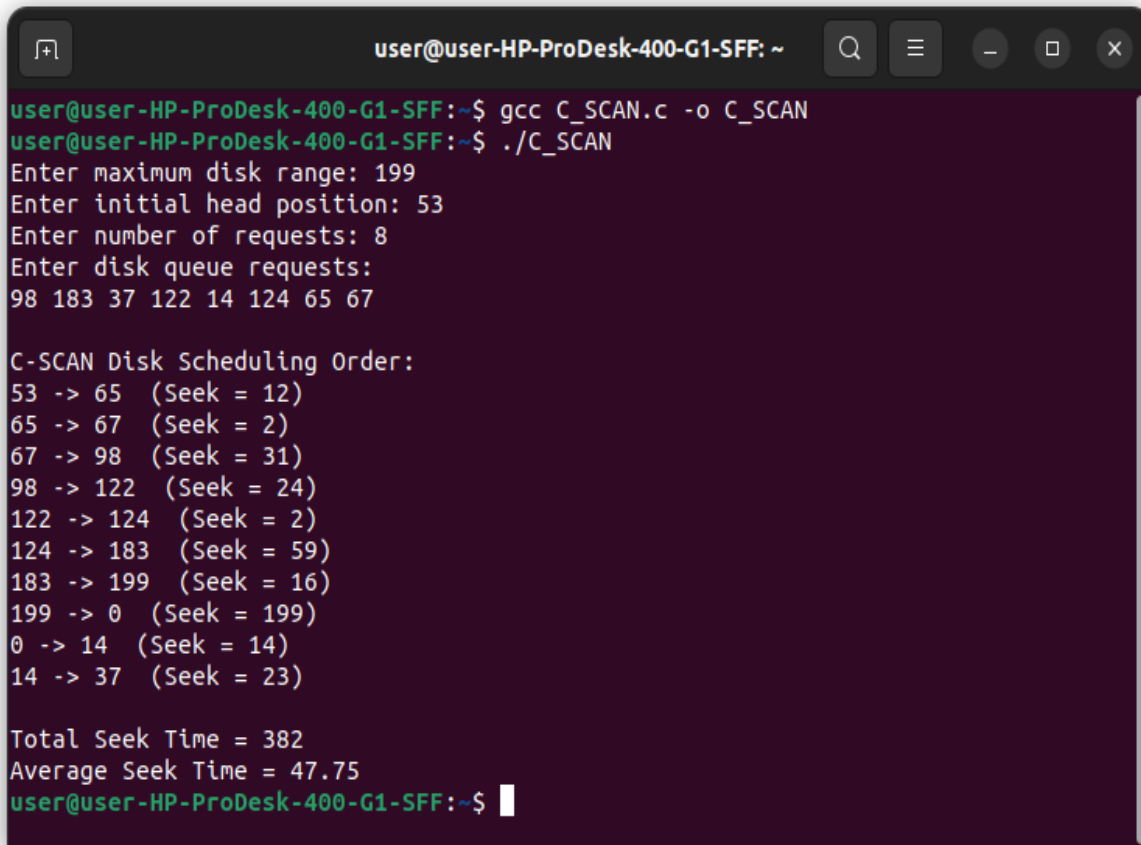
for(i = 0; i < k - 1; i++)
{
    diff = abs(queue[i + 1] - queue[i]);
    seek += diff;

    printf("%d -> %d (Seek = %d)\n",
           queue[i], queue[i + 1], diff);
}

printf("\nTotal Seek Time = %d\n", seek);

avg = (float)seek / n;
printf("Average Seek Time = %.2f\n", avg);

return 0;
}
```



```
user@user-HP-ProDesk-400-G1-SFF: ~  
user@user-HP-ProDesk-400-G1-SFF:~$ gcc C_SCAN.c -o C_SCAN  
user@user-HP-ProDesk-400-G1-SFF:~$ ./C_SCAN  
Enter maximum disk range: 199  
Enter initial head position: 53  
Enter number of requests: 8  
Enter disk queue requests:  
98 183 37 122 14 124 65 67  
  
C-SCAN Disk Scheduling Order:  
53 -> 65 (Seek = 12)  
65 -> 67 (Seek = 2)  
67 -> 98 (Seek = 31)  
98 -> 122 (Seek = 24)  
122 -> 124 (Seek = 2)  
124 -> 183 (Seek = 59)  
183 -> 199 (Seek = 16)  
199 -> 0 (Seek = 199)  
0 -> 14 (Seek = 14)  
14 -> 37 (Seek = 23)  
  
Total Seek Time = 382  
Average Seek Time = 47.75  
user@user-HP-ProDesk-400-G1-SFF:~$
```

RESULT:

Thus, the program to simulate disk scheduling algorithms. a) FCFS b) SCAN c) C-SCAN have been implemented and the output was verified successfully

VIVA QUESTIONS

1. Define Artificial Intelligence.
2. What is meant by a Turing Test?
3. What are software agents?
4. What is a rational agent?
5. Why are condition-action rules important in the design of an agent?
6. How to measure the performance of an agent?
7. Define Omniscience and Information Gathering.
8. What is important for task environment?
9. List the properties of environments.
10. List the ways to formulate a problem?
11. List the different types of agents.
12. What are the applications of AI?
13. Name the elements of an agent and list out the characteristics of an intelligent agent.
14. Define Problem Formulation.
15. What are the components to define a problem? Define them.
16. What is an optimal solution?
17. Define abstraction.
18. List the criteria to measure the performance of search strategies.
19. Differentiate uninformed search and informed search algorithms.
20. Define Breadth First Search.
21. Define Depth First Search.
22. Define Backtracking Search.
23. Define Depth-Limited Search.
24. Define Iterative Deepening Depth-First Search.
25. Define Bi-directional Search.
26. Define Best-First Search.
27. State the significance of using heuristic functions.
28. Define Hill Climbing Search Algorithm.
29. Define A* Search Algorithm.
30. What do you mean by local maxima with respect to search technique?
31. Define Local Search Algorithm.
32. List out the advantages of local search algorithms.
33. Define Greedy Best-First Search.
34. List the disadvantages of Hill Climbing.
35. Explain Recursive Best-First Search.
36. How can we avoid ridge and plateau in Hill Climbing?
37. Define Simulated Annealing.
38. Define Quiescence Problem.
39. Define Stochastic Beam Search.
40. Define Genetic Algorithm.
41. Define Local Beam Search.
42. Define Alpha-Beta Pruning.
43. Define Game.
44. List out the classification of CSP with respect to constraints.
45. Define Minimax Algorithm.
46. Define a CSP.
47. Define Forward Pruning.
48. What is Skolemization?
49. What is First-Order Logic?
50. What are the elements and symbols of First-Order Logic?

51. What are the three families of First-Order Inference Algorithms?
52. What are the four parts of knowledge in First-Order Logic?
53. What is the significance of using the unification algorithm?
54. Define the First-Order Definite Clause.
55. Define Semantic Networks.
56. What is Data-Driven Search?
57. Compare Forward Chaining and Backward Chaining.
58. Define Resolution.
59. Define Knowledge Base.
60. What is a Pure Symbol?
61. What are the basic elements of First-Order Logic?
62. What is Classical Planning?
63. What is Partial Order Planning?
64. What is a Job Shop Scheduling Problem?
65. Define Purely Reactive Agents.
66. What are the two types of information source?
67. What are the characteristics of subsumption architecture?
68. State the advantage of vertically layered architecture.
69. Explore some interesting properties of agents and perception.
70. What are four classes of agents?
71. What are logical formulae and logical deduction?
72. Define Belief-Desire-Intention (BDI) Architectures.
73. State the advantage of horizontal layered architectures.
74. Define Agent Communication.
75. Define Coherence.
76. Define the property of Coordination.
77. What are the three aspects to the formal study of communication?
78. Define Ontology.
79. Define Bargaining.
80. List the various applications of Artificial Intelligence.
81. Define Language Modeling.
82. What is Natural Language Processing (NLP)?
83. What is Information Retrieval?
84. What are the objectives of NLP?
85. What is Information Extraction?
86. What are the features of NLP?
87. What is an N-gram Language Model?
88. What is meant by Machine Translation?
89. What are the three main applications of Machine Translation?
90. Define Speech Recognition.
91. What is a Robot?
92. Mention the basic hardware components of a Robot.
93. Define Robotic Perception.
94. Define Pruning.
95. What is Mini-Max Strategy?
96. What is Game Playing?
97. What is Learning?
98. What are the types of Learning?
99. What is Adaptive Learning?
100. What do you mean by Default Reasoning?

Viva Questions

Linux System Structure & Commands

1. **What is Linux?**
Linux is an open-source UNIX-like operating system.
2. **What is the kernel?**
The kernel is the core component that manages hardware and software resources.
3. **What does `pwd` command do?**
It displays the current working directory.
4. **What does `ls` command do?**
It lists files and directories.
5. **What is the use of `mkdir`?**
It creates a new directory.
6. **What is the use of `rmdir`?**
It removes an empty directory.
7. **What does `cp` command do?**
It copies files or directories.
8. **What is the purpose of `mv` command?**
It moves or renames files.
9. **What does `rm` command do?**
It deletes files or directories.
10. **What is a shell?**
A shell is a command interpreter between user and OS.

Shell Scripting

11. **What is shell scripting?**
It is writing commands in a file for automated execution.
12. **What is the extension of a shell script?**
Usually `.sh`.
13. **What is shebang (`#!/bin/bash`)?**
It specifies the interpreter for the script.
14. **How are variables declared in shell scripts?**
Using `variable=value`.
15. **What does `echo` do?**
It displays text on the terminal.
16. **What is `$1` in shell scripting?**
It represents the first command-line argument.
17. **What is a loop?**
A loop repeatedly executes a block of statements.
18. **Name two looping constructs in shell scripting.**
`for` and `while`.
19. **What is an if statement?**
It performs conditional execution.
20. **How do you execute a shell script?**
Using `./script.sh`.

Processes and CPU Scheduling

21. **What is a process?**
A process is a program in execution.
22. **What is a Process Control Block (PCB)?**
It stores information about a process.
23. **What is CPU scheduling?**
It selects a process for CPU execution.
24. **What is FCFS scheduling?**
First Come First Serve executes processes in arrival order.
25. **What is SJF scheduling?**
Shortest Job First executes the shortest burst-time process first.
26. **What is Round Robin scheduling?**
It allocates CPU time using fixed time slices.
27. **What is Priority Scheduling?**
Processes are executed based on priority values.
28. **What is waiting time?**
Time spent in the ready queue.
29. **What is turnaround time?**
Total time from submission to completion.
30. **What is response time?**
Time until a process first receives CPU.

Interprocess Communication (IPC)

31. **What is IPC?**
It enables communication between processes.
32. **What is a pipe?**
A pipe is a unidirectional IPC mechanism.
33. **What is a named pipe?**
A FIFO pipe identified by a filename.
34. **What is shared memory?**
Memory shared among multiple processes.
35. **What is a message queue?**
A queue used to exchange messages between processes.
36. **Which IPC method is fastest?**
Shared memory.
37. **What is synchronization in IPC?**
Coordinating process execution safely.
38. **What is a producer-consumer problem?**
A synchronization problem involving shared buffers.
39. **What is data consistency?**
Maintaining correct shared data values.
40. **Why is IPC required?**
To share data and resources among processes.

Process Synchronization

41. **What is process synchronization?**
Coordinating concurrent process execution.

- 42. **What is a critical section?**
Code segment accessing shared resources.
- 43. **What is a semaphore?**
An integer variable used for synchronization.
- 44. **What are semaphore operations?**
Wait(P) and Signal(V).
- 45. **What is a binary semaphore?**
A semaphore with values 0 or 1.
- 46. **What is a counting semaphore?**
A semaphore with multiple resource counts.
- 47. **What is a monitor?**
A high-level synchronization construct.
- 48. **What is mutual exclusion?**
Allowing only one process in a critical section.
- 49. **What is race condition?**
Unpredictable results due to concurrent access.
- 50. **What is busy waiting?**
Continuous checking for a condition.

Deadlocks

- 51. **What is a deadlock?**
A situation where processes wait indefinitely.
- 52. **What are the four deadlock conditions?**
Mutual exclusion, hold-and-wait, no preemption, circular wait.
- 53. **What is deadlock prevention?**
Eliminating one of the deadlock conditions.
- 54. **What is deadlock avoidance?**
Avoiding unsafe states dynamically.
- 55. **What is Banker's Algorithm?**
A deadlock avoidance algorithm.
- 56. **What is a safe state?**
A state where all processes can finish.
- 57. **What is an unsafe state?**
A state that may lead to deadlock.
- 58. **What is deadlock detection?**
Identifying deadlocks after occurrence.
- 59. **What is resource allocation graph?**
A graph representing process-resource relationships.
- 60. **How can deadlocks be recovered?**
By process termination or resource preemption.

Memory Management

- 61. **What is memory management?**
Managing allocation and deallocation of memory.
- 62. **What is fragmentation?**
Wasted memory space during allocation.
- 63. **What is external fragmentation?**
Free memory scattered into small blocks.

- 64. **What is internal fragmentation?**
Unused space within allocated memory.
- 65. **What is First Fit?**
Allocates the first sufficient memory block.
- 66. **What is Best Fit?**
Allocates the smallest suitable block.
- 67. **What is Worst Fit?**
Allocates the largest available block.
- 68. **Which allocation method is fastest?**
First Fit.
- 69. **Which allocation method minimizes wastage?**
Best Fit.
- 70. **What is virtual memory?**
Memory management using disk as extension of RAM.

Page Replacement Algorithms

- 71. **What is paging?**
Dividing memory into fixed-size pages.
- 72. **What is a page fault?**
Accessing a page not in memory.
- 73. **What is page replacement?**
Replacing pages when memory is full.
- 74. **What is FIFO page replacement?**
Replaces the oldest page first.
- 75. **What is LRU page replacement?**
Replaces the least recently used page.
- 76. **What is Optimal page replacement?**
Replaces the page used farthest in future.
- 77. **Which page replacement gives minimum faults?**
Optimal algorithm.
- 78. **What is a frame?**
A fixed-size block in physical memory.
- 79. **What is thrashing?**
Excessive page swapping causing poor performance.
- 80. **What is demand paging?**
Loading pages only when required.

File Systems

- 81. **What is a file system?**
A method of organizing and storing files.
- 82. **What is a file?**
A collection of related information.
- 83. **What is a directory?**
A structure used to organize files.
- 84. **What is contiguous allocation?**
File blocks stored in consecutive locations.
- 85. **What is linked allocation?**
File blocks linked using pointers.

- 86. **What is indexed allocation?**
An index block stores file block addresses.
- 87. **What is a file descriptor?**
A unique identifier for an open file.
- 88. **What is file allocation?**
Assigning disk space to files.
- 89. **What is free space management?**
Tracking available disk blocks.
- 90. **What is inode in Linux?**
A data structure storing file metadata.

Disk Scheduling

- 91. **What is disk scheduling?**
Determining the order of disk access requests.
- 92. **What is FCFS disk scheduling?**
Requests are served in arrival order.
- 93. **What is SSTF?**
Shortest Seek Time First scheduling.
- 94. **What is SCAN algorithm?**
Disk arm moves in one direction then reverses.
- 95. **What is C-SCAN?**
Circular SCAN services requests in one direction only.
- 96. **What is seek time?**
Time required to move the disk head.
- 97. **What is rotational latency?**
Waiting time for the desired sector.
- 98. **What is disk throughput?**
Number of requests serviced per unit time.
- 99. **What is a process scheduler simulation?**
A program that mimics CPU scheduling behavior.
- 100. **What is a simple file system project?**
A software model that manages files and directories virtually.