MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

Department of Computer Science and Engineering (AIML)

OPERATING SYSTEMS LAB

II B.TECH-II SEMESTER

MLRS-BT25

A.Y.2026-2027

INDEX

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

CERTIFICATE

This is to certify that this manual is a bonafide record of practical work carried out in the **Operating System Laboratory** for the **II B.Tech (Computer Science and Engineering (AI&ML)) IV Semester** Program during the academic year **2026–2027.**This manual has been prepared by **Mrs. G. Asha Bhavani (Assistant Professor),** Department of Computer Science and Engineering (AI&ML), with my/our own efforts and to the best of our knowledge.

Signature of Lab Faculty **Signature of HOD**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

PREFACE

This laboratory lays the foundation for the Computer Science and Engineering (AI&ML) students during Second year of their course.

Operating system Lab can be divided into 2 groups: Cycle 1 and Cycle 2. In Cycle -I, students will know how to write CPU Scheduling algorithms, I/O System calls of UNIX/LINUX operating system and Deadlocks.

In cycle -II, students will design Page replacement polices, IPC Mechanism and Memory Management techniques. After performing all the experiments included in this Laboratory, it is hoped the student receives good training to handle any Principles of programming and Data Structures.

By,

Mrs. G. Asha Bhavani

ACKNOWLEDGEMENT

It was really a good experience, working at Operating System Lab. First, I would like to thank Mrs G. Asha Bhavani, Assistant Professor, Department of Computer Science and Engineering(AI&ML), Marri Laxman Reddy Institute of technology & Management for giving the technical support in preparing the document.

I express my sincere thanks to Dr. B. Ravi Prasad, Head of the Department of CSM, Marri Laxman Reddy Institute of technology & Management, for his concern towards me and gave me opportunity to prepare Operating System laboratory manual.

I am deeply indebted and gratefully acknowledge the constant support and valuable patronage of Dr. B. Ravi Prasad, Dean Academics, Marri Laxman Reddy Institute of technology & Management. I am unboundedly grateful to him for timely corrections and scholarly guidance.

I express my heartful thanks to Dr. P. Sridhar, Director, and Dr. R. Murali Prasad, Principal, Marri Laxman Reddy Institute of technology & Management, for giving me this wonderful opportunity for preparing the Concrete Technology laboratory manual.

At last, but not the least I would like to thank the entire CSM Department faculties those who had inspired and helped me to achieve my goal.

By,

Mrs. G. Asha Bhavani,

Department of CSE (AI&ML)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

GENERAL INSTRUCTIONS

1. Students are instructed to come to Soft Computing laboratory on time. Late comers are not entertained in the lab.

2. Students should be punctual to the lab. If not, the conducted experiments will not be repeated.

3. Students are expected to come prepared at home with the experiments which are going to be performed.

4. Students are instructed to display their identity cards before entering into the lab.

5. Students are instructed not to bring mobile phones to the lab.

6. Any damage/loss of system parts like keyboard, mouse during the lab session, it is student's responsibility and penalty or fine will be collected from the student.

7. Students should update the records and lab observation books session wise. Before leaving the lab the student should get his/her lab observation book signed by the faculty.

8. Students should submit the lab records by the next lab to the concerned faculty members in the staff room for their correction and return.

9. Students should not move around the lab during the lab session.

10. If any emergency arises, the student should take the permission from faculty member concerned in written format.

11. The faculty members may suspend any student from the lab session on disciplinary grounds.

12. Never copy the output from other students. Write down your own outputs.

13. Handle all computer systems, keyboards, mice, and peripherals with care.

- Do not attempt to open or tamper with any hardware components.
- Use only the assigned computer system; do not switch systems without permission.

4. Software and Data Safety

- Use only authorized software installed by the lab administrator.
- Do not attempt to install, uninstall, or modify any software without approval.
- Save your work frequently and ensure backups of important files.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

SAFETY MEASURES

To ensure the safe and efficient use of the Computer Science and Engineering(AI&ML) laboratory, all students must strictly adhere to the following safety guidelines:

1. General Conduct
   - Maintain silence and discipline during lab sessions.
   - Do not bring food, drinks, or chewing gum into the lab.
   - Use lab resources responsibly and follow all instructions provided by the instructor or lab assistant.
2. Electrical Safety
   - Do not touch electrical switches, sockets, or plugs with wet hands.
   - Avoid overloading power sockets with unauthorized devices.
   - Immediately report any loose connections, sparks, or unusual noises from equipment.

3. Computer and Equipment Handling

   - Handle all computer systems, keyboards, mice, and peripherals with care.
   - Do not attempt to open or tamper with any hardware components.
   - Use only the assigned computer system; do not switch systems without permission.

4. Software and Data Safety
   - Use only authorized software installed by the lab administrator.
   - Do not attempt to install, uninstall, or modify any software without approval.
   - Save your work frequently and ensure backups of important files.

5. Cybersecurity and Network Usage

   - Keep your login credentials confidential.
   - Do not attempt to access restricted websites or server
   - Avoid activities such as hacking, gaming, or the use of pirated content.

6. Emergency Preparedness

- Be familiar with the location of emergency exits, fire extinguishers, and first aid kits.
- In the event of a fire, electrical hazard, or any emergency, remain calm and inform the lab instructor immediately.

- Follow the evacuation procedure as instructed.

7. Post-Lab Procedures
   - Log out of your session and shut down the system properly after use.
   - Leave your workstation clean and organized.
   - Return any borrowed materials or equipment to their proper place.

8. Hygiene and Cleanliness

- Wash or sanitize your hands before and after using shared devices.
- Do not write or place unnecessary items on the workstation.
- Report any spills or cleanliness issues to the lab staff.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

VISION & MISSION OF THE INSTITUTE

Vision of the Institute:

To be a globally recognized institution that fosters innovation, excellence, and leadership in education, research, and technology development, empowering students to create sustainable solutions for the advancement of society.

Mission of the Institute:

To foster a transformative learning environment that empowers students to excel in engineering, innovation, and leadership.

To produce skilled, ethical, and socially responsible engineers who contribute to sustainable technological advancements and address global challenges.

To shape future leaders through cutting-edge research, industry collaboration, and community engagement.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

VISION & MISSION OF THE DEPARTMENT

Vision of the Department:

To nurture globally competent professionals in Artificial Intelligence and Machine Learning through excellence in education, research, and innovation, committed to developing sustainable and impactful solutions for the betterment of society.

Mission of the Department:

To provide a transformative learning environment that equips students with in-depth knowledge and practical skills in Artificial Intelligence and Machine Learning, fostering innovation, leadership, and lifelong learning.

To advance AI and ML through cutting-edge research, strong industry collaboration, and community engagement, preparing students to address real-world challenges on a global scale.

To produce competent and ethical AI professionals who contribute to technological progress while addressing societal and environmental challenges with sustainable solutions.

To foster a research-driven culture by partnering with industry and academia, encouraging entrepreneurship, and engaging in community-centered technology development.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

Program Educational Objectives (PEOs)

PEO 1	**Professional Competence** Graduates will possess strong theoretical and practical knowledge in Artificial Intelligence and Machine Learning, enabling them to solve complex real-world problems, pursue higher education, or excel in professional careers
PEO 2	**Innovation and Research Orientation:** Graduates will engage in innovative practices, cutting-edge research, and contribute to the advancement of AI and ML technologies through collaboration with industry and academia.
PEO 3	**Leadership and Lifelong Learning:** Graduates will exhibit leadership qualities, effective communication, and teamwork skills, and will continuously upgrade their knowledge to adapt to evolving technological landscapes.
PEO 4	**Entrepreneurial and Community Engagement:** Graduates will leverage entrepreneurial skills and a sense of civic responsibility to create AI-driven solutions that benefit local and global communities.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

Program Outcomes (POs)

PO:

PO1.Engineering Knowledge:
Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO 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.

PO 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.

PO 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.

PO 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.

PO 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.

PO 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.

PO 8.Ethics:
Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9. Individual and Team Work:
Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO 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.

PO 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.

PO 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.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: Able to identify, analyze and solve the problems related to Artificial Intelligence and Machine Learning by applying the fundamental knowledge of Computer Science and Engineering.
1. Problem Identification and Formulation
2. Application of Computer Science Fundamentals
3. AI/ML Techniques and Tools Proficiency
4. Analytical and Critical Thinking

PSO2: Build innovative tools and techniques to develop project models in the areas related to Deep Learning, Machine learning, Artificial Intelligence.
1. Innovation and Tool Development
2. Implement end-to-end project models using real-world datasets in domains like image processing, NLP, or predictive analytics.
3. Advanced Technical Proficiency
4. Evaluation and Optimization.

PSO 3: Make use of the Artificial Intelligence and Machine Learning knowledge to assess societal, environmental, health, safety issues, Sustainable development goals with professional ethics and can also pursue higher studies, involve in research activities, be employable or entrepreneur.
1. Application of AI/ML for Societal and Environmental Impact
2. Ethical and Responsible AI Practice
3. Lifelong Learning and Research Orientation
4. Employability and Entrepreneurship

2540582: OPERATING SYSTEM LAB

II Year B.Tech.CSM II– Sem. **L T P C**

0 0 2 1

Pre-requisites:

- A course on "Programming for Problem Solving"
- A course on "Data Structures".

Co-requisites: A course on "Operating Systems".

Course Objectives:

1. To provide an understanding of the design aspects of operating system concepts through simulation.
2. Introduce basic Unix commands, system call interface for process management, interprocess communication and I/O in Unix.

Course Outcomes: The students must be able to

1. Apply various CPU scheduling algorithms and analyze their performance using c programing.

2. Develop C progrms utilizing UNIX/LINUX system calls to perform file operations, process control, I/O management.

3. Simulate deadlock avoidance and resource allocation strategies such as the banker's algorithm effectively.

4. Implement inter-process communication and synchronization mechanism including semaphores, pipes, message queues, and shared memory.

5. Demonstrate memory management and page replacement techniques through C programming, analyzing efficiency and system behavior.

List of Experiments

1. Write C programs to simulate the following CPU Scheduling algorithms
   a)FCFS b)SJF c)Round Robin d) priority
2. Write programs using the I/O system calls of UNIX/LINUX operating system (open, read, write, close, fork, seek, stat, exit).
3. Write a C program to simulate Bankers Algorithm for Deadlock Avoidance.
4. Write a C program to implement the Producer — Consumer problem using semaphores using UNIX/LINUX system calls.
5. Write C programs to illustrate the following IPC mechanisms

a) Pipes b) FIFOs c)Message Queues d) Shared Memory

6. Write C programs to simulate the following memory management techniques a)Paging b)Segmentation.
7. Write a C programe to simulate page replacement policies a) FCFS b) LRU c) Optimal

Text Books

1. Operating System Principles- Abraham Silberchatz, Peter B. Galvin, Greg Gagne 7th Edition, JohnWiley
2. Advanced programming in the Unix environment, W.R.Stevens, Pearson education.

References

1. Operating Systems – Internals and Design Principles Stallings, Fifth Edition–2005, Pearson Education/PHI
2. Operating System A Design Approach-Crowley,TMH.
3. Modern Operating Systems, Andrew S Tanenbaum 2nd edition, Pearson/PHI
4. Unix programming environment, Kernighan and Pike, PHI. / Pearson Education
5. Unix Internals The New Frontiers, U.Vahalia, Pearson Education

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

OPERATING SYSTEM LABORATORY

Virtual lab details

Name of the Virtual Lab: Operating System Lab

Virtual Lab Host Institute: IIT Bombay Virtual Labs

URL/Link to Lab: https: https://khushalip.github.io/OS-lab/index.html.

Academic Year: 2026-2027

Semester: III SEM

List of Experiments Available in Virtual Lab

- Deadlock and concurrency
- Scheduling algorithms
- Page replacement algorithms
- Disk scheduling algorithms

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

OPERATING SYSTEM LABORATORY

LAB PLANNER

S.No	Experiment	CO	Virtual Lab Availability	Date planned	Date conducted
1	Write C programs to simulate the following CPU Scheduling algorithms. a) FCFS	CO1	yes		
2	Write C programs to simulate the following CPU Scheduling algorithms. b) SJF	CO1	yes		
3	Write C programs to simulate the following CPU Scheduling algorithms. c) Round Robin d) priority	CO1	yes		
4	Write programs using the I/O system calls ofUNIX/LINUX operating system. (open, read, write, close, fentl,)	CO2			
5	Write programs using the I/O system calls ofUNIX/LINUX operating system. (close, fentl, seek)	CO2			
6	Write programs using the I/O system calls ofUNIX/LINUX operating system. (stat, opendir, readdir)	CO2			
7	Write a C program to simulate Bankers Algorithm for Deadlock Avoidance	CO3	yes		
8	Write a C program to simulate Bankers Algorithm for Deadlock Prevention	CO3	yes		
9	**LAB INTERNAL-1**				
10	Write a C program to implement the Producer and Consumer problem UNIX/LINUX system calls.	CO4			
11	Write C programs to illustrate	CO4	yes		

	the following IPC mechanisms a) Pipes b) FIFOs		
12	Write C programs to illustrate the following IPC mechanisms c) Message Queues	CO4	
13	Write C programs to illustrate the following IPC mechanisms d) Shared Memory	CO4	yes
14	Write C programs to simulate the following memory management techniques a) Paging	CO5	yes
15	6. Write C programs to simulate the following memory management techniques a) segmentation	CO5	yes
16	7. Write a C program to simulate the following Page replacement policies. a) FIFO b) LRU c) Optimal	CO5	yes
17	**LAB INTERNAL-2**		

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

OPERATING SYSTEM LABORATORY
LAB PLANNER

Date planed	**10/7**			**17/7**			**24/7**																										
Date conducted																																	
Roll Number	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*		Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	
257Y16601	1	1	Y	2	1	N	3	2	Y	4	2	N	5	3	Y		6	4	Y	7	4	Y	8	4	Y	9	5	N	10	5	Y		
257Y16602	2	1	N	3	2	Y	4	2	N	5	3	Y	6	4	Y	M	7	4	Y	8	4	Y	9	5	N	10	5	Y	1	1	Y	**M**
257Y16603	3	2	Y	4	2	N	5	3	Y	6	4	Y	7	4	Y	I	8	4	Y	9	5	N	10	5	Y	1	1	Y	2	1	N	**I**
257Y16604	4	2	N	5	3	Y	6	4	Y	7	4	Y	8	4	Y	D	9	5	N	10	5	Y	1	1	Y	2	1	N	3	2	Y	D
257Y16605	5	3	Y	6	4	Y	7	4	Y	8	4	Y	9	5	N	-	10	5	Y	1	1	Y	2	1	N	3	2	Y	4	2	N	**-**
257Y16606	6	4	Y	7	4	Y	8	4	Y	9	5	N	10	5	Y	I	1	1	Y	2	1	N	3	2	Y	4	2	N	5	3	Y	**II**

Note:VL*-Virtual Lab Availabilty

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

RUBRICS USED TO ASSESS LEARNINGS IN LABORATORIES

1. RUBRICS FOR DAY TO DAY EVALUATION

Parameter	Max Marks	Level-1 (Very Poor)	Level-2 (Poor)	Level-3 (Average)	Level-4 (Good)	Level-5 (Excellent)
Observation Book	05	No observations or irrelevant data. (0–1)	Incomplete or incorrect data. (2)	Basic values with some errors. (3)	Mostly correct with good format. (4)	Fully correct, clear, and well-formatted. (5)
Record Writing	05	Not submitted. (0-1)	Submitted but mostly incomplete. (2)	Submitted with some missing/wrong parts. (3)	Submitted with minor issues. (4)	Fully complete, correct algorithm & flowchart. (5)
Result	05	No result or major errors. (0–1)	Result partially obtained. (2)	Acceptable result with limited error. (3)	Near-correct result and reasonable error. (4)	Accurate result. (5)
Viva-Voce	05	Did not answer any questions. (1)	Answered very few questions. (2)	Answered some questions with help. (3)	Answered most questions correctly. (4)	Answered all questions accurately. (5)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

OPERATING SYSTEM LABORATORY

RUBRICS FOR INTERNAL EVALUATION

Criterion	Max Marks	Level-1 *(Very Poor)*	Level-2 *(Poor)*	Level-3 *(Average)*	Level-4 *(Good)*	Level-5 *(Excellent)*
Design/Tool/Apparatus Selection	2 Marks	Incorrect tool/design and no reasoning. **(0)**	Tool/design selection attempted with unclear logic. **(0.5)**	Satisfactory selection with partial justification. **(1)**	Correct selection and proper analysis with few errors. **(1.5)**	Smart selectio with accurate relevant analysis. **(2)**
Execution (Code/Debug/Run) /Analysis/Method Used	4 Marks	Did not attempt or completely failed to execute. **(0)**	Attempted but unable to proceed or with major errors. **(1)**	Partial execution with some logic/syntax errors. **(2)**	Mostly correct execution with minimal help. **(3)**	Fully correct and independently executed program. **(4)**
Results& Documentation	2 Marks	Incomplete or poorly presented. **(0)**	Basic structure but lacks clarity or formatting. **(0.5)**	Complete but generic or with formatting issues. **(1)**	Well-structured and mostly clear. **(1.5)**	Well-organized, professional, and engaging documentatio **(2)**
Viva-Voce (Understanding of Concepts)	2 Marks	No understanding; could not answer questions. **(0)**	Answered a few with difficulty. **(0.5)**	Answered half the questions with basic clarity. **(1)**	Good understanding with confident answers. **(1.5)**	Answered all questions wit clarity and depth. **(2)**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

OPERATING SYSTEM LABORATORY
3. RUBRICS FOR SEMESTER END EXAMINATIONS

Criterion	Max Marks	Level-1 (Very Poor *(0–2 marks)*	Level-2 (Poor) *(3–4 marks)*	Level-3 (Average) *(5–6 marks)*	Level-4 (Good) *(7–9 marks)*	Level-5 (Excellent) *(10–12 marks)*
Preparedness for the Experiment	12 marks	No clarity on objective or procedure. Unable to explain basics.	Limited idea of the objective/procedure. Needed prompting.	Has basic understanding; minor gaps in concept or preparation.	Well-prepared, with clear understanding of steps and background.	Fully prepare with strong conceptual clarity and confident explanation.
Performance in the Laboratory	12 marks	Unable to perform experiment. Relied entirely on examiner's help.	Performed with multiple errors and constant support.	Performed with some errors; required occasional help.	Performed mostly independently with minimal support.	Performed independentl efficiently, and with precision.
Calculations & Graphs	12 marks	No or incorrect calculations. Graphs missing or irrelevant.	Multiple calculation errors. Graphs/plots inaccurate or poorly labeled.	Calculations partially correct. Graphs present but with some flaws.	Correct calculations and graphs with minor errors.	Accurate calculations and well-labeled graph with proper interpretation
Results & Error Analysis	12 marks	No result or invalid result. No error analysis attempted.	Incorrect result with vague or no error discussion.	Acceptable result. Error analysis attempted but limited.	Correct result with sound error discussion.	Accurate result with detailed and relevant erro analysis.
Viva-Voce (Subject Knowledge)	12 marks	Unable to answer any questions. No conceptual understanding.	Answered few questions with poor logic.	Answered half of the questions with average understanding.	Answered most questions with clarity and confidence.	Answered all questions wit depth, clarity and reasonin

EXPERIMENT-1

AIM: Write a C program simulate the following CPU scheduling algorithms:

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

a) **FCFS**

OBJECTIVE: Assume all the processes arrive at the same time.

FCFS CPU SCHEDULING ALGORITHM

For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

CPU SCHEDULING:

Maximum CPU utilization obtained with multiprogramming

CPU–I/O Burst Cycle – Process execution consists of a cycle of

CPU execution and I/O wait

CPU burst distribution

a) First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order: P1 , P2 , P3

The Gantt Chart for the schedule is:

Waiting time for P1 = 0; P2 = 24; P3 = 27

Average waiting time: (0 + 24 + 27)/3 = 17

ALGORITHM :

1. Start
2. Declare the array size
3. Read the number of processes to be inserted
4. Read the Burst times of processes
5. Calculate the waiting time of each process

   wt[i+1]=bt[i]+wt[i]

6. Calculate the turnaround time of each process

   tt[i+1]=tt[i]+bt[i+1]

7. Calculate the average waiting time and average turnaround time.
8. Display the values
9. Stop

PROGRAM:

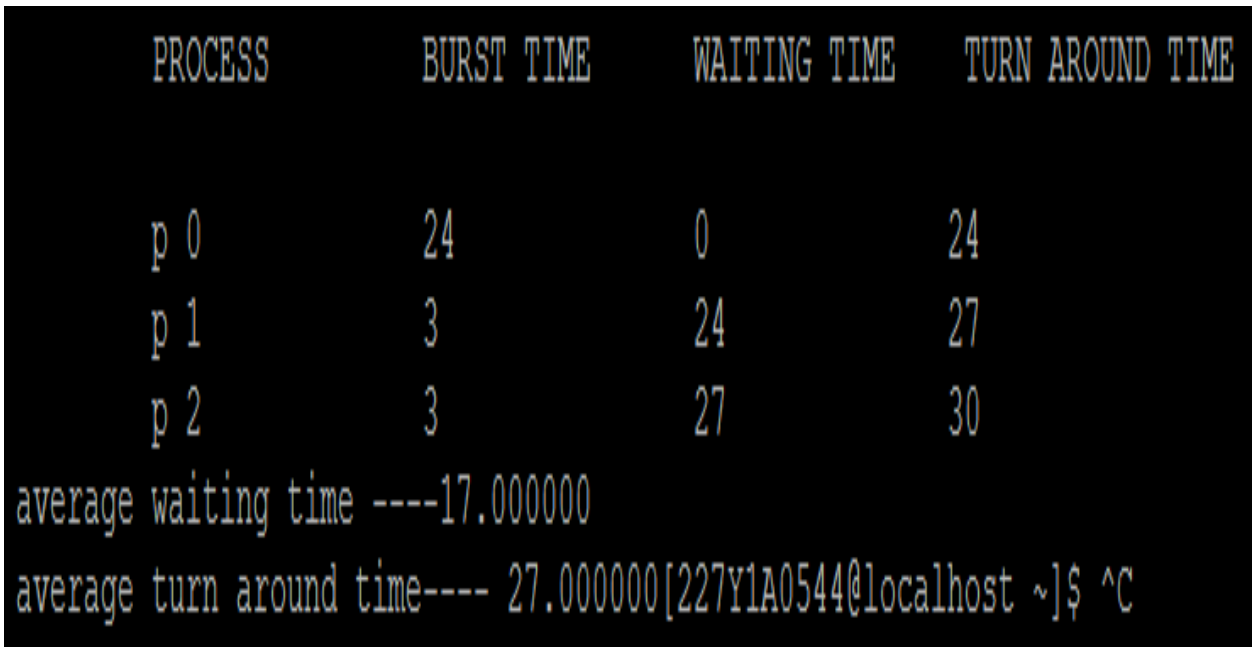
```
#include<stdio.h>
void main()
{
int i,j,bt[10],n,wt[10],tt[10];
float w1=0,t1=0;
float aw,at;
```

```
printf("enter no. of processes:\n");
scanf("%d",&n);
printf("enter the burst time of processes:");
for(i=0;i<n;i++)
scanf("%d",&bt[i]);
for(i=0;i<n;i++)
{
wt[0]=0;
tt[0]=bt[0];
wt[i+1]=bt[i]+wt[i];
tt[i+1]=tt[i]+bt[i+1];
w1=w1+wt[i];
t1=t1+tt[i];
}
aw=(float)w1/n;
at=(float)t1/n;
printf("\nbt\t wt\t tt\n");
for(i=0;i<n;i++)
printf("%d\t %d\t %d\n",bt[i],wt[i],tt[i]);
printf("aw=%f\n at=%f\n",aw,at);
}
```

INPUT

```
enter the no of processes:     3
enter burst time for process 0:        24
enter burst time for process 1:        3
enter burst time for process 2:        3
```

OUTPUT:



b) SJF

SJF CPU SCHEDULING ALGORITHM

For SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly.

HARDWARE REQUIREMENTS: Intel based Desktop Pc,RAM of 512 MB

SOFTWARE REQUIREMENTS: Turbo C/ Borland C.

THEORY: Example of Non Preemptive SJF

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	3.0	4

P1	P3	P2	P4

0 7 8 12 16

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	3.0	4

P1	P2	P3	P2	P4	P1

Average waiting time = (9 + 1 + 0 +2)/4 = 3

ALGORITHM

1. Start
2. Declare the array size
3. Read the number of processes to be inserted
4. Read the Burst times of processes
5. Sort the Burst times in ascending order and process with shortest burst time is first executed.
6. Calculate the waiting time of each process

   wt[i+1]=bt[i]+wt[i]

7. Calculate the turnaround time of each process

   tt[i+1]=tt[i]+bt[i+1]

8. Calculate the average waiting time and average turnaround time.
9. Display the values
10. Stop

PROGRAM:

```
#include<stdio.h>
//#include<conio.h>
void main()
{
int i,j,bt[10],t,n,wt[10],tt[10],w1=0,t1=0;
float aw,at;
//clrscr();
printf("enter no. of processes:\n");
scanf("%d",&n);
printf("enter the burst time of processes:");
for(i=0;i<n;i++)
scanf("%d",&bt[i]);
```

```
for(i=0;i<n;i++)
{
for(j=i;j<n;j++)
if(bt[i]>bt[j])
{
t=bt[i];
bt[i]=bt[j];
bt[j]=t;
}
}
for(i=0;i<n;i++)
printf("%d",bt[i]);
for(i=0;i<n;i++)
{
wt[0]=0;
tt[0]=bt[0];
wt[i+1]=bt[i]+wt[i];
tt[i+1]=tt[i]+bt[i+1];
w1=w1+wt[i];
t1=t1+tt[i];
}
aw=(float)w1/n;
at=(float)t1/n;
printf("\nbt\t wt\t tt\n");
for(i=0;i<n;i++)
printf("%d\t %d\t %d\n",bt[i],wt[i],tt[i]);
```

```
printf("aw=%f\n at=%f\n",aw,at);
//getch();
}
```

INPUT&OUTPUT:

```
 enter the number of processes:4
enter burst time for process    0:6
enter burst time for process    1:8
enter burst time for process    2:7
enter burst time for process    3:3

         PROCESS        BURST TIME      Waiting time    Turn around time

         p3             3               0               3
         p0             6               3               9
         p2             7               9               16
         p1             8               16              24
 Average waiting time--- :7.000000
 Average turn around time --:13.000000[227Y1A0544@localhost ~]$
```

C) ROUND ROBIN

Description: Assume all the processes arrive at the same time.

Round Robin Cpu Scheduling Algorithm:

For round robin scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the size of the time slice. Time slices are assigned to each process in equal portions and in circular order, handling all processes execution. This allows every process to get an equal chance. Calculate the waiting time and turnaround time of each of the processes accordingly.

HARDWARE REQUIREMENTS: Intel based Desktop Pc RAM of 512 MB

SOFTWARE REQUIREMENTS: Turbo C/ Borland C.

THEORY: Round Robin:

Example of RR with time quantum=3

Process	Burst time
aaa	4
Bbb	3
Ccc	2
Ddd	5
Eee	1

ALGORITHM

1. Start

2. Declare the array size

3. Read the number of processes to be inserted

4. Read the burst times of the processes

5. Read the Time Quantum

6. If the burst time of a process is greater than time Quantum then subtract time quantum form the burst time

    Else

        Assign the burst time to time quantum.

7. Calculate the average waiting time and turn around time of the processes.

8. Display the values

9. Stop

PROGRAM:

```
#include<stdio.h>

//#include<conio.h>

void main()

{

int st[10],bt[10],wt[10],tat[10],n,tq;
```

```
int i,count=0,swt=0,stat=0,temp,sq=0;
float awt=0.0,atat=0.0;
//clrscr();
printf("Enter number of processes:");
scanf("%d",&n);
printf("Enter burst time for sequences:");
for(i=0;i<n;i++)
{
scanf("%d",&bt[i]);
st[i]=bt[i];
}
printf("Enter time quantum:");
scanf("%d",&tq);
while(1)
{
for(i=0,count=0;i<n;i++)
{
temp=tq;
if(st[i]==0)
{
count++;
continue;
}
if(st[i]>tq)
st[i]=st[i]-tq;
else
```

```
if(st[i]>=0)
{
temp=st[i];
st[i]=0;
}
sq=sq+temp;
tat[i]=sq;
}
if(n==count)
break;
}
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
swt=swt+wt[i];
stat=stat+tat[i];
}
awt=(float)swt/n;
atat=(float)stat/n;
printf("Process_no Burst time Wait time Turn around time");
for(i=0;i<n;i++)
printf("\n%d\t %d\t %d\t %d",i+1,bt[i],wt[i],tat[i]);
printf("\nAvg wait time is %f Avg turn around time is %f",awt,atat);
//getch();
}
```

INPUT & OUTPUT:

```
Enter number of processes:3
Enter burst time for sequences:24
3
3
Enter time quantum:4
Process_no Burst time Wait time Turn around time
1              24             6              30
2              3              4              7
3              3              7              10
Avg wait time is 5.666667 Avg turn around time is 15.666667
```

D) PRIORITY:

Hardware Requirements: Intel based Desktop Pc, RAM of 512 MB

Software Requirements: Turbo C/ Borland C.

Theory:

In Priority Scheduling, each process is given a priority, and higher priority methods are executed first, while equal priorities are executed First Come First Served or Round Robin.

There are several ways that priorities can be assigned:

Internal priorities are assigned by technical quantities such as memory usage, and file/IO operations.

External priorities are assigned by politics, commerce, or user preference, such as importance and amount being paid for process access (the latter usually being for mainframes).

Algorithm:

1. Start

2. Declare the array size

3. Read the number of processes to be inserted

4. Read the Priorities of processes

5. sort the priorities and Burst times in ascending order

5. calculate the waiting time of each process

wt[i+1]=bt[i]+wt[i]

6. calculate the turnaround time of each process

tt[i+1]=tt[i]+bt[i+1]

7. Calculate the average waiting time and average turnaround time.

8. Display the values

9. Stop

PROGRAM:

```
#include<stdio.h>
//#include<conio.h>
void main()
 {
  int i,j,pno[10],prior[10],bt[10],n,wt[10],tt[10],w1=0,t1=0,s;
  float aw,at;
  //clrscr();
  printf("enter the number of processes:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
  printf("The process %d:\n",i+1);
   printf("Enter the burst time of processes:");
   scanf("%d",&bt[i]);
    printf("Enter the priority of processes %d:",i+1);
    scanf("%d",&prior[i]);
```

```
 pno[i]=i+1;
}
 for(i=0;i<n;i++)
  {
   for(j=0;j<n;j++)
   {
    if(prior[i]<prior[j])
    {
        s=prior[i];
        prior[i]=prior[j];
        prior[j]=s;

         s=bt[i];
         bt[i]=bt[j];
         bt[j]=s;
          s=pno[i];
          pno[i]=pno[j];
          pno[j]=s;
         }
        }
    }
    for(i=0;i<n;i++)
    {
      wt[0]=0;
      tt[0]=bt[0];
        wt[i+1]=bt[i]+wt[i];
```

```
        tt[i+1]=tt[i]+bt[i+1];
        w1=w1+wt[i];
        t1=t1+tt[i];
        aw=(float)w1/n;
        at=(float)t1/n;
        }
        printf(" \n job \t bt \t wt \t tat \t prior\n");
          for(i=0;i<n;i++)
           printf("%d   \t %d   \t %d  \t %d  \t %d\n",pno[i],bt[i],wt[i],tt[i],prior[i]);
           printf("aw=%f \t at=%f \n",aw,at);
  //getch();
   }
```

Input & Output:

```
Enter the number of processes --- 5
Enter the Burst Time & Priority of Process 0 --- 10 3
Enter the Burst Time & Priority of Process 1 --- 1 1
Enter the Burst Time & Priority of Process 2 --- 2 4
Enter the Burst Time & Priority of Process 3 --- 1 5
Enter the Burst Time & Priority of Process 4 --- 5 2

PROCESS         PRIORITY        BURST TIME      WAITING TIME    TURNAROUNDTIME
1                1               1               0               1
4                2               5               1               6
0                3               10              6               16
2                4               2               16              18
3                5               1               18              19
Average Waiting Time is --- 8.200000
AverageTurnaround Time is --- 12.000000[227Y1A0571@localhost os]$ ^C
```

VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	What is FCFS?	CO1	Remember
2	List the differences of FCFS and SJF.	CO1	Understand
3	Expand SJF.	CO1	Remember
4	What are the dis-advantages of RR Scheduling Algorithm?	CO1	Remember
5	What are the advantages of RR Scheduling Algoritm?	CO1	Remember
6	List an optimal scheduling algorithm in terms of minimizing the average waiting time of a given set of processes.	CO1	Understand
7	Which algorithm is identified as optimal scheduling alg.	CO1	Remember
8	In terms of average wait time name the optimum scheduling algorithm .	CO1	Remember
9	What are the dis-advantages of SJF Scheduling Algorithm?	CO1	Remember
10	What are the advantages of SJF Scheduling Algorithm?	CO1	Remember
11	Define CPU Scheduling algorithm?	CO1	Remember
12	What is First-Come-First-Served (FCFS) Scheduling?	CO1	Remember
13	Why CPU scheduling is required?	CO1	Understand
14	Which technique was introduced because a single job could not keep both the CPU and the I/O devices busy?	CO1	Remember
15	List CPU performance measuring factors.	CO1	Understand
16	What is avg waiting time?	CO1	Remember
17	Maximum CPU utilization obtained by which factors.	CO1	Understand
18	What is Burst Time.	CO1	Remember

19	What is FCFS?	CO1	Remember
20	List the differences of FCFS and SJF.	CO1	Understand

EXPERIMENT-2

AIM: Write programs using the I/O system calls of UNIX/LINUX operating system.

OBJECTIVE: To implement I/O system calls

PROGRAM:

```
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
int main()
{
int fd[2];
char buf1[25] = "just a test\n";
char buf2[100];
fd[0] = open("tfile",O_RDWR);
fd[1] = open("tfile",O_RDWR);
write(fd[0],buf1,strlen(buf1));
printf("\nEnter your text now…");
gets(buf1);
write(fd[0],buf1,strlen(buf1));
write(1, buf2, read(fd[1],buf2,sizeof(buf2)));
close(fd[0]);
close(fd[1]);
printf("\n");
return 0;
}
```

VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	Whether a system call is a routine built into the kernel and performs a basic function?	CO2	Remember
2	How does System Call Work?	CO2	Understand
3	When we execute a C program, CPU runs in which mode.	CO2	Remember
4	In which mode, the kernel runs on behalf of the user.	CO2	Remember
5	What is common in Linux and Unix OS.	CO2	Understand
6	The chmod command invokes which system call.	CO2	Remember
7	For reading input, which system call is used?	CO2	Remember
8	Which system call is used for opening or creating a file?	CO2	Remember
9	List the modes of file opening	CO2	Remember
10	Which of the following mode is used for opening a file in both reading and writing?	CO2	Remember
11	Define process blocking.	CO2	Remember
12	List the modes of file closing.	CO2	Remember
13	What are the advantages of a multiprocessor system?	CO2	Understand
14	Which system call is used for positioning the offset pointer?	CO2	Remember
15	What are the differences between process and thread?	CO2	Understand
16	Define System Call.	CO2	Remember
17	What are services provided by OS?	CO2	Understand
18	List the various types of System Calls.	CO2	Understand
19	What are the features of OS?	CO2	Understand
20	List the features of System Calls.	CO2	Understand

EXPERIMENT-3

AIM: Write a C program to simulate Bankers Algorithm for Deadlock Avoidance.

OBJECTIVE: Write a C program to simulate Bankers Algorithm for Deadlock Avoidance

DESCRIPTION

In a multiprograming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are 4held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.

NAME OF EXPERIMENT: Simulate Banker's Algorithm for Deadlock Avoidance.

AIM: Simulate Banker's Algorithm for Deadlock Avoidance to find whether the system is in safe state or not.

HARDWARE REQUIREMENTS: Intel based Desktop Pc

RAM of 512 MB

SOFTWARE REQUIREMENTS: Turbo C/ Borland C.

THEORY:

DEAD LOCK AVOIDANCE:

To implement deadlock avoidance & Prevention by using Banker's Algorithm.

Banker's Algorithm:

When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system.When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

Data structures: **n-Number of process, m-number of resource types.**

- Available: Available[j]=k, k – instance of resource type Rj is available.
- Max: If max[i, j]=k, Pi may request at most k instances resource Rj.
- Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj
- Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj,
  Need[I, j]=Max[I, j]-Allocation[I, j];

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.

2. Find an i such that both
   - Finish[i] =False
   - Need<=Work

   If no such I exists go to step 4.

3.
4. work=work+Allocation, Finish[i] =True;
5. if Finish[1]=True for all I, then the system is in safe state.

Resource request algorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to step 2. Otherwise raise an error condition.
2. if Request<=Available go to step 3. Otherwise Pi must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;
   Available=Available-Request I;

   Allocation I =Allocation+Request I;

   Need i=Need i-Request I;

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

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.

3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. or not if we allow the request.
10. stop the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct da {
int max[10],al[10],need[10],before[10],after[10];
}p[10];
void main() {
int i,j,k,l,r,n,tot[10],av[10],cn=0,cz=0,temp=0,c=0;
//clrscr();
printf("\n Enter the no of processes:");
scanf("%d",&n);
printf("\n Enter the no of resources:");
scanf("%d",&r);
for(i=0;i<n;i++) {
printf("process %d \n",i+1);
for(j=0;j<r;j++) {
printf("maximum value for resource %d:",j+1);
scanf("%d",&p[i].max[j]);
}
for(j=0;j<r;j++) {
printf("allocated from resource %d:",j+1);
scanf("%d",&p[i].al[j]);
```

```
p[i].need[j]=p[i].max[j]-p[i].al[j];
}
}
for(i=0;i<r;i++) {
printf("Enter total value of resource %d:",i+1);
scanf("%d",&tot[i]);
}
for(i=0;i<r;i++) {
for(j=0;j<n;j++)
temp=temp+p[j].al[i];
av[i]=tot[i]-temp;
temp=0;
}
printf("\n\t max allocated needed total avail");
for(i=0;i<n;i++) {
printf("\n P%d \t",i+1);
for(j=0;j<r;j++)
printf("%d",p[i].max[j]);
printf("\t");
for(j=0;j<r;j++)
printf("%d",p[i].al[j]);
printf("\t");
for(j=0;j<r;j++)
printf("%d",p[i].need[j]);
printf("\t");
for(j=0;j<r;j++)
{
if(i==0)
```

```
printf("%d",tot[j]);
}
printf(" ");
for(j=0;j<r;j++) {
if(i==0)
printf("%d",av[j]);
}
}
printf("\n\n\t AVAIL BEFORE \t AVAIL AFTER");
for(l=0;l<n;l++)
{
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
if(p[i].need[j]>av[j])
cn++;
if(p[i].max[j]==0)
cz++;
}
if(cn==0 && cz!=r)
{
for(j=0;j<r;j++)
{
p[i].before[j]=av[j]-p[i].need[j];
p[i].after[j]=p[i].before[j]+p[i].max[j];
av[j]=p[i].after[j];
p[i].max[j]=0;
```

```
}
printf("\n p%d \t",i+1);
for(j=0;j<r;j++)
printf("%d",p[i].before[j]);
printf("\t");
for(j=0;j<r;j++)
printf("%d",p[i].after[j]);
cn=0;
cz=0;
c++;
break;
}
else {
cn=0;cz=0;
}
}
}
if(c==n)
printf("\n the above sequence is a safe sequence");
else
printf("\n deadlock occured");
//getch();
}
```

INPUT:

Enter the no of processes:

3

Enter the no of resources:

3

process 1

maximum value for resource 1:3

maximum value for resource 2:4

maximum value for resource 3:5

allocated from resource 1:2

allocated from resource 2:1

allocated from resource 3:3

process 2

maximum value for resource 1:5

maximum value for resource 2:6

maximum value for resource 3:7

allocated from resource 1:2

allocated from resource 2:3

allocated from resource 3:4

process 3

maximum value for resource 1:5

maximum value for resource 2:7

maximum value for resource 3:8

allocated from resource 1:2

allocated from resource 2:3

allocated from resource 3:4

Enter total value of resource 1:12

Enter total value of resource 2:13

Enter total value of resource 3:14

OUTPUT:

max allocated needed total avail

P1	345	213	132	121314 663
P2	567	234	333	

P3 578 234 344

	AVAIL BEFORE	AVAIL AFTER
p1	531	876
p2	543	101010
p3	766	121314

the above sequence is a safe sequence

VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	Differentiate deadlock avoidance and prevention.	CO3	Understand
2	Give example of deadlock occurrence.	CO3	Remember
3	How do we calculate the need for process?	CO3	Remember
4	What is the name of the algorithm to avoid deadlock?	CO3	Remember
5	Define Banker's algorithm.	CO3	Remember
6	Each request requires that the system to decide whether the current request can be satisfied or must wait to avoid a future possible deadlock.	CO3	Understand
7	Define mutual exclusion.	CO3	Remember
8	What is Deadlock?	CO3	Remember
9	Define Safe State.	CO3	Remember
10	What is Unsafe state?	CO3	Remember
11	Are all unsafe states are deadlocks?	CO3	Understand
12	If no cycle exists in the resource allocation graph	CO3	Remember
13	The resource allocation graph is not applicable to a resource allocation system :	CO3	Remember
14	What is resource allocation?	CO3	Remember
15	List Data structures available in the Banker's algorithm.	CO3	Remember
16	What is the content of the matrix Need?	CO3	Remember
17	What are the necessary conditions of Deadlock?	CO3	Remember
18	How can we prevent deadlock occurrence.	CO3	Understand
19	What is Synchronization?	CO3	Remember
20	Give classical problems of synchronization	CO3	Remember

EXPERIMENT-4

AIM: Write a program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

OBJECTIVE:

To implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

DESCRIPTION:

Producer consumer problem is also known as bounded buffer problem. In this problem we have two processes, producer and consumer, who share a fixed size buffer. Producer work is to produce data or items and put in buffer. Consumer work is to remove data from buffer and consume it. We have to make sure that producer do not produce data when buffer is full and consumer do not remove data when buffer is empty.

The producer should go to sleep when buffer is full. Next time when consumer removes data it notifies the producer and producer starts producing data again. The consumer should go to sleep when buffer is empty. Next time when producer add data it notifies the consumer and consumer starts consuming data. This solution can be achieved using semaphores.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
        int n;
        void producer();
        void consumer();
        int wait(int);
        int signal(int);
        printf("\n1.Producer\n2.Consumer\n3.Exit");
        while(1)
```

```
        {
                printf("\nEnter your choice:");
                scanf("%d",&n);
                switch(n)
                {
                        case 1:  if((mutex==1)&&(empty!=0))
                                                producer();
                                        else
                                                printf("Buffer is full!!");
                                        break;
                        case 2:  if((mutex==1)&&(full!=0))
                                                consumer();
                                        else
                                                printf("Buffer is empty!!");
                                        break;
                        case 3:
                                        exit(0);
                                        break;
                }
        }

        return 0;
}

int wait(int s)
{
        return (--s);
}
```

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

void producer()
{
        mutex=wait(mutex);
        full=signal(full);
        empty=wait(empty);
        x++;
        printf("\nProducer produces the item %d",x);
        mutex=signal(mutex);
}

void consumer()
{
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        printf("\nConsumer consumes item %d",x);
        x--;
        mutex=signal(mutex);
}
```

Output

1.Producer

2.Consumer

3.Exit

Enter your choice:1

Producer produces the item 1

Enter your choice:2

Consumer consumes item 1

Enter your choice:2

Buffer is empty!!

Enter your choice:1

Producer produces the item 1

Enter your choice:1

Producer produces the item 2

Enter your choice:1

Producer produces the item 3

Enter your choice:1

Buffer is full!!

Enter your choice:3

VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	What is safe state?	CO4	Understand
2	What are the conditions that cause deadlock?	CO4	Understand
3	How do we calculate the need for process?	CO4	Apply
4	What is Semaphore?	CO4	Understand
5	Define Race Condition.	CO4	Remember
6	Define Lock.	CO4	Remember
7	Define Orphan Process.	CO4	Remember
8	What is monitor?	CO4	Understand
9	Define aging.	CO4	Remember
10	Which condition is required for a deadlock to be possible?	CO4	Understand
11	What is safety alg.	CO4	Understand
12	Define Process synchronization.	CO4	Remember
13	What is Concurrency?	CO4	Understand
14	Define Preemption.	CO4	Remember
15	What is no preemption?	CO4	Understand
16	List the advantages of Semaphores.	CO4	Understand
17	Define Message Queue.	CO4	Remember
18	What is Pipe?	CO4	Understand
19	What is IPC?	CO4	Understand
20	Define Inter Process Communication.	CO4	Remember

EXPERIMENT-5

AIM: Write a C program to illustrate the following IPC mechanisms

a) Pipes b) FIFOs c) Message Queues d) Shared Memory

OBJECTIVE:

C program to illustrate the Pipes IPC mechanism

PROGRAM

```
#include <stdio.h>
#include <unistd.h>
#define MSGSIZE 16
char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";

int main()
{
   char inbuf[MSGSIZE];
   int p[2], i;

   if (pipe(p) < 0)
      exit(1);

   /* continued */
   /* write pipe */
```

```
    write(p[1], msg1, MSGSIZE);
    write(p[1], msg2, MSGSIZE);
    write(p[1], msg3, MSGSIZE);

    for (i = 0; i < 3; i++) {
        /* read pipe */
        read(p[0], inbuf, MSGSIZE);
        printf("% s\n", inbuf);
    }
    return 0;
}
```

Output:

hello, world #1

hello, world #2

hello, world #3

OBJECTIVE

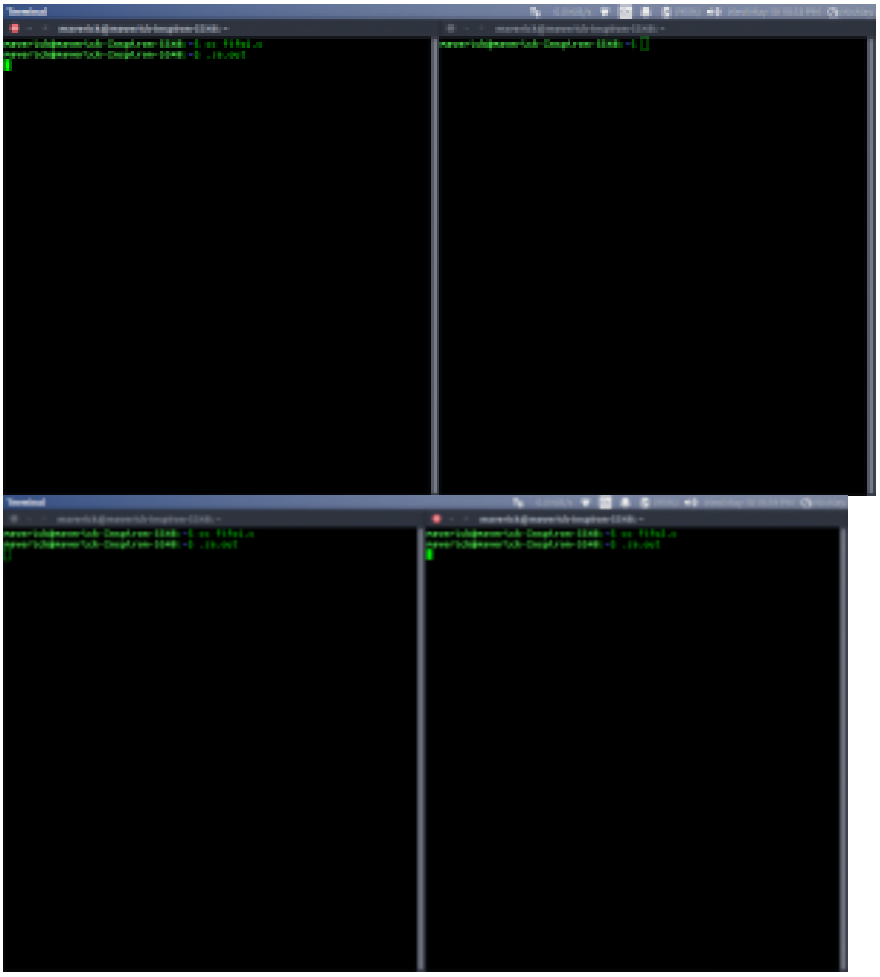
b) C program to illustrate the FIFO IPC mechanism

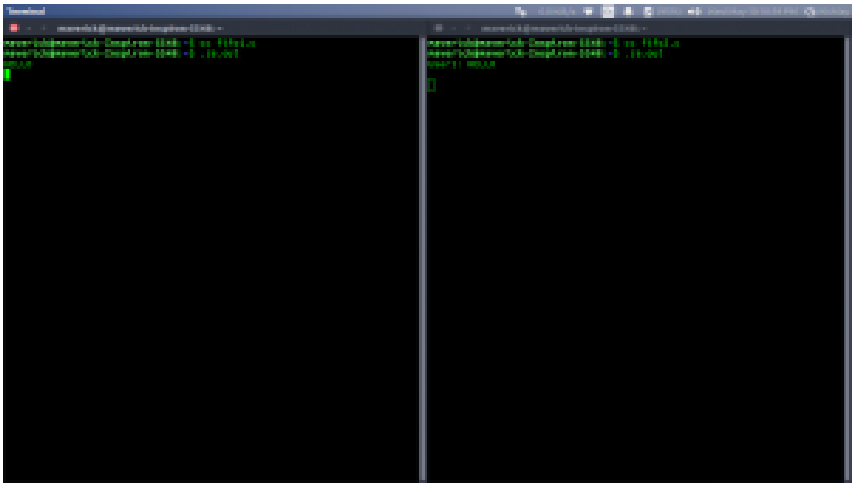
PROGRAM

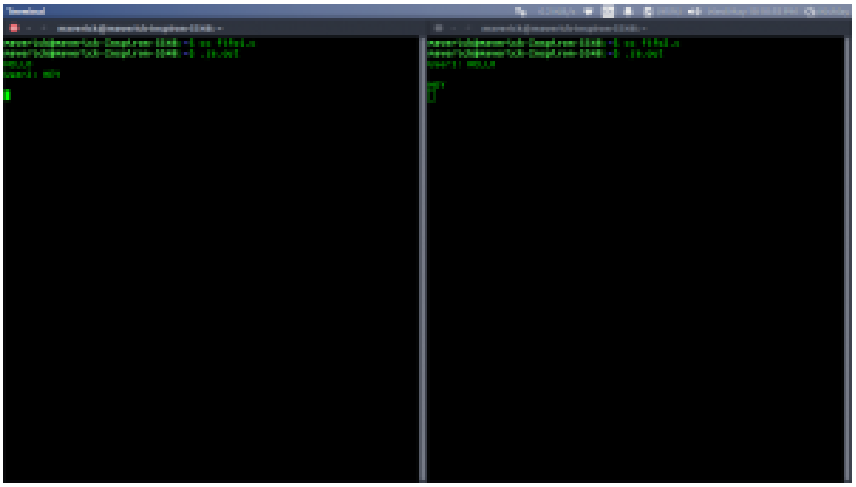
```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;

    // FIFO file path
```

```
    char * myfifo = "/tmp/myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>, <permission>)
    mkfifo(myfifo, 0666);

    char arr1[80], arr2[80];
    while (1)
    {
        // Open FIFO for write only
        fd = open(myfifo, O_WRONLY);

        // Take an input arr2ing from user.
        // 80 is maximum length
        fgets(arr2, 80, stdin);

        // Write the input arr2ing on FIFO
        // and close it
        write(fd, arr2, strlen(arr2)+1);
        close(fd);

        // Open FIFO for Read only
        fd = open(myfifo, O_RDONLY);

        // Read from FIFO
        read(fd, arr1, sizeof(arr1));

        // Print the read message
        printf("User2: %s\n", arr1);
        close(fd);
    }
    return 0;
}
```

Output:







OBJECTIVE

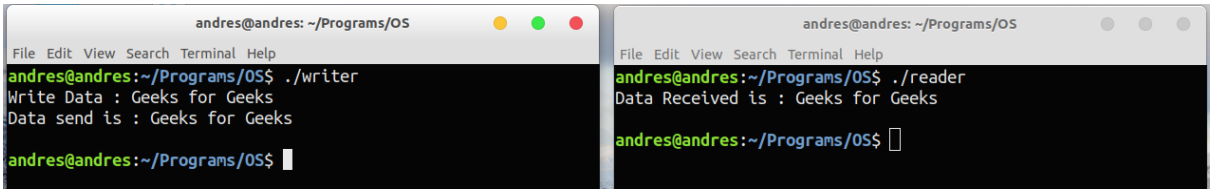
c) C program to illustrate the Message Queue IPC mechanism

PROGRAM:

```
// C Program for Message Queue (Writer Process)
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;

    printf("Write Data : ");
    gets(message.mesg_text);
```

```
    // msgsnd to send message
    msgsnd(msgid, &message, sizeof(message), 0);

    // display the message
    printf("Data send is : %s \n", message.mesg_text);

    return 0;
}


// C Program for Message Queue (Reader Process)
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);

    // msgrcv to receive message
    msgrcv(msgid, &message, sizeof(message), 1, 0);

    // display the message
    printf("Data Received is : %s \n",
                    message.mesg_text);

    // to destroy the message queue
    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}
```

Output:



OBJECTIVE

d) C program to illustrate the Shared Memory IPC mechanism

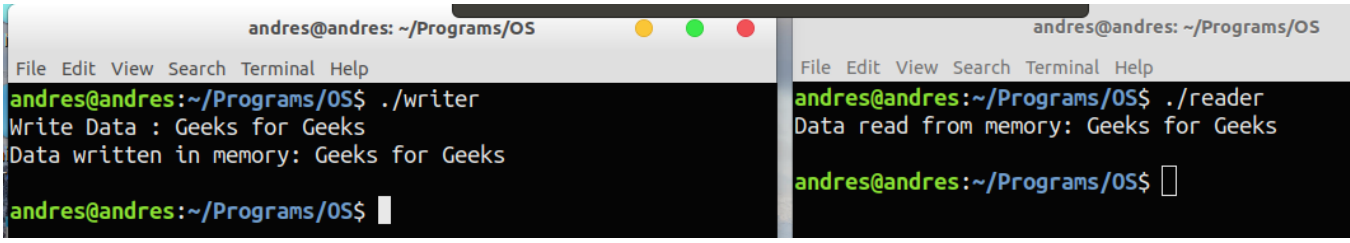
PROGRAM:

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;

int main()
{
   // ftok to generate unique key
   key_t key = ftok("shmfile",65);

   // shmget returns an identifier in shmid
   int shmid = shmget(key,1024,0666|IPC_CREAT);

   // shmat to attach to shared memory
   char *str = (char*) shmat(shmid,(void*)0,0);

   cout<<"Write Data : ";
   gets(str);

   printf("Data written in memory: %s\n",str);

   //detach from shared memory
   shmdt(str);

   return 0;
}
```

SHARED MEMORY FOR READER PROCESS

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;

int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);

    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);

    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);

    printf("Data read from memory: %s\n",str);

    //detach from shared memory
    shmdt(str);

    // destroy the shared memory
    shmctl(shmid,IPC_RMID,NULL);

    return 0;
}
```

Output:



VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	What are local and global page replacements?	CO4	Remember
2	Define latency, transfer and seek time with respect to disk I/O.	CO4	Understand
3	Describe the Buddy system of memory allocation.	CO4	Understand
4	What is time-stamping?	CO4	
5	How are the wait/signal operations for monitor different from those for semaphores?	CO4	Understand
6	In the context of memory management, what are placement and replacement algorithms?	CO4	Understand
7	In loading programs into memory, what is the difference between load-time dynamic linking and run-time dynamic linking?	CO4	Understand
8	What are demand-paging and pre-paging?	CO4	Remember
9	What is fixed partitioning?	CO4	Remember
10	What is page fault?	CO4	Remember
11	What has triggered the need for multitasking in PCs?	CO4	Remember
12	What are the four layers that Windows NT have in order to achieve independence?	CO4	Remember
13	Explain compaction.	CO4	Understand
14	What are page frames?	CO4	Remember
15	What are pages?	CO4	Remember
16	Differentiate between logical and physical address.	CO4	Understand
17	When does page fault error occur?	CO4	Remember
18	Define thrashing?	CO4	Remember
19	What is the criteria for the best page replacement algorithm?	CO4	Remember
20	What is Belady's anomaly ?	CO4	Remember

EXPERIMENT -6

AIM: Write a C program to simulate the following techniques of memory management

a) Paging b) Segmentation

6.1 OBJECTIVE

C program to simulate paging technique of memory management.

6.2 DESCRIPTION

In computer operating systems, paging is one of the memory management schemes by which a computer stores and retrieves data from the secondary storage for use in main memory. In the paging memory-management scheme, the operating system retrieves data from secondary storage in same-size blocks called pages. Paging is a memory-management scheme that permits the physical address space a process to be noncontiguous. The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages. When a process is to be executed, its pages are loaded into any available memory frames from their source.

AIM: To simulate paging technique of memory management.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
main()
{
int ms, ps, nop, np, rempages, i, j, x, y, pa, offset; int s[10], fno[10][20];
clrscr();
printf("\nEnter the memory size -- "); scanf("%d",&ms);
printf("\nEnter the page size -- "); scanf("%d",&ps);
nop = ms/ps;
printf("\nThe no. of pages available in memory are -- %d ",nop);
printf("\nEnter number of processes -- "); scanf("%d",&np);
rempages = nop;
for(i=1;i<=np;i++)
```

```
{
printf("\nEnter no. of pages required for p[%d]-- ",i); scanf("%d",&s[i]);
if(s[i] >rempages)
{
printf("\nMemory is Full"); break;
}
rempages = rempages - s[i];
printf("\nEnter pagetable for p[%d] -,i);
for(j=0;j<s[i];j++)
scanf("%d",&fno[i][j]);
}
printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset ");
scanf("%d %d %d",&x,&y, &offset);
if(x>np || y>=s[i] || offset>=ps)
printf("\nInvalid Process or Page Number or offset");
else{
pa=fno[x][y]*ps+offset;
printf("\nThe Physical Address is -- %d",pa);
}
getch();}
```

INPUT

Enter the memory size – 1000

Enter the page size -- 100

The no. of pages available in memory are -- 10

Enter number of processes -- 3

Enter no. of pages required for p[1] -- 4

Enter pagetable for p[1] --- 8 6 9 5

Enter no. of pages required for p[2] -- 5

Enter pagetable for p[2] --- 1 4 5 7 3

Enter no. of pages required for p[3] -- 5

OUTPUT

Memory is Full

Enter Logical Address to find Physical Address

Enter process no. and pagenumber and offset -- 2 3 60

The Physical Address is -- 760

OBJECTIVE: To implement the memory management policy-segmentation

PROGRAM LOGIC:

1. Start the program.
2. Get the number of segments.
3. Get the base address and length for each segment.
4. Get the logical address.
5. Check whether the segment number is within the limit, if not display the error message.
6. Check whether the byte reference is within the limit, if not display the error message.
7. Calculate the physical memory and display it.
8. Stop the program

SOURCE CODE:

```
#include<stdio.h>
 #include <conio.h>
#include<math.h>
 int sost;
 void gstinfo();
void ptladdr();
```

```
struct segtab
{ int sno;
 int baddr;
int limit;
 int val[10];
}st[10];
void gstinfo()
{ int i,j;
printf("\n\tEnter the size of the segment table: ");
scanf("%d",&sost);
for(i=1;i<=sost;i++)
 {
 printf("\n\tEnter the information about segment: %d",i);
 st[i].sno = i;
printf("\n\tEnter the base Address: ");
scanf("%d",&st[i].baddr);
printf("\n\tEnter the Limit: ");
scanf("%d",&st[i].limit);
for(j=0;j<=sost;i++)
printf("\t\t%d \t\t%d\t\t%d\n\n",st[i].sno,st[i].baddr,st[i].limit);
 printf("\n\nEnter the logical Address: ");
scanf("%d",&swd);
n=swd;
 while (n != 0)
{
 n=n/10; d++;
 }
 s = swd/pow(10,d-1);
```

```
 disp = swd%(int)pow(10,d-1);
 if(s<=sost)
{
if(disp < st[s].limit)
{
 paddr = st[s].baddr + disp;
printf("\n\t\tLogical Address is: %d",swd);
printf("\n\t\tMapped Physical address is: %d",paddr);
printf("\n\tThe value is: %d",( st[s].val[disp] ) );
 }
 Else
 printf("\n\t\tLimit of segment %d is high\n\n",s);
}
else
 printf("\n\t\tInvalid Segment Address \n");
 }
 void main()
 { char ch;
 clrscr();
 gstinfo();
do
{
 ptladdr();
printf("\n\t Do U want to Continue(Y/N)");
flushall();
scanf("%c",&ch);
}while (ch == 'Y' || ch == 'y' );
getch();
```

}

INPUT AND OUTPUT:

Enter the size of the segment table: 3

Enter the information about segment: 1

Enter the base Address: 4

Enter the Limit: 5

Enter the 4 address Value: 11

Enter the 5 address Value: 12

Enter the 6 address Value: 13

Enter the 7 address Value: 14

Enter the 8 address Value: 15

Enter the information about segment: 2

Enter the base Address: 5

Enter the Limit: 4

Enter the 5 address Value: 21

Enter the 6 address Value: 31

Enter the 7 address Value: 41

Enter the 8 address Value: 51

Enter the information about segment: 3

Enter the base Address: 3

Enter the Limit: 4

Enter the 3 address Value: 31

Enter the 4 address Value: 41

Enter the 5 address Value: 41

Enter the 6 address Value: 51

SEGMENT TABLE SEG.NO BASE ADDRESS LIMIT

1 4 5

2 5 4

3 3 4

Enter the logical Address: 3

Logical Address is: 3

Mapped Physical address is: 3

The value is: 31

Do U want to Continue(Y/N)

SEGMENT TABLE SEG.NO BASE ADDRESS LIMIT

1 4 5

2 5 4

3 3 4

Enter the logical Address: 1

Logical Address is: 1

Mapped Physical address is: 4

The value is: 11 Do U want to Continue(Y/N)

VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	What are local and global page replacements?	CO5	Understand
2	Define latency, transfer and seek time with respect to disk I/O.	CO5	Remember
3	Describe the Buddy system of memory allocation.	CO5	Understand
4	What is time-stamping?	CO5	Remember
5	How are the wait/signal operations for monitor different from those for semaphores?	CO5	Understand
6	In the context of memory management, what are placement and replacement algorithms?	CO5	Remember
7	In loading programs into memory, what is the difference between load-time dynamic linking and run-time dynamic linking?	CO5	Understand
8	What are demand-paging and pre-paging?	CO5	Remember
9	What is fixed partitioning?	CO5	Remember
10	What is page fault?	CO5	Remember
11	What has triggered the need for multitasking in PCs?	CO5	Remember

12	What are the four layers that Windows NT have in order to achieve independence?	CO5	Remember
13	Explain compaction.	CO5	Remember
14	What are page frames?	CO5	Remember
15	What are pages?	CO5	Remember
16	Differentiate between logical and physical address.	CO5	Understand
17	When does page fault error occur?	CO5	Remember
18	What is thrashing?	CO5	Remember
19	What is the criteria for the best page replacement algorithm?	CO5	Remember
20	What is Belady's anomaly ?	CO5	Remember

EXPERIMENT -7

AIM: Write a C program to simulate the following Page replacement policies.

a) FIFO b) LRU c) Optimal

```
#include <stdio.h>
int main() {
    int pages[50], frames[10];
    int n, capacity;
    int i, j, k = 0, found, faults = 0;
    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", &capacity);
    // Initialize frames with -1
    for (i = 0; i < capacity; i++) {
        frames[i] = -1;
    }
    printf("\nFIFO Page Replacement Process:\n");
    for (i = 0; i < n; i++) {
        found = 0;
        // Check if page already exists
        for (j = 0; j < capacity; j++) {
```

```
            if (frames[j] == pages[i]) {
                found = 1;
                break;
            }
        }
        // If page fault occurs
        if (!found) {
            frames[k] = pages[i];
            k = (k + 1) % capacity;   // FIFO replacement
            faults++;
        }
        // Print current frame status
        for (j = 0; j < capacity; j++) {
            if (frames[j] == -1)
                printf("- ");
            else
                printf("%d ", frames[j]);
        }
        printf("\n");
    }
    printf("\nTotal Page Faults = %d\n", faults);
    return 0;
}
```

Input:

Enter number of pages: 12

Enter page reference string: 1 2 3 4 1 2 5 1 2 3 4 5

Enter number of frames: 3

Output:

FIFO Page Replacement Process:

1 - -

1 2 -

1 2 3

4 2 3

4 1 3

4 1 2

5 1 2

5 1 2

5 1 2

5 3 2

5 3 4

5 3 4

Total Page Faults = 9

b) LRU

Program:

```
#include <stdio.h>
int main() {
    int pages[50], frames[10], time[10];
    int n, capacity;
    int i, j, pos, faults = 0, count = 0;
    int found, min;
    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", &capacity);
// Initialize frames and time arrays
for (i = 0; i < capacity; i++) {
    frames[i] = -1;
    time[i] = 0;
}
printf("\nLRU Page Replacement Process:\n");
for (i = 0; i < n; i++) {
    found = 0;
    // Check if page already exists
    for (j = 0; j < capacity; j++) {
        if (frames[j] == pages[i]) {
            count++;
            time[j] = count;   // Update recent use time
            found = 1;
            break;
        }
    }
    // If page fault occurs
    if (!found) {
        min = time[0];
        pos = 0;
```

```
            for (j = 1; j < capacity; j++) {
                if (time[j] < min) {
                    min = time[j];
                    pos = j;
                }
            }
            count++;
            frames[pos] = pages[i];
            time[pos] = count;
            faults++;
        }
        // Print current frame status
        for (j = 0; j < capacity; j++) {
            if (frames[j] == -1)
                printf("- ");
            else
                printf("%d ", frames[j]);
        }
        printf("\n");
    }
    printf("\nTotal Page Faults = %d\n", faults);
    return 0;
}
```

Input:

Enter number of pages: 12

Enter page reference string: 1 2 3 4 1 2 5 1 2 3 4 5

Enter number of frames: 3

Output:

LRU Page Replacement Process:

1 - -

1 2 -

1 2 3

4 2 3

4 1 3

4 1 2

5 1 2

5 1 2

5 1 2

3 1 2

3 4 2

3 4 5

Total Page Faults = 10

c) Write a C program to simulate the following Page replacement policie using Optimal

Program:

```
#include <stdio.h>
int main() {
    int pages[50], frames[10];
    int n, capacity;
    int i, j, k, pos, faults = 0;
    int found, farthest, index;
    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", &capacity);
// Initialize frames
for (i = 0; i < capacity; i++) {
    frames[i] = -1;
}
printf("\nOptimal Page Replacement Process:\n");
for (i = 0; i < n; i++) {
    found = 0;
    // Check if page already exists
    for (j = 0; j < capacity; j++) {
        if (frames[j] == pages[i]) {
            found = 1;
            break;
        }
    }
    // If page fault occurs
    if (!found) {
        pos = -1;
        farthest = i + 1;
        for (j = 0; j < capacity; j++) {
            index = -1;
            // Find future use of current frame page
            for (k = i + 1; k < n; k++) {
```

```
            if (frames[j] == pages[k]) {
                index = k;
                break;
            }
        }
        // If page is never used again
        if (index == -1) {
            pos = j;
            break;
        }
        // Find farthest future use
        if (index > farthest) {
            farthest = index;
            pos = j;
        }
    }
    // If no page selected, replace first
    if (pos == -1)
        pos = 0;
    frames[pos] = pages[i];
    faults++;
}
// Print current frame status
for (j = 0; j < capacity; j++) {
    if (frames[j] == -1)
        printf("- ");
```

```
            else
                printf("%d ", frames[j]);
        }
        printf("\n");
    }
    printf("\nTotal Page Faults = %d\n", faults);
    return 0;}
```

Input Format:

Enter number of pages: 12

Enter page reference string: 1 2 3 4 1 2 5 1 2 3 4 5

Enter number of frames: 3

Output Format:

Optimal Page Replacement Process:

1 - -

1 2 -

1 2 3

1 2 4

1 2 4

1 2 4

1 2 5

1 2 5

1 2 5

3 2 5

4 2 5

4 2 5

Total Page Faults = 7

VIVA QUESTIONS

S.No	Question	CO	Bloom's Taxonomy
1	What is page replacement in operating systems?	CO5	Remember
2	What is a page fault?	CO5	Remember
3	Why are page replacement algorithms needed?	CO5	Understand
4	Explain FIFO page replacement policy.	CO5	Understand
5	Which page is replaced first in FIFO?	CO5	Remember
6	What is Belady's anomaly?	CO5	Understand
7	Explain LRU page replacement policy.	CO5	Understand
8	Which page is replaced in LRU?	CO5	Remember
9	Why is LRU better than FIFO?	CO5	Analyze
10	What is the role of time/counter in LRU?	CO5	Understand
11	Explain Optimal page replacement.	CO5	Understand
12	Why is Optimal considered the best algorithm?	CO5	Analyze
13	Can Optimal be implemented in real systems? Why?	CO5	Analyze
14	Which page is replaced in Optimal policy?	CO5	Remember
15	Compare FIFO and LRU.	CO5	Analyze
16	Compare LRU and Optimal.	CO5	Analyze
17	Which algorithm gives minimum page faults?	CO5	Evaluate
18	Which algorithm is easiest to implement?	CO5	Evaluate

S.No	Question	CO	Bloom's Taxonomy
19	What are the disadvantages of FIFO?	CO5	Analyze
20	Which page replacement policy is most practical ?	CO5	Evaluate

OPEN ENDED EXPERIMENT

1.Write programs using the I/O system calls of UNIX/LINUX operating system

(open, read, write, close, fcntl, seek, stat, opendir, readdir)

Fork()

```
#include<stdio.h>
#include<unistd.h>
int main()
{
fork();
printf("hello\n PID= %d\n",getpid());
return 0;
}
```

```
[227Y1A0544@mlritm ~]$ vi fork1.c
[227Y1A0544@mlritm ~]$ cc fork1.c
[227Y1A0544@mlritm ~]$ ./a.out
hello
 PID= 28592
hello
 PID= 28593
[227Y1A0544@mlritm ~]$
```

Output:

Fork ()

Example2:

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

```
#include<unistd.h>
int main()
{
fork();
fork();

printf("hello\n PID= %d\n",getpid());
return 0;
}
```

Output:

```
[227Y1A0544@mlritm ~]$ vi fork2.c
[227Y1A0544@mlritm ~]$ cc fork2.c
[227Y1A0544@mlritm ~]$ ./a.out
hello
 PID= 28627
hello
hello
 PID= 28629
 PID= 28628
hello
 PID= 28630
[227Y1A0544@mlritm ~]$
```

Read ()

```
#include<unistd.h>
int main()
{
int n;
char b[20];
n=read(0,b,30);
write(1,b,10);
}
```

Output:

```
[227Y1A0544@mlritm ~]$ vi rsc2.c
[227Y1A0544@mlritm ~]$ cc rsc2.c
[227Y1A0544@mlritm ~]$ ./a.out
1234567890abcd
1234567890[227Y1A0544@mlritm ~]$
```

Example 2:

```
#include<unistd.h>
int main()
{
int n;
char b[30];
n=read(0,b,30);
write(1,b,n);
}
```

```
[227Y1A0544@mlritm ~]$ vi rsc3.c
[227Y1A0544@mlritm ~]$ cc rsc3.c
[227Y1A0544@mlritm ~]$ ./a.out
1234567890abcd
1234567890abcd
[227Y1A0544@mlritm ~]$
```

```
#include<unistd.h>
int main()
{
int n;
char b[10];
n=read(0,b,10);
write(1,b,n);
}
```

output:

```
[227Y1A0544@mlritm ~]$ vi rsc3.c
[227Y1A0544@mlritm ~]$ cc rsc3.c
[227Y1A0544@mlritm ~]$ ./a.out
1234567890abcd
1234567890[227Y1A0544@mlritm ~]$ abcd
bash: abcd: command not found...
```

Read()

```
#include<unistd.h>
int main()
{
int n;
char b[30];
n=read(0,b,10);
write(1,b,10);
}
```

Output:

```
[227Y1A0544@mlritm ~]$ vi rscl.c
[227Y1A0544@mlritm ~]$ cc rscl.c
[227Y1A0544@mlritm ~]$ ./a.out
12345
12345
[227Y1A0544@mlritm ~]$ ./a.out
1234567890
1234567890[227Y1A0544@mlritm ~]$
[227Y1A0544@mlritm ~]$ ./a.out
1234567890abcdef
1234567890[227Y1A0544@mlritm ~]$ abcdef
bash: abcdef: command not found...
[227Y1A0544@mlritm ~]$
```

Open,read,close

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

int main() {
   int file_descriptor;
   char buffer[1024];
   ssize_t bytes_read;

   file_descriptor = open("sample.txt", O_RDONLY);
   if (file_descriptor == -1) {
      perror("open");
      exit(1);
   }

bytes_read = read(file_descriptor, buffer, sizeof(buffer));
   if (bytes_read == -1) {
      perror("read");
      close(file_descriptor);
      exit(1);
   }

   close(file_descriptor);

   printf("Read %zd bytes: %s\n", bytes_read, buffer);
```

return 0;

}

Output:

```
[227Y1A0544@mlritm ~]$ vi sample.txt
```

Marri laxman reddy

```
[227Y1A0544@mlritm ~]$ vi  prog1.c
[227Y1A0544@mlritm ~]$ cc  prog1.c
[227Y1A0544@mlritm ~]$ ./a.out
Read 20 bytes: marri
laxman reddy
```

Write()

Example1:

#include<unistd.h>

int main()

{

write(1,"hello",5);

}

Output:

```
[227Y1A0544@mlritm ~]$ vi w1.c
[227Y1A0544@mlritm ~]$ cc w1.c
[227Y1A0544@mlritm ~]$ ./a.out
hello[227Y1A0544@mlritm ~]$
```

Example2:

#include<unistd.h>

int main()

{

write(1,"hello",2);

}

Output:

```
     227Y1A0544@mlritm ~]$ vi w2.c
[227Y1A0544@mlritm ~]$ cc w2.c
[227Y1A0544@mlritm ~]$ ./a.out
he[227Y1A0544@mlritm ~]$
```

Write()

Example3:

#include<unistd.h>

int main()

{

write(1,"hello",20);

}

Output:

```
[227Y1A0544@mlritm ~]$ vi w3.c
[227Y1A0544@mlritm ~]$ gcc w3.c
[227Y1A0544@mlritm ~]$ ./a.out
hello4[227Y1A0544@mlritm ~]$ PuTTY
```

2. Demonstrate process synchronization using semaphores for shared buffer coordination.

Program:

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define SIZE 5

int buffer[SIZE], in = 0, out = 0;

```
sem_t empty, full;
pthread_mutex_t mutex;
void* producer(void* arg) {
int i = 1;
while (1) {
sem_wait(&empty);
pthread_mutex_lock(&mutex);
buffer[in] = i;
printf("Produced: %d\n", i++);
in = (in + 1) % SIZE;
pthread_mutex_unlock(&mutex);
sem_post(&full);
sleep(1);
}
}
void* consumer(void* arg) {
while (1) {
sem_wait(&full);
pthread_mutex_lock(&mutex);
int item = buffer[out];
printf("Consumed: %d\n", item);
out = (out + 1) % SIZE;
pthread_mutex_unlock(&mutex);
sem_post(&empty);
sleep(2);
}
}
int main() {
```

```
pthread_t prod, cons;
sem_init(&empty, 0, SIZE);
sem_init(&full, 0, 0);
pthread_mutex_init(&mutex, NULL);
pthread_create(&prod, NULL, producer, NULL);
pthread_create(&cons, NULL, consumer, NULL);
pthread_join(prod, NULL);
pthread_join(cons, NULL);
return 0;
}
```

Expected output:

```
Produced: 1
Produced: 2
Consumed: 1
Produced: 3
Consumed: 2
Produced: 4
Consumed: 3
```

3. Create a basic shell that accepts user commands and executes them using system calls.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#define MAX 100
int main() {
char command[MAX];
char *args[10];
while (1) {
```

```
printf("myshell> ");
fgets(command, MAX, stdin);
command[strcspn(command, "\n")] = 0;
if (strcmp(command, "exit") == 0)
break;
int i = 0;
args[i] = strtok(command, " ");
while (args[i] != NULL)
args[++i] = strtok(NULL, " ");
pid_t pid = fork();
if (pid == 0) {
execvp(args[0], args);
perror("exec failed");
exit(1);
} else {
wait(NULL);
}
}
return 0;
}
```

Output:

```
Enter number of pages: 12
Enter page reference string: 1 2 3 1 4 5 2 1 2 3 4 5
Frames: 1 -1 -1
Frames: 1 2 -1
Frames: 1 2 3
Frames: 1 2 3
Frames: 4 2 3
```

Frames: 4 5 3

Frames: 2 5 3

Frames: 1 5 3

Frames: 1 2 3

Frames: 1 2 3

Frames: 4 2 3

Frames: 4 5 3

Total Page Faults = 9