Acharya Institute of Technology

Acharya Dr Sarvepalli. Radhakrishnan Road
Acharya P.O Soladevanahalli, Bengaluru-560107

LABORATORY MANUAL

Operating System

BCS303

III Semester

AY: 2025-26

Name	:_______________________________________
USN	:_______________________________________
Branch & Section	:_______________________________________

Prepared by:

1. Mrs. Deeksha Satish

Assistant Professor, Dept. of CS&E

2. Mrs. Anitta Anthony

Assistant Professor, Dept. of CS&E

Reviewed by:

Mrs. Sneha N P
Assistant Professor, Dept. of CS&E

Approved By

Dr. Kala Venugopal
H.O.D, Dept. of CS&E

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
(Accredited by NBA & NAAC)

TABLE OF CONTENTS

I. Vision, Mission and Motto of the Institute:

Institute Vision:

Acharya Institute of Technology, committed to the cause of value-based education in all disciplines, envisions itself as a fountainhead of innovative human enterprise, with inspirational initiatives for Academic Excellence.

Institute Mission:

Acharya Institute of Technology strives to provide excellent academic ambiance to the students for achieving global standards of technical education, foster intellectual and personal development, meaningful research and ethical service to sustainable societal needs.

Motto of the Institute:

Nurturing Aspirations Supporting Growth

II. Vision and Mission of the Department:

Department Vision:

Envisions to be recognized for quality education and research in the field of Computing, leading to creation of competent engineers, who are innovative and adaptable to the changing demands of industry and society.

Department Mission:

- Act as a nurturing ground for young computing aspirants to attain the excellence by imparting quality education and professional ethics.
- Collaborate with industries and provide exposure to latest tools/ technologies.
- Create an environment conducive for research and continuous learning

III. Program Educational Objectives:

PEO-1: Students shall, have a successful career in academia, R&D organizations, IT industry or pursue higher studies in specialized field of Computer Science and Engineering and allied disciplines.

PEO-2: Students shall, be competent, creative and a valued professional in the chosen field

PEO-3: Students shall, engage in life-long learning, professional development and adapt to the working environment quickly

PEO-4: Students shall, become effective collaborators and exhibit high level of professionalism by leading or participating in addressing technical, business, environmental and societal challenges.

IV. Program Outcomes:

ENGINEERING GRADUATES WILL BE ABLE TO:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

V. Program Specific Outcomes:

PSO-1: Students shall apply the knowledge of hardware, system software, algorithms, networking and data bases.

PSO-2: Students shall design, analyze and develop efficient, secure algorithms using appropriate data structures, databases for processing of data.

PSO-3: Students shall be Capable of developing stand alone, embedded and web-based solutions having an easy to operate interface using Software Engineering practices and contemporary computer programming languages.

VI. Course Details:

i. Course Details:

Course Title	Course Code	Core/Elective	Semester	Academic Year
Operating Systems	BCS303	IPCC	3	2025-26

Contact Hours/week	Lecture	Tutorials	Practical
5	3	0	2

ii. Course Administrator/Coordinator Details:

DETAILS OF THE FACULTY CO-ORDINATORS/COURSE ADMINISTRATORS				
S.N	**NAME OF THE FACULTY**	**DESIGNATION**	**DEPARTMENT**	**E-MAIL-ID & CONTACT NUMBER**
1.	**Mrs. Deeksha Satish**	**Assistant Professor**	Computer Science and Engineering	EMAIL-ID: deeksha2789@acharya.ac.in
				CONTACT NUMBER: 9902164935
2.	**Mrs. Anitta Antony**	**Assistant Professor**	Computer Science and Engineering	EMAIL-ID: anitta2415@acharya.ac.in
				CONTACT NUMBER: 8848417242

iii. Course Related Specific details:

List Of Prerequisites:	
1.	Basic C Programming skills (loops, functions, pointers, arrays, structures).
2.	Basic Understanding of Hardware, Software, Types of software, CPU, memory and storage.

Recommended Text Books:	
1.	Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Principles 8th edition, Wiley-India, 2015

Recommended Reference Books:	
1	Ann McHoes Ida M Fylnn, Understanding Operating System, Cengage Learning, 6th Edition
2	D.M Dhamdhere, Operating Systems: A Concept Based Approach 3rd Ed, McGraw- Hill, 2013.
3	P.C.P. Bhatt, An Introduction to Operating Systems: Concepts and Practice 4th Edition, PHI(EEE), 2014.
4	William Stallings Operating Systems: Internals and Design Principles, 6th Edition, Pearson.

iv. Course Objectives:

Course Objectives:	
The course will enable the students to:	
1	To Demonstrate the need for OS and different types of OS
2	To discuss suitable techniques for management of different types of resources.
3	To demonstrate different APIs/Commands related to processor, memory, storage and file system management.

v. Course Outcomes:

Course Outcomes:		
After the completion of the course, Students will be able to:		
COURSE OUTCOME NUMBER	**COURSE OUTCOME STATEMENT**	**RBT Level**
CO1	Illustrate the fundamentals of the operating system, its various types, services and its importance in computer systems.	BL2
CO2	Apply the concepts of Inter process communication, multi-threading programming and process scheduling for process management.	BL3
CO3	Illustrate various process synchronization techniques, including algorithms for deadlock avoidance, prevention, and recovery.	BL2
CO4	Explain the different strategies used for managing main memory and virtual memory in operating systems	BL2
CO5	Explain the different principles of file systems, methods for managing secondary storage, and techniques used to safeguard information	BL2
CO6	Simulate process-related system calls and inter-process communication (IPC) mechanisms using the C programming language.	BL3
CO7	Simulate the Bankers algorithm, memory allocation techniques, file organization, allocation techniques and CPU/disk scheduling algorithms using C programming lang	BL3

vi. CO-PO-PSO Mapping:

COs	Program Outcomes												Program Specific Outcomes		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO-1	3								3	3			2	1	
CO-2	3	2	2	1	2				3	3		2	2	1	
CO-3	3	2	2	1	2				3	3		2	1	2	
CO-4	3	2	2	1	2				3	3		2			
CO-5	3	1	2	1	2				3	3		2	2		
CO-6	3	3	1	1	3				3	3		2	1	1	
CO-7	3	3	1	1	3				3	3		2	2	1	

vii. List of the Programs/Experiments:

Table for listing programs/Experiments for IPCC subject labs

Sl. No	Experiments
	Module – 1
1.	Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process)
	Module – 2
2.	Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.
	Module – 3
3.	Develop a C program to simulate producer-consumer problem using semaphores.
4.	Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.
5.	Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.
	Module – 4
6.	Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit.
7.	Develop a C program to simulate page replacement algorithms: a) FIFO b) LRU

Module - 5	
8.	Simulate following File Organization Techniques a) Single level directory b) Two level directory.
9.	Develop a C program to simulate the Linked file allocation strategies.
10.	Develop a C program to simulate SCAN disk scheduling algorithm.

viii. Assessment and Rubrics:

Sl No	Parameters	Mark (15)	4-5	2-3	0-1
1.	Writing Program/Logic (present week's/previous week's)	5	The student is able to write the program without errors or with minimal errors	The student is able to write the program with minor error	The student has written incomplete program with error or not attempted to write the program.
2.	Implementation in the target language with different inputs	5	Student is able to execute, debug, and test the program for all possible inputs/test cases.	Student is able to execute the program, but fails to debug, and test the program for all possible inputs/test cases.	Student has executed the program partially (fails to meet desired output) or does not execute the program.
3.	Record and Viva	5	Student Communicates the concepts and program effectively both orally and written.	Student is not able to effectively Communicates the concepts and program effectively both orally and written.	Student fails to submit the record on time or Does not answer any viva questions

4	Parameters	Marks (50)	10(Write-up)	30(Execution)	10(Viva)
5	Internal Assessment	10	Conducted for 50 Marks and Scaled down to 10 marks (20 Write-Up+20 Execution+10 Viva)		

VII. Do's and Don'ts in the laboratory:

Do's:

Sl. No	Do's statement
1	Maintain silence in the laboratory.
2	Bring observation book and lab record book and get them signed every time.
3	Fill the login details in the register provided.
4	Leave the bags and other belongings outside the lab, in designated places.
5	Know the location of the fire extinguisher and the first aid box and how to use them in case of an emergency.
6	Report any broken plugs or exposed electrical wires to your faculty/instructor immediately.
7	Shutdown the PC and arrange the chairs properly while leaving the laboratory.

Don'ts:

Sl. No	Do's statement
1	Do not eat or drink in the laboratory.
2	Avoid stepping on electrical wires or any other computer cables.
3	Do not connect pen drive or any other external storage devices to the computer.
4	Do not open the system unit casing or monitor casing, particularly when the power is turned on. Some internal components hold electric voltages of up to 30000 volts, which can be fatal.
5	Do not install any software without your faculty/instructor's permission.
6	Do not touch, connect or disconnect any plug or cable without your faculty/instructor's permission.
7	Usage of mobile phone inside the laboratory is prohibited and if found, phone shall be confiscated.

VIII. Introduction to Equipment's /Software used in the laboratory:

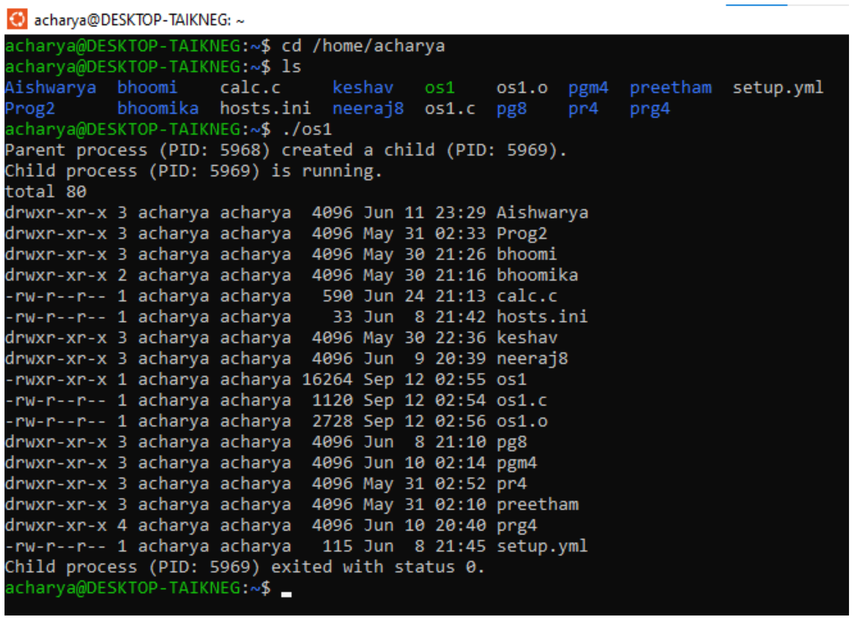
1. Linux OS
2. Code blocks / online C compiler

XI. Laboratory Experiments/programs:

1. Develop a c program to implement the Process system calls (fork (), exec (), wait (), create process, terminate process)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
  int main()
 {
   pid_t child_pid;
   int status;
     // Create a child process
   child_pid = fork();
   if (child_pid < 0)
  {
      // Fork failed
      perror("Fork failed");
      exit(1);
   } else if (child_pid == 0)
    {
      // Child process
      printf("Child process (PID: %d) is running.\n", getpid());
      // Replace the child process with a new program
      execlp("/bin/ls", "ls", "-l", NULL);
      // If exec fails, the code below will be executed
      perror("Exec failed");
      exit(1);
   } else
    {
        // Parent process
        printf("Parent process (PID: %d) created a child (PID: %d).\n", getpid(), child_pid);
        // Wait for the child process to finish
        wait(&status);
        if (WIFEXITED(status)) {
            printf("Child process (PID: %d) exited with status %d.\n", child_pid, WEXITSTATUS(status));
        }
    else {
            printf("Child process (PID: %d) did not exit normally.\n", child_pid);
        }
    }
    return 0; }
```

OUTPUT:



```
acharya@DESKTOP-TAIKNEG: ~
acharya@DESKTOP-TAIKNEG:~$ cd /home/acharya
acharya@DESKTOP-TAIKNEG:~$ ls
Aishwarya  bhoomi    calc.c     keshav   os1    os1.o  pgm4  preetham  setup.yml
Prog2      bhoomika  hosts.ini  neeraj8  os1.c  pg8    pr4   prg4
acharya@DESKTOP-TAIKNEG:~$ ./os1
Parent process (PID: 5968) created a child (PID: 5969).
Child process (PID: 5969) is running.
total 80
drwxr-xr-x 3 acharya acharya  4096 Jun 11 23:29 Aishwarya
drwxr-xr-x 3 acharya acharya  4096 May 31 02:33 Prog2
drwxr-xr-x 3 acharya acharya  4096 May 30 21:26 bhoomi
drwxr-xr-x 2 acharya acharya  4096 May 30 21:16 bhoomika
-rw-r--r-- 1 acharya acharya   590 Jun 24 21:13 calc.c
-rw-r--r-- 1 acharya acharya    33 Jun  8 21:42 hosts.ini
drwxr-xr-x 3 acharya acharya  4096 May 30 22:36 keshav
drwxr-xr-x 3 acharya acharya  4096 Jun  9 20:39 neeraj8
-rwxr-xr-x 1 acharya acharya 16264 Sep 12 02:55 os1
-rw-r--r-- 1 acharya acharya  1120 Sep 12 02:54 os1.c
-rw-r--r-- 1 acharya acharya  2728 Sep 12 02:56 os1.o
drwxr-xr-x 3 acharya acharya  4096 Jun  8 21:10 pg8
drwxr-xr-x 3 acharya acharya  4096 Jun 10 02:14 pgm4
drwxr-xr-x 3 acharya acharya  4096 May 31 02:52 pr4
drwxr-xr-x 3 acharya acharya  4096 May 31 02:10 preetham
drwxr-xr-x 4 acharya acharya  4096 Jun 10 20:40 prg4
-rw-r--r-- 1 acharya acharya   115 Jun  8 21:45 setup.yml
Child process (PID: 5969) exited with status 0.
acharya@DESKTOP-TAIKNEG:~$
```

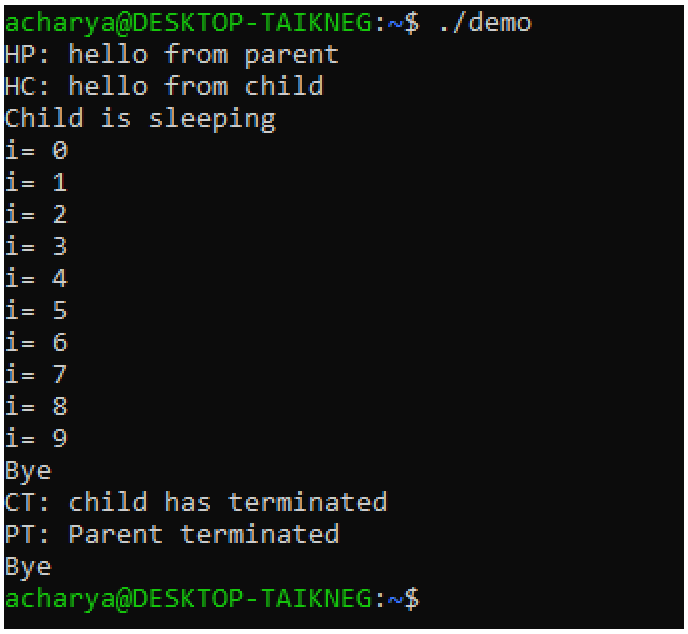
Question to think:

1. Why is wait() necessary in the parent process after fork() and exec()?
2. What problems may occur if the parent does not call wait()?

// C program to demonstrate working of wait()

```
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
  int main()
{
      if (fork()== 0)
                   {
                            printf("HC: hello from child\n");
                            printf("Child is sleeping\n");

                            for(int i=0;i<10;i++)
                            {
                                     printf("i= %d\n",i);
                                     sleep(1);
                            }
                   //sleep(2);
                   }
                   else
                   {
                            printf("HP: hello from parent\n");
                            wait(NULL);
                            printf("CT: child has terminated\n");
                            printf("PT: Parent terminated ");
                   }
         printf("Bye\n");
          return 0;
}
```

OUTPUT:



```
acharya@DESKTOP-TAIKNEG:~$ ./demo
HP: hello from parent
HC: hello from child
Child is sleeping
i= 0
i= 1
i= 2
i= 3
i= 4
i= 5
i= 6
i= 7
i= 8
i= 9
Bye
CT: child has terminated
PT: Parent terminated
Bye
acharya@DESKTOP-TAIKNEG:~$
```

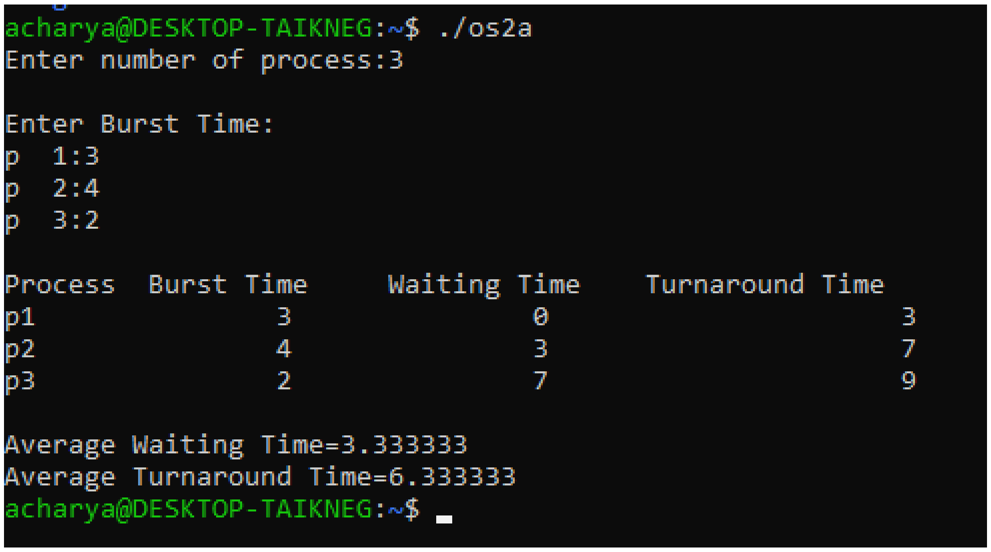
2. Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.

A) FCFS (First Come First Serve)

```
#include<stdio.h>
int main()
{
 int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
 float avg_wt,avg_tat;
 printf("Enter number of process:");
 scanf("%d",&n);
 printf("\nEnter Burst Time:\n");

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

```
            printf("p % d:",i+1);
            scanf("%d",&bt[i]);
            p[i]=i+1; //contains process number
    }

wt[0]=0; //waiting time for first process will be zero
//calculate waiting time

 for(i=1;i<n;i++)
 {
            wt[i]=0;
            for(j=0;j<i;j++)
            wt[i]+=bt[j];
            total+=wt[i];
 }
 avg_wt=(float)total/n; //average waiting time
 total=0;
 printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
            for(i=0;i<n;i++)
            {
                    tat[i]=bt[i]+wt[i]; //calculate turnaround time
                    total+=tat[i];
                    printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
 }
 avg_tat=(float)total/n; //average turnaround time
 printf("\n\nAverage Waiting Time=%f",avg_wt);
 printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

OUTPUT:



```
acharya@DESKTOP-TAIKNEG:~$ ./os2a
Enter number of process:3

Enter Burst Time:
p  1:3
p  2:4
p  3:2

Process  Burst Time     Waiting Time    Turnaround Time
p1               3               0                       3
p2               4               3                       7
p3               2               7                       9

Average Waiting Time=3.333333
Average Turnaround Time=6.333333
acharya@DESKTOP-TAIKNEG:~$ 
```

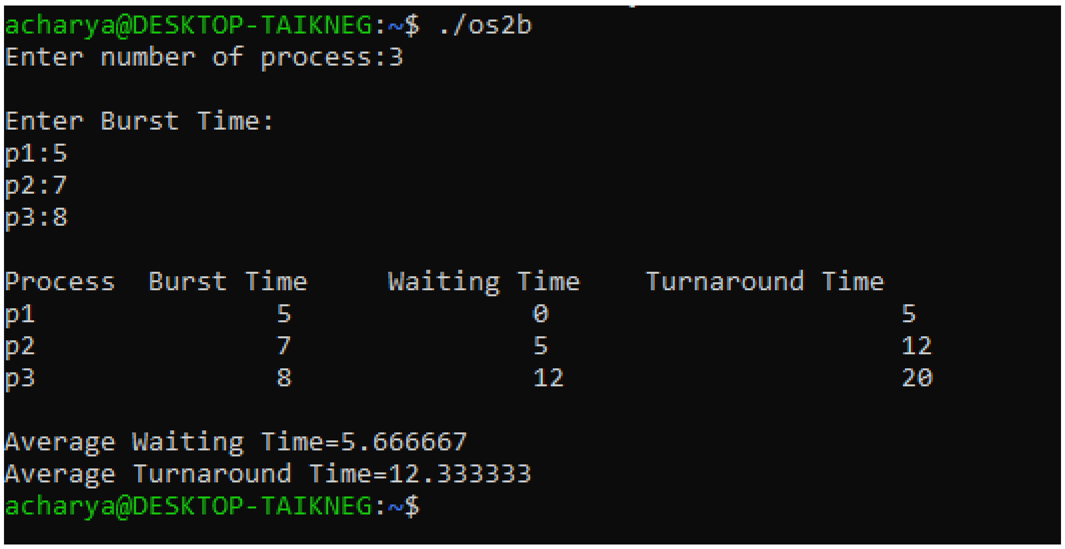
B) SJF (Shortest Job First)

```
#include<stdio.h>
int main()
{
 int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
 float avg_wt,avg_tat;
 printf("Enter number of process:");
 scanf("%d",&n);
 printf("\nEnter Burst Time:\n");
 for(i=0;i<n;i++)
 {
 printf("p%d:",i+1);
 scanf("%d",&bt[i]);
 p[i]=i+1; //contains process number
 }
//sorting burst time in ascending order using selection sort
```

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

        for(j=i+1;j<n;j++)
        {
                if(bt[j]<bt[pos])
                pos=j;
        }

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
}
wt[0]=0; //waiting time for first process will be zero

//calculate waiting time

for(i=1;i<n;i++)
{
        wt[i]=0;
        for(j=0;j<i;j++)
        wt[i]+=bt[j];
        total+=wt[i];
}
avg_wt=(float)total/n; //average waiting time
```

```
 total=0;
 printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

for(i=0;i<n;i++)
{
        tat[i]=bt[i]+wt[i]; //calculate turnaround time
        total+=tat[i];
        printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
 }

 avg_tat=(float)total/n; //average turnaround time
 printf("\n\nAverage Waiting Time=%f",avg_wt);
 printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

OUTPUT:

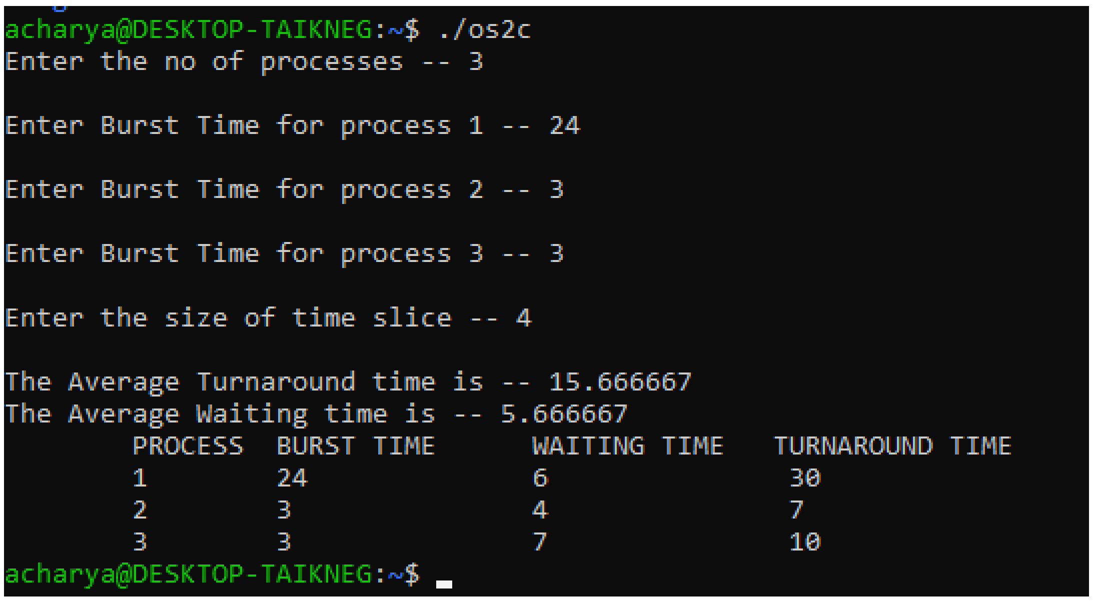


```
acharya@DESKTOP-TAIKNEG:~$ ./os2b
Enter number of process:3

Enter Burst Time:
p1:5
p2:7
p3:8

Process  Burst Time     Waiting Time    Turnaround Time
p1               5               0                      5
p2               7               5                      12
p3               8               12                     20

Average Waiting Time=5.666667
Average Turnaround Time=12.333333
acharya@DESKTOP-TAIKNEG:~$
```

C) Round Robin

```
 #include<stdio.h>

int main()
{
        int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
        float awt=0,att=0,temp=0;
        printf("Enter the no of processes -- ");
        scanf("%d",&n);

        for(i=0;i<n;i++)
        {
                printf("\nEnter Burst Time for process %d -- ", i+1);
                scanf("%d",&bu[i]);
                ct[i]=bu[i];
        }
printf("\nEnter the size of time slice -- ");
scanf("%d",&t);
max=bu[0];

for(i=1;i<n;i++)
if(max<bu[i])
max=bu[i];
for(j=0;j<(max/t)+1;j++)
for(i=0;i<n;i++)
if(bu[i]!=0)
if(bu[i]<=t) {
tat[i]=temp+bu[i];
temp=temp+bu[i];
bu[i]=0;
}

else
{
        bu[i]=bu[i]-t;
        temp=temp+t;
}
for(i=0;i<n;i++)
{
        wa[i]=tat[i]-ct[i];
        att+=tat[i];
        awt+=wa[i];
}

printf("\nThe Average Turnaround time is -- %f",att/n);
printf("\nThe Average Waiting time is -- %f ",awt/n);
printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
for(i=0;i<n;i++)
        printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
}
```

OUTPUT:



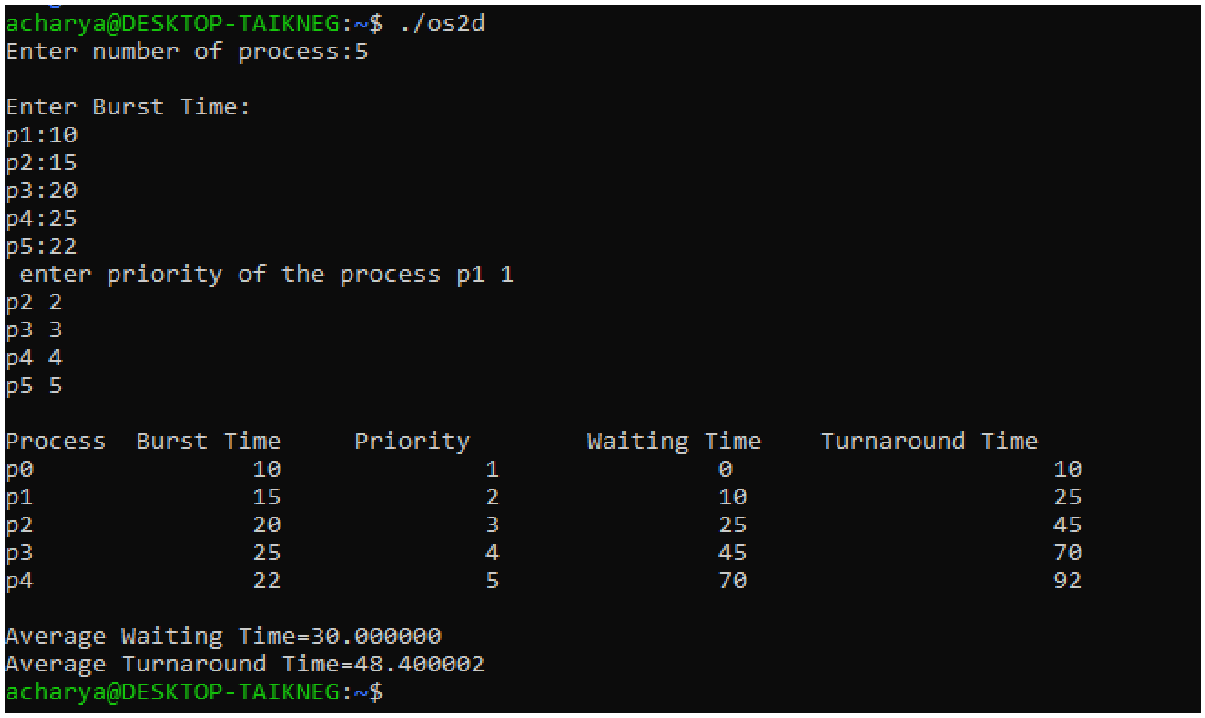
```
acharya@DESKTOP-TAIKNEG:~$ ./os2c
Enter the no of processes -- 3

Enter Burst Time for process 1 -- 24

Enter Burst Time for process 2 -- 3

Enter Burst Time for process 3 -- 3

Enter the size of time slice -- 4

The Average Turnaround time is -- 15.666667
The Average Waiting time is -- 5.666667
        PROCESS  BURST TIME       WAITING TIME   TURNAROUND TIME
        1        24               6               30
        2        3                4               7
        3        3                7               10
acharya@DESKTOP-TAIKNEG:~$
```

D) Priority

```
#include<stdio.h>
int main()
{
 int bt[20],p[20],wt[20],tat[20],pri[20],i,j,k,n,total=0,pos,temp;
 float avg_wt,avg_tat;
 printf("Enter number of process:");
 scanf("%d",&n);
 printf("\nEnter Burst Time:\n");
 for(i=0;i<n;i++)
 {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1; //contains process number
 }
printf(" enter priority of the process ");
for(i=0;i<n;i++)
{
        p[i] = i;
        //printf("Priority of Process");
        printf("p%d ",i+1);
        scanf("%d",&pri[i]);
}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(pri[i] > pri[k])
{
 temp=p[i];
 p[i]=p[k];
 p[k]=temp;
 temp=bt[i];
 bt[i]=bt[k];
 bt[k]=temp;
temp=pri[i];
 pri[i]=pri[k];
 pri[k]=temp;
}
wt[0]=0; //waiting time for first process will be zero
 //calculate waiting time
 for(i=1;i<n;i++)
 {
        wt[i]=0;
        for(j=0;j<i;j++)
                wt[i]+=bt[j];
                total+=wt[i];
 }
avg_wt=(float)total/n; //average waiting time
 total=0;
 printf("\nProcess\t Burst Time \tPriority \tWaiting Time\tTurnaround Time");
 for(i=0;i<n;i++)
 {
        tat[i]=bt[i]+wt[i]; //calculate turnaround time
        total+=tat[i];
        printf("\np%d\t\t %d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],pri[i],wt[i],tat[i]);
 }
 avg_tat=(float)total/n; //average turnaround time
 printf("\n\nAverage Waiting Time=%f",avg_wt);
 printf("\nAverage Turnaround Time=%f\n",avg_tat);
```

}

OUTPUT:



```
acharya@DESKTOP-TAIKNEG:~$ ./os2d
Enter number of process:5

Enter Burst Time:
p1:10
p2:15
p3:20
p4:25
p5:22
 enter priority of the process p1 1
p2 2
p3 3
p4 4
p5 5

Process  Burst Time     Priority        Waiting Time    Turnaround Time
p0              10             1               0                10
p1              15             2              10                25
p2              20             3              25                45
p3              25             4              45                70
p4              22             5              70                92

Average Waiting Time=30.000000
Average Turnaround Time=48.400002
acharya@DESKTOP-TAIKNEG:~$
```

Question to think:

Scenario: Imagine a hospital emergency room where patients arrive with different conditions:

Some need long treatment times, some need short treatments.

Some are critical (high priority), while others can wait.

Doctors attend patients one after another.

Question:

Which CPU scheduling algorithm (FCFS, SJF, Round Robin, Priority) would best model this scenario?

Justify your answer with reasons.

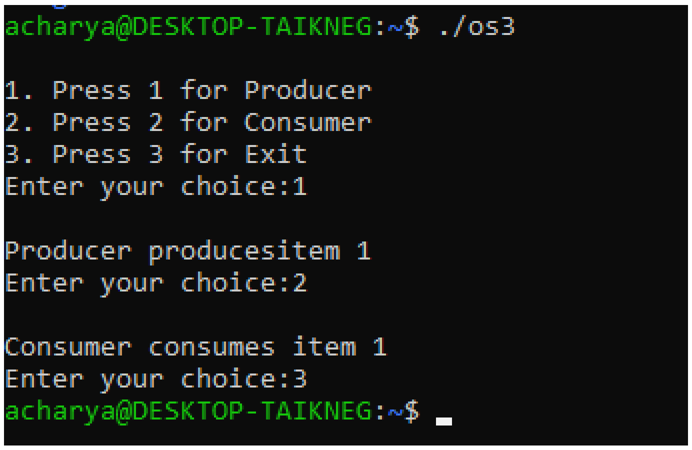
3.Develop a C program to simulate producer-consumer problem using semaphores.

```
#include <stdio.h>

#include <stdlib.h>

// Initialize a mutex to 1

int mutex = 1;


// Number of full slots as 0

int full = 0;


// Number of empty slots as size

// of buffer

int empty = 10, x = 0;


// Function to produce an item and add it to the buffer

void producer()

{

// Decrease mutex value by 1

--mutex;


// Increase the number of full slots by 1

++full;


// Decrease the number of empty slots by 1

--empty;


// Item produced

x++;


printf("\nProducer produces item %d",x);
```

```
// Increase mutex value by 1
++mutex;
}
// Function to consume an item and
// remove it from buffer
void consumer()
{
// Decrease mutex value by 1
--mutex;
// Decrease the number of full
// slots by 1
--full;
// Increase the number of empty
// slots by 1
++empty;
printf("\nConsumer consumes item %d",x);
x--;
// Increase mutex value by 1
++mutex;
}

// Driver Code
int main()
{
int n, i;
printf("\n1. Press 1 for Producer"
"\n2. Press 2 for Consumer"
"\n3. Press 3 for Exit");
```

```
for (i = 1; i > 0; i++)
{
printf("\nEnter your choice:");
scanf("%d", &n);

switch (n)
{
case 1:

// If mutex is 1 and empty is non-zero, then it is possible to produce
if ((mutex == 1)&& (empty != 0))
{
        producer();
}
// Otherwise, print buffer is full
else {
printf("Buffer is full!");
}
break;
case 2:
// If mutex is 1 and full is non-zero, then it is possible to consume
if ((mutex == 1) && (full != 0)) {
consumer();
}
// Otherwise, print Buffer is empty
else {
printf("Buffer is empty!");
}
break;
// Exit Condition
case 3:
exit(0);
break;
}
}
}
```

OUTPUT:



```
acharya@DESKTOP-TAIKNEG:~$ ./os3

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:1

Producer producesitem 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:3
acharya@DESKTOP-TAIKNEG:~$
```

Question to think:

Scenario: A **logistics warehouse** handles packages:

- **Workers** (producers) bring in packages and place them on a **conveyor belt** (buffer).
- **Delivery staff** (consumers) pick packages to load onto trucks.

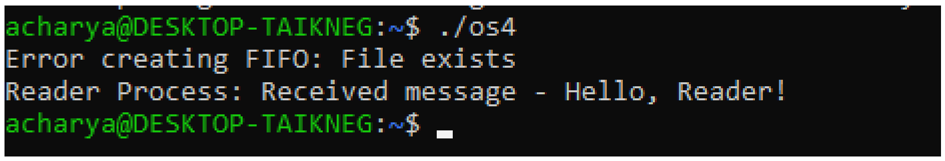
Question

1. How do **semaphores** prevent a delivery staff member from picking up a package that hasn't been placed yet?
2. What happens if **mutual exclusion** is not enforced while multiple workers place packages simultaneously?

4) Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#define FIFO_PATH "myfifo"

void writerProcess() {
    int fd;
    char buffer[] = "Hello, Reader!";
    // Create a FIFO (named pipe)
    if (mkfifo(FIFO_PATH, 0666) == -1) {
        perror("Error creating FIFO");
        exit(EXIT_FAILURE);
    }
    // Open the FIFO for writing
    fd = open(FIFO_PATH, O_WRONLY);
    if (fd == -1) {
        perror("Error opening FIFO for writing");
        exit(EXIT_FAILURE);
    }
    // Write data to the FIFO
    write(fd, buffer, sizeof(buffer));
    // Close the FIFO
    close(fd);
    // Remove the FIFO
    unlink(FIFO_PATH);
}

void readerProcess() {
    int fd;
    char buffer[50];
    // Open the FIFO for reading
    fd = open(FIFO_PATH, O_RDONLY);



  if (fd == -1) {
        perror("Error opening FIFO for reading");
        exit(EXIT_FAILURE);
    }
    // Read data from the FIFO
    read(fd, buffer, sizeof(buffer));
    // Display the read data
    printf("Reader Process: Received message - %s\n", buffer);
    // Close the FIFO
    close(fd);
}

int main() {
    pid_t pid;
    // Fork a child process
    pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process (writer)
        writerProcess();
    } else {
        // Parent process (reader)
        readerProcess();
    }
    return 0;
}
```

OUTPUT:



```
acharya@DESKTOP-TAIKNEG:~$ ./os4
Error creating FIFO: File exists
Reader Process: Received message - Hello, Reader!
acharya@DESKTOP-TAIKNEG:~$
```

Question to think:

What happens if the reader process tries to read before the writer writes any data? How can this be handled?

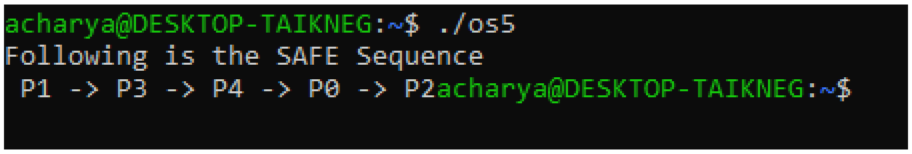
5) Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.

```
#include <stdio.h>
int main()
{
  // P0, P1, P2, P3, P4 are the Process names here
  int n, m, i, j, k;
  n = 5; // Number of processes
  m = 3; // Number of resources
  int alloc[5][3] = { { 0, 1, 0 }, // P0   // Allocation Matrix
                      { 2, 0, 0 }, // P1
                      { 3, 0, 2 }, // P2
                      { 2, 1, 1 }, // P3
                      { 0, 0, 2 } }; // P4
  int max[5][3] = { { 7, 5, 3 }, // P0   // MAX Matrix
                    { 3, 2, 2 }, // P1
                    { 9, 0, 2 }, // P2
                    { 2, 2, 2 }, // P3
                    { 4, 3, 3 } }; // P4
  int avail[3] = { 3, 3, 2 }; // Available Resources
  int f[n], ans[n], ind = 0;
  for (k = 0; k < n; k++) {
    f[k] = 0;
  }
  int need[n][m];
  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++)

      need[i][j] = max[i][j] - alloc[i][j];
```

```
}
int y = 0;
for (k = 0; k < 5; k++) {
    for (i = 0; i < n; i++) {
        if (f[i] == 0) {
            int flag = 0;
            for (j = 0; j < m; j++) {
                if (need[i][j] > avail[j]){
                    flag = 1;
                     break;
                }
            }
            if (flag == 0) {
                ans[ind++] = i;
                for (y = 0; y < m; y++)
                    avail[y] += alloc[i][y];
                f[i] = 1;
            }
        }
    }
}
  int flag = 1;
  for(int i=0;i<n;i++)
{
  if(f[i]==0)
  {

    flag=0;
     printf("The following system is not safe");
```

```
          break;
        }
      }
        if(flag==1)
      {
        printf("Following is the SAFE Sequence\n");
        for (i = 0; i < n - 1; i++)
          printf(" P%d ->", ans[i]);
        printf(" P%d", ans[n - 1]);
      }
      return (0);
  }
```

OUTPUT:



```
acharya@DESKTOP-TAIKNEG:~$ ./os5
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2acharya@DESKTOP-TAIKNEG:~$
```

Question to think:

If a process requests more resources than its declared maximum, what should the system do, and why?

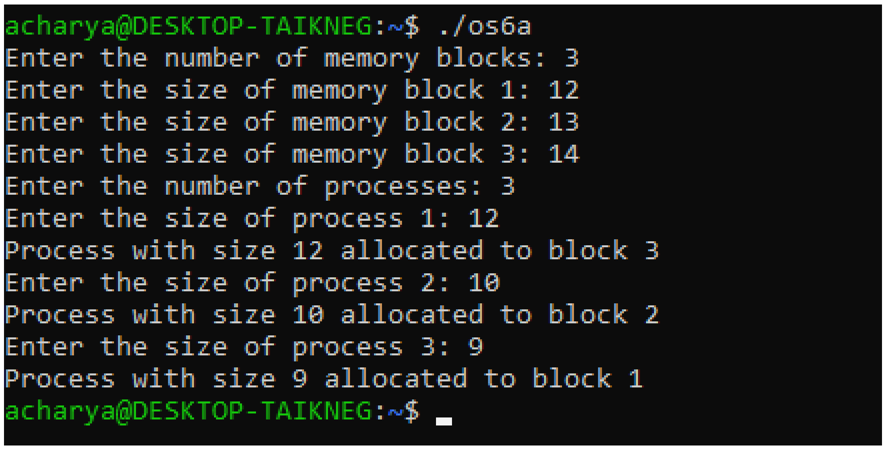
6) Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit.

A) Worst fit

```
#include <stdio.h>
#define MAX_BLOCKS 100
struct MemoryBlock
{
    int block_id;
    int size;
    int allocated;
};
void worstFit(struct MemoryBlock blocks[], int m, int processSize)
{
    int worstIdx = -1;
    for (int i = 0; i < m; i++)
    {
        if (blocks[i].allocated == 0 && blocks[i].size >= processSize)
        {
            if (worstIdx == -1 || blocks[i].size > blocks[worstIdx].size)
            {
                worstIdx = i;
            }
        }
    }
    if (worstIdx != -1)
    {
        blocks[worstIdx].allocated = 1;
        printf("Process with size %d allocated to block %d\n", processSize, worstIdx + 1);
    }
    else
    {
        printf("Cannot allocate the process with size %d\n", processSize);
    }
}

int main()
{
    int m;
    printf("Enter the number of memory blocks: ");
    scanf("%d", &m);
    struct MemoryBlock blocks[MAX_BLOCKS];
    for (int i = 0; i < m; i++)
    {
        blocks[i].block_id = i + 1;
        printf("Enter the size of memory block %d: ", i + 1);
        scanf("%d", &blocks[i].size);
        blocks[i].allocated = 0;
    }
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        int processSize;
        printf("Enter the size of process %d: ", i + 1);
        scanf("%d", &processSize);
        worstFit(blocks, m, processSize);
}
```

```
return 0;

}
```

OUTPUT:

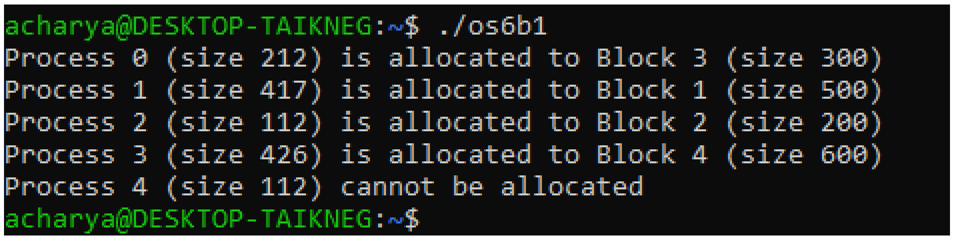


```
acharya@DESKTOP-TAIKNEG:~$ ./os6a
Enter the number of memory blocks: 3
Enter the size of memory block 1: 12
Enter the size of memory block 2: 13
Enter the size of memory block 3: 14
Enter the number of processes: 3
Enter the size of process 1: 12
Process with size 12 allocated to block 3
Enter the size of process 2: 10
Process with size 10 allocated to block 2
Enter the size of process 3: 9
Process with size 9 allocated to block 1
acharya@DESKTOP-TAIKNEG:~$
```

B) Best fit

```
#include <stdio.h>
// Define the maximum number of memory blocks
#define MAX_BLOCKS 100
// Structure to represent memory blocks
struct MemoryBlock {
        int block_id;
        int size;
        int allocated;
};

// Function to perform best-fit memory allocation
void bestFit(struct MemoryBlock blocks[], int m, int processSize) {
        int bestIdx = -1;
        for (int i = 0; i < m; i++) {
                if (blocks[i].allocated == 0 && blocks[i].size >= processSize) {
                        if (bestIdx == -1 || blocks[i].size < blocks[bestIdx].size) {
                                bestIdx = i;
                        }
                }
        }

if (bestIdx != -1) {
        // Allocate the memory block
        blocks[bestIdx].allocated = 1;
        printf("Process with size %d allocated to block %d\n", processSize, bestIdx + 1);
}
else {
        printf("Cannot allocate the process with size %d\n", processSize);
        }
}

int main() {
        int m; // Number of memory blocks
        printf("Enter the number of memory blocks: ");
        scanf("%d", &m);
        struct MemoryBlock blocks[MAX_BLOCKS];
        for (int i = 0; i < m; i++)
        {
                blocks[i].block_id = i + 1;
                printf("Enter the size of memory block %d: ", i + 1);
                scanf("%d", &blocks[i].size);
                blocks[i].allocated = 0; // Initialize as unallocated
        }
        int n; // Number of processes
        printf("Enter the number of processes: ");
        scanf("%d", &n);
        for (int i = 0; i < n; i++)
        {
                int processSize;
                printf("Enter the size of process %d: ", i + 1);
                scanf("%d", &processSize);
                bestFit(blocks, m, processSize);
        }
        return 0;
}
```

OUTPUT:



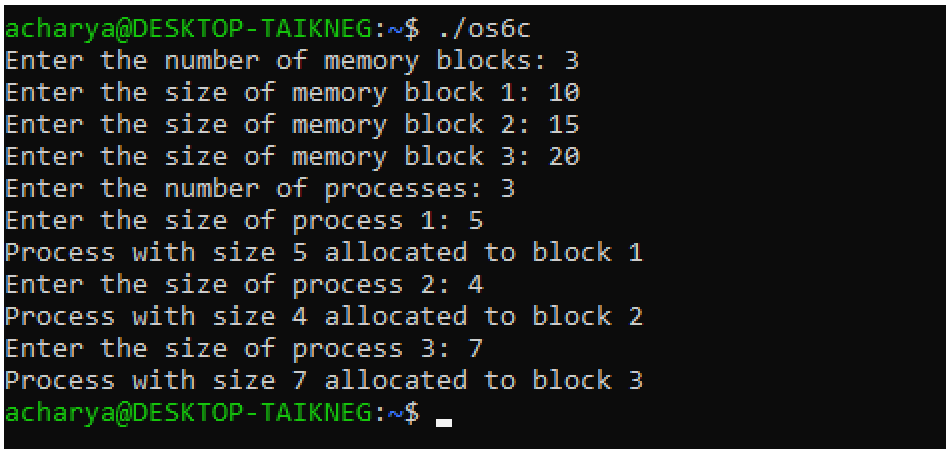
```
acharya@DESKTOP-TAIKNEG:~$ ./os6b1
Process 0 (size 212) is allocated to Block 3 (size 300)
Process 1 (size 417) is allocated to Block 1 (size 500)
Process 2 (size 112) is allocated to Block 2 (size 200)
Process 3 (size 426) is allocated to Block 4 (size 600)
Process 4 (size 112) cannot be allocated
acharya@DESKTOP-TAIKNEG:~$
```

C. First fit

```
#include <stdio.h>
// Define the maximum number of memory blocks
#define MAX_BLOCKS 100
// Structure to represent memory blocks
struct MemoryBlock {
        int block_id;
        int size;
        int allocated;
};
// Function to perform first-fit memory allocation
void firstFit(struct MemoryBlock blocks[], int m, int processSize) {
        for (int i = 0; i < m; i++) {
                if (blocks[i].allocated == 0 && blocks[i].size >= processSize) {
                        // Allocate the memory block
                        blocks[i].allocated = 1;
                        printf("Process with size %d allocated to block %d\n", processSize, i + 1);
                return;
                }
        }

        printf("Cannot allocate the process with size %d\n", processSize);
}
```

```
int main() {
        int m; // Number of memory blocks
        printf("Enter the number of memory blocks: ");
        scanf("%d", &m);
        struct MemoryBlock blocks[MAX_BLOCKS];
        for (int i = 0; i < m; i++) {
                blocks[i].block_id = i + 1;
                printf("Enter the size of memory block %d: ", i + 1);
                scanf("%d", &blocks[i].size);
                blocks[i].allocated = 0; // Initialize as unallocated
        }
        int n; // Number of processes
        printf("Enter the number of processes: ");
        scanf("%d", &n);
        for (int i = 0; i < n; i++) {
                int processSize;
                printf("Enter the size of process %d: ", i + 1);
                scanf("%d", &processSize);
                firstFit(blocks, m, processSize);
        }
        return 0;
}
```

OUTPUT:



```
acharya@DESKTOP-TAIKNEG:~$ ./os6c
Enter the number of memory blocks: 3
Enter the size of memory block 1: 10
Enter the size of memory block 2: 15
Enter the size of memory block 3: 20
Enter the number of processes: 3
Enter the size of process 1: 5
Process with size 5 allocated to block 1
Enter the size of process 2: 4
Process with size 4 allocated to block 2
Enter the size of process 3: 7
Process with size 7 allocated to block 3
acharya@DESKTOP-TAIKNEG:~$
```

Question to think:

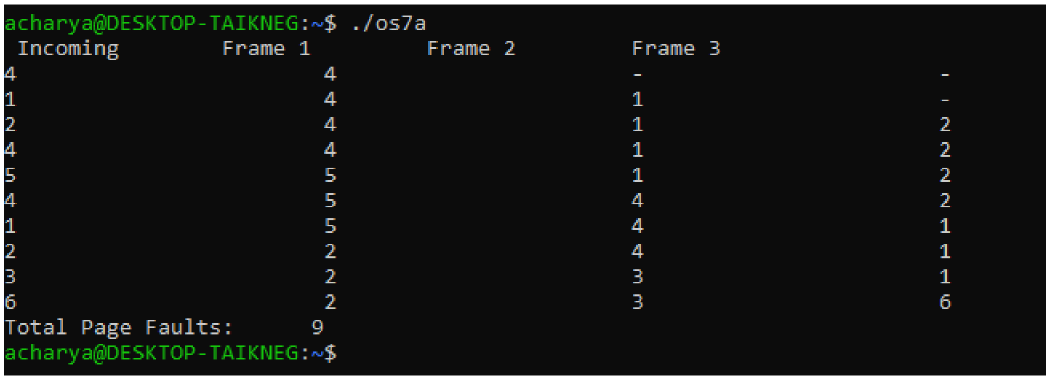
A system frequently runs programs of varying memory sizes. As a system designer, which memory allocation strategy (Worst Fit, Best Fit, or First Fit) would you implement to optimize memory usage and reduce allocation time? Justify your choice by applying the characteristics of each technique to this scenario

7.Develop a C program to simulate page replacement algorithms: a) FIFO b) LRU

A) FIFO

```
#include <stdio.h>
int main()
{
 int incomingStream[] = {4 , 1 , 2 , 4 , 5,4,1,2,3,6};
 int pageFaults = 0;
 int frames = 3;
 int m, n, s, pages;
 pages = sizeof(incomingStream)/sizeof(incomingStream[0]);
 printf(" Incoming \t Frame 1 \t Frame 2 \t Frame 3 ");
 int temp[ frames ];
 for(m = 0; m < frames; m++)
 {
            temp[m] = -1;
 }
 for(m = 0; m < pages; m++)
 {
            s = 0;
 for(n = 0; n < frames; n++)
 {
             if(incomingStream[m] == temp[n])
             {
                         s++;
                         pageFaults--;
             }
 }
pageFaults++;
if((pageFaults <= frames) && (s == 0))
{
            temp[m] = incomingStream[m];
}
            else if(s == 0)
{
            temp[(pageFaults - 1) % frames] = incomingStream[m];
}
printf("\n");
printf("%d\t\t\t",incomingStream[m]);
for(n = 0; n < frames; n++)
{
             if(temp[n] != -1)
             printf(" %d\t\t\t", temp[n]);
             else
             printf(" - \t\t\t");
}
}
printf("\nTotal Page Faults:\t%d\n", pageFaults);
return 0;
}
```

OUTPUT:



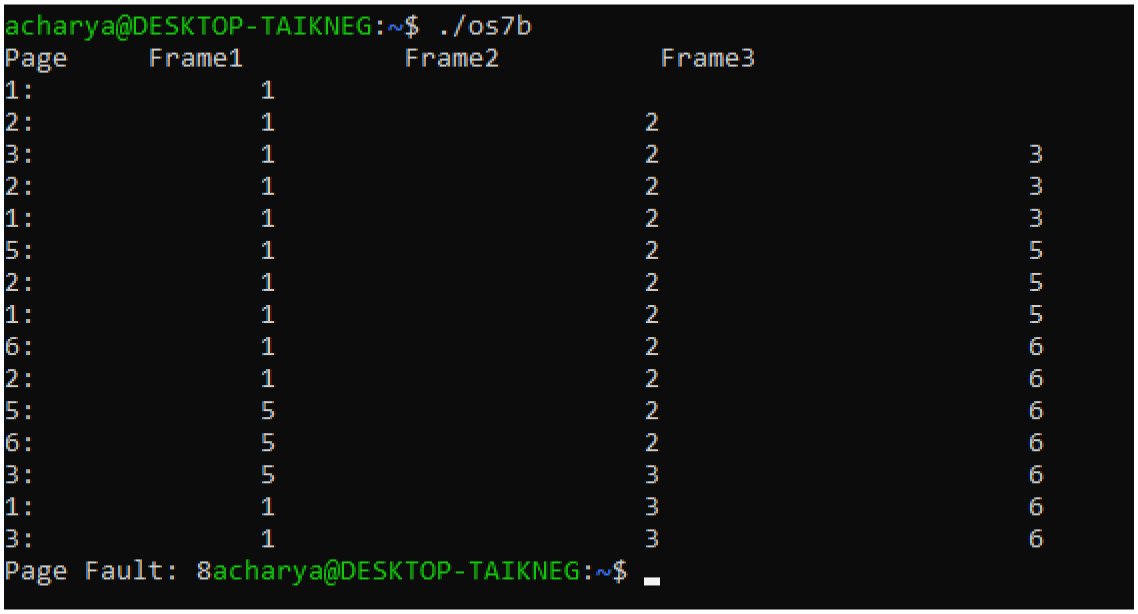
```
acharya@DESKTOP-TAIKNEG:~$ ./os7a
 Incoming        Frame 1         Frame 2         Frame 3
4                        4                       -                       -
1                        4                       1                       -
2                        4                       1                       2
4                        4                       1                       2
5                        5                       1                       2
4                        5                       4                       2
1                        5                       4                       1
2                        2                       4                       1
3                        2                       3                       1
6                        2                       3                       6
Total Page Faults:      9
acharya@DESKTOP-TAIKNEG:~$
```

B) LRU

```
#include<stdio.h>
#include<limits.h>
int checkHit(int incomingPage, int queue[], int occupied){

 for(int i = 0; i < occupied; i++){
 if(incomingPage == queue[i])
 return 1;
 }
 return 0;
}
void printFrame(int queue[], int occupied)
{
 for(int i = 0; i < occupied; i++)
 printf("%d\t\t\t",queue[i]);
}
int main()
{
// int incomingStream[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1};
// int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3, 6, 1, 2, 4, 3};
 int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3};
 int n = sizeof(incomingStream)/sizeof(incomingStream[0]);
 int frames = 3;
 int queue[n];
 int distance[n];
 int occupied = 0;
 int pagefault = 0;
 printf("Page\t Frame1 \t Frame2 \t Frame3\n");
 for(int i = 0;i < n; i++)
```

```
{
printf("%d: \t\t",incomingStream[i]);
// what if currently in frame 7
// next item that appears also 7
// didnt write condition for HIT
if(checkHit(incomingStream[i], queue, occupied)){
printFrame(queue, occupied);
}
// filling when frame(s) is/are empty
else if(occupied < frames){
queue[occupied] = incomingStream[i];
pagefault++;
occupied++;
printFrame(queue, occupied);
}
else{

int max = INT_MIN;
int index;
// get LRU distance for each item in frame
for (int j = 0; j < frames; j++)
{
distance[j] = 0;
// traverse in reverse direction to find
// at what distance frame item occurred last
for(int k = i - 1; k >= 0; k--)
{
++distance[j];
if(queue[j] == incomingStream[k])
break;
}
// find frame item with max distance for LRU
// also notes the index of frame item in queue
// which appears furthest(max distance)
if(distance[j] > max){
max = distance[j];
index = j;
}
}
queue[index] = incomingStream[i];
printFrame(queue, occupied);
pagefault++;
}

printf("\n");
}
printf("Page Fault: %d",pagefault);
return 0;
}
```

OUTPUT:



```
acharya@DESKTOP-TAIKNEG:~$ ./os7b
Page      Frame1          Frame2          Frame3
1:              1
2:              1                         2
3:              1                         2                         3
2:              1                         2                         3
1:              1                         2                         3
5:              1                         2                         5
2:              1                         2                         5
1:              1                         2                         5
6:              1                         2                         6
2:              1                         2                         6
5:              5                         2                         6
6:              5                         2                         6
3:              5                         3                         6
1:              1                         3                         6
3:              1                         3                         6
Page Fault: 8acharya@DESKTOP-TAIKNEG:~$
```

Question to think:

A company is developing an operating system for devices with limited memory (e.g., smartwatches). Considering the memory constraints and access patterns of such devices, would you choose the FIFO or LRU page replacement algorithm? Apply the principles of each algorithm to justify which one would be more suitable in this context.

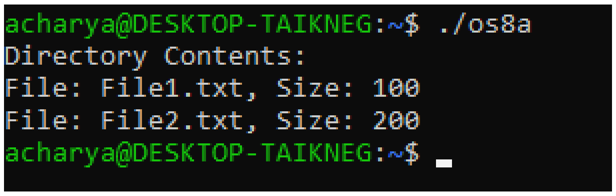
8. Simulate following File Organization Techniques a) Single level directory b) Two level directory.

a) Single level directory

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Maximum number of files in the directory

#define MAX_FILES 100

// Maximum file name length

#define MAX_NAME_LENGTH 256

// File structure to represent files

typedef struct File {

        char name[MAX_NAME_LENGTH];

        int size;

        char content[1024]; // For simplicity, we use a fixed content size

} File;

// Directory structure to hold files

typedef struct Directory {

        File files[MAX_FILES];

        int num_files;

} Directory;

// Function to create a new file

File createFile(const char* name, int size, const char* content)

{

        File newFile;

        strncpy(newFile.name, name, MAX_NAME_LENGTH);

        newFile.size = size;

        strncpy(newFile.content, content, sizeof(newFile.content));

        return newFile;

}
```

```
// Function to add a file to the directory
void addFileToDirectory(Directory* directory, File file) {
        if (directory->num_files < MAX_FILES) {
        directory->files[directory->num_files] = file;
        directory->num_files++;
        }
        else {
        printf("Directory is full. Cannot add more files.\n");
        }
}
// Function to display the contents of the directory
void displayDirectoryContents(const Directory* directory) {
        printf("Directory Contents:\n");
        for (int i = 0; i < directory->num_files; i++) {
                printf("File: %s, Size: %d\n", directory->files[i].name, directory->files[i].size);
        }
}
int main() {
        Directory directory;
        directory.num_files = 0;
        // Create and add files to the directory
        File file1 = createFile("File1.txt", 100, "This is the content of File1.");
        addFileToDirectory(&directory, file1);
        File file2 = createFile("File2.txt", 200, "Content of File2 goes here.");
        addFileToDirectory(&directory, file2);
        // Display the directory contents
        displayDirectoryContents(&directory);
        return 0;
}
```

OUTPUT:

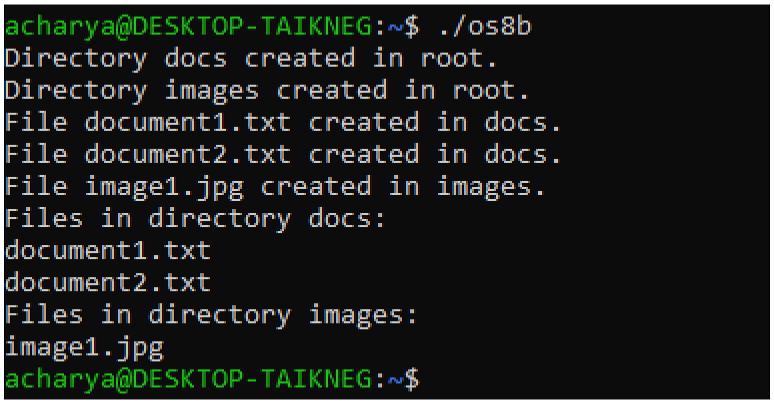


```
acharya@DESKTOP-TAIKNEG:~$ ./os8a
Directory Contents:
File: File1.txt, Size: 100
File: File2.txt, Size: 200
acharya@DESKTOP-TAIKNEG:~$
```

b) Two level directory

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_DIRS 100
#define MAX_FILES 100
struct FileEntry {
        char name[50];
        char content[1000];
};
struct Directory {
        char name[50];
        struct FileEntry files[MAX_FILES];
        int num_files;
};
int num_dirs = 0;
struct Directory directories[MAX_DIRS];
void createDirectory(char parent_name[], char dir_name[]) {
        if (num_dirs >= MAX_DIRS) {
        printf("Error: Maximum directories reached.\n");
        return;
}
for (int i = 0; i < num_dirs; i++) {
        if (strcmp(directories[i].name, parent_name) == 0) {
            if (directories[i].num_files >= MAX_FILES) {
                printf("Error: Maximum files reached in %s.\n", parent_name);
                return;
            }
            strcpy(directories[num_dirs].name, dir_name);
            directories[i].files[directories[i].num_files].content[0] = '\0';
            directories[i].num_files++;
            num_dirs++;
            printf("Directory %s created in %s.\n", dir_name, parent_name);
            return;
        }
    }

        printf("Error: Parent directory not found.\n");
    }

    void createFile(char dir_name[], char file_name[]) {
            for (int i = 0; i < num_dirs; i++) {
                if (strcmp(directories[i].name, dir_name) == 0) {
                    if (directories[i].num_files >= MAX_FILES) {
                            printf("Error: Maximum files reached in %s.\n", dir_name);
                            return;
                    }
                strcpy(directories[i].files[directories[i].num_files].name, file_name);
                directories[i].files[directories[i].num_files].content[0] = '\0';
                directories[i].num_files++;
                printf("File %s created in %s.\n", file_name, dir_name);
                return;
                }
            }
        printf("Error: Directory not found.\n");
}
void listFiles(char dir_name[]) {
        for (int i = 0; i < num_dirs; i++) {
            if (strcmp(directories[i].name, dir_name) == 0) {
                printf("Files in directory %s:\n", dir_name);
                for (int j = 0; j < directories[i].num_files; j++) {
                        printf("%s\n", directories[i].files[j].name);
                }
                return;
            }
        }
        printf("Error: Directory not found.\n");
}

int main() {
        strcpy(directories[0].name, "root");
        directories[0].num_files = 0;
        num_dirs++;
        char parent[50], dir[50], file[50];
        createDirectory("root", "docs");
        createDirectory("root", "images");
        createFile("docs", "document1.txt");
        createFile("docs", "document2.txt");
        createFile("images", "image1.jpg");
        listFiles("docs");
        listFiles("images");
        return 0;
}
```

OUTPUT:



```
acharya@DESKTOP-TAIKNEG:~$ ./os8b
Directory docs created in root.
Directory images created in root.
File document1.txt created in docs.
File document2.txt created in docs.
File image1.jpg created in images.
Files in directory docs:
document1.txt
document2.txt
Files in directory images:
image1.jpg
acharya@DESKTOP-TAIKNEG:~$
```

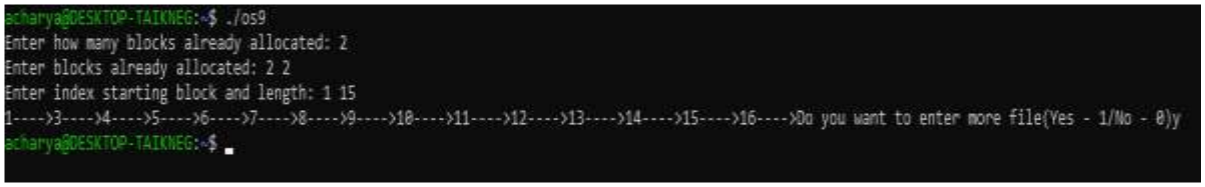
Question to think: -

A startup is designing an operating system for embedded devices used in home automation (e.g., smart thermostats or lights), which need to manage a limited number of files efficiently. Considering the trade-offs between simplicity and scalability, would you recommend using a single-level or two- level directory structure? Apply the characteristics of each to support your recommendation.

9) Develop a C program to simulate the Linked file allocation strategies.

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int f[50], p, i, st, len, j, c, k, a;
    for (i = 0; i < 50; i++)
        f[i] = 0;
    printf("Enter how many blocks already allocated: ");
    scanf("%d", &p);
    printf("Enter blocks already allocated: ");
    for (i = 0; i < p; i++) {
        scanf("%d", &a);
        f[a] = 1;
    }
x:
    printf("Enter index starting block and length: ");
    scanf("%d%d", &st, &len);
    k = len;
    if (f[st] == 0)
    {
        for (j = st; j < (st + k); j++){
            if (f[j] == 0){
                f[j] = 1;
                printf("%d---->", j);
            }
            else{
                //printf("%d Block is already allocated \n", j);
                k++;
            }
        }
    }
    else
        printf("%d starting block is already allocated \n", st);
    printf("Do you want to enter more file(Yes - 1/No - 0)");
    scanf("%d", &c);
    if (c == 1)
        goto x;
    else
        exit(0);
    return 0;
}
```

OUTPUT:



```
acharya@DESKTOP-TAIKNEG:~$ ./os9
Enter how many blocks already allocated: 2
Enter blocks already allocated: 2 2
Enter index starting block and length: 1 15
1---->3---->4---->5---->6---->7---->8---->9---->10---->11---->12---->13---->14---->15---->16---->Do you want to enter more file(Yes - 1/No - 0)y
acharya@DESKTOP-TAIKNEG:~$
```

Question to think:-

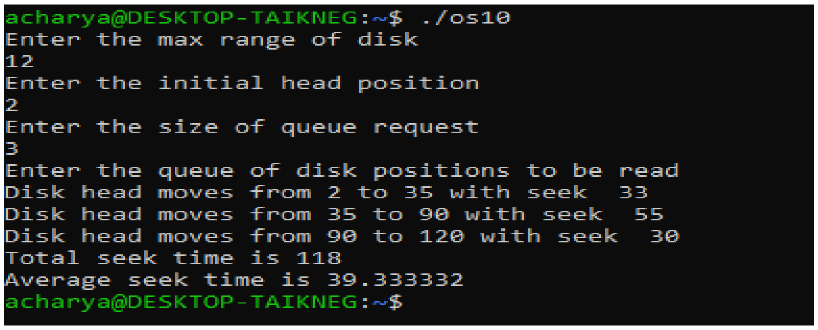
A media streaming application stores large audio and video files that are frequently accessed sequentially. As a system designer, would link file allocation be an appropriate strategy for managing these files on disk? Apply your understanding of linked allocation to justify your answer in terms of performance, access patterns, and storage management.

10) Develop a C program to simulate SCAN disk scheduling algorithm.

```
#include<stdio.h>

#include<stdlib.h>

int main()

{

    int queue[20],n,head,i,j,k,seek=0,max,diff,temp,queue1[20],queue2[20],

    temp1=0,temp2=0;

    float avg;

    printf("Enter the max range of disk\n");

    scanf("%d",&max);

    printf("Enter the initial head position\n");

    scanf("%d",&head);

    printf("Enter the size of queue request\n");

    scanf("%d",&n);

    printf("Enter the queue of disk positions to be read\n");

    int pos[] = {90,120,35,122,38,128,65,68};

    for(i=1;i<=n;i++)

    {

        //scanf("%d",&temp);

        temp = pos[i-1];

        if(temp>=head)

        {

            queue1[temp1]=temp;

             temp1++;

        }

        else

        {

            queue2[temp2]=temp;

            temp2++;
    }
  }
  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;
      }
   }
  }
  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;
      }
    }
  }
  for(i=1,j=0;j<temp1;i++,j++)
  queue[i]=queue1[j];
 //queue[i]=max;
```

```
  //queue[i+1]=0;
   for(i=temp1+1,j=0;j<temp2;i++,j++)
      queue[i]=queue2[j];
      queue[0]=head;
      for(j=0;j<=n-1;j++)
      {
      diff=abs(queue[j+1]-queue[j]);
      seek+=diff;
      printf("Disk head moves from %d to %d with seek  %d\n",queue[j],queue[j+1],diff);
      }
           printf("Total seek time is %d\n",seek);
           avg=seek/(float)n;
           printf("Average seek time is %f\n",avg);
           return 0;
}
```

OUTPUT:



```
acharya@DESKTOP-TAIKNEG:~$ ./os10
Enter the max range of disk
12
Enter the initial head position
2
Enter the size of queue request
3
Enter the queue of disk positions to be read
Disk head moves from 2 to 35 with seek  33
Disk head moves from 35 to 90 with seek  55
Disk head moves from 90 to 120 with seek  30
Total seek time is 118
Average seek time is 39.333332
acharya@DESKTOP-TAIKNEG:~$
```

Question to think:

A data center manages a high-volume database server with frequent and unpredictable read/write requests across the disk. As a system engineer, would you choose the SCAN disk scheduling algorithm to manage disk I/O operations? Apply the characteristics of SCAN to justify your decision in terms of seek time, fairness, and request handling efficiency.