GLOBAL ACADEMY OF TECHNOLOGY

Autonomous Institution affiliated to VTU, Belagavi,
Accredited by NAAC with 'A' grade
Ideal Homes Township , RR Nagar, Bengaluru – 560098

Department of Computer Science and Engineering (AI&ML)

OPERATING SYSTEMS Laboratory Manual

III Semester
Course Code: BCI24302

Academic Year 2025-26

Version 1.0
W.e.f, March 2025

Prepared By
Mr. Dinesh M
Assistant Professor , Dept. of CSE(AI&ML)

Approved by
HOD,
Dept. of CSE(AI&ML)

DOCUMENT LOG

Name of the document	**Operating Systems Laboratory Manual**
Scheme	**2024**
Current version number and date	**Version 1.0 / 10.03.2025**
Subject code	**BCI24302**
Prepared by	**Mr Dinesh M , Dept. of CSE(AI&ML)**
Approved by	**HOD, Dept. of CSE(AI&ML)**

Table of Contents

Vision of the Institute

Become a premier institution imparting quality education in engineering and management tomeet the changing needs of society.

Mission of the Institute

M1: Create environment conductive for continuous learning through quality teaching andlearning processes supported by modern infrastructure.

M2: Promote research and innovation through collaboration with industries.

M3: Inculcate ethical values and environmental conscious through holistic education programs.

Vision of the Department

To Become a leading hub of excellence in education, research in the field of Computer Science and Engineering providing AI-driven solutions with holistic development for the needs of the Society.

Mission of the Department

- To Provide aspiring engineers for industry and academia by offering excellent education in emerging AI techniques.
- To equip value-added technical and research-oriented education by satisfying societal needs
- To educate with professional integrity values and ethics for the environment awareness.

PROGRAM EDUCATIONAL OBJECTIVES(PEOs)

- **PEO1:** Design and Develop innovative intelligent systems for the welfare of the Society.
- **PEO2:** Engage in lifelong learning process through higher education and to inculcate innovative ideas in research field.
- **PEO3:** Lead the IT Industry with management and Entrepreneurship skills.

PROGRAM SPECIFIC OUTCOMES(PSOs)

- **PSO1:** To Produce graduates with industry-ready skills with the knowledge of Computer Science & Machine Learning technology based to solve real world problems.
- **PSO2:** Ability to develop many successful applications and designing efficient algorithms for intelligent systems within the realm of artificial intelligence incorporating in data analytics, Natural language processing and Internet of Things.

PROGRAM OUTCOMES (PO's)

PO1: Engineering Knowledge:
Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.

PO2: Problem Analysis:
Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development (WK1 to WK4).

PO3: Design/Development of Solutions:
Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required (WK5).

PO4: Conduct Investigations of Complex Problems:
Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions (WK8).

PO5: Engineering Tool Usage:
Create, select, and apply appropriate techniques, resources, and modern engineering & IT tools, including prediction, and modelling recognizing their limitations to solve complex engineering problems (WK2 and WK6).

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 (WK1, WK5, and WK7).

PO7: Ethics:
Apply ethical principles and commit to professional ethics, human values, diversity, and inclusion; adhere to national & international laws (WK9).

PO8: Individual and Collaborative Teamwork:
Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.

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.

PO10: Project Management and Finance:
Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.

PO11: Life-Long Learning:
Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies, and iii) critical thinking in the broadest context of technological change (WK8).

COURSE OUTCOMES:

Upon successful completion of this course, students are able to:

CO1	Explain core OS concepts – processes, scheduling, synchronization, memory, file systems, storage, and protection.
CO2	Apply scheduling and synchronization techniques for process and resource management.
CO3	Demonstrate memory allocation, paging, segmentation, and file system organization.
CO4	Analyze the performance of OS algorithms in scheduling, memory, and file systems.

WKs are Washington Accord's Knowledge & Attitude Profiles ranging from WK1 to WK9

Syllabus

Department: **Computer Science and Engineering (AI & ML)**			
Semester: **III**	Course Code: **BCI24302**		Contact Hrs /week: **3**
Course Description: **Operating Systems**			No. of Credits:**4** L : T : P : S = **3:0:2:0**
Course Category: **IPCC**			Total no. of Hours = 50
CIE: **50 Marks**		SEE: **50 Marks**	Exam Hours: **03**
Course Pre-requisites: Programming in C			

Course Learning Objectives:

The course will enable students to:

CLO1	To Demonstrate the need for OS and different types of OS
CLO2	To discuss suitable techniques for management of different resources
CLO3	To demonstrate different APIs/Commands related to processor

Sl. No.	**EXPERIMENTS**
1	Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process)
2	Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority
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.
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
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.

Mapping of CO-PO:

POs → COs ↓	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11
CO1	3	2	2	-	2	-	-	-	-	-	2
CO2	2	3	2	-	2	-	-	-	-	-	2
CO3	2	2	2	-	2	-	-	-	-	-	2
CO4	2	3	2	-	3	-	-	-	-	-	2

Correlation Weightage: 1 – Low, 2 – Moderate, 3 - High

MARKS DISTRIBUTION

	Component	Marks	Total Marks
	CIE Test-1	**30**	
	CIE Test-2	**30**	
CIE	Lab	**20**	**50**
SEE	Semester End Examination	**50**	**50**
Grand Total			**100**

LAB EVALUATION PROCESS

Continuous Internal Evaluation (CIE)

Evaluation of CIE		
S. No	**Activity**	**Marks**
1	Average of Weekly Entries	10
2	Internal Assessment reduced to	10
TOTAL		**20**

LAB RUBRICS

Rubrics for Evaluation of Observation

Attribute	Max. Marks	Good	Satisfactory	Poor
		4-5	2-3	0-1
Writeup	**05**	Writes the program without errors.	Writes the program with few mistakes.	Unable to write the program.
	Max. Marks	**9-10**	**5-8**	**0-4**
Execution of program	**10**	• Debugs the program independently. • Executed the program for all possible inputs.	• Works with little help from faculty. • All possible input cases not covered.	• Unable to complete the execution of the program within the lab session.
	Max. Marks	**3-5**	**1-2**	**0**
Viva Voce	**05**	• Able to explain the logic of the program. • Answered all questions.	• Partially understood the logic of the program. • Answered few questions.	• Not understood the logic of the program. • Not answering any questions

Rubrics for Evaluation of Record

Attribute	Max. Marks	9-10	5-8	0-4
Complete Program with Algorithm (wherever applicable) and all possible Outputs	**10**	• Complete program, algorithm and outputs.	• Writes program with few mistakes • Outputs are not clear. • Late submissions.	• Not submitted the records. • Incomplete program • Not written algorithms

Rubrics for Evaluation of Internal Test

Attribute	Max Marks	Good	Satisfactory	Poor
		4-5	2-3	0-1
Writeup	**05**	• Completes source code. • No syntax and logical errors. • All possible inputs listed with expected output.	• Completes source code. • Syntax and logical errors exist. • All possible inputs listed with expected output.	• Incomplete source code. • Change of program.
	Max Marks	**9 -10**	**5 - 8**	**0-4**
Execution	**10**	Able to debug the program and get the right set of outputs.	• Program, executed for only few input cases. • With few errors.	• Program with many errors. • Not executed.
	Max Marks	**3-5**	**1-2**	**0**
Viva Voce	**05**	Answering all questions.	Answering few questions.	Not Able to answer basic questions.

Evaluation Sheet

Sl. No	Date	Particulars	Page No.	Marks W+E+V (5M+10M+5M)	Total (20M)	Sign
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
Average (Max. 20)						

CHAPTER 1 INTRODUCTION

The Operating System is a fundamental component of computer science and engineering, serving as the interface between hardware and software. This course provides an in-depth understanding of how operating systems function, manage resources, and provide services for computer programs. Students will explore core concepts such as process management, memory management, file systems, input/output handling, and security. By studying both theoretical foundations and practical implementations, learners will gain the skills necessary to analyze, design, and work with modern operating systems, equipping them for further study and professional practice in system-level programming and software development..

1. Process Creation and System Calls (fork, exec, wait, create, terminate)

Modern operating systems enable multitasking by managing multiple processes concurrently. Process system calls like fork(), exec(), and wait() are fundamental in creating and controlling processes. The fork() system call creates a new child process, exec() replaces the process image, and wait() is used for synchronization. These system calls form the core of process management in Unix-like systems.

Application: Essential in developing multi-process applications like shells, servers, and system daemons.

2. CPU Scheduling Algorithms: FCFS, SJF, Round Robin, Priority

CPU scheduling determines which process will use the CPU at a given time. Different algorithms like First-Come First-Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling impact system responsiveness, throughput, and waiting time.

Application: These strategies are implemented in OS kernels and real-time systems to optimize CPU utilization and improve performance under different workloads.

3. Producer-Consumer Problem using Semaphores

The producer-consumer problem models synchronization between processes sharing a bounded buffer. Semaphores, a type of signaling mechanism, help coordinate access without conflicts or data corruption. Implementing this model illustrates the concept of mutual exclusion and process synchronization in concurrent programming.

Application: Found in concurrent applications like thread pools, message queues, and buffering systems in OS and embedded devices.

4. Interprocess Communication (IPC) using mkfifo and File Descriptors

Interprocess Communication allows processes to exchange data and coordinate actions. Using named pipes (FIFOs) and system calls like open(), read(), and write(), processes can communicate through shared file-like interfaces.

Application: Used in client-server models, daemon communication, shell pipelines, and message passing in distributed systems.

5. Banker's Algorithm for Deadlock Avoidance

The Banker's Algorithm is a resource allocation and deadlock avoidance algorithm. It ensures that the system remains in a safe state before allocating resources to any process.

Application: Helps in systems where resource requests can be predicted and avoided from causing deadlocks, such as in database transactions and multi-threaded systems.

6. Contiguous Memory Allocation Techniques: Worst Fit, Best Fit, First Fit

Memory management is crucial for efficient program execution. In contiguous allocation, memory is assigned in a single block. The First Fit, Best Fit, and Worst Fit strategies determine how free memory blocks are selected for allocation.

Application: Forms the basis of memory allocators in early OSes and is useful for understanding fragmentation issues and allocation efficiency.

7. Page Replacement Algorithms: FIFO and LRU

In virtual memory systems, page replacement algorithms decide which memory page to replace when a page fault occurs. FIFO (First-In-First-Out) and LRU (Least Recently Used) are classic algorithms that balance performance and simplicity.

Application: Used in virtual memory management in modern OSes like Linux and Windows to optimize memory access patterns.

8. File Organization Techniques: Single-Level and Two-Level Directory

File systems manage how files are stored and accessed. Single-level directories use a flat namespace, while two-level directories provide user-based isolation. Simulating these systems highlights how directory structures impact file access, security, and naming conflicts.

Application: Used in simple embedded systems (single-level) and multi-user systems (two-level or hierarchical directories).

9. Linked File Allocation Strategy

In linked allocation, each file is a linked list of disk blocks, where each block contains a pointer to the next. This avoids external fragmentation and simplifies allocation but has slower random access times.

Application: Suitable for sequential file access systems and early versions of file systems like MS-DOS.

10. SCAN Disk Scheduling Algorithm

Disk scheduling algorithms manage read/write requests to a disk. The SCAN algorithm (also known as the elevator algorithm) moves the disk arm back and forth across the disk, servicing requests in one direction before reversing.

Application: SCAN and its variants are used in modern disk scheduling to reduce seek time and improve throughput in HDDs.

CHAPTER - 2 SYLLABUS PROGRAMS (SOURCE CODE)

LAB PROGRAMS

PROGRAM 1

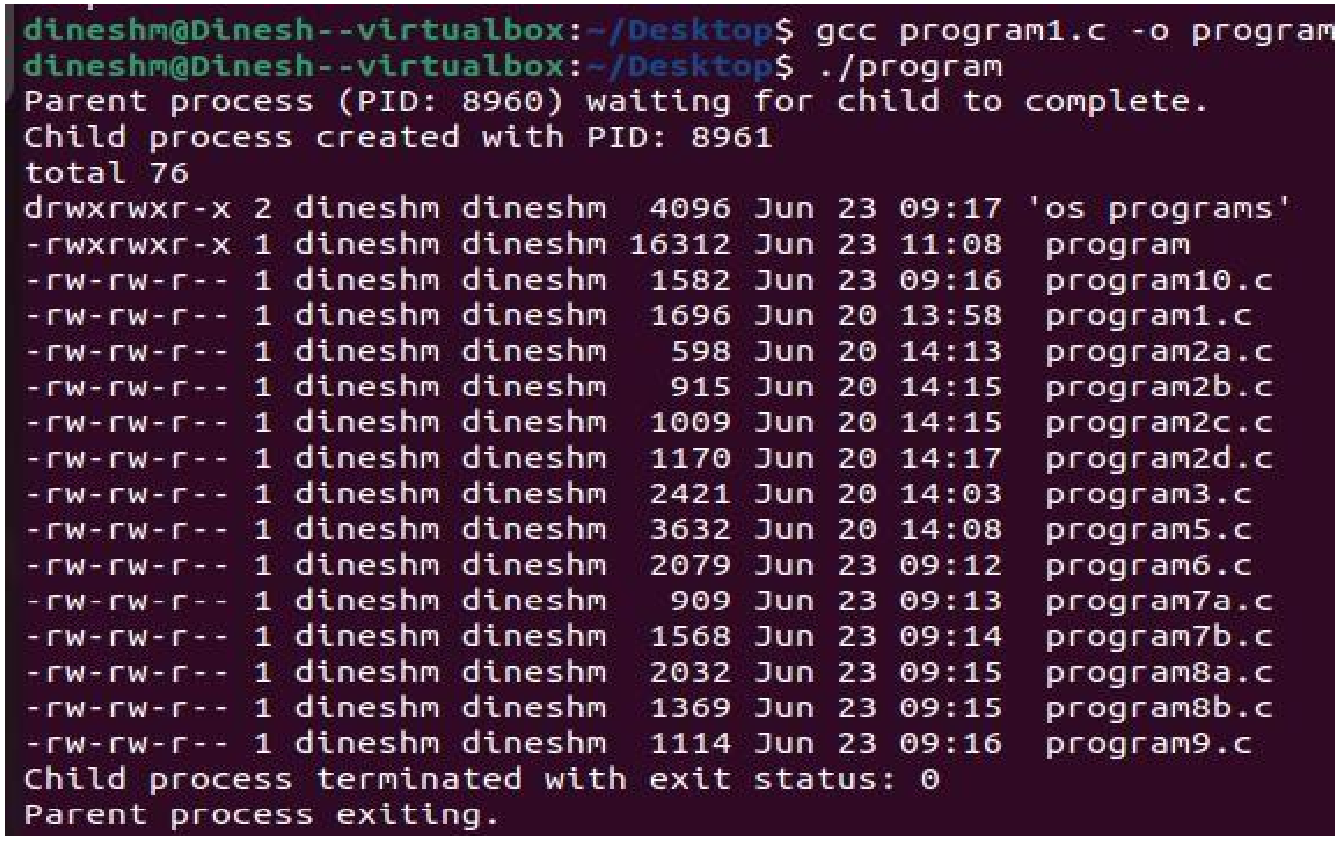
Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process)

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main() {
    pid_t pid;
    pid = fork();
    if (pid < 0)
        { perror("Fork
        failed"); exit(1);
    } else if (pid == 0) {
        printf("Child  process  created  with  PID:  %d\n",
getpid());
        char *args[] = {"ls", "-l", NULL};
        if (execvp(args[0], args) == -1)
            { perror("Exec failed");
            exit(1);
        }
    } else {
        printf("Parent process (PID: %d) waiting for child to
complete.\n", getpid());
```

```
        int status;
        waitpid(pid, &status, 0);
        if (WIFEXITED(status)) {
            printf("Child    process    terminated    with    exit
status: %d\n", WEXITSTATUS(status));
        } else {
            printf("Child process terminated abnormally.\n");
        }
        printf("Parent process exiting.\n");
    }
    return 0;
}
```

Expected Output:



```
dineshm@Dinesh--virtualbox:~/Desktop$ gcc program1.c -o program
dineshm@Dinesh--virtualbox:~/Desktop$ ./program
Parent process (PID: 8960) waiting for child to complete.
Child process created with PID: 8961
total 76
drwxrwxr-x 2 dineshm dineshm  4096 Jun 23 09:17 'os programs'
-rwxrwxr-x 1 dineshm dineshm 16312 Jun 23 11:08  program
-rw-rw-r-- 1 dineshm dineshm  1582 Jun 23 09:16  program10.c
-rw-rw-r-- 1 dineshm dineshm  1696 Jun 20 13:58  program1.c
-rw-rw-r-- 1 dineshm dineshm   598 Jun 20 14:13  program2a.c
-rw-rw-r-- 1 dineshm dineshm   915 Jun 20 14:15  program2b.c
-rw-rw-r-- 1 dineshm dineshm  1009 Jun 20 14:15  program2c.c
-rw-rw-r-- 1 dineshm dineshm  1170 Jun 20 14:17  program2d.c
-rw-rw-r-- 1 dineshm dineshm  2421 Jun 20 14:03  program3.c
-rw-rw-r-- 1 dineshm dineshm  3632 Jun 20 14:08  program5.c
-rw-rw-r-- 1 dineshm dineshm  2079 Jun 23 09:12  program6.c
-rw-rw-r-- 1 dineshm dineshm   909 Jun 23 09:13  program7a.c
-rw-rw-r-- 1 dineshm dineshm  1568 Jun 23 09:14  program7b.c
-rw-rw-r-- 1 dineshm dineshm  2032 Jun 23 09:15  program8a.c
-rw-rw-r-- 1 dineshm dineshm  1369 Jun 23 09:15  program8b.c
-rw-rw-r-- 1 dineshm dineshm  1114 Jun 23 09:16  program9.c
Child process terminated with exit status: 0
Parent process exiting.
```

PROGRAM 2

Simulate the following CPU scheduling algorithms to find turnaround time and waiting time

a) FCFS
b) SJF
c) Round Robin
d) Priority.

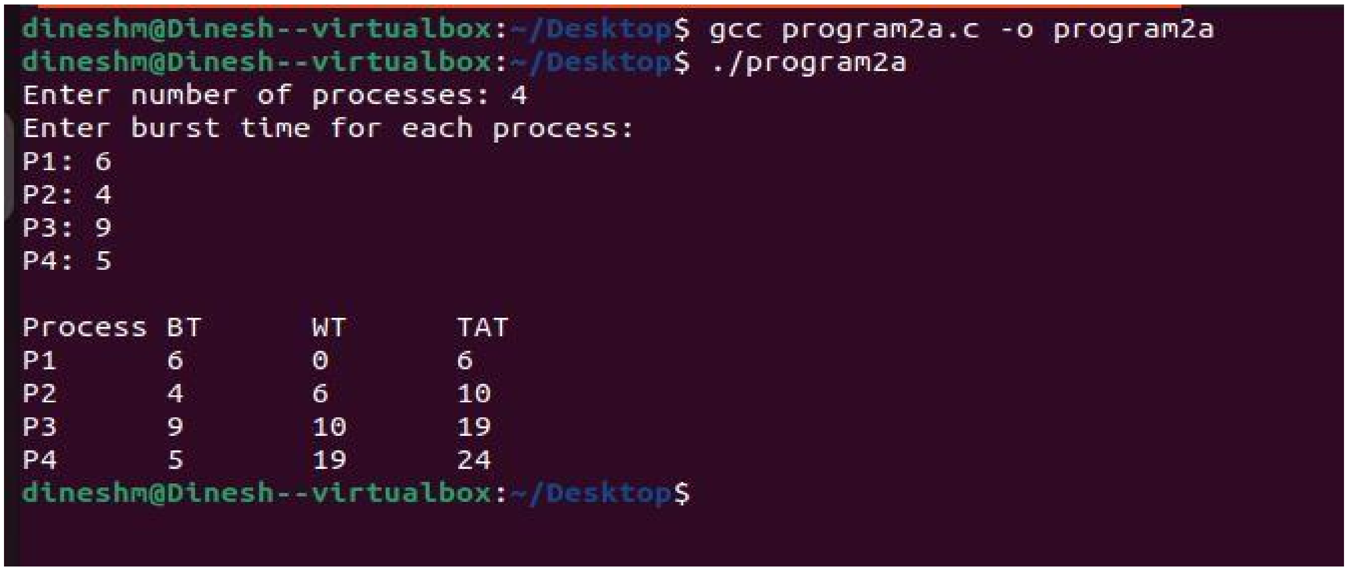
Source code:

a) FCFS

```
#include <stdio.h>
int main() {
    int n, i;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    int bt[n], wt[n], tat[n];
    printf("Enter burst time for each process:\n");
    for (i = 0; i < n; i++) {
        printf("P%d: ", i + 1);
        scanf("%d", &bt[i]);
    }
    wt[0] = 0;
    for (i = 1; i < n; i++)
        wt[i] = wt[i - 1] + bt[i - 1];
    for (i = 0; i < n; i++)
        tat[i] = wt[i] + bt[i];
    printf("\nProcess\tBT\tWT\tTAT\n");
    for (i = 0; i < n; i++)
```

```
        printf("P%d\t%d\t%d\t%d\n", i + 1, bt[i], wt[i], tat[i]);
    return 0;
}
```

Expected Output:

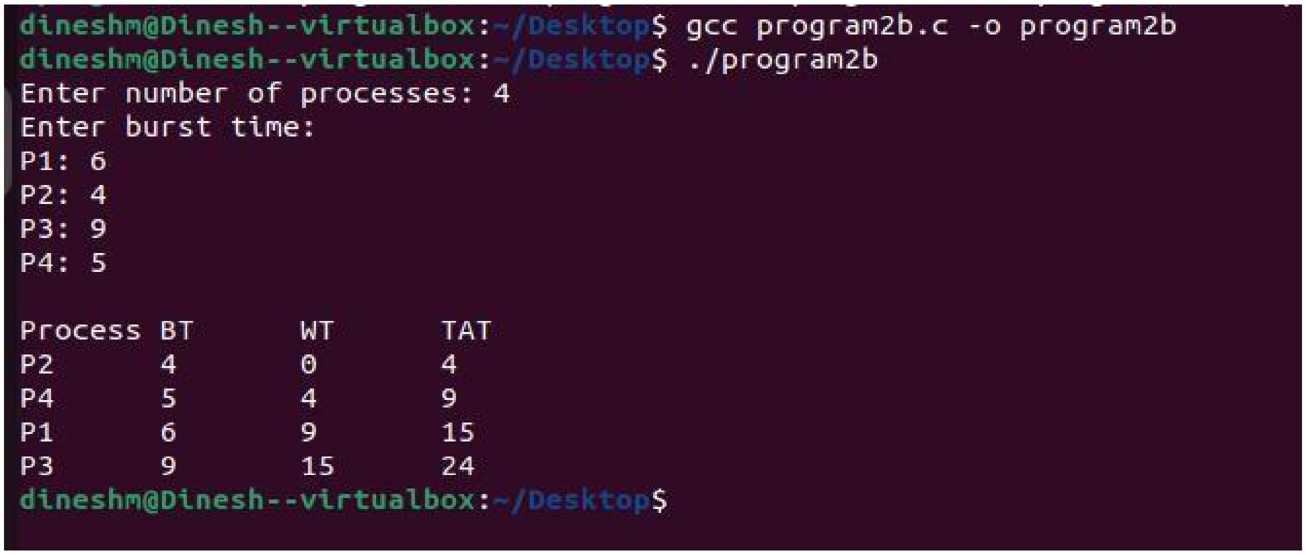


```
dineshm@Dinesh--virtualbox:~/Desktop$ gcc program2a.c -o program2a
dineshm@Dinesh--virtualbox:~/Desktop$ ./program2a
Enter number of processes: 4
Enter burst time for each process:
P1: 6
P2: 4
P3: 9
P4: 5

Process BT      WT      TAT
P1      6       0       6
P2      4       6       10
P3      9       10      19
P4      5       19      24
dineshm@Dinesh--virtualbox:~/Desktop$
```

B) SJF.

```
#include <stdio.h>
int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    int bt[n], p[n], wt[n], tat[n], i, j, temp;
    printf("Enter burst time:\n");
    for (i = 0; i < n; i++)
        { printf("P%d: ", i + 1);
        scanf("%d", &bt[i]);
        p[i] = i + 1;
    }
    // Sort by burst time
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (bt[j] > bt[j + 1]) {
               temp = bt[j]; bt[j] = bt[j + 1]; bt[j + 1] = temp;
               temp = p[j];  p[j] = p[j + 1];    p[j + 1] = temp;
            }
        }
    }
    wt[0] = 0;
    for (i = 1; i < n; i++)
        wt[i] = wt[i - 1] + bt[i - 1];
    for (i = 0; i < n; i++)
        tat[i] = wt[i] + bt[i];
    printf("\nProcess\tBT\tWT\tTAT\n");
    for (i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\n", p[i], bt[i], wt[i], tat[i]);
    return 0;
}
```

Expected Output:



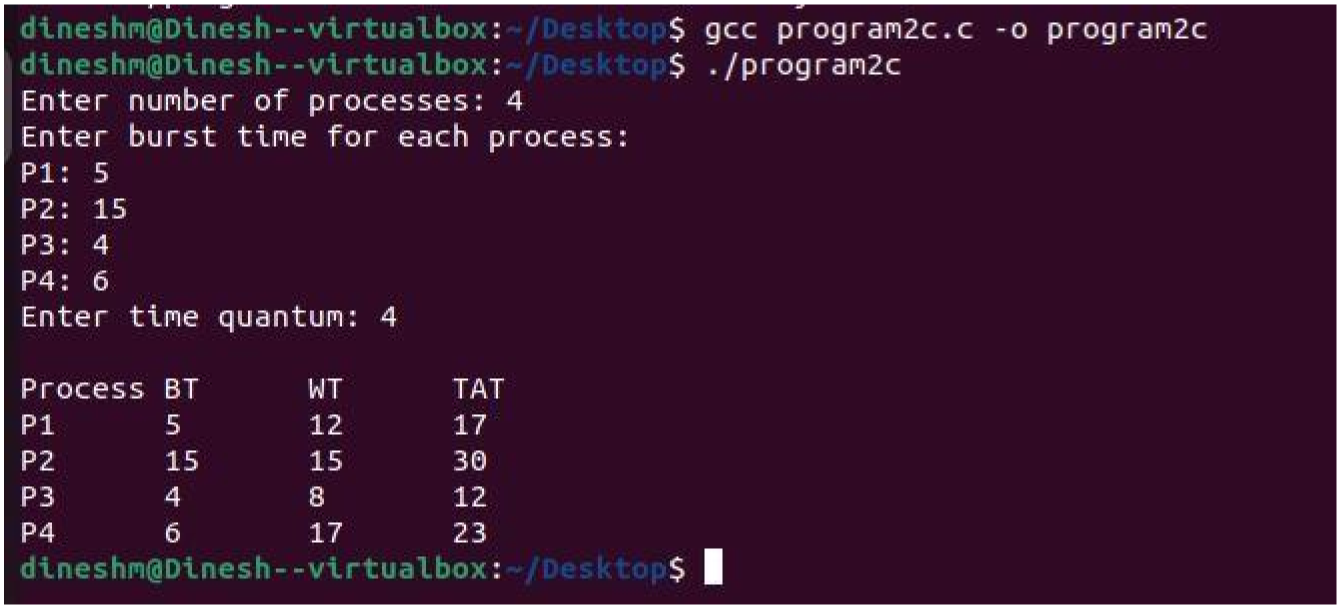
```
dineshm@Dinesh--virtualbox:~/Desktop$ gcc program2b.c -o program2b
dineshm@Dinesh--virtualbox:~/Desktop$ ./program2b
Enter number of processes: 4
Enter burst time:
P1: 6
P2: 4
P3: 9
P4: 5

Process BT      WT      TAT
P2      4       0       4
P4      5       4       9
P1      6       9       15
P3      9       15      24
dineshm@Dinesh--virtualbox:~/Desktop$
```

C) Round Robin.

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

```
int main() {
    int n, tq, i, time = 0, remain;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    int bt[n], rt[n], wt[n], tat[n];
    printf("Enter burst time for each process:\n");
    for (i = 0; i < n; i++) {
        printf("P%d: ", i + 1);
        scanf("%d", &bt[i]);
        rt[i] = bt[i];
    }
    printf("Enter time quantum: ");
    scanf("%d", &tq);
    remain = n;
    while (remain != 0) {
        for (i = 0; i < n; i++)
            { if (rt[i] > 0) {
                if (rt[i] > tq)
                    { time += tq;
                    rt[i] -= tq;
                } else {
                    time += rt[i];
                    tat[i] = time;
                    wt[i] = tat[i] - bt[i];
                    rt[i] = 0;
                    remain--;
                }
            }
        }
    }
    printf("\nProcess\tBT\tWT\tTAT\n");
    for (i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\n", i + 1, bt[i], wt[i], tat[i]);
    return 0;
}
```

Expected Output:

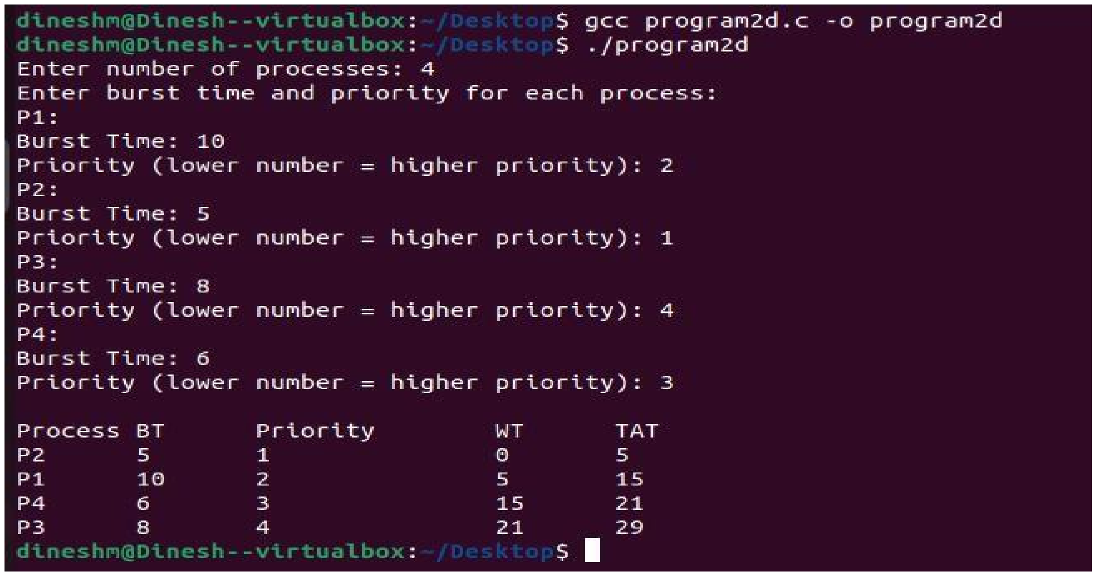


```
dineshm@Dinesh--virtualbox:~/Desktop$ gcc program2c.c -o program2c
dineshm@Dinesh--virtualbox:~/Desktop$ ./program2c
Enter number of processes: 4
Enter burst time for each process:
P1: 5
P2: 15
P3: 4
P4: 6
Enter time quantum: 4

Process BT      WT      TAT
P1      5       12      17
P2      15      15      30
P3      4       8       12
P4      6       17      23
dineshm@Dinesh--virtualbox:~/Desktop$
```

D) Priority.

```
#include <stdio.h>
int main() {
    int n, i, j;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    int bt[n], p[n], priority[n], wt[n], tat[n], temp;
    printf("Enter burst time and priority for each process:\n");
    for (i = 0; i < n; i++) {
        printf("P%d:\n", i + 1);
        printf("Burst Time: "); scanf("%d", &bt[i]);
        printf("Priority (lower number = higher priority): ");
scanf("%d", &priority[i]);
        p[i] = i + 1;
    }
    // Sort based on priority
    for (i = 0; i < n - 1; i++) {
        for (j = i + 1; j < n; j++) {
            if (priority[i] > priority[j]) {
                temp = priority[i]; priority[i] = priority[j];
priority[j] = temp;
                temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
                temp = p[i]; p[i] = p[j]; p[j] = temp;
            }
        }
    }
    wt[0] = 0;
    for (i = 1; i < n; i++)
        wt[i] = wt[i - 1] + bt[i - 1];
    for (i = 0; i < n; i++)
        tat[i] = wt[i] + bt[i];
    printf("\nProcess\tBT\tPriority\tWT\tTAT\n");
    for (i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t\t%d\t%d\n", p[i], bt[i],
priority[i], wt[i], tat[i]);
    return 0;
}
```

Expected Output:



```
dineshm@Dinesh--virtualbox:~/Desktop$ gcc program2d.c -o program2d
dineshm@Dinesh--virtualbox:~/Desktop$ ./program2d
Enter number of processes: 4
Enter burst time and priority for each process:
P1:
Burst Time: 10
Priority (lower number = higher priority): 2
P2:
Burst Time: 5
Priority (lower number = higher priority): 1
P3:
Burst Time: 8
Priority (lower number = higher priority): 4
P4:
Burst Time: 6
Priority (lower number = higher priority): 3

Process BT      Priority        WT      TAT
P2      5       1               0       5
P1      10      2               5       15
P4      6       3               15      21
P3      8       4               21      29
dineshm@Dinesh--virtualbox:~/Desktop$
```

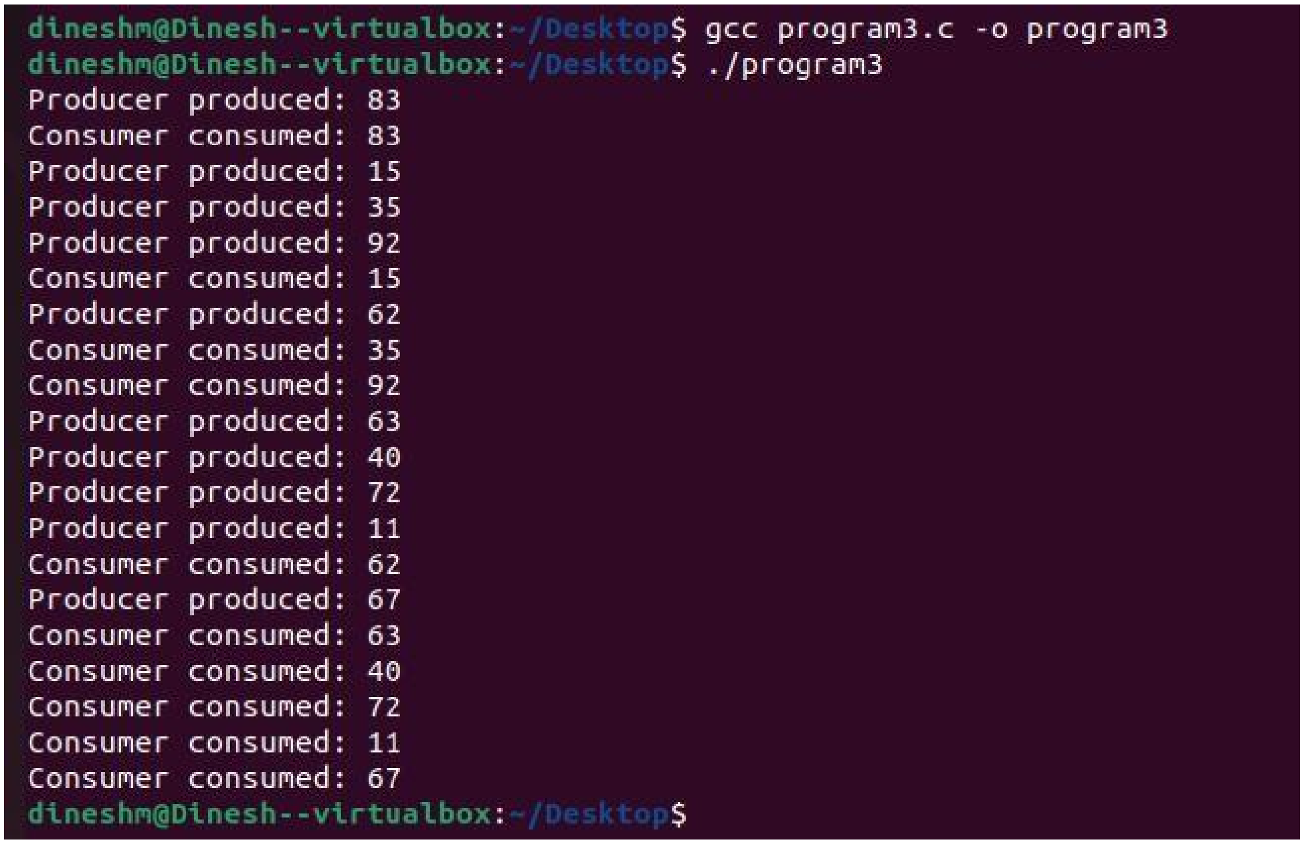
PROGRAM 3

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

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define MAX_BUFFER_SIZE 5 // Maximum size of the buffer
int buffer[MAX_BUFFER_SIZE]; // Shared buffer
int in = 0; // Index for next item to produce
int out = 0; // Index for next item to consume

sem_t empty; // Semaphore to count empty slots in the buffer
sem_t full; // Semaphore to count full slots in the buffer
pthread_mutex_t mutex; // Mutex for critical section

void*  producer(void*  arg)
    { int item;
    for (int i = 0; i < 10; i++) { // Produce 10 items
        item = rand() % 100; // Generate a random item
        sem_wait(&empty); // Wait for an empty slot
        pthread_mutex_lock(&mutex); // Enter critical section

        // Produce the item and put it in the buffer
        buffer[in] = item;
        printf("Producer produced: %d\n", item);
        in = (in + 1) % MAX_BUFFER_SIZE; // Circular buffer
        pthread_mutex_unlock(&mutex); // Exit critical section
        sem_post(&full); // Signal that a new item is produced
        sleep(rand() % 2); // Sleep for a random time
    }
    return NULL;
}

void* consumer(void* arg)
    { int item;
    for (int i = 0; i < 10; i++) { // Consume 10 items
        sem_wait(&full); // Wait for a full slot
        pthread_mutex_lock(&mutex); // Enter critical section
        // Consume the item from the buffer
        item = buffer[out];
        printf("Consumer consumed: %d\n", item);
        out = (out + 1) % MAX_BUFFER_SIZE; // Circular buffer
        pthread_mutex_unlock(&mutex); // Exit critical section
        sem_post(&empty); // Signal that an item is consumed
        sleep(rand() % 2); // Sleep for a random time
    }
    return NULL;
}
int main() {
    pthread_t prod_thread, cons_thread;
    // Initialize semaphores and mutex
    sem_init(&empty, 0, MAX_BUFFER_SIZE);
    sem_init(&full, 0, 0); // Initially, no items are produced
    pthread_mutex_init(&mutex, NULL); // Initialize the mutex
    // Create producer and consumer threads
    pthread_create(&prod_thread, NULL, producer, NULL);
    pthread_create(&cons_thread, NULL, consumer, NULL);
    // Wait for threads to finish
    pthread_join(prod_thread, NULL);
    pthread_join(cons_thread, NULL);
    // Cleanup
    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);
    return 0;
}
```

Expected Output:



```
dineshm@Dinesh--virtualbox:~/Desktop$ gcc program3.c -o program3
dineshm@Dinesh--virtualbox:~/Desktop$ ./program3
Producer produced: 83
Consumer consumed: 83
Producer produced: 15
Producer produced: 35
Producer produced: 92
Consumer consumed: 15
Producer produced: 62
Consumer consumed: 35
Consumer consumed: 92
Producer produced: 63
Producer produced: 40
Producer produced: 72
Producer produced: 11
Consumer consumed: 62
Producer produced: 67
Consumer consumed: 63
Consumer consumed: 40
Consumer consumed: 72
Consumer consumed: 11
Consumer consumed: 67
dineshm@Dinesh--virtualbox:~/Desktop$
```

PROGRAM 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.

Source code:

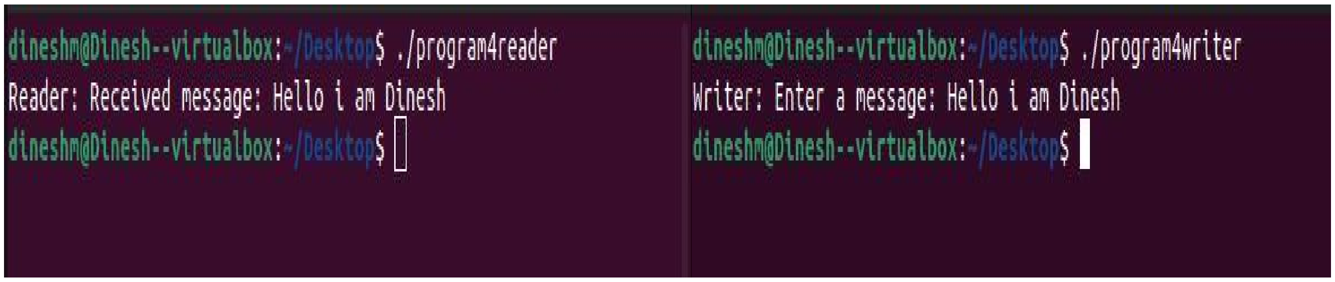
A) Writer.c program

```
#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

int main() {

    int fd;

    char *fifo = "/tmp/myfifo";

    mkfifo(fifo, 0666);

    char arr1[80];

    printf("Writer: Enter a message: ");

    fgets(arr1, 80, stdin);

    fd = open(fifo, O_WRONLY);

    write(fd, arr1, sizeof(arr1));

    close(fd);

    return 0;
}
```

B)Reader.c program

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

int main() {
    int fd;
    char *fifo = "/tmp/myfifo";
    char arr2[80];

    fd = open(fifo, O_RDONLY);
    read(fd, arr2, sizeof(arr2));
    printf("Reader: Received message: %s", arr2);
    close(fd);

    return 0;
}
```

Expected Output:



```
dineshm@Dinesh--virtualbox:~/Desktop$ ./program4reader
Reader: Received message: Hello i am Dinesh
dineshm@Dinesh--virtualbox:~/Desktop$
```

```
dineshm@Dinesh--virtualbox:~/Desktop$ ./program4writer
Writer: Enter a message: Hello i am Dinesh
dineshm@Dinesh--virtualbox:~/Desktop$
```

PROGRAM 5

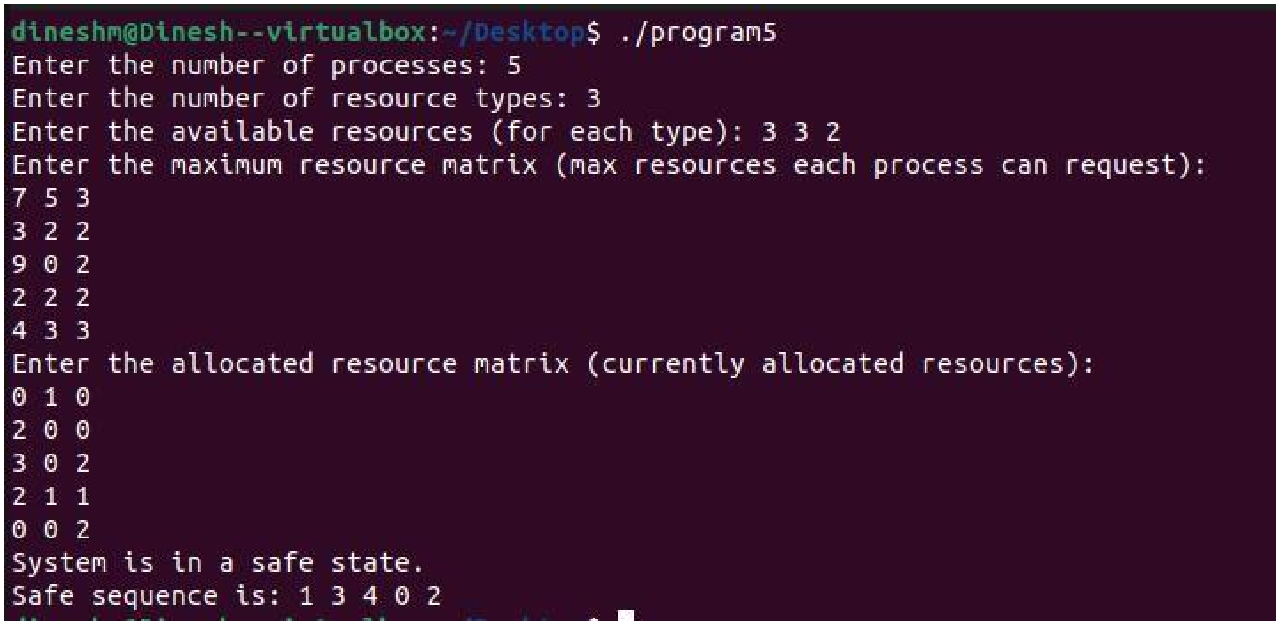
Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.

Source code:

```
#include <stdio.h>

#define MAX_PROCESSES 5

#define MAX_RESOURCES 3

void calculateNeed(int need[MAX_PROCESSES][MAX_RESOURCES], int
max[MAX_PROCESSES][MAX_RESOURCES],int
allocated[MAX_PROCESSES][MAX_RESOURCES], int n, int m)

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

        for (int j = 0; j < m; j++) {

            need[i][j] = max[i][j] - allocated[i][j];

        }

    }

}

int isSafe(int processes[],int avail[],int max[][MAX_RESOURCES],
int allocated[][MAX_RESOURCES], int n, int m) {

    int need[MAX_PROCESSES][MAX_RESOURCES];

    calculateNeed(need, max, allocated, n, m);

    int finish[MAX_PROCESSES] = {0};

    int safeSeq[MAX_PROCESSES];

    int work[MAX_RESOURCES];

    for (int i = 0; i < m; i++)

        { work[i] = avail[i];

    }

    int count = 0;

    while (count < n) {

        int found = 0;
        for (int p = 0; p < n; p++)
            { if (finish[p] == 0) {
                int j;
                for (j = 0; j < m; j++) {
                    if (need[p][j] > work[j])
                        break;
                }
                if (j == m) {
                    for (int k = 0; k < m; k++)
                        work[k] += allocated[p][k];

                    safeSeq[count++] = p;
                    finish[p] = 1;
                    found = 1;
                }
            }
        }
        if (found == 0) {
            printf("System is not in a safe state.\n");
            return 0;
        }
    }
    printf("System is in a safe state.\nSafe sequence is: ");
    for (int i = 0; i < n; i++)
        printf("%d ", safeSeq[i]);
    printf("\n");

    return 1;
```

```
}
int main() {
    int n, m;
    int processes[MAX_PROCESSES];
    int avail[MAX_RESOURCES];
    int max[MAX_PROCESSES][MAX_RESOURCES];
    int allocated[MAX_PROCESSES][MAX_RESOURCES];
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the number of resource types: ");
    scanf("%d", &m);
    for (int i = 0; i < n; i++)
        { processes[i] = i;
    }
    printf("Enter the available resources: ");
    for (int i = 0; i < m; i++) {
        scanf("%d", &avail[i]);
    }
    printf("Enter the maximum resource matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++)
            { scanf("%d", &max[i][j]);
        }
    }
    printf("Enter the allocated resource matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++)
            { scanf("%d", &allocated[i][j]);
        }
    }
    isSafe(processes, avail, max, allocated, n, m);
    return 0;
}
```

Expected output:



```
dineshm@Dinesh--virtualbox:~/Desktop$ ./program5
Enter the number of processes: 5
Enter the number of resource types: 3
Enter the available resources (for each type): 3 3 2
Enter the maximum resource matrix (max resources each process can request):
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the allocated resource matrix (currently allocated resources):
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
System is in a safe state.
Safe sequence is: 1 3 4 0 2
```

PROGRAM 6

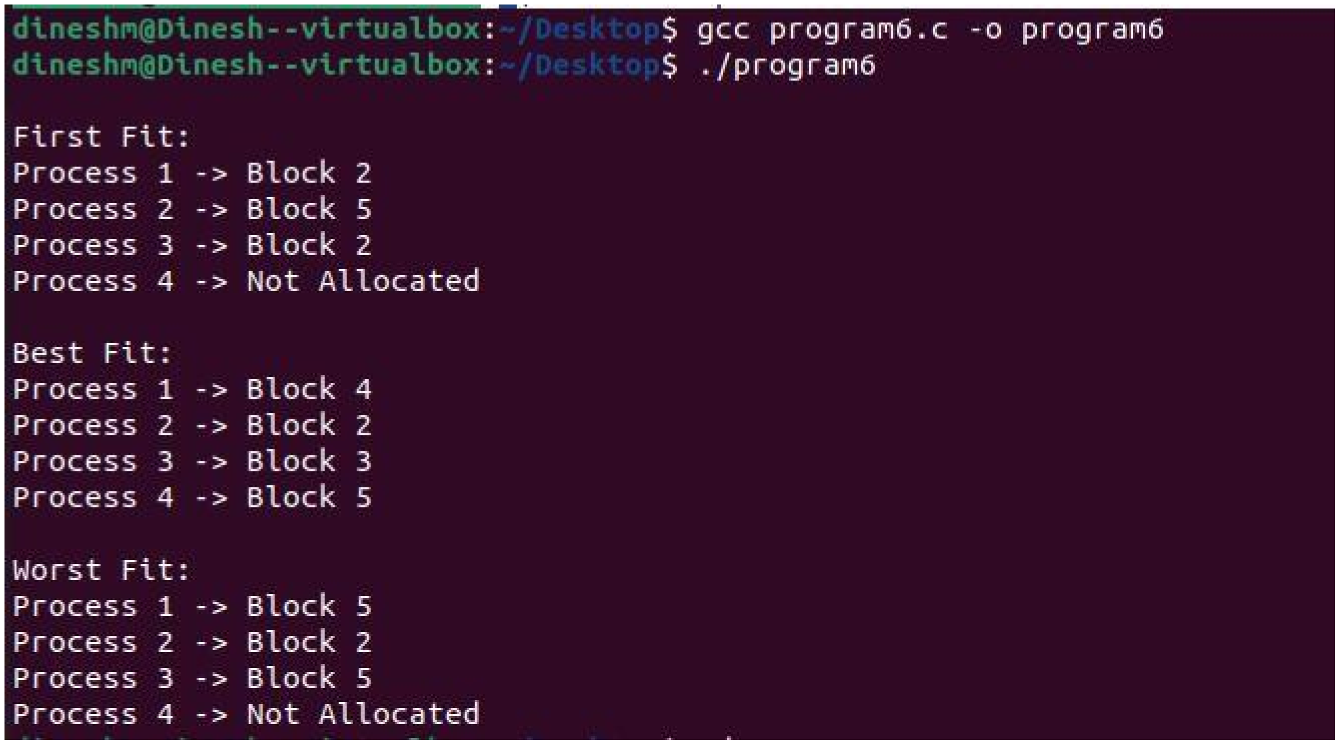
Develop a C program to simulate the following contiguous memory allocation Techniques:
a) Worst fit
b) Best fit
c) First fit.

<u>Source Code:</u>

```
#include <stdio.h>
#define MAX 25

void firstFit(int block[], int m, int process[], int n)
    { int alloc[n], i, j;
    for (i = 0; i < n; i++) alloc[i] = -1;

    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            if (block[j] >= process[i])
                { alloc[i] = j;
                block[j] -= process[i];
                break;
            }
        }
    }
    printf("\nFirst Fit:\n");
    for (i = 0; i < n; i++) {
        if (alloc[i] != -1)
            printf("Process %d -> Block %d\n", i + 1, alloc[i] +
1);
        else
            printf("Process %d -> Not Allocated\n", i + 1);
    }
}
```

```
void bestFit(int block[], int m, int process[], int n)
    { int alloc[n], i, j, best;
    for (i = 0; i < n; i++) alloc[i] = -1;
    for (i = 0; i < n; i++) {
        best = -1;
        for (j = 0; j < m; j++) {
            if (block[j] >= process[i]) {
                if (best == -1 || block[j] < block[best])
                    best = j;
            }
        }
        if (best != -1)
            { alloc[i] = best;
            block[best] -= process[i];
        }
    }
    printf("\nBest Fit:\n");
    for (i = 0; i < n; i++) {
        if (alloc[i] != -1)
            printf("Process %d -> Block %d\n", i + 1, alloc[i] +
1);
        else
            printf("Process %d -> Not Allocated\n", i + 1);
    }
}
void worstFit(int block[], int m, int process[], int n)
    { int alloc[n], i, j, worst;
    for (i = 0; i < n; i++) alloc[i] = -1;

    for (i = 0; i < n; i++)
        { worst = -1;
        for (j = 0; j < m; j++) {
            if (block[j] >= process[i]) {
                if (worst == -1 || block[j] > block[worst])
                    worst = j;
            }
        }
        if (worst != -1)
            { alloc[i] = worst;
            block[worst] -= process[i];
        }
    }
    printf("\nWorst Fit:\n");
    for (i = 0; i < n; i++) {
        if (alloc[i] != -1)
            printf("Process %d -> Block %d\n", i + 1, alloc[i] +
1);
        else
            printf("Process %d -> Not Allocated\n", i + 1);
    }
}

int main() {
    int block[MAX] = {100, 500, 200, 300, 600};
    int process[MAX] = {212, 417, 112, 426};
    int m = 5, n = 4;
    int block1[MAX], block2[MAX], block3[MAX];
    for (int i = 0; i < m; i++) {
        block1[i] = block[i];
        block2[i] = block[i];
        block3[i] = block[i];
    }
    firstFit(block1, m, process, n);
    bestFit(block2, m, process, n);
    worstFit(block3, m, process, n);
    return 0;
}
```

Expected otuptut:



```
dineshm@Dinesh--virtualbox:~/Desktop$ gcc program6.c -o program6
dineshm@Dinesh--virtualbox:~/Desktop$ ./program6

First Fit:
Process 1 -> Block 2
Process 2 -> Block 5
Process 3 -> Block 2
Process 4 -> Not Allocated

Best Fit:
Process 1 -> Block 4
Process 2 -> Block 2
Process 3 -> Block 3
Process 4 -> Block 5

Worst Fit:
Process 1 -> Block 5
Process 2 -> Block 2
Process 3 -> Block 5
Process 4 -> Not Allocated
```

PROGRAM 7

Develop a C program to simulate page replacement algorithms:

a) FIFO
b) LRU

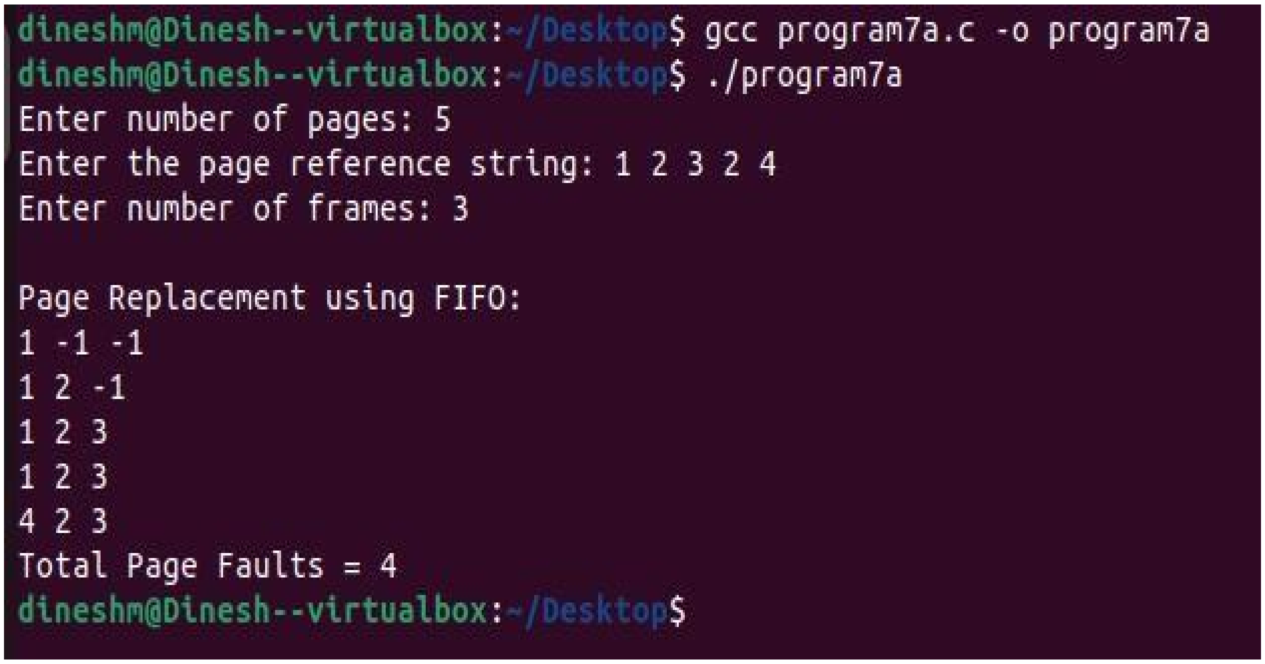
Source Code:

A) FIFO

```
#include <stdio.h>
int main() {
    int pages[50], n, frames[10], f, i, j, k = 0, count = 0,
flag;
    printf("Enter number of pages: ");
    scanf("%d", &n);
    printf("Enter the page reference string: ");
    for (i = 0; i < n; i++)
        scanf("%d", &pages[i]);
    printf("Enter number of frames: ");
    scanf("%d", &f);
    for (i = 0; i < f; i++)
        frames[i] = -1;
    printf("\nPage Replacement using FIFO:\n");
    for (i = 0; i < n; i++) {
        flag = 0;
        for (j = 0; j < f; j++) {
            if (frames[j] == pages[i])
                { flag = 1;
                break;
            }
        }
        if (!flag) {
            frames[k] = pages[i];
            k = (k + 1) % f;
```

```
                count++;
            }
            for (j = 0; j < f; j++)
                printf("%d ", frames[j]);
            printf("\n");
        }
    printf("Total Page Faults = %d\n", count);
    return 0;
}
```

Expected output:

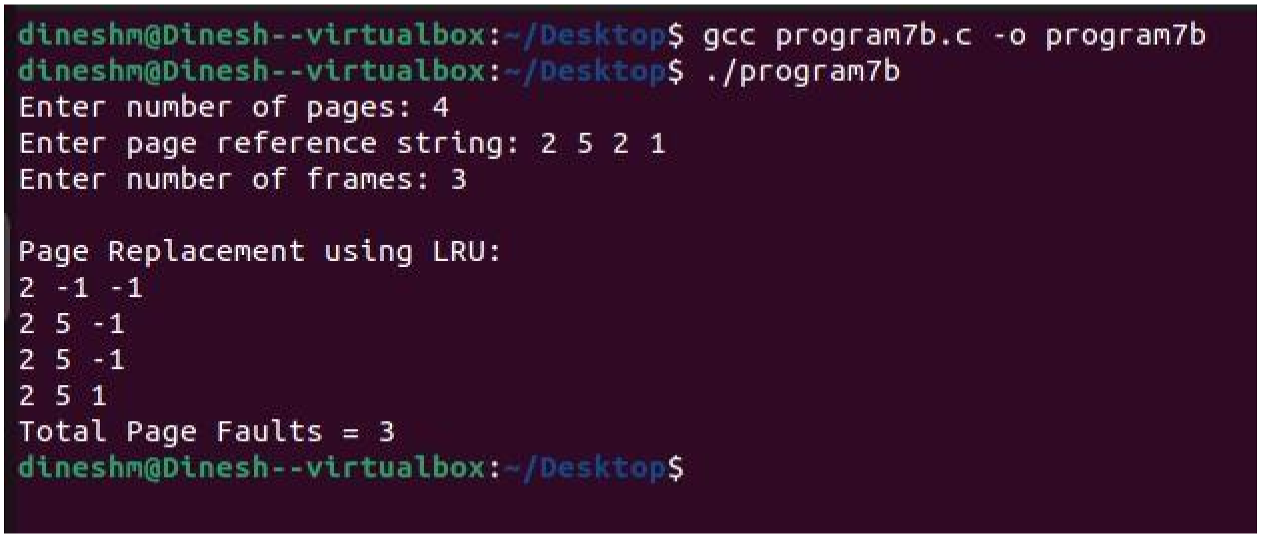


```
dineshm@Dinesh--virtualbox:~/Desktop$ gcc program7a.c -o program7a
dineshm@Dinesh--virtualbox:~/Desktop$ ./program7a
Enter number of pages: 5
Enter the page reference string: 1 2 3 2 4
Enter number of frames: 3

Page Replacement using FIFO:
1 -1 -1
1 2 -1
1 2 3
1 2 3
4 2 3
Total Page Faults = 4
dineshm@Dinesh--virtualbox:~/Desktop$
```

B) LRU

```
#include <stdio.h>

int findLRU(int time[], int n) {

    int i, minimum = time[0], pos = 0;

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

        if (time[i] < minimum) {
            minimum = time[i];
            pos = i;
        }
    }
    return pos;
}
int main() {
    int pages[50], frames[10], time[10], n, f, count = 0, flag1,
flag2, i, j, pos;
    printf("Enter number of pages: ");
    scanf("%d", &n);
    printf("Enter page reference string: ");
    for (i = 0; i < n; i++)
        scanf("%d", &pages[i]);

    printf("Enter number of frames: ");
    scanf("%d", &f);
    for (i = 0; i < f; i++)
        { frames[i] = -1;
        time[i] = 0;
    }
    printf("\nPage Replacement using LRU:\n");
    for (i = 0, j = 0; i < n; i++) {
        flag1 = flag2 = 0;
        for (j = 0; j < f; j++) {
            if (frames[j] == pages[i])
                { flag1 = flag2 = 1;
                time[j] = i;
                break;
            }
        }
        if (!flag1) {
            for (j = 0; j < f; j++)
                { if (frames[j] == -1)
                {
                    frames[j] = pages[i];
                    time[j] = i;
                    flag2 = 1;
                    count++;
                    break;
                }
            }
        }
        if (!flag2) {
            pos = findLRU(time, f);
            frames[pos] = pages[i];
            time[pos] = i;
            count++;
        }
        for (j = 0; j < f; j++)
            printf("%d ", frames[j]);
        printf("\n");
    }
    printf("Total Page Faults = %d\n", count);
    return 0;
}
```

Expected output:



```
dineshm@Dinesh--virtualbox:~/Desktop$ gcc program7b.c -o program7b
dineshm@Dinesh--virtualbox:~/Desktop$ ./program7b
Enter number of pages: 4
Enter page reference string: 2 5 2 1
Enter number of frames: 3

Page Replacement using LRU:
2 -1 -1
2 5 -1
2 5 -1
2 5 1
Total Page Faults = 3
dineshm@Dinesh--virtualbox:~/Desktop$
```

PROGRAM 8

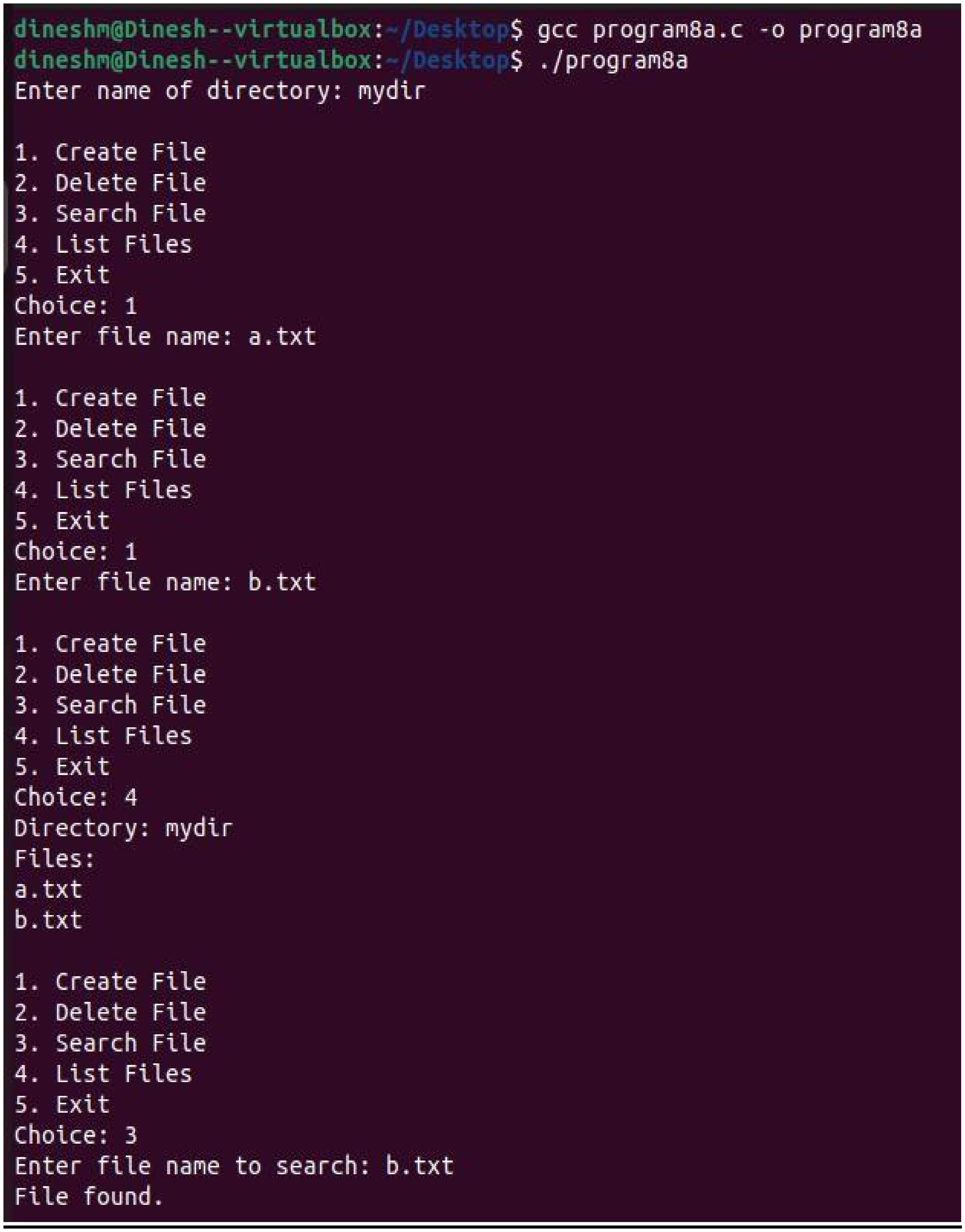
Simulate following File Organization Techniques
a) Single level directory
b) Two level directory

Source code

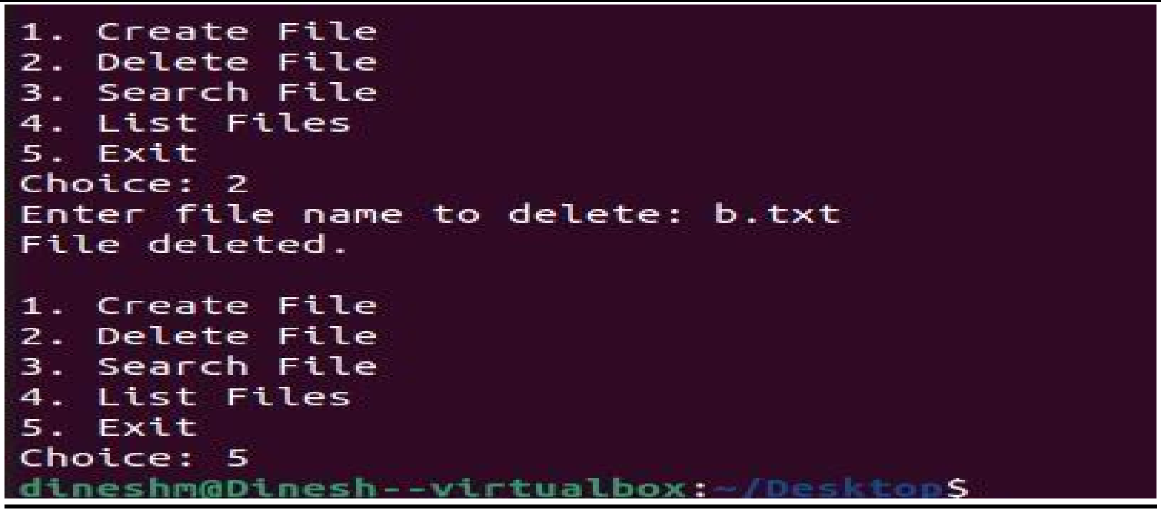
a) Single level directory

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

struct directory {
    char dname[10], fname[10][10];
    int fcount;
};

int main() {
    struct directory dir;
    int i;

    printf("Enter name of directory: ");
    scanf("%s", dir.dname);
    dir.fcount = 0;

    int ch;
    do {
        printf("\n1. Create File\n2. Delete File\n3. Search
File\n4. List Files\n5. Exit\nChoice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("Enter file name: ");
                scanf("%s", dir.fname[dir.fcount]);
                dir.fcount++;
                break;
            case 2: {
                char f[10];
                printf("Enter file name to delete: ");
                scanf("%s", f);
                int found = 0;
                for (i = 0; i < dir.fcount; i++) {
                    if (strcmp(f, dir.fname[i]) == 0)
                        { strcpy(dir.fname[i],
dir.fname[dir.fcount - 1]);
                        dir.fcount--;
                        found = 1;
                        printf("File deleted.\n");
                        break;
                    }
                }
                if (!found) printf("File not found.\n");
                break;
            }
            case 3: {
                char f[10];
                printf("Enter file name to search: ");
                scanf("%s", f);
                int found = 0;
                for (i = 0; i < dir.fcount; i++) {
                    if (strcmp(f, dir.fname[i]) == 0)
                        { printf("File found.\n");
                        found = 1;
                        break;
                    }
                }
                if (!found) printf("File not found.\n");
                break;
            }
            case 4:
                printf("Directory: %s\nFiles:\n", dir.dname);
                for (i = 0; i < dir.fcount; i++) {
                    printf("%s\n", dir.fname[i]);
                }
                break;
        }
    } while (ch != 5);

    return 0;
}
```

Expected output:



```
dineshm@Dinesh--virtualbox:~/Desktop$ gcc program8a.c -o program8a
dineshm@Dinesh--virtualbox:~/Desktop$ ./program8a
Enter name of directory: mydir

1. Create File
2. Delete File
3. Search File
4. List Files
5. Exit
Choice: 1
Enter file name: a.txt

1. Create File
2. Delete File
3. Search File
4. List Files
5. Exit
Choice: 1
Enter file name: b.txt

1. Create File
2. Delete File
3. Search File
4. List Files
5. Exit
Choice: 4
Directory: mydir
Files:
a.txt
b.txt

1. Create File
2. Delete File
3. Search File
4. List Files
5. Exit
Choice: 3
Enter file name to search: b.txt
File found.
```

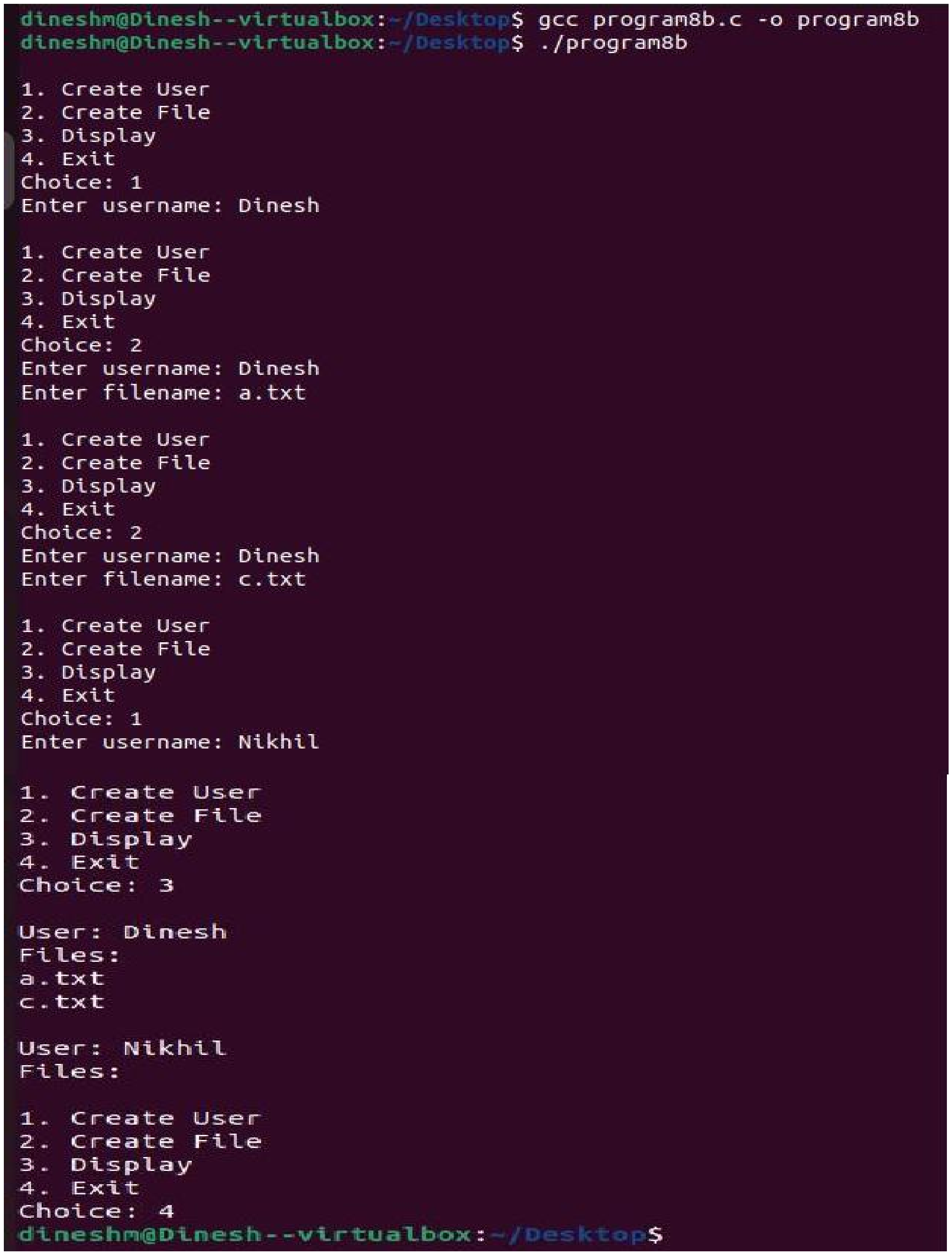


```
1. Create File
2. Delete File
3. Search File
4. List Files
5. Exit
Choice: 2
Enter file name to delete: b.txt
File deleted.

1. Create File
2. Delete File
3. Search File
4. List Files
5. Exit
Choice: 5
dineshm@Dinesh--virtualbox:~/Desktop$
```

b) Two level directory

```
#include <stdio.h>
#include <string.h>
struct file {
    char fname[10][10];
    int fcount;
};
struct user {
    char uname[10];
    struct file f;
};
int main() {
    struct user u[10];
    int ucount = 0, i, j;
    int ch;
    do {
        printf("\n1. Create User\n2. Create File\n3. Display\n4.
Exit\nChoice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("Enter username: ");
                scanf("%s", u[ucount].uname);
                u[ucount].f.fcount = 0;
                ucount++;
                break;
            case 2: {
                char name[10];
                printf("Enter username: ");
                scanf("%s", name);
                for (i = 0; i < ucount; i++) {
                    if (strcmp(name, u[i].uname) == 0)
                        { printf("Enter filename: ");

        scanf("%s",u[i].f.fname[u[i].f.fcount++]);
                        break;
                    }
                }
                break;
            }
            case 3:
                for (i = 0; i < ucount; i++) {
                    printf("\nUser: %s\nFiles:\n", u[i].uname);
                    for (j = 0; j < u[i].f.fcount; j++)
                        printf("%s\n", u[i].f.fname[j]);
                }
                break;
        }
    } while (ch != 4);

    return 0;
}
```

Expected output:



```
dineshm@Dinesh--virtualbox:~/Desktop$ gcc program8b.c -o program8b
dineshm@Dinesh--virtualbox:~/Desktop$ ./program8b

1. Create User
2. Create File
3. Display
4. Exit
Choice: 1
Enter username: Dinesh

1. Create User
2. Create File
3. Display
4. Exit
Choice: 2
Enter username: Dinesh
Enter filename: a.txt

1. Create User
2. Create File
3. Display
4. Exit
Choice: 2
Enter username: Dinesh
Enter filename: c.txt

1. Create User
2. Create File
3. Display
4. Exit
Choice: 1
Enter username: Nikhil

1. Create User
2. Create File
3. Display
4. Exit
Choice: 3

User: Dinesh
Files:
a.txt
c.txt

User: Nikhil
Files:

1. Create User
2. Create File
3. Display
4. Exit
Choice: 4
dineshm@Dinesh--virtualbox:~/Desktop$
```

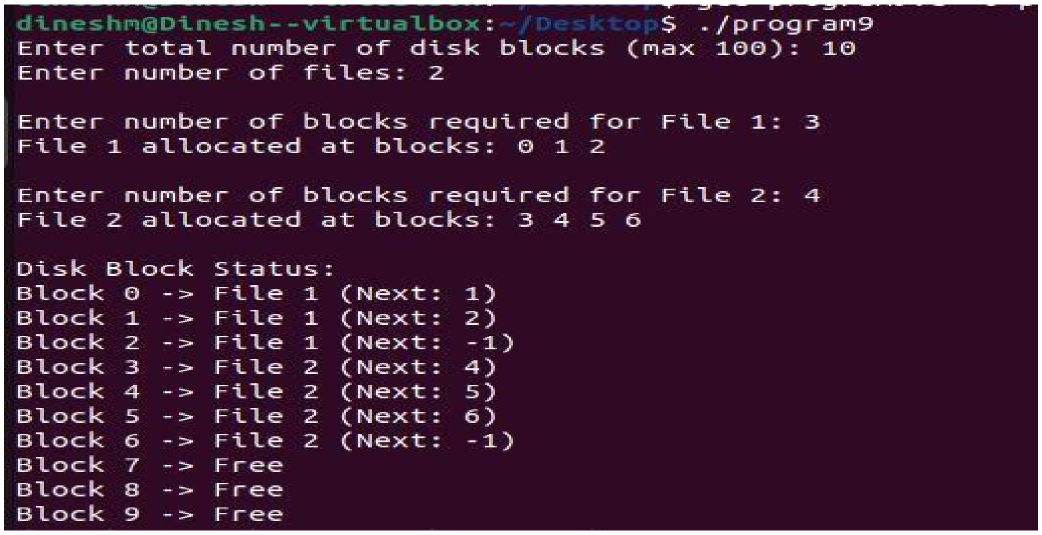
PROGRAM 9

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

Source code

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_BLOCKS 100
// Structure for a disk block
struct Block {
    int data;       // Can be used to represent file ID
    int next;       // Index of the next block, -1 if end of file
};
int main() {
    struct Block disk[MAX_BLOCKS];   // Simulate the disk blocks
    int used[MAX_BLOCKS] = {0};      // Tracks used/free blocks
    int totalBlocks, numFiles;
    printf("Enter total number of disk blocks (max %d): ",
MAX_BLOCKS);
    scanf("%d", &totalBlocks);
    if (totalBlocks > MAX_BLOCKS || totalBlocks <= 0)
        { printf("Invalid block size.\n");
        return 1;
    }
    printf("Enter number of files: ");
    scanf("%d", &numFiles);
    for (int i = 0; i < numFiles; i++)
        { int fileSize;
        printf("\nEnter number of blocks required for File %d: ", i +
1);
        scanf("%d", &fileSize);
        if (fileSize > totalBlocks) {
            printf("Not enough blocks available!\n");
            continue;
        }
        int allocated = 0;
        int prev = -1, start = -1;
        for (int j = 0; j < totalBlocks && allocated < fileSize; j++)
{           if (!used[j])
                { used[j] =
                1;
                disk[j].data = i + 1;    // Store file ID
                disk[j].next = -1;
                if (prev != -1)
                    disk[prev].next = j;
                if (start == -1)
                    start = j;
                prev = j;
                allocated++;
            }
        }
        if (allocated < fileSize) {
            printf("Not enough space for File %d. Reverting
allocation.\n", i + 1);
            // Revert allocations
            for (int k = 0; k < totalBlocks; k++) {
                if (used[k] && disk[k].data == i + 1)
                    { used[k] = 0;
                    disk[k].next = -1;
                }
            }
        } else {
            printf("File %d allocated at blocks: ", i + 1);
            int ptr = start;
            while (ptr != -1) {
                printf("%d ", ptr);
                ptr = disk[ptr].next;
            }
            printf("\n");
        }
    }
    // Optional: Display final disk status
    printf("\nDisk Block Status:\n");
    for (int i = 0; i < totalBlocks; i++) {
        if (used[i])
            printf("Block %d -> File %d (Next: %d)\n", i,
disk[i].data, disk[i].next);
        else
            printf("Block %d -> Free\n", i);
    }
    return 0;
}
```

Expected output:



```
dineshm@Dinesh--virtualbox:~/Desktop$ ./program9
Enter total number of disk blocks (max 100): 10
Enter number of files: 2

Enter number of blocks required for File 1: 3
File 1 allocated at blocks: 0 1 2

Enter number of blocks required for File 2: 4
File 2 allocated at blocks: 3 4 5 6

Disk Block Status:
Block 0 -> File 1 (Next: 1)
Block 1 -> File 1 (Next: 2)
Block 2 -> File 1 (Next: -1)
Block 3 -> File 2 (Next: 4)
Block 4 -> File 2 (Next: 5)
Block 5 -> File 2 (Next: 6)
Block 6 -> File 2 (Next: -1)
Block 7 -> Free
Block 8 -> Free
Block 9 -> Free
```

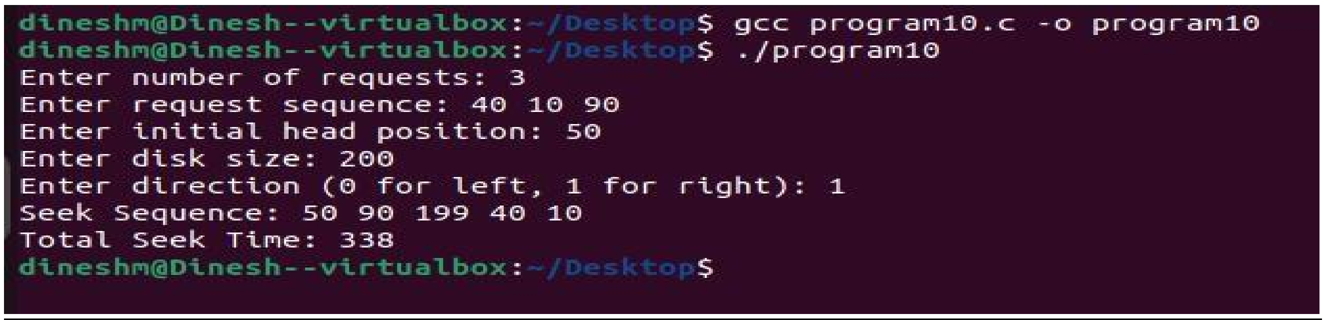
PROGRAM 10

Develop a C program to simulate SCAN disk scheduling algorithm.

<u>Source code</u>

```
#include <stdio.h>
#include <stdlib.h>
int compare(const void *a, const void *b)
    { return (*(int*)a - *(int*)b);
}
int main() {
    int req[100], n, head, i, direction, disk_size;
    int seek = 0;
    printf("Enter number of requests: ");
    scanf("%d", &n);
    printf("Enter request sequence: ");
    for (i = 0; i < n; i++) scanf("%d", &req[i]);
    printf("Enter initial head position: ");
    scanf("%d", &head);
    printf("Enter disk size: ");
    scanf("%d", &disk_size);
    printf("Enter direction (0 for left, 1 for right): ");
    scanf("%d", &direction);
    req[n] = head;
    req[n + 1] = (direction == 1) ? disk_size - 1 : 0;
    n += 2;
    qsort(req, n, sizeof(int), compare);
    int index;
    for (i = 0; i < n; i++)
        if (req[i] == head) {
            index = i;
            break;
        }
    printf("Seek Sequence: ");
    if (direction == 1) {
        for (i = index; i < n; i++)
            { printf("%d ", req[i]);
            seek += abs(head - req[i]);
            head = req[i];
        }
        for (i = index - 1; i >= 0; i--) {
            printf("%d ", req[i]);
            seek += abs(head - req[i]);
            head = req[i];
        }
    } else {
        for (i = index; i >= 0; i--) {
            printf("%d ", req[i]);
            seek += abs(head - req[i]);
            head = req[i];
        }
        for (i = index + 1; i < n; i++)
            { printf("%d ", req[i]);
            seek += abs(head - req[i]);
            head = req[i];
        }
    }
    printf("\nTotal Seek Time: %d\n", seek);
    return 0;
}
```

Expected output:



```
dineshm@Dinesh--virtualbox:~/Desktop$ gcc program10.c -o program10
dineshm@Dinesh--virtualbox:~/Desktop$ ./program10
Enter number of requests: 3
Enter request sequence: 40 10 90
Enter initial head position: 50
Enter disk size: 200
Enter direction (0 for left, 1 for right): 1
Seek Sequence: 50 90 199 40 10
Total Seek Time: 338
dineshm@Dinesh--virtualbox:~/Desktop$
```

VIVA Questions

1. **What happens when fork() is called?**
   → A new child process is created.
2. **Why do both parent and child continue after fork()?**
   → Because fork() returns separately to both.

3. **Why can Priority Scheduling lead to starvation?**
   → Low-priority jobs may never get CPU time.
4. **What is aging in scheduling?**
   → Gradually increasing the priority of waiting processes to avoid starvation.

5. **What is the main problem in producer-consumer setup?**
   → Managing access to a shared buffer.
6. **What happens if you don't use semaphores?**
   → Race conditions or data corruption.

7. **What is IPC used for?**
   → For processes to exchange data.
8. **Why are FIFOs better for unrelated processes?**
   → They can be accessed via file paths and are persistent.

9. **What is the Banker's Algorithm used for?**
   → To prevent deadlock by granting safe resource requests.
10. **What is a "safe state"?**
    → A state where all processes can eventually finish.

11. **What is fragmentation?**
    → Unused memory holes that waste space.
12. **Which fit is fastest to implement?**
    → First Fit is simple and quick.

13. **Why is LRU better than FIFO?**

    → It keeps frequently used pages longer, reducing page faults.
14. **What causes a page fault?**

    → Accessing a page not currently in memory.

15. **What is a two-level directory?**

→ Each user gets their own directory, improving file organization and isolation.

16. **What problem happens in single-level directories?**

→ File name conflicts due to a shared namespace.

17. **How does linked allocation work?**

→ Each block points to the next block of the file.

18. **What happens if a pointer is corrupted in linked allocation?**

→ The file becomes unreadable from that point forward.

19. **What is the SCAN algorithm?**

→ The disk arm moves across the disk servicing requests in one direction, then reverses.

20. **How does SCAN improve performance over FCFS?**

→ It reduces long seek times by organizing requests directionally.

21. **What is a zombie process?**

→ A child that has exited but is not yet cleaned up by the parent.

22. **What causes an orphan process?**

→ When the parent dies before the child finishes.

23. **What does exec() do in a process?**

→ It replaces the current process with a new program.

24. **How does the parent know the child has finished?**

→ Using the wait() system call.

25. **Why is virtual memory used?**

→ To give the illusion of a larger memory space than physically available.