VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

OPERATING SYSTEMS

Submitted by

ZAIN MAHEDI(1WA23CS056)

in partial fulfillment for the award of the degree of

# BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

Feb-2025 to June-2025

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “OPERATING SYSTEMS – 23CS4PCOPS” carried out by Zain Mahedi(1wa23cs056), who is Bonafide student of B. M. S. College of Engineering. It is in partial fulfilment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year Feb 2025- June 2025. The Lab report has been approved as it satisfies the academic requirements in respect of a OPERATING SYSTEMS - (23CS4PCOPS) work prescribed for the said degree.

Sandhya A Kulkarni Dr. Kavitha Sooda

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

Index Sheet

|  |  |  |
| --- | --- | --- |
| Sl.  No. | Experiment Title | Page No. |
| 1. | Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.  →FCFS  → SJF (pre-emptive & Non-preemptive) | 1-10 |
| 2. | Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.  → Priority (pre-emptive & Non-pre-emptive)  →Round Robin (Experiment with different quantum sizes for RR algorithm) | 11-19 |
| 3. | Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue. | 20-24 |
| 4. | Write a C program to simulate Real-Time CPU Scheduling algorithms:   1. Rate- Monotonic 2. Earliest-deadline First 3. Proportional scheduling | 24-34 |
| 5. | Write a C program to simulate producer-consumer problem using semaphores | 35-38 |
| 6. | Write a C program to simulate the concept of Dining Philosophers problem. | 39-43 |
| 7. | Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance. | 44-47 |
| 8. | Write a C program to simulate deadlock detection | 48-51 |
| 9. | Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit   1. Best-fit 2. First-fit | 52-56 |
| 10. | Write a C program to simulate page replacement algorithms a) FIFO   1. LRU 2. Optimal | 57-63 |

Course Outcomes

|  |  |
| --- | --- |
| C01 | Apply the different concepts and functionalities of Operating System |
| C02 | Analyse various Operating system strategies and techniques |
| C03 | Demonstrate the different functionalities of Operating System. |
| C04 | Conduct practical experiments to implement the functionalities of Operating system. |

# Program – 1

Write a C program to simulate the FCFS CPU scheduling algorithm to find turnaround  
time and waiting time.

Code

#include <stdio.h>  
#include <limits.h>  
  
typedef struct {  
 int id, arrival, burst, remaining, waiting, turnaround, completion, response, started;  
} Process;  
  
void swap(Process \*a, Process \*b) {  
 Process temp = \*a;  
 \*a = \*b;  
 \*b = temp;  
}  
void sortByArrival(Process p[], int n) {  
 for (int i = 0; i < n - 1; i++) {  
 for (int j = 0; j < n - i - 1; j++) {  
 if (p[j].arrival > p[j + 1].arrival) {  
 swap(&p[j], &p[j + 1]);  
 }  
 }  
 }  
}  
void fcfs(Process p[], int n) {  
 sortByArrival(p, n);  
 int time = 0;  
  
 for (int i = 0; i < n; i++) {  
 if (time < p[i].arrival)  
 time = p[i].arrival;  
  
 p[i].response = time - p[i].arrival;  
 p[i].completion = time + p[i].burst;  
 p[i].turnaround = p[i].completion - p[i].arrival;  
 p[i].waiting = p[i].turnaround - p[i].burst;  
 time = p[i].completion;  
 }  
}  
void sjfNonPreemptive(Process p[], int n) {  
 int completed = 0, time = 0;  
  
 while (completed < n) {  
 int minIndex = -1, minBurst = INT\_MAX;  
  
 for (int i = 0; i < n; i++) {  
 if (p[i].arrival <= time && p[i].completion == 0) {  
 if (p[i].burst < minBurst || (p[i].burst == minBurst && p[i].arrival < p[minIndex].arrival)) {  
 minBurst = p[i].burst;  
 minIndex = i;  
 }  
 }  
 }  
  
 if (minIndex == -1) {  
 time++;  
 continue;  
 }  
 p[minIndex].response = time - p[minIndex].arrival;  
 time += p[minIndex].burst;  
 p[minIndex].completion = time;  
 p[minIndex].turnaround = p[minIndex].completion - p[minIndex].arrival;  
 p[minIndex].waiting = p[minIndex].turnaround - p[minIndex].burst;  
 completed++;  
 }  
}  
void sjfPreemptive(Process p[], int n) {  
 int completed = 0, time = 0, minIndex = -1, minBurst = INT\_MAX;  
  
 while (completed < n) {  
 minIndex = -1, minBurst = INT\_MAX;  
  
 for (int i = 0; i < n; i++) {  
 if (p[i].arrival <= time && p[i].remaining > 0) {  
 if (p[i].remaining < minBurst || (p[i].remaining == minBurst && p[i].arrival < p[minIndex].arrival)) {  
 minBurst = p[i].remaining;  
 minIndex = i;  
 }  
 }  
 }  
 if (minIndex == -1) {  
 time++;  
 continue;  
 }  
 if (p[minIndex].started == 0) {  
 p[minIndex].response = time - p[minIndex].arrival;  
 p[minIndex].started = 1;  
 }  
  
 p[minIndex].remaining--;  
 time++;  
  
 if (p[minIndex].remaining == 0) {  
 p[minIndex].completion = time;  
 p[minIndex].turnaround = p[minIndex].completion - p[minIndex].arrival;  
 p[minIndex].waiting = p[minIndex].turnaround - p[minIndex].burst;  
 completed++;  
 }  
 }  
}  
  
void displayResults(Process p[], int n, const char \*title) {  
 printf("\n--- %s ---\n", title);  
 printf("\nPID\tAT\tBT\tCT\tTAT\tWT\tRT\n");  
  
 float totalWT = 0, totalTAT = 0, totalRT = 0;  
 for (int i = 0; i < n; i++) {  
 printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].arrival, p[i].burst, p[i].completion, p[i].turnaround, p[i].waiting, p[i].response);  
 totalWT += p[i].waiting;  
 totalTAT += p[i].turnaround;  
 totalRT += p[i].response;  
 }  
  
 printf("Average Waiting Time: %.2f\n", totalWT / n);  
 printf("Average Turnaround Time: %.2f\n", totalTAT / n);  
 printf("Average Response Time: %.2f\n", totalRT / n);  
}  
  
int main() {  
 int n, choice;  
 printf("Enter number of processes: ");  
 scanf("%d", &n);  
  
 Process p[n], temp[n];  
  
 printf("Enter Arrival Time and Burst Time:\n");  
 for (int i = 0; i < n; i++) {  
 p[i].id = i + 1;   
 scanf("%d %d", &p[i].arrival, &p[i].burst);  
 p[i].remaining = p[i].burst;  
 p[i].waiting = p[i].turnaround = p[i].completion = p[i].response = p[i].started = 0;  
 temp[i] = p[i]; // Copy for reuse  
 }  
  
 while (1) {  
 printf("\nCPU Scheduling Algorithms:\n");  
 printf("1. First Come First Serve (FCFS)\n");  
 printf("2. Shortest Job First (Non-Preemptive)\n");  
 printf("3. Shortest Job First (Preemptive)\n");  
 printf("4. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 for (int i = 0; i < n; i++) p[i] = temp[i]; // Restore original processes  
  
 switch (choice) {  
 case 1:  
 fcfs(p, n);  
 displayResults(p, n, "First Come First Serve (FCFS)");  
 break;  
 case 2:  
 sjfNonPreemptive(p, n);  
 displayResults(p, n, "Shortest Job First (Non-Preemptive)");  
 break;  
 case 3:  
 sjfPreemptive(p, n);  
 displayResults(p, n, "Shortest Job First (Preemptive)");  
 break;  
 case 4:  
 return 0;  
 default:  
 printf("Invalid choice! Try again.\n");  
 }  
 }  
  
 return 0;  
}

Output

A screenshot of a computer program

AI-generated content may be incorrect.

# Program – 2

Write a C program to simulate the SJF CPU scheduling algorithm to find turnaround  
time and waiting time.

Code

#include <stdio.h>  
#include <limits.h>  
  
typedef struct {  
 int id, arrival, burst, remaining, waiting, turnaround, completion, response, started;  
} Process;  
  
void swap(Process \*a, Process \*b) {  
 Process temp = \*a;  
 \*a = \*b;  
 \*b = temp;  
}  
  
void sortByArrival(Process p[], int n) {  
 for (int i = 0; i < n - 1; i++) {  
 for (int j = 0; j < n - i - 1; j++) {  
 if (p[j].arrival > p[j + 1].arrival) {  
 swap(&p[j], &p[j + 1]);  
 }  
 }  
 }  
}  
  
void fcfs(Process p[], int n) {  
 sortByArrival(p, n);  
 int time = 0;  
  
 for (int i = 0; i < n; i++) {  
 if (time < p[i].arrival)  
 time = p[i].arrival;  
  
 p[i].response = time - p[i].arrival;  
 p[i].completion = time + p[i].burst;  
 p[i].turnaround = p[i].completion - p[i].arrival;  
 p[i].waiting = p[i].turnaround - p[i].burst;  
 time = p[i].completion;  
 }  
}  
  
void sjfNonPreemptive(Process p[], int n) {  
 int completed = 0, time = 0;  
  
 while (completed < n) {  
 int minIndex = -1, minBurst = INT\_MAX;  
  
 for (int i = 0; i < n; i++) {  
 if (p[i].arrival <= time && p[i].completion == 0) {  
 if (p[i].burst < minBurst || (p[i].burst == minBurst && p[i].arrival < p[minIndex].arrival)) {  
 minBurst = p[i].burst;  
 minIndex = i;  
 }  
 }  
 }  
  
 if (minIndex == -1) {  
 time++;  
 continue;  
 }  
  
 p[minIndex].response = time - p[minIndex].arrival;  
 time += p[minIndex].burst;  
 p[minIndex].completion = time;  
 p[minIndex].turnaround = p[minIndex].completion - p[minIndex].arrival;  
 p[minIndex].waiting = p[minIndex].turnaround - p[minIndex].burst;  
 completed++;  
 }  
}  
  
void sjfPreemptive(Process p[], int n) {  
 int completed = 0, time = 0, minIndex = -1, minBurst = INT\_MAX;  
  
 while (completed < n) {  
 minIndex = -1, minBurst = INT\_MAX;  
  
 for (int i = 0; i < n; i++) {  
 if (p[i].arrival <= time && p[i].remaining > 0) {  
 if (p[i].remaining < minBurst || (p[i].remaining == minBurst && p[i].arrival < p[minIndex].arrival)) {  
 minBurst = p[i].remaining;  
 minIndex = i;  
 }  
 }  
 }  
  
 if (minIndex == -1) {  
 time++;  
 continue;  
 }  
  
 if (p[minIndex].started == 0) {  
 p[minIndex].response = time - p[minIndex].arrival;  
 p[minIndex].started = 1;  
 }  
  
 p[minIndex].remaining--;  
 time++;  
  
 if (p[minIndex].remaining == 0) {  
 p[minIndex].completion = time;  
 p[minIndex].turnaround = p[minIndex].completion - p[minIndex].arrival;  
 p[minIndex].waiting = p[minIndex].turnaround - p[minIndex].burst;  
 completed++;  
 }  
 }  
}  
void displayResults(Process p[], int n, const char \*title) {  
 printf("\n--- %s ---\n", title);  
 printf("\nPID\tAT\tBT\tCT\tTAT\tWT\tRT\n");  
  
 float totalWT = 0, totalTAT = 0, totalRT = 0;  
 for (int i = 0; i < n; i++) {  
 printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].arrival, p[i].burst, p[i].completion, p[i].turnaround, p[i].waiting, p[i].response);  
 totalWT += p[i].waiting;  
 totalTAT += p[i].turnaround;  
 totalRT += p[i].response;  
 }  
 printf("Average Waiting Time: %.2f\n", totalWT / n);  
 printf("Average Turnaround Time: %.2f\n", totalTAT / n);  
 printf("Average Response Time: %.2f\n", totalRT / n);  
}  
  
int main() {  
 int n, choice;  
 printf("Enter number of processes: ");  
 scanf("%d", &n);  
 Process p[n], temp[n];  
 printf("Enter Arrival Time and Burst Time:\n");  
 for (int i = 0; i < n; i++) {  
 p[i].id = i + 1;   
 scanf("%d %d", &p[i].arrival, &p[i].burst);  
 p[i].remaining = p[i].burst;  
 p[i].waiting = p[i].turnaround = p[i].completion = p[i].response = p[i].started = 0;  
 temp[i] = p[i]; // Copy for reuse  
 }  
 while (1) {  
 printf("\nCPU Scheduling Algorithms:\n");  
 printf("1. First Come First Serve (FCFS)\n");  
 printf("2. Shortest Job First (Non-Preemptive)\n");  
 printf("3. Shortest Job First (Preemptive)\n");  
 printf("4. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 for (int i = 0; i < n; i++) p[i] = temp[i]; // Restore original processes  
  
 switch (choice) {  
 case 1:  
 fcfs(p, n);  
 displayResults(p, n, "First Come First Serve (FCFS)");  
 break;  
 case 2:  
 sjfNonPreemptive(p, n);  
 displayResults(p, n, "Shortest Job First (Non-Preemptive)");  
 break;  
 case 3:  
 sjfPreemptive(p, n);  
 displayResults(p, n, "Shortest Job First (Preemptive)");  
 break;  
 case 4:  
 return 0;  
 default:  
 printf("Invalid choice! Try again.\n");  
 }  
 }  
  
 return 0;  
}

Output

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

# Program – 3

Write a C program to simulate the Priority CPU scheduling algorithm to find turnaround  
time and waiting time.

Code

#include <limits.h>  
#include <stdio.h>  
  
struct Process {  
 int id, arrivalTime, burstTime, priority;  
 int completionTime, turnaroundTime, waitingTime, responseTime;  
};  
  
void swap(struct Process \*a, struct Process \*b) {  
 struct Process temp = \*a;  
 \*a = \*b;  
 \*b = temp;  
}  
  
void sortByArrival(struct Process proc[], int n) {  
 for (int i = 0; i < n - 1; i++) {  
 for (int j = i + 1; j < n; j++) {  
 if (proc[i].arrivalTime > proc[j].arrivalTime) {  
 swap(&proc[i], &proc[j]);  
 }  
 }  
 }  
}  
  
void prioritySchedulingNonPreemptive(struct Process proc[], int n) {  
 int time = 0, completed = 0;  
 float totalTAT = 0, totalWT = 0;  
  
 sortByArrival(proc, n);  
  
 while (completed < n) {  
 int idx = -1, highestPriority = INT\_MAX;  
  
 for (int i = 0; i < n; i++) {  
 if (proc[i].arrivalTime <= time && proc[i].completionTime == 0) {  
 if (proc[i].priority < highestPriority) {  
 highestPriority = proc[i].priority;  
 idx = i;  
 }  
 }  
 }  
  
 if (idx == -1) {  
 time++;  
 } else {  
 proc[idx].completionTime = time + proc[idx].burstTime;  
 proc[idx].turnaroundTime =  
 proc[idx].completionTime - proc[idx].arrivalTime;  
 proc[idx].waitingTime = proc[idx].turnaroundTime - proc[idx].burstTime;  
 proc[idx].responseTime = proc[idx].waitingTime;  
  
 totalTAT += proc[idx].turnaroundTime;  
 totalWT += proc[idx].waitingTime;  
 time = proc[idx].completionTime;  
 completed++;  
 }  
 }  
  
 printf("\nNon-Preemptive Priority Scheduling:\n");  
 printf("PID\tAT\tBT\tP\tCT\tTAT\tWT\tRT\n");  
 for (int i = 0; i < n; i++) {  
 printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n", proc[i].id, proc[i].arrivalTime,  
 proc[i].burstTime, proc[i].priority, proc[i].completionTime,  
 proc[i].turnaroundTime, proc[i].waitingTime, proc[i].responseTime);  
 }  
  
 printf("\nAverage Turnaround Time: %.2f", totalTAT / n);  
 printf("\nAverage Waiting Time: %.2f\n", totalWT / n);  
}  
  
void prioritySchedulingPreemptive(struct Process proc[], int n) {  
 int remainingTime[n], completed = 0, time = 0, shortest = -1;  
 float totalTAT = 0, totalWT = 0;  
  
 for (int i = 0; i < n; i++) {  
 remainingTime[i] = proc[i].burstTime;  
 }  
  
 while (completed < n) {  
 int highestPriority = INT\_MAX;  
 shortest = -1;  
  
 for (int i = 0; i < n; i++) {  
 if (proc[i].arrivalTime <= time && remainingTime[i] > 0) {  
 if (proc[i].priority < highestPriority) {  
 highestPriority = proc[i].priority;  
 shortest = i;  
 }  
 }  
 }  
  
 if (shortest == -1) {  
 time++;  
 } else {  
 if (remainingTime[shortest] == proc[shortest].burstTime) {  
 proc[shortest].responseTime = time - proc[shortest].arrivalTime;  
 }  
  
 remainingTime[shortest]--;  
 time++;  
  
 if (remainingTime[shortest] == 0) {  
 completed++;  
 proc[shortest].completionTime = time;  
 proc[shortest].turnaroundTime =  
 proc[shortest].completionTime - proc[shortest].arrivalTime;  
 proc[shortest].waitingTime =  
 proc[shortest].turnaroundTime - proc[shortest].burstTime;  
  
 totalTAT += proc[shortest].turnaroundTime;  
 totalWT += proc[shortest].waitingTime;  
 }  
 }  
 }  
  
 printf("\nPreemptive Priority Scheduling:\n");  
 printf("PID\tAT\tBT\tP\tCT\tTAT\tWT\tRT\n");  
 for (int i = 0; i < n; i++) {  
 printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n", proc[i].id, proc[i].arrivalTime,  
 proc[i].burstTime, proc[i].priority, proc[i].completionTime,  
 proc[i].turnaroundTime, proc[i].waitingTime, proc[i].responseTime);  
 }  
  
 printf("\nAverage Turnaround Time: %.2f", totalTAT / n);  
 printf("\nAverage Waiting Time: %.2f\n", totalWT / n);  
}  
  
int main() {  
 int n, choice;  
  
 printf("Enter number of processes: ");  
 scanf("%d", &n);  
  
 struct Process proc[n];  
  
 printf("Enter Arrival Time, Burst Time, and Priority for each process:\n");  
 for (int i = 0; i < n; i++) {  
 proc[i].id = i + 1;  
 printf("Process %d - Arrival Time: ", i + 1);  
 scanf("%d", &proc[i].arrivalTime);  
 printf("Process %d - Burst Time: ", i + 1);  
 scanf("%d", &proc[i].burstTime);  
 printf("Process %d - Priority: ", i + 1);  
 scanf("%d", &proc[i].priority);  
 }  
  
 printf("\nChoose Scheduling Type:\n1. Non-Preemptive\n2. Preemptive\nEnter "  
 "choice: ");  
 scanf("%d", &choice);  
  
 if (choice == 1) {  
 prioritySchedulingNonPreemptive(proc, n);  
 } else if (choice == 2) {  
 prioritySchedulingPreemptive(proc, n);  
 } else {  
 printf("Invalid choice!\n");  
 }  
  
 return 0;  
}

Output

A screenshot of a computer

AI-generated content may be incorrect.

# Program – 4

Write a C program to simulate the Round Robin CPU scheduling algorithm to find  
turnaround time and waiting time.

Code

#include <stdio.h>  
  
#define MAX 100  
  
void roundRobin(int n, int at[], int bt[], int quant) {  
 int ct[n], tat[n], wt[n], rem\_bt[n];  
 int queue[MAX], front = 0, rear = 0;  
 int time = 0, completed = 0, visited[n];  
  
 for (int i = 0; i < n; i++) {  
 rem\_bt[i] = bt[i];  
 visited[i] = 0;  
 }  
  
 queue[rear++] = 0;  
 visited[0] = 1;  
  
 while (completed < n) {  
 int index = queue[front++];  
   
 if (rem\_bt[index] > quant) {  
 time += quant;  
 rem\_bt[index] -= quant;  
 } else {  
 time += rem\_bt[index];  
 rem\_bt[index] = 0;  
 ct[index] = time;  
 completed++;  
 }  
  
 for (int i = 0; i < n; i++) {  
 if (at[i] <= time && rem\_bt[i] > 0 && !visited[i]) {  
 queue[rear++] = i;  
 visited[i] = 1;  
 }  
 }  
  
 if (rem\_bt[index] > 0) {  
 queue[rear++] = index;  
 }  
  
 if (front == rear) {  
 for (int i = 0; i < n; i++) {  
 if (rem\_bt[i] > 0) {  
 queue[rear++] = i;  
 visited[i] = 1;  
 break;  
 }  
 }  
 }  
 }  
  
 float total\_tat = 0, total\_wt = 0;  
 printf("P#\tAT\tBT\tCT\tTAT\tWT\n");  
 for (int i = 0; i < n; i++) {  
 tat[i] = ct[i] - at[i];  
 wt[i] = tat[i] - bt[i];  
 total\_tat += tat[i];  
 total\_wt += wt[i];  
 printf("%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], ct[i], tat[i], wt[i]);  
 }  
  
 printf("Average TAT: %.2f\n", total\_tat / n);  
 printf("Average WT: %.2f\n", total\_wt / n);  
}  
  
int main() {  
 int n, quant;  
 printf("Enter number of processes: ");  
 scanf("%d", &n);  
  
 int at[n], bt[n];  
 for (int i = 0; i < n; i++) {  
 printf("Enter AT and BT for process %d: ", i + 1);  
 scanf("%d %d", &at[i], &bt[i]);  
 }  
  
 printf("Enter time quantum: ");  
 scanf("%d", &quant);  
  
 roundRobin(n, at, bt, quant);  
 return 0;  
}

Output

A black screen with white text

AI-generated content may be incorrect.

# Program – 5

Write a C program to simulate Rate Monotonic Real-Time CPU Scheduling algorithms:

Code

#include <stdio.h>  
#include<math.h>  
  
#define MAX\_PROCESSES 10  
  
typedef struct {  
 int id;  
 int burst\_time;  
 int period;  
 int remaining\_time;  
 int next\_deadline;  
} Process;  
  
void sort\_by\_period(Process processes[], int n) {  
 for (int i = 0; i < n - 1; i++) {  
 for (int j = 0; j < n - i - 1; j++) {  
 if (processes[j].period > processes[j + 1].period) {  
 Process temp = processes[j];  
 processes[j] = processes[j + 1];  
 processes[j + 1] = temp;  
 }  
 }  
 }  
}  
  
int gcd(int a, int b) {  
 return b == 0 ? a : gcd(b, a % b);  
}  
  
int lcm(int a, int b) {  
 return (a \* b) / gcd(a, b);  
}  
  
int calculate\_lcm(Process processes[], int n) {  
 int result = processes[0].period;  
 for (int i = 1; i < n; i++) {  
 result = lcm(result, processes[i].period);  
 }  
 return result;  
}  
  
double utilization\_factor(Process processes[], int n) {  
 double sum = 0;  
 for (int i = 0; i < n; i++) {  
 sum += (double)processes[i].burst\_time / processes[i].period;  
 }  
 return sum;  
}  
  
double rms\_threshold(int n) {  
 return n \* (pow(2.0, 1.0 / n) - 1);  
}  
  
void rate\_monotonic\_scheduling(Process processes[], int n) {  
 int lcm\_period = calculate\_lcm(processes, n);  
 printf("LCM=%d\n\n", lcm\_period);  
 printf("Rate Monotone Scheduling:\n");  
 printf("PID Burst Period\n");  
 for (int i = 0; i < n; i++) {  
 printf("%d %d %d\n", processes[i].id, processes[i].burst\_time, processes[i].period);  
 }  
 double utilization = utilization\_factor(processes, n);  
 double threshold = rms\_threshold(n);  
 printf("\n%.6f <= %.6f => %s\n", utilization, threshold, (utilization <= threshold) ? "true" : "false");  
  
 if (utilization > threshold) {  
 printf("\nSystem may not be schedulable!\n");  
 return;  
 }  
 int timeline = 0, executed = 0;  
 while (timeline < lcm\_period) {  
 int selected = -1;  
 for (int i = 0; i < n; i++) {  
 if (timeline % processes[i].period == 0) {  
 processes[i].remaining\_time = processes[i].burst\_time;  
 }  
 if (processes[i].remaining\_time > 0) {  
 selected = i;  
 break;  
 }  
 }  
 if (selected != -1) {  
 printf("Time %d: Process %d is running\n", timeline, processes[selected].id);  
 processes[selected].remaining\_time--;  
 executed++;  
 } else {  
 printf("Time %d: CPU is idle\n", timeline);  
 }  
 timeline++;  
 }  
}  
int main() {  
 int n;  
 Process processes[MAX\_PROCESSES];  
  
 printf("Enter the number of processes: ");  
 scanf("%d", &n);  
  
 printf("Enter the CPU burst times:\n");  
 for (int i = 0; i < n; i++) {  
 processes[i].id = i + 1;  
 scanf("%d", &processes[i].burst\_time);  
 processes[i].remaining\_time = processes[i].burst\_time;  
 }  
 printf("Enter the time periods:\n");  
 for (int i = 0; i < n; i++) {  
 scanf("%d", &processes[i].period);  
 }  
 sort\_by\_period(processes, n);  
 rate\_monotonic\_scheduling(processes, n);  
  
 return 0;  
}

Output

A screenshot of a computer program

AI-generated content may be incorrect.

# Program – 6

Write a C program to simulate: Producer-Consumer problem using semaphores.

Code

#include <stdio.h>  
  
int mutex = 1, full = 0, empty = 3, x = 0;  
  
void wait(int \*s) {  
 --(\*s);  
}  
void signal(int \*s) {  
 ++(\*s);  
}  
void producer() {  
 wait(&empty);  
 wait(&mutex);  
 x++;  
 printf("The item produced is %d\n", x);  
 signal(&mutex);  
 signal(&full);  
}  
void consumer() {  
 wait(&full);  
 wait(&mutex);  
 printf("Consumed item %d\n", x);  
 x--;  
 signal(&mutex);  
 signal(&empty);  
}  
int main() {  
 int choice;  
 do {  
 printf("\n1. Produce\n2. Consume\n3. Exit\nEnter choice: ");  
 scanf("%d", &choice);  
 switch (choice) {  
 case 1:  
 if ((mutex == 1) && (empty != 0)) {  
 producer();  
 } else {  
 printf("The buffer is full\n");  
 }  
 break;  
 case 2:  
 if ((mutex == 1) && (full != 0)) {  
 consumer();  
 } else {  
 printf("The buffer is empty\n");  
 }  
 break;  
 case 3:  
 printf("Exiting.\n");  
 break;  
 default:  
 printf("Invalid choice.\n");  
 }  
 } while (choice != 3);  
 return 0;  
}

Output

A screen shot of a computer

AI-generated content may be incorrect.

# Program – 7

Write a C program to simulate: Dining-Philosopher’s problem using semaphores.

Code

#include <pthread.h>  
#include <semaphore.h>  
#include <stdio.h>  
#include <unistd.h>  
  
#define N 5  
#define THINKING 2  
#define HUNGRY 1  
#define EATING 0  
#define LEFT (phnum + 4) % N  
#define RIGHT (phnum + 1) % N  
  
int state[N];  
int phil[N] = { 0, 1, 2, 3, 4 };  
  
sem\_t mutex;  
sem\_t S[N];  
  
void\* philosopher(void\* num);  
void take\_fork(int phnum);  
void put\_fork(int phnum);  
void test(int phnum);  
  
int main() {  
 int i;  
 pthread\_t thread\_id[N];  
  
 // initialize the semaphores  
 sem\_init(&mutex, 0, 1);  
  
 for (i = 0; i < N; i++)  
 sem\_init(&S[i], 0, 0);  
  
 for (i = 0; i < N; i++) {  
 // create philosopher processes  
 pthread\_create(&thread\_id[i], NULL, philosopher, &phil[i]);  
 printf("Philosopher %d is thinking\n", i + 1);  
 }  
  
 for (i = 0; i < N; i++)  
 pthread\_join(thread\_id[i], NULL);  
  
 return 0;  
}  
  
void test(int phnum) {  
 if (state[phnum] == HUNGRY  
 && state[LEFT] != EATING  
 && state[RIGHT] != EATING) {  
 // state that eating  
 state[phnum] = EATING;  
 sleep(2);  
 printf("Philosopher %d takes fork %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);  
 printf("Philosopher %d is Eating\n", phnum + 1);  
 sem\_post(&S[phnum]);  
 }  
}  
  
// take up chopsticks  
void take\_fork(int phnum) {  
 sem\_wait(&mutex);  
 // state that hungry  
 state[phnum] = HUNGRY;  
 printf("Philosopher %d is Hungry\n", phnum + 1);  
  
 // eat if neighbours are not eating  
 test(phnum);  
 sem\_post(&mutex);  
 // if unable to eat wait to be signalled  
 sem\_wait(&S[phnum]);  
 sleep(1);  
}  
  
// put down chopsticks  
void put\_fork(int phnum) {  
 sem\_wait(&mutex);  
  
 // state that thinking  
 state[phnum] = THINKING;  
  
 printf("Philosopher %d putting fork %d and %d down\n", phnum + 1, LEFT + 1, phnum + 1);  
 printf("Philosopher %d is thinking\n", phnum + 1);  
  
 test(LEFT);  
 test(RIGHT);  
  
 sem\_post(&mutex);  
}  
  
void\* philosopher(void\* num) {  
 int\* i = (int\*)num;  
 while (1) {  
 sleep(1);  
 take\_fork(\*i);  
 sleep(0);  
 put\_fork(\*i);  
 }  
}

Output

A screenshot of a computer screen

AI-generated content may be incorrect.

# Program – 8

Write a C program to simulate the Worst-fit contiguous memory allocation techniques.

Code

//Worst-fit  
  
#include <stdio.h>  
  
struct Block {  
 int block\_no;  
 int block\_size;  
 int is\_free;  
};  
  
struct File {  
 int file\_no;  
 int file\_size;  
};  
  
void worstFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {  
 printf("\nMemory Management Scheme - Worst Fit\n");  
 printf("File\_no\tFile\_size\tBlock\_no\tBlock\_size\tFragment\n");  
  
 for (int i = 0; i < n\_files; i++) {  
 int worst\_fit\_block = -1;  
 int max\_fragment = -1;  
  
 for (int j = 0; j < n\_blocks; j++) {  
 if (blocks[j].is\_free && blocks[j].block\_size >= files[i].file\_size) {  
 int fragment = blocks[j].block\_size - files[i].file\_size;  
 if (fragment > max\_fragment) {  
 max\_fragment = fragment;  
 worst\_fit\_block = j;  
 }  
 }  
 }  
  
 if (worst\_fit\_block != -1) {  
 blocks[worst\_fit\_block].is\_free = 0;  
 printf("%d\t%d\t\t%d\t\t%d\t\t%d\n",  
 files[i].file\_no,  
 files[i].file\_size,  
 blocks[worst\_fit\_block].block\_no,  
 blocks[worst\_fit\_block].block\_size,  
 max\_fragment);  
 } else {  
 printf("%d\t%d\t\tNot Allocated\n", files[i].file\_no, files[i].file\_size);  
 }  
 }  
}  
  
int main() {  
 int n\_blocks, n\_files;  
  
 printf("Enter the number of blocks: ");  
 scanf("%d", &n\_blocks);  
  
 struct Block blocks[n\_blocks];  
  
 for (int i = 0; i < n\_blocks; i++) {  
 blocks[i].block\_no = i + 1;  
 printf("Enter the size of block %d: ", i + 1);  
 scanf("%d", &blocks[i].block\_size);  
 blocks[i].is\_free = 1;  
 }  
  
 printf("Enter the number of files: ");  
 scanf("%d", &n\_files);  
  
 struct File files[n\_files];  
  
 for (int i = 0; i < n\_files; i++) {  
 files[i].file\_no = i + 1;  
 printf("Enter the size of file %d: ", i + 1);  
 scanf("%d", &files[i].file\_size);  
 }  
  
 worstFit(blocks, n\_blocks, files, n\_files);  
  
 return 0;  
}

Output

A screenshot of a computer screen

AI-generated content may be incorrect.

# Program – 9

Write a C program to simulate the Best-fit contiguous memory allocation techniques.

Code

#include <stdio.h>  
  
struct Block {  
 int block\_no;  
 int block\_size;  
 int is\_free;  
};  
  
struct File {  
 int file\_no;  
 int file\_size;  
};  
  
void bestFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {  
 printf("\nMemory Management Scheme - Best Fit\n");  
 printf("File\_no\tFile\_size\tBlock\_no\tBlock\_size\tFragment\n");  
  
 for (int i = 0; i < n\_files; i++) {  
 int best\_fit\_block = -1;  
 int min\_fragment = 10000; // Large initial value  
  
 for (int j = 0; j < n\_blocks; j++) {  
 if (blocks[j].is\_free && blocks[j].block\_size >= files[i].file\_size) {  
 int fragment = blocks[j].block\_size - files[i].file\_size;  
 if (fragment < min\_fragment) {  
 min\_fragment = fragment;  
 best\_fit\_block = j;  
 }  
 }  
 }  
  
 if (best\_fit\_block != -1) {  
 blocks[best\_fit\_block].is\_free = 0;  
 printf("%d\t%d\t\t%d\t\t%d\t\t%d\n",  
 files[i].file\_no,  
 files[i].file\_size,  
 blocks[best\_fit\_block].block\_no,  
 blocks[best\_fit\_block].block\_size,  
 min\_fragment);  
 } else {  
 printf("%d\t%d\t\tNot Allocated\n", files[i].file\_no, files[i].file\_size);  
 }  
 }  
}  
  
int main() {  
 int n\_blocks, n\_files;  
  
 printf("Enter the number of blocks: ");  
 scanf("%d", &n\_blocks);  
  
 struct Block blocks[n\_blocks];  
  
 for (int i = 0; i < n\_blocks; i++) {  
 blocks[i].block\_no = i + 1;  
 printf("Enter the size of block %d: ", i + 1);  
 scanf("%d", &blocks[i].block\_size);  
 blocks[i].is\_free = 1;  
 }  
  
 printf("Enter the number of files: ");  
 scanf("%d", &n\_files);  
  
 struct File files[n\_files];  
  
 for (int i = 0; i < n\_files; i++) {  
 files[i].file\_no = i + 1;  
 printf("Enter the size of file %d: ", i + 1);  
 scanf("%d", &files[i].file\_size);  
 }  
  
 bestFit(blocks, n\_blocks, files, n\_files);  
  
 return 0;  
}

Output

A screenshot of a computer

AI-generated content may be incorrect.

# Program – 10

Write a C program to simulate the First-fit contiguous memory allocation techniques.

Code

// First-fit  
  
#include <stdio.h>  
  
struct Block {  
 int block\_no;  
 int block\_size;  
 int is\_free;  
};  
  
struct File {  
 int file\_no;  
 int file\_size;  
};  
  
void firstFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {  
 printf("\nMemory Management Scheme - First Fit\n");  
 printf("File\_no\tFile\_size\tBlock\_no\tBlock\_size\tFragment\n");  
  
 for (int i = 0; i < n\_files; i++) {  
 int allocated = 0;  
  
 for (int j = 0; j < n\_blocks; j++) {  
 if (blocks[j].is\_free && blocks[j].block\_size >= files[i].file\_size) {  
 int fragment = blocks[j].block\_size - files[i].file\_size;  
 blocks[j].is\_free = 0;  
  
 printf("%d\t%d\t\t%d\t\t%d\t\t%d\n",  
 files[i].file\_no,  
 files[i].file\_size,  
 blocks[j].block\_no,  
 blocks[j].block\_size,  
 fragment);  
  
 allocated = 1;  
 break;  
 }  
 }  
  
 if (!allocated) {  
 printf("%d\t%d\t\tNot Allocated\n", files[i].file\_no, files[i].file\_size);  
 }  
 }  
}  
  
int main() {  
 int n\_blocks, n\_files;  
  
 printf("Enter the number of blocks: ");  
 scanf("%d", &n\_blocks);  
  
 struct Block blocks[n\_blocks];  
  
 for (int i = 0; i < n\_blocks; i++) {  
 blocks[i].block\_no = i + 1;  
 printf("Enter the size of block %d: ", i + 1);  
 scanf("%d", &blocks[i].block\_size);  
 blocks[i].is\_free = 1;  
 }  
  
 printf("Enter the number of files: ");  
 scanf("%d", &n\_files);  
  
 struct File files[n\_files];  
  
 for (int i = 0; i < n\_files; i++) {  
 files[i].file\_no = i + 1;  
 printf("Enter the size of file %d: ", i + 1);  
 scanf("%d", &files[i].file\_size);  
 }  
  
 firstFit(blocks, n\_blocks, files, n\_files);  
  
 return 0;  
}

Output

A screen shot of a computer

AI-generated content may be incorrect.

# Program – 11

Write a C program to simulate page replacement algorithms. FIFO

Code

#include <stdio.h>  
  
int main() {  
 int n, frames, i, j, k, found, index = 0, page\_faults = 0, hits = 0;  
 char pages[100];  
   
 printf("Enter the size of the pages:\n");  
 scanf("%d", &n);  
 printf("Enter the page strings:\n");  
 scanf("%s", pages);  
 printf("Enter the no of page frames:\n");  
 scanf("%d", &frames);  
  
 int mem[frames];  
 for (i = 0; i < frames; i++) mem[i] = -1;  
  
 for (i = 0; i < n; i++) {  
 found = 0;  
 for (j = 0; j < frames; j++) {  
 if (mem[j] == pages[i] - '0') {  
 hits++;  
 found = 1;  
 break;  
 }  
 }  
 if (!found) {  
 mem[index] = pages[i] - '0';  
 index = (index + 1) % frames;  
 page\_faults++;  
 }  
 }  
  
 printf("FIFO Page Faults: %d, Page Hits: %d\n", page\_faults, hits);  
 return 0;  
}

Output

A screenshot of a computer screen

AI-generated content may be incorrect.

# Program – 12

Write a C program to simulate page replacement algorithms. LRU

Code

#include <stdio.h>  
  
int main() {  
 int n, frames, i, j, k, page\_faults = 0, hits = 0;  
 char pages[100];  
 int mem[10], used[10];  
  
 printf("Enter the size of the pages:\n");  
 scanf("%d", &n);  
 printf("Enter the page strings:\n");  
 scanf("%s", pages);  
 printf("Enter the no of page frames:\n");  
 scanf("%d", &frames);  
 for (i = 0; i < frames; i++) {  
 mem[i] = -1;  
 used[i] = -1;  
 }  
 for (i = 0; i < n; i++) {  
 int page = pages[i] - '0';  
 int found = 0;  
  
 for (j = 0; j < frames; j++) {  
 if (mem[j] == page) {  
 hits++;  
 used[j] = i;  
 found = 1;  
 break;  
 }  
 }  
 if (!found) {  
 int lru = 0;  
 for (j = 1; j < frames; j++) {  
 if (used[j] < used[lru]) lru = j;  
 }  
 mem[lru] = page;  
 used[lru] = i;  
 page\_faults++;  
 }  
 }  
 printf("LU Page Faults: %d, Page Hits: %d\n", page\_faults, hits);  
 return 0;  
}

Output

A screenshot of a computer screen

AI-generated content may be incorrect.

# Program – 13

Write a C program to simulate page replacement algorithms. Optimal

Code

#include <stdio.h>  
  
int main() {  
 int n, frames, i, j, k, page\_faults = 0, hits = 0;  
  
 printf("Enter the size of the pages:\n");  
 scanf("%d", &n);  
  
 char pages[n + 1];  
 printf("Enter the page strings:\n");  
 scanf("%s", pages);  
  
 printf("Enter the no of page frames:\n");  
 scanf("%d", &frames);  
  
 int mem[frames], next\_use[frames];  
 for (i = 0; i < frames; i++) {  
 mem[i] = -1;  
 }  
  
 for (i = 0; i < n; i++) {  
 int page = pages[i] - '0';  
 int found = 0;  
  
 for (j = 0; j < frames; j++) {  
 if (mem[j] == page) {  
 hits++;  
 found = 1;  
 break;  
 }  
 }  
  
 if (!found) {  
 if (page\_faults < frames) {  
 mem[page\_faults++] = page;  
 } else {  
 for (j = 0; j < frames; j++) {  
 next\_use[j] = -1;  
 for (k = i + 1; k < n; k++) {  
 if (mem[j] == pages[k] - '0') {  
 next\_use[j] = k;  
 break;  
 }  
 }  
 }  
  
 int farthest = 0;  
 for (j = 1; j < frames; j++) {  
 if (next\_use[j] > next\_use[farthest]) {  
 farthest = j;  
 }  
 }  
  
 mem[farthest] = page;  
 page\_faults++;  
 }  
 }  
 }  
  
 printf("Optimal Page Faults: %d, Page Hits: %d\n", page\_faults, hits);  
 return 0;  
}

Output

A screenshot of a computer screen

AI-generated content may be incorrect.

# Program – 14

Write a C program to simulate: Bankers’ algorithm for the purpose of deadlock  
avoidance.

Code

#include <stdio.h>  
#include <stdbool.h>  
  
int main() {  
 int n, m, i, j, k;  
 printf("Enter number of processes: ");  
 scanf("%d", &n);  
 printf("Enter number of resources: ");  
 scanf("%d", &m);  
  
 int alloc[n][m], max[n][m], avail[m];  
 int need[n][m];  
  
 printf("Enter allocation matrix (%d x %d):\n", n, m);  
 for (i = 0; i < n; i++) {  
 printf("Allocation for process %d: ", i);  
 for (j = 0; j < m; j++)  
 scanf("%d", &alloc[i][j]);  
 }  
  
 printf("Enter max matrix (%d x %d):\n", n, m);  
 for (i = 0; i < n; i++) {  
 printf("Max for process %d: ", i);  
 for (j = 0; j < m; j++)  
 scanf("%d", &max[i][j]);  
 }  
  
 printf("Enter available resources (%d values): ", m);  
 for (i = 0; i < m; i++)  
 scanf("%d", &avail[i]);  
  
 for (i = 0; i < n; i++)  
 for (j = 0; j < m; j++)  
 need[i][j] = max[i][j] - alloc[i][j];  
  
 bool finish[n];  
 int safeSeq[n];  
 int count = 0;  
  
 for (i = 0; i < n; i++)  
 finish[i] = false;  
  
 while (count < n) {  
 bool found = false;  
 for (i = 0; i < n; i++) {  
 if (!finish[i]) {  
 for (j = 0; j < m; j++)  
 if (need[i][j] > avail[j])  
 break;  
  
 if (j == m) {  
 for (k = 0; k < m; k++)  
 avail[k] += alloc[i][k];  
  
 safeSeq[count++] = i;  
 finish[i] = true;  
 found = true;  
 }  
 }  
 }  
  
 if (!found) {  
 printf("System is not in safe state.\n");  
 return 1;  
 }  
 }  
  
 printf("System is in safe state.\n");  
 printf("Safe sequence is: ");  
 for (i = 0; i < n; i++) {  
 printf("P%d", safeSeq[i]);  
 if (i != n - 1)  
 printf(" -> ");  
 }  
 printf("\n");  
  
 return 0;  
}

Output

A screenshot of a computer

AI-generated content may be incorrect.

# Program – 15

Write a C program to simulate: Bankers’ algorithm for the purpose of Deadlock  
Detection

Code

#include <stdio.h>  
#include <stdbool.h>  
  
int main() {  
 int n, m, i, j, k;  
  
 printf("Enter number of processes and resources:\n");  
 scanf("%d %d", &n, &m);  
  
 int allocation[n][m], request[n][m], available[m];  
 int work[m];  
 bool finish[n];  
  
 printf("Enter allocation matrix:\n");  
 for (i = 0; i < n; i++)  
 for (j = 0; j < m; j++)  
 scanf("%d", &allocation[i][j]);  
  
 printf("Enter request matrix:\n");  
 for (i = 0; i < n; i++)  
 for (j = 0; j < m; j++)  
 scanf("%d", &request[i][j]);  
  
 printf("Enter available matrix:\n");  
 for (i = 0; i < m; i++) {  
 scanf("%d", &available[i]);  
 work[i] = available[i];  
 }  
  
 for (i = 0; i < n; i++) {  
 bool zero\_allocation = true;  
 for (j = 0; j < m; j++) {  
 if (allocation[i][j] != 0) {  
 zero\_allocation = false;  
 break;  
 }  
 }  
 finish[i] = zero\_allocation;  
 }  
  
 bool found\_process;  
 do {  
 found\_process = false;  
 for (i = 0; i < n; i++) {  
 if (!finish[i]) {  
 bool can\_allocate = true;  
 for (j = 0; j < m; j++) {  
 if (request[i][j] > work[j]) {  
 can\_allocate = false;  
 break;  
 }  
 }  
 if (can\_allocate) {  
 for (k = 0; k < m; k++)  
 work[k] += allocation[i][k];  
 finish[i] = true;  
 printf("Process %d can finish.\n", i);  
 found\_process = true;  
 }  
 }  
 }  
 } while (found\_process);  
  
 bool deadlock = false;  
 for (i = 0; i < n; i++) {  
 if (!finish[i]) {  
 deadlock = true;  
 break;  
 }  
 }  
  
 if (deadlock)  
 printf("System is in a deadlock state.\n");  
 else  
 printf("System is not in a deadlock state.\n");  
  
 return 0;  
}

Output

A screenshot of a computer program

AI-generated content may be incorrect.