

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT
on
OPERATING SYSTEMS

Submitted by

VEDIKA S S (1WA23CS037)

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 Vedika S S (1WA23CS037), 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.

Dr Seema Patil
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1.	<p>Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.</p> <ul style="list-style-type: none"> → FCFS → SJF (pre-emptive & Non-preemptive) 	1-6
2.	<p>Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.</p> <ul style="list-style-type: none"> → Priority (pre-emptive & Non-pre-emptive) 	7-11
3.	<p>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 RR and FCFS scheduling for the processes in each queue.</p> <p>Write a C program to simulate Real-Time CPU Scheduling algorithms:</p> <ul style="list-style-type: none"> a) Rate- Monotonic b) Earliest-deadline First 	12-24
4.	<p>Write a C program to simulate producer-consumer problem using semaphores</p> <p>Write a C program to simulate the concept of Dining Philosophers problem.</p>	25-33
5.	<p>Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.</p> <p>Write C program to stimulate deadlock detection</p>	34-40
6.	<p>Write a C program to simulate the following contiguous memory allocation techniques</p> <ul style="list-style-type: none"> a) Worst-fit b) Best-fit c) First-fit <p>Write a C program to simulate page replacement algorithms</p> <ul style="list-style-type: none"> a) FIFO 	41-48
7.	<p>Write a C program to simulate page replacement algorithms</p> <ul style="list-style-type: none"> b) LRU c) Optimal 	49-55

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.

```
int main() {Program -1
```

Question: 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)

Code:

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

struct process {
    int id, AT, BT, CT, TAT, WT, RT, remaining_BT;
    int completed;
};

void sort_by_AT(struct process p[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (p[i].AT > p[j].AT) {
                struct process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
}

void calculate_FCFS(struct process p[], int n) {
    sort_by_AT(p, n);
    int currentTime = 0;

    for (int i = 0; i < n; i++) {
        if (currentTime < p[i].AT)
            currentTime = p[i].AT;
```

```

p[i].RT = currentTime - p[i].AT;
p[i].CT = currentTime + p[i].BT;
currentTime = p[i].CT;
p[i].TAT = p[i].CT - p[i].AT;
p[i].WT = p[i].TAT - p[i].BT;
}

}

void calculate_SJF_NonPreemptive(struct process p[], int n) {
    int completed = 0, currentTime = 0;
    while (completed < n) {
        int shortest = -1, minBT = 10000;
        for (int i = 0; i < n; i++) {
            if (!p[i].completed && p[i].AT <= currentTime && p[i].BT < minBT) {
                minBT = p[i].BT;
                shortest = i;
            }
        }
        if (shortest == -1) {
            currentTime++;
        } else {
            p[shortest].RT = currentTime - p[shortest].AT;
            p[shortest].CT = currentTime + p[shortest].BT;
            currentTime = p[shortest].CT;
            p[shortest].TAT = p[shortest].CT - p[shortest].AT;
            p[shortest].WT = p[shortest].TAT - p[shortest].BT;
            p[shortest].completed = 1;
            completed++;
        }
    }
}

void calculate_SJF_Preemptive(struct process p[], int n) {

```

```

int completed = 0, currentTime = 0;
for (int i = 0; i < n; i++) {
    p[i].remaining_BT = p[i].BT;
}
while (completed < n) {
    int shortest = -1, minBT = 10000;
    for (int i = 0; i < n; i++) {
        if (!p[i].completed && p[i].AT <= currentTime && p[i].remaining_BT < minBT) {
            minBT = p[i].remaining_BT;
            shortest = i;
        }
    }
    if (shortest == -1) {
        currentTime++;
    } else {
        if (p[shortest].remaining_BT == p[shortest].BT)
            p[shortest].RT = currentTime - p[shortest].AT;

        p[shortest].remaining_BT--;
        currentTime++;

        if (p[shortest].remaining_BT == 0) {
            p[shortest].CT = currentTime;
            p[shortest].TAT = p[shortest].CT - p[shortest].AT;
            p[shortest].WT = p[shortest].TAT - p[shortest].BT;
            p[shortest].completed = 1;
            completed++;
        }
    }
}
}

```

```

void display(struct process p[], int n) {
    printf("\nProcess\tAT\tBT\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\n", p[i].id, p[i].AT, p[i].BT, p[i].CT, p[i].TAT,
p[i].WT, p[i].RT);
    }
}

int main() {
    int n, choice;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct process p[n];
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter Arrival Time (AT) for process %d: ", i + 1);
        scanf("%d", &p[i].AT);
        printf("Enter Burst Time (BT) for process %d: ", i + 1);
        scanf("%d", &p[i].BT);
        p[i].completed = 0;
    }

    while (1) {
        printf("\nMenu:\n");
        printf("1. First Come First Serve (FCFS)\n");
        printf("2. Shortest Job First (SJF) - Non Preemptive\n");
        printf("3. Shortest Job First (SJF) - Preemptive\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
    }
}

```

```
switch (choice) {  
    case 1:  
        calculate_FCFS(p, n);  
        display(p, n);  
        break;  
    case 2:  
        calculate_SJF_NonPreemptive(p, n);  
        display(p, n);  
        break;  
    case 3:  
        calculate_SJF_Preemptive(p, n);  
        display(p, n);  
        break;  
    case 4:  
        exit(0);  
    default:  
        printf("Invalid choice. Try again.\n");  
}  
}  
  
return 0;  
}
```

OUTPUT:

```
C:\Users\Admin\Desktop\OS LAB\FCFS.exe"
Menu:
1. First Come First Serve (FCFS)
2. Shortest Job First (SJF) - Non Preemptive
3. Shortest Job First (SJF) - Preemptive
4. Exit
Enter choice: 1

Process AT BT CT TAT WT RT
1 0 4 4 4 0 0
2 2 5 9 7 2 2
3 4 6 15 11 5 5

Menu:
1. First Come First Serve (FCFS)
2. Shortest Job First (SJF) - Non Preemptive
3. Shortest Job First (SJF) - Preemptive
4. Exit
Enter choice: 2

Process AT BT CT TAT WT RT
1 0 4 4 4 0 0
2 2 5 9 7 2 2
3 4 6 15 11 5 5

Menu:
1. First Come First Serve (FCFS)
2. Shortest Job First (SJF) - Non Preemptive
3. Shortest Job First (SJF) - Preemptive
4. Exit
Enter choice: 3
```

6/3/23

papergrid

Write a C program to simulate the following non-preemptive CPU scheduling algorithm to find turnaround time and waiting time.

i) FCFS

ii) SJF (Preemptive & Non-preemptive)

```
#include <stdio.h>
#include <math.h>
#define MAX 10
```

struct process {
 int ID, AT, BT, CT, TAT, WT, remaining_BT;
};

void sort_by_AT(process p[], int n)
{
 for (int i = 0; i < n - 1; i++)
 for (int j = i + 1; j < n; j++)
 if (p[i].AT > p[j].AT)
 swap_process(p[i], p[j]);
}

void calculate_FCFS(struct process p[], int n)
{
 int completed = 0, current_time = 0;
 while (completed < n)
 {
 int shortest = 1, minBT = 1000;
 for (int i = 0; i < n; i++)
 if (p[i].completed == 0 & p[i].BT < minBT)
 shortest = p[i].ID;
 shortest = i;

 if (shortest == -1)
 current_time++;
 else
 {
 p[shortest].RT = current_time - p[shortest].AT;
 p[shortest].CT = current_time + p[shortest].BT;
 current_time = p[shortest].CT;
 p[shortest].TAT = p[shortest].CT - p[shortest].AT;
 p[shortest].WT = p[shortest].TAT - p[shortest].BT;
 p[shortest].completed = 1;
 completed++;
 }
 }
}

Date: / /

SJF

```
void calculate_SJF_NonPreemptive(struct process p[], int n)  
{  
    int completed = 0, current_time = 0;  
    while (completed < n)  
    {  
        int shortest = 1, minBT = 1000;  
        for (int i = 0; i < n; i++)  
            if (p[i].completed == 0 & p[i].BT < minBT)  
                shortest = p[i].ID;  
        shortest = i;  
  
        if (shortest == -1)  
            current_time++;  
        else  
        {  
            p[shortest].RT = current_time - p[shortest].AT;  
            p[shortest].CT = current_time + p[shortest].BT;  
            current_time = p[shortest].CT;  
            p[shortest].TAT = p[shortest].CT - p[shortest].AT;  
            p[shortest].WT = p[shortest].TAT - p[shortest].BT;  
            p[shortest].completed = 1;  
            completed++;  
        }  
    }  
}
```

SJF (Non Preemptive)

```
void calculate_SJF_NonPreemptive(struct process p[], int n)  
{  
    int completed = 0, current_time = 0;  
    while (completed < n)  
    {  
        int shortest = 1, minBT = 1000;  
        for (int i = 0; i < n; i++)  
            if (p[i].completed == 0 & p[i].BT < minBT)  
                shortest = p[i].ID;  
        shortest = i;  
  
        if (shortest == -1)  
            current_time++;  
        else  
        {  
            p[shortest].RT = current_time - p[shortest].AT;  
            p[shortest].CT = current_time + p[shortest].BT;  
            current_time = p[shortest].CT;  
            p[shortest].TAT = p[shortest].CT - p[shortest].AT;  
            p[shortest].WT = p[shortest].TAT - p[shortest].BT;  
            p[shortest].completed = 1;  
            completed++;  
        }  
    }  
}
```

SJF (Preemptive)

```
void calculate_SJF_Preemptive(struct process p[], int n)  
{  
    int completed = 0, current_time = 0;  
    while (completed < n)  
    {  
        int shortest = 1, minBT = 1000;  
        for (int i = 0; i < n; i++)  
            if (p[i].remaining_BT == p[i].BT)  
                shortest = i;  
        shortest = i;  
  
        if (shortest == -1)  
            current_time++;  
        else  
        {  
            if (p[shortest].remaining_BT == p[shortest].BT)  
                p[shortest].CT = current_time + p[shortest].BT;  
            p[shortest].remaining_BT -= current_time;  
            current_time++;  
  
            if (p[shortest].remaining_BT == 0)  
                p[shortest].CT = current_time;  
            p[shortest].TAT = p[shortest].CT - p[shortest].AT;  
            p[shortest].WT = p[shortest].TAT - p[shortest].AT;  
            p[shortest].completed = 1;  
            completed++;  
        }  
    }  
}
```

Output

Enter Arrival Time (AT) for process 1: 0
Enter Burst Time (BT) for process 1: 4
Enter Arrival Time (AT) for process 2: 2
Enter Burst Time (BT) for process 2: 5
Enter Arrival Time (AT) for process 3: 4
Enter Burst Time (BT) for process 3: 6

Menu:

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF) - Non Preemptive
3. Shortest Job First (SJF) - Preemptive
4. Exit

Enter choice: 1

Process	AT	BT	CT	TAT	WT	RT
1	0	4	4	4	0	0
2	2	5	9	7	2	2
3	4	6	15	11	5	5

Menu:

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF) - Non Preemptive
3. Shortest Job First (SJF) - Preemptive
4. Exit

Enter choice: 2

Process	AT	BT	CT	TAT	WT	RT
1	0	4	4	4	0	0
2	2	5	9	7	2	2
3	4	6	15	11	5	5

Menu:

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF) - Non Preemptive
3. Shortest Job First (SJF) - Preemptive
4. Exit

Enter choice: 2

Process	AT	BT	CT	TAT	WT	PT
1	0	4	4	4	0	0
2	2	5	9	7	2	2
3	4	6	15	11	5	5

Process 1 has arrived at time 0 and burst time is 4. It will start execution at time 0 and complete at time 4. Its turnaround time is 4 and waiting time is 0.

Process 2 has arrived at time 2 and burst time is 5. It will start execution at time 2 and complete at time 7. Its turnaround time is 5 and waiting time is 2.

Process 3 has arrived at time 4 and burst time is 6. It will start execution at time 4 and complete at time 15. Its turnaround time is 11 and waiting time is 5.

Program -2

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

→ Priority (pre-emptive & Non-pre-emptive)

Code:

```
#include <stdio.h>
#define MAX 10

typedef struct {
    int pid, at, bt, pt, remaining_bt, ct, tat, wt, rt, is_completed, st;
} Process;

void nonPreemptivePriority(Process p[], int n) {
    int time = 0, completed = 0;
    while (completed < n) {
        int highest_priority = -1, selected = -1;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && !p[i].is_completed && p[i].pt > highest_priority) {
                highest_priority = p[i].pt;
                selected = i;
            }
        }

        if (selected == -1) {
            time++;
            continue;
        }

        if (p[selected].rt == -1) {
            p[selected].st = time; // Start time
            p[selected].rt = time - p[selected].at; // Response Time = Start Time - Arrival Time
        }

        completed++;
    }
}
```

```

time += p[selected].bt;
p[selected].ct = time;
p[selected].tat = p[selected].ct - p[selected].at;
p[selected].wt = p[selected].tat - p[selected].bt;
p[selected].is_completed = 1;
completed++;
}
}

void preemptivePriority(Process p[], int n) {
    int time = 0, completed = 0;
    while (completed < n) {
        int highest_priority = -1, selected = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && p[i].remaining_bt > 0 && p[i].pt > highest_priority) {
                highest_priority = p[i].pt;
                selected = i;
            }
        }
        if (selected == -1) {
            time++;
            continue;
        }
        if (p[selected].rt == -1) {
            p[selected].st = time; // Start time
            p[selected].rt = time - p[selected].at; // Response Time = Start Time - Arrival Time
        }
        p[selected].remaining_bt--;
        time++;
        if (p[selected].remaining_bt == 0) {
            p[selected].ct = time;
        }
    }
}

```

```

    p[selected].tat = p[selected].ct - p[selected].at;
    p[selected].wt = p[selected].tat - p[selected].bt;
    completed++;
}

}

void displayProcesses(Process p[], int n) {
    float avg_tat = 0, avg_wt = 0, avg_rt = 0;

    printf("\nPID\tAT\tBT\tPriority\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",
               p[i].pid, p[i].at, p[i].bt, p[i].pt, p[i].ct, p[i].tat, p[i].wt, p[i].rt);
        avg_tat += p[i].tat;
        avg_wt += p[i].wt;
        avg_rt += p[i].rt;
    }
    printf("\nAverage TAT: %.2f", avg_tat / n);
    printf("\nAverage WT: %.2f", avg_wt / n);
    printf("\nAverage RT: %.2f\n", avg_rt / n);
}

int main() {
    Process p[MAX];
    int n, choice;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("\nEnter Arrival Time, Burst Time, and Priority for Process %d:\n", p[i].pid);
        printf("Arrival Time: ");
        scanf("%d", &p[i].at);
    }
}

```

```

printf("Burst Time: ");
scanf("%d", &p[i].bt);
printf("Priority (higher number means higher priority): ");
scanf("%d", &p[i].pt);
p[i].remaining_bt = p[i].bt;
p[i].is_completed = 0;
p[i].rt = -1;
}
while (1) {
    printf("\nPriority Scheduling Menu:\n");
    printf("1. Non-Preemptive Priority Scheduling\n");
    printf("2. Preemptive Priority Scheduling\n");
    printf("3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            nonPreemptivePriority(p, n);
            printf("Non-Preemptive Scheduling Completed!\n");
            displayProcesses(p, n);
            break;
        case 2:
            preemptivePriority(p, n);
            printf("Preemptive Scheduling Completed!\n");
            displayProcesses(p, n);
            break;
        case 3:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice! Try again.\n");
    }
}

```

```

    }
}

return 0;
}

```

OUTPUT:

GUvenBMSCEDesktopOS\priorityScheduling.exe

```

Enter the number of processes: 5
Enter Arrival Time, Burst Time, and Priority for Process 1:
Arrival Time: 0
burst time: 4
Priority (higher number means higher priority): 2
Enter Arrival Time, Burst Time, and Priority for Process 2:
Arrival Time: 1
burst time: 3
Priority (higher number means higher priority): 1
Enter Arrival Time, Burst Time, and Priority for Process 3:
Arrival Time: 2
burst time: 5
Priority (higher number means higher priority): 4
Enter Arrival Time, Burst Time, and Priority for Process 4:
Arrival Time: 3
burst time: 5
Priority (higher number means higher priority): 3
Enter Arrival Time, Burst Time, and Priority for Process 5:
Arrival Time: 4
burst time: 2
Priority (higher number means higher priority): 5
Priority Scheduling Menu:
1. Non-Preemptive Priority Scheduling
2. Preemptive Priority Scheduling
Enter your choice: 1
Non-Preemptive Scheduling Completed!

```

PID	AT	BT	Priority	CT	TAT	WT	RT
1	0	4	2	4	4	0	0
2	1	3	1	15	14	11	11
3	2	1	4	12	10	9	9
4	3	5	3	9	6	1	1
5	4	2	5	11	7	5	5

```

Average TAT: 8.20
Average WT: 5.20
Average RT: 5.20
Priority Scheduling Menu:
1. Non-Preemptive Priority Scheduling
2. Preemptive Priority Scheduling
Enter your choice: 2
Non-Preemptive Scheduling Completed!

```

20/12/25

higher wt → higher priority

papergrid

Date: / /

```

Write a C program to simulate the CPU scheduling algorithm known to find TAT & WT.
→ Priority (pre-emptive & non-preemptive)

#include <stdio.h>
#define MAX 10

typedef struct {
    int pid, at, bt, pt, remaining_bt, ct, tat,
    wt, rt, is_completed, st;
} Process;

void nonPreemptivePriority( Process p[], int n)
{
    int time = 0, completed = 0;
    while (completed < n)
    {
        int highest_priority = -1, selected = -1;
        for (int i = 0; i < n; i++)
        {
            if (p[i].at <= time && p[i].is_completed == 0)
            {
                if (p[i].pt > highest_priority)
                {
                    highest_priority = p[i].pt;
                    selected = i;
                }
            }
        }
        if (selected == -1)
        {
            time++;
            continue;
        }
        if (p[selected].rt == -1)
        {
            p[selected].st = time;
            p[selected].rt = time - p[selected].at;
        }
        time += p[selected].bt;
        p[selected].ct = time;
        p[selected].tat = p[selected].ct - p[selected].at;
        p[selected].wt = p[selected].tat - p[selected].rt;
        p[selected].is_completed = 1;
        completed++;
    }
}

```

```

p[Selected].rt = -1;
p[Selected].st = time;
p[Selected].rt = time - p[Selected].at;
time += p[Selected].bt;
p[Selected].ct = time;
p[Selected].tat = p[Selected].ct - p[Selected].at;
p[Selected].wt = p[Selected].tat - p[Selected].rt;
completed++;

```

void displayingProcess(Process p[], int n)
{
 float avg_tat = 0, avg_wt = 0, avg_rt = 0;
 printf ("\\n\\nPID AT BT Priority CT TAT WT RT\\n");
 for (int i = 0; i < n; i++)
 {
 cout << i+1 << " " << p[i].at << " " << p[i].bt << " "
 << p[i].pt << " " << p[i].ct << " " << p[i].tat << " "
 << p[i].wt << " " << p[i].rt << "\\n";
 }
 printf ("\\n Average TAT: %.2f ", avg_tat);
 printf ("\\n Average WT: %.2f ", avg_wt);
 printf ("\\n Average RT: %.2f ", avg_rt);
}

Output:

Non Preemptive Scheduling

Process	AT	BT	Priority	CT	TAT	WT	RT
1	0	4	2	4	4	0	0
2	1	3	1	15	14	11	11
3	2	1	4	12	10	9	9
4	3	5	3	9	6	1	1
5	4	2	5	11	7	5	5

Preemptive

Process	AT	BT	Priority	CT	TAT	WT	RT
1	0	4	2	15	15	11	0
2	1	3	3	12	11	8	11
3	2	1	4	3	1	0	9
4	3	5	5	8	5	0	1
5	4	2	5	10	6	4	5

Non Preemptive

Process	AT	BT	Priority	CT	TAT	WT	RT
1	0	4	2	4	4	0	0
2	1	3	1	15	14	11	11
3	2	1	4	12	10	9	9
4	3	5	3	9	6	1	1
5	4	2	5	11	7	5	5

Burst Time

Process	AT	BT	Priority	CT	TAT	WT	RT
1	0	4	2	4	4	0	0
2	1	3	1	15	14	11	11
3	2	1	4	12	10	9	9
4	3	5	3	9	6	1	1
5	4	2	5	11	7	5	5

Program -3

Question: 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 RR and FCFS scheduling for the processes in each queue.

Code:

```
#include <stdio.h>

#define MAX_PROCESSES 10
#define TIME_QUANTUM 2

typedef struct {

    int burst_time, arrival_time, queue_type, waiting_time, turnaround_time, response_time,
    remaining_time;

} Process;

void round_robin(Process processes[], int n, int time_quantum, int *time) {

    int done, i;
    do {
        done = 1;
        for (i = 0; i < n; i++) {
            if (processes[i].remaining_time > 0) {
                done = 0;
                if (processes[i].remaining_time > time_quantum) {
                    *time += time_quantum;
                    processes[i].remaining_time -= time_quantum;
                } else {
                    *time += processes[i].remaining_time;
                    processes[i].waiting_time = *time - processes[i].arrival_time -
processes[i].burst_time;
                }
            }
        }
    } while (!done);
}
```

```

        processes[i].turnaround_time = *time - processes[i].arrival_time;
        processes[i].response_time = processes[i].waiting_time;
        processes[i].remaining_time = 0;
    }
}

}

} while (!done);

}

void fcfs(Process processes[], int n, int *time) {
    for (int i = 0; i < n; i++) {
        if (*time < processes[i].arrival_time) {
            *time = processes[i].arrival_time;
        }
        processes[i].waiting_time = *time - processes[i].arrival_time;
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
        processes[i].response_time = processes[i].waiting_time;
        *time += processes[i].burst_time;
    }
}

int main() {
    Process processes[MAX_PROCESSES], system_queue[MAX_PROCESSES],
user_queue[MAX_PROCESSES];
    int n, sys_count = 0, user_count = 0, time = 0;
    float avg_waiting = 0, avg_turnaround = 0, avg_response = 0, throughput;

    printf("Enter number of processes: ");
}

```

```

scanf("%d", &n);

for (int i = 0; i < n; i++) {
    printf("Enter Burst Time, Arrival Time and Queue of P%d: ", i + 1);
    scanf("%d %d %d", &processes[i].burst_time, &processes[i].arrival_time,
&processes[i].queue_type);
    processes[i].remaining_time = processes[i].burst_time;

    if (processes[i].queue_type == 1) {
        system_queue[sys_count++] = processes[i];
    } else {
        user_queue[user_count++] = processes[i];
    }
}

// Sort user processes by arrival time for FCFS
for (int i = 0; i < user_count - 1; i++) {
    for (int j = 0; j < user_count - i - 1; j++) {
        if (user_queue[j].arrival_time > user_queue[j + 1].arrival_time) {
            Process temp = user_queue[j];
            user_queue[j] = user_queue[j + 1];
            user_queue[j + 1] = temp;
        }
    }
}

printf("\nQueue 1 is System Process\nQueue 2 is User Process\n");
round_robin(system_queue, sys_count, TIME_QUANTUM, &time);

```

```

fcfs(user_queue, user_count, &time);

printf("\nProcess Waiting Time Turn Around Time Response Time\n");

for (int i = 0; i < sys_count; i++) {
    avg_waiting += system_queue[i].waiting_time;
    avg_turnaround += system_queue[i].turnaround_time;
    avg_response += system_queue[i].response_time;
    printf("%d      %d      %d      %d\n", i + 1, system_queue[i].waiting_time,
           system_queue[i].turnaround_time, system_queue[i].response_time);
}

for (int i = 0; i < user_count; i++) {
    avg_waiting += user_queue[i].waiting_time;
    avg_turnaround += user_queue[i].turnaround_time;
    avg_response += user_queue[i].response_time;
    printf("%d      %d      %d      %d\n", i + 1 + sys_count,
           user_queue[i].waiting_time, user_queue[i].turnaround_time, user_queue[i].response_time);
}

avg_waiting /= n;
avg_turnaround /= n;
avg_response /= n;
throughput = (float)n / time;

printf("\nAverage Waiting Time: %.2f", avg_waiting);
printf("\nAverage Turn Around Time: %.2f", avg_turnaround);
printf("\nAverage Response Time: %.2f", avg_response);
printf("\nThroughput: %.2f", throughput);

```

```
printf("\nProcess returned %d (0x%08x) execution time: %.3f s\n", time, time, (float)time);
```

```
return 0;
```

```
}
```

OUTPUT:

```
C:\Users\Admin\Desktop\OS LAB\Multilevel.exe*
Enter number of processes: 4
Enter Burst Time, Arrival Time and Queue of P1: 2 0 1
Enter Burst Time, Arrival Time and Queue of P2: 1 0 2
Enter Burst Time, Arrival Time and Queue of P3: 5 0 1
Enter Burst Time, Arrival Time and Queue of P4: 3 0 2

Queue 1 is System Process
Queue 2 is User Process

Process Waiting Time Turn Around Time Response Time
1 0 2 0
2 2 7 2
3 7 8 7
4 8 11 8

Average Waiting Time: 4.25
Average Turn Around Time: 7.00
Average Response Time: 4.25
Throughput: 0.36
Process returned 11 (0x0000000B) execution time: 11.000 s
Press any key to continue.
```

```
#include <stdio.h>
#define MAX_PROCESSES 10
#define TIME_QUANTUM 2
#define MAX_WAITING 1000000000000000000L

typedef struct {
    int burst_time, arrival_time, queue_type,
        waiting_time, turnaround_time, response_time,
        remaining_time;
} Process;

Process processes[10], user_queue[MAX_PROCESSES];
int n, sys_count = 0, user_count = 0, time = 0;
float avg_waiting = 0, avg_turnaround = 0,
    avg_response = 0, throughput;

void round_robin(Process processes[], int n, int time)
{
    int done = 0;
    do {
        done = 1;
        for (int i = 0; i < n; i++) {
            if (processes[i].remaining_time) {
                done = 0;
                if (processes[i].remaining_time >
                    TIME_QUANTUM) {
                    user_queue[sys_count].waiting_time =
                        processes[i].arrival_time -
                        processes[i].burst_time -
                        processes[i].remaining_time;
                    processes[i].remaining_time -=
                        TIME_QUANTUM;
                    processes[i].response_time =
                        processes[i].waiting_time;
                    processes[i].remaining_time = 0;
                } else {
                    user_queue[sys_count].waiting_time =
                        processes[i].arrival_time -
                        processes[i].burst_time -
                        processes[i].remaining_time;
                    processes[i].remaining_time = 0;
                }
            }
        }
    } while (!done);
}

int main()
{
    Process processes[MAX_PROCESSES], system_queue[MAX_PROCESSES];
    int n, sys_count = 0, user_count = 0, time = 0;
    float avg_waiting = 0, avg_turnaround = 0,
        avg_response = 0, throughput;

    printf("Enter number of processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("Enter Burst Time, Arrival Time and
Queue of P%d: ", i + 1);
        scanf("%d %d %d", &processes[i].burst_time,
            &processes[i].arrival_time, &processes[i].queue_type);
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].arrival_time = process[i].burst_time;
        processes[i].response_time = process[i].waiting_time;
        processes[i].remaining_time = 0;
    }

    if (processes[0].queue_type == 2)
        sys_count++;
    else
        user_count++;

    user_queue[sys_count] = processes[0];
    sys_count++;

    while (user_count < n) {
        round_robin(system_queue, sys_count, TIME_QUANTUM,
            &time);
        user_queue[user_count].waiting_time =
            user_queue[user_count].arrival_time - time;
        user_queue[user_count].turnaround_time =
            user_queue[user_count].arrival_time +
            user_queue[user_count].waiting_time;
        user_queue[user_count].response_time =
            user_queue[user_count].turnaround_time -
            user_queue[user_count].arrival_time;
        user_count++;
    }

    for (int i = 0; i < user_count; i++)
        avg_waiting += user_queue[i].waiting_time;
        avg_turnaround += user_queue[i].turnaround_time;
        avg_response += user_queue[i].response_time;
    printf("Avg Waiting Time: %.2f", avg_waiting / user_count);
    printf("Avg Turnaround Time: %.2f", avg_turnaround / user_count);
    printf("Avg Response Time: %.2f", avg_response / user_count);
    printf("Throughput: %.2f", throughput);
    printf("Process returned %d (0x%08x) execution
time: %.3f s\n", time, time, (float)time);
    return 0;
}
```

```
void fofc(Process processes[], int n, int *time)
{
    for (int i = 0; i < n; i++) {
        if (*time < processes[i].arrival_time)
            *time = processes[i].arrival_time;
        if (processes[i].remaining_time >= time)
            processes[i].remaining_time -= time;
        else
            processes[i].remaining_time = 0;
        processes[i].response_time = processes[i].waiting_time +
            processes[i].burst_time;
        *time += processes[i].burst_time;
    }
}

int main()
{
    Process processes[MAX_PROCESSES], system_queue[MAX_PROCESSES];
    int n, sys_count = 0, user_count = 0, time = 0;
    float avg_waiting = 0, avg_turnaround = 0,
        avg_response = 0, throughput;

    printf("Enter number of processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("Enter Burst Time, Arrival Time and
Queue of P%d: ", i + 1);
        scanf("%d %d %d", &processes[i].burst_time,
            &processes[i].arrival_time, &processes[i].queue_type);
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].arrival_time = process[i].burst_time;
        processes[i].response_time = process[i].waiting_time;
        if (processes[0].queue_type == 2)
            sys_count++;
        else
            user_count++;
        user_queue[user_count] = processes[i];
    }

    while (user_count < n) {
        fofc(user_queue, user_count, &time);
        user_queue[user_count].waiting_time =
            user_queue[user_count].arrival_time - time;
        user_queue[user_count].turnaround_time =
            user_queue[user_count].arrival_time +
            user_queue[user_count].waiting_time;
        user_queue[user_count].response_time =
            user_queue[user_count].turnaround_time -
            user_queue[user_count].arrival_time;
        user_count++;
    }

    for (int i = 0; i < user_count; i++)
        avg_waiting += user_queue[i].waiting_time;
        avg_turnaround += user_queue[i].turnaround_time;
        avg_response += user_queue[i].response_time;
    printf("Avg Waiting Time: %.2f", avg_waiting / user_count);
    printf("Avg Turnaround Time: %.2f", avg_turnaround / user_count);
    printf("Avg Response Time: %.2f", avg_response / user_count);
    printf("Throughput: %.2f", throughput);
    printf("Process returned %d (0x%08x) execution
time: %.3f s\n", time, time, (float)time);
    return 0;
}
```

```
papergrid
Date: 1/1
for (int i = 0; i < user_count; i++) {
    for (int j = 0; j < user_count - i - 1; j++) {
        if (user_queue[j].arrival_time < user_queue[j + 1].arrival_time) {
            Process temp = user_queue[j];
            user_queue[j] = user_queue[j + 1];
            user_queue[j + 1] = temp;
        }
    }
}
printf("In Queue 1 is System Process\nQueue 2 is User Process");
round_robin(system_queue, sys_count, TIME_QUANTUM, &time);
fofc(user_queue, user_count, &time);
printf("InProcess Waiting Time Turn Around Time Response
Time\n");
for (int i = 0; i < user_count; i++) {
    for (int j = 0; j < user_count - i - 1; j++) {
        avg_waiting += user_queue[i].waiting_time;
        avg_turnaround += user_queue[i].turnaround_time;
        avg_response += user_queue[i].response_time;
    }
    printf("%d %d %d %d\n", i + 1, avg_waiting / user_count,
        avg_turnaround / user_count, avg_response / user_count);
}
for (int i = 0; i < sys_count; i++) {
    avg_waiting += system_queue[i].waiting_time;
    avg_turnaround += system_queue[i].turnaround_time;
    avg_response += system_queue[i].response_time;
}
printf("Avg Waiting Time: %.2f", avg_waiting / sys_count);
printf("Avg Turnaround Time: %.2f", avg_turnaround / sys_count);
printf("Avg Response Time: %.2f", avg_response / sys_count);
printf("Throughput: %.2f", throughput);
printf("Process returned %d (0x%08x) execution time: %.3f s\n",
    time, time, (float)time);
return 0;
}
```

```
papergrid
Date: 1/1
avg_waiting /> n
avg_turnaround /> n
avg_response /> n
Throughput = (Read) n / time;
printf("Avg Waiting Time: %.2f", avg_waiting);
printf("Avg Turnaround Time: %.2f", avg_turnaround);
printf("Avg Response Time: %.2f", avg_response);
printf("Throughput: %.2f", throughput);
printf("Process returned %d (0x%08x) execution
time: %.3f s\n", time, time, (float)time);
return 0;
}

Output
Enter number of processes: 4
Enter Burst Time, Arrival Time and Queue of P1: 2 0 1
Enter Burst Time, Arrival Time and Queue of P2: 1 0 2
Enter Burst Time, Arrival Time and Queue of P3: 5 0 1
Enter Burst Time, Arrival Time and Queue of P4: 3 0 2

Queue 1 is System Process
Queue 2 is User Process

Process WaitingTime TurnAroundTime ResponseTime
1 0 2 0
2 2 7 2
3 7 8 7
4 8 11 8

Avg Waiting Time: 4.25
Avg Turnaround Time: 7.00
Avg Response Time: 4.25
Throughput: 0.36
Process returned 11 (0x0000000B) execution time: 11.000 s
Process returned 11 (0x0000000B) execution time: 11.000 s
Process returned 0 (0x0) execution time: 145.173 s
Press any key to continue.
```

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

- a) Rate- Monotonic
- b) Earliest-deadline First

Code:

a) Rate Monotonic

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#define MAX_PROCESS 10

int num_of_process;
int execution_time[MAX_PROCESS], period[MAX_PROCESS],
    remain_time[MAX_PROCESS], deadline[MAX_PROCESS],
    remain_deadline[MAX_PROCESS];

void get_process_info() {
    printf("Enter total number of processes (maximum %d): ", MAX_PROCESS);
    scanf("%d", &num_of_process);

    if (num_of_process < 1) {
        exit(0);
    }

    for (int i = 0; i < num_of_process; i++) {
        printf("\nProcess %d:\n", i + 1);
        printf("==> Execution time: ");
        scanf("%d", &execution_time[i]);
        remain_time[i] = execution_time[i];
```

```

printf("==> Period: ");
scanf("%d", &period[i]);
}

}

int max(int a, int b, int c) {
    int max;
    if (a >= b && a >= c)
        max = a;
    else if (b >= a && b >= c)
        max = b;
    else if (c >= a && c >= b)
        max = c;
    return max;
}

void print_schedule(int process_list[], int cycles) {
    printf("\nScheduling:\n\n");
    printf("Time: ");
    for (int i = 0; i < cycles; i++) {
        if (i < 10)
            printf("| 0%d ", i);
        else
            printf("| %d ", i);
    }
    printf("|\n");

    for (int i = 0; i < num_of_process; i++) {
        printf("P[%d]: ", i + 1);
        for (int j = 0; j < cycles; j++) {
            if (process_list[j] == i + 1)

```

```

        printf("|#####");
    else
        printf("|   ");
    }
    printf("\n");
}

void rate_monotonic(int time) {
    int process_list[100] = {0}, min = 999, next_process = 0;
    float utilization = 0;

    for (int i = 0; i < num_of_process; i++) {
        utilization += (1.0 * execution_time[i]) / period[i];
    }

    int n = num_of_process;
    float m = n * (pow(2, 1.0 / n) - 1);

    if (utilization > m) {
        printf("\nGiven problem is not schedulable under the said scheduling algorithm.\n");
    }

    for (int i = 0; i < time; i++) {
        min = 1000;
        for (int j = 0; j < num_of_process; j++) {
            if (remain_time[j] > 0) {
                if (min > period[j]) {
                    min = period[j];
                    next_process = j;
                }
            }
        }
    }
}

```

```

        }

    }

if (remain_time[next_process] > 0) {
    process_list[i] = next_process + 1;
    remain_time[next_process] -= 1;
}

for (int k = 0; k < num_of_process; k++) {
    if ((i + 1) % period[k] == 0) {
        remain_time[k] = execution_time[k];
        next_process = k;
    }
}
print_schedule(process_list, time);
}

int main() {
    int observation_time;
    get_process_info();
    observation_time = max(period[0], period[1], period[2]);
    rate_monotonic(observation_time);
    return 0;
}

```

OUTPUT:

Screenshot of a terminal window showing the execution of a Rate Monotonic CPU Scheduling algorithm. The terminal shows process details, scheduling timeline, and a sample C code for the algorithm.

```

C:\Users\Aman\Desktop\GSLAB\Rate Monotonic
-> Period 10
Process 2:
=> Execution time: 1
=> Period: 5
Process 3:
=> Execution time: 5
=> Period: 30
-> Period 30
Process 4:
=> Execution time: 2
=> Period: 15
Scheduling:
Time: 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
P[1]: #####
P[2]: #####
P[3]: #####
P[4]: #####
Process returned 0 (0x0) execution time : 25.854 s
press any key to continue.

```

Handwritten notes on the right side of the screenshot show the Rate Monotonic CPU Scheduling Algorithm pseudocode and a max function implementation.

```

Rate Monotonic CPU Scheduling Algorithm
Date: 1/1/25
#include <math.h>
#include <math.h>
#include <math.h>
#define MAX_PROCESS 10

int num_of_processes;
int execution_time[MAX_PROCESS], period_time[MAX_PROCESS];
float remain_time[MAX_PROCESS], deadline[MAX_PROCESS];
float remain_deadline[MAX_PROCESS];

void get_process_info() {
    printf("Enter total number of processes(maximum %d):\n", MAX_PROCESS);
    scanf("%d", &num_of_processes);
    if (num_of_processes < 2) {
        exit(0);
    }
    for (int i = 0; i < num_of_processes; i++) {
        printf("\nEnter Period %d: ", i + 1);
        scanf("%d", &period_time[i]);
        printf("\nEnter Execution time %d: ", i + 1);
        scanf("%d", &execution_time[i]);
        remain_time[i] = execution_time[i];
        printf("\nEnter Period %d: ", i + 1);
        scanf("%d", &deadline[i]);
    }
}

int max(int a, int b, int c) {
    int max;
    if (a >= b && a >= c) {
        max = a;
    } else if (b >= a && b >= c) {
        max = b;
    } else {
        max = c;
    }
    return max;
}

Output
Enter total number of processes(maximum 10): 4
Process 1:
=> Execution time: 2
=> Period: 2
Process 2:
=> Execution time: 1
=> Period: 5
Process 3:
=> Execution time: 5
=> Period: 30
Process 4:
=> Execution time: 2
=> Period: 15
Scheduling
Time: 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
P[1]: #####
P[2]: #####
P[3]: #####
P[4]: #####

```

Handwritten notes below the screenshot show two versions of the C code for the Rate Monotonic algorithm: one for printing the schedule and one for calculating utilization.

```

void print_schedule(int process_list[], int num_of_processes) {
    printf("In Scheduling:\n");
    printf("Time: ");
    for (int i = 0; i < num_of_processes; i++) {
        if (i < 10)
            printf("0%d ", i);
        else
            printf("1%d ", i);
    }
    printf("\n");
    for (int i = 0; i < num_of_processes; i++) {
        printf("P[%d]: ", i + 1);
        for (int j = 0; j < cycles; j++) {
            if (process_list[j] == i + 1)
                printf("1#%d ", i);
            else
                printf("1#%d ", i);
        }
        printf("\n");
    }
    printf("3/\n");
}

void rate_monotonic(int time) {
    int process_list[100] = {0}, min = 999, next_process = 0;
    float utilization = 0;
    for (int i = 0; i < num_of_processes; i++)
        utilization += (1.0 * execution_time[i]) / time;
    int num_of_processes;
    float m = n * (pow(2, 1.0 / n) - 1);
    if (utilization > m) {
        printf("Given problem is not schedulable\n");
    }
}

for (int i = 0; i < time; i++) {
    min = 1000;
    for (int j = 0; j < num_of_processes; j++) {
        if (remain_time[j] > 0) {
            if (min > period[j]) {
                min = period[j];
                next_process = j;
            }
        }
    }
    if (remain_time[next_process] > 0) {
        process_list[i % next_process + 1] = next_process + 1;
        remain_time[next_process] -= 1;
    }
    for (int k = 0; k < num_of_processes; k++) {
        if ((i + 1) % period[k] == 0) {
            remain_time[k] = execution_time[k];
            next_process = k;
        }
    }
}
printf(schedule(process_list, time));
}

int main() {
    int observation_time;
    get_process_info();
    observation_time = max(period[0], period[1], period[2]);
    rate_monotonic(observation_time);
    return 0;
}

```

b)Earliest Deadline

```
#include <stdio.h>

int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int lcm(int a, int b) {
    return (a * b) / gcd(a, b);
}

struct Process {
    int id, burst_time, deadline, period;
};

void earliest_deadline_first(struct Process p[], int n, int time_limit) {
    int time = 0;
    printf("Earliest Deadline Scheduling:\n");
    printf("PID\tBurst\tDeadline\tPeriod\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\n", p[i].id, p[i].burst_time, p[i].deadline, p[i].period);
    }

    printf("\nScheduling occurs for %d ms\n", time_limit);
    while (time < time_limit) {
        int earliest = -1;
        for (int i = 0; i < n; i++) {
```

```

        if (p[i].burst_time > 0) {
            if (earliest == -1 || p[i].deadline < p[earliest].deadline) {
                earliest = i;
            }
        }
    }

if (earliest == -1) break;

printf("%dms: Task %d is running.\n", time, p[earliest].id);
p[earliest].burst_time--;
time++;
}

}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[n];
    printf("Enter the CPU burst times:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processes[i].burst_time);
        processes[i].id = i + 1;
    }

    printf("Enter the deadlines:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processes[i].deadline);
    }
}

```

```

printf("Enter the time periods:\n");
for (int i = 0; i < n; i++) {
    scanf("%d", &processes[i].period);
}

int hyperperiod = processes[0].period;
for (int i = 1; i < n; i++) {
    hyperperiod = lcm(hyperperiod, processes[i].period);
}

printf("\nSystem will execute for hyperperiod (LCM of periods): %d ms\n", hyperperiod);

earliest_deadline_first(processes, n, hyperperiod);

return 0;
}

```

OUTPUT:

```

C:\Users\Vedika\OneDriveD Enter the number of processes: 3
Enter the CPU burst times:
2 3 4
Enter the deadlines:
1 2 3
Enter the time periods:
1 2 3

System will execute for hyperperiod (LCM of periods): 6 ms
Earliest Deadline Scheduling:
PID    Burst    Deadline    Period
1      2        1          1
2      3        2          2
3      4        3          3

Scheduling occurs for 6 ms
0ms: Task 1 is running.
1ms: Task 1 is running.
2ms: Task 2 is running.
3ms: Task 2 is running.
4ms: Task 2 is running.
5ms: Task 3 is running.

Process returned 0 (0x0) execution time : 14.084 s
Press any key to continue.

```

Handwritten code for Earliest Deadline Scheduling:

```

#include <stdint.h>
int gcd (int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int lcm (int a, int b) {
    return (a * b) / gcd(a, b);
}

struct Process {
    int id, burst_time, deadline, period,
    int earliest;
};

void earliest_deadline_first (struct Process p[],
    int n, int time_limit) {
    int time = 0;
    printf ("Earliest deadline Scheduling:\n");
    printf ("PID    Burst    Deadline    Period\n");
    for (int i = 0; i < n; i++) {
        printf ("%d    %d    %d    %d\n",
            p[i].id, p[i].burst_time, p[i].deadline,
            p[i].period);
    }

    printf ("Scheduling occurs for %d ms\n",
        time_limit);
    while (time < time_limit) {
        int earliest = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].burst_time > 0) {
                if (earliest == -1 || p[i].deadline < p[earliest].deadline)
                    earliest = i;
            }
        }

        printf ("Time: %d ms, Task %d is running.\n",
            time, p[earliest].id);
        p[earliest].burst_time -= 1;
        time += 1;
    }
}

```

Handwritten code for Earliest Deadline Scheduling (Continued):

```

if (earliest == -1) break;
printf ("%d ms, Task %d is running.\n",
    time, p[earliest].id);
p[earliest].burst_time -= 1;
time += 1;

int main () {
    int n;
    printf ("Enter the number of processes: ");
    scanf ("%d", &n);
    struct Process processes [n];
    printf ("Enter the CPU burst times: ");
    for (int i = 0; i < n; i++) {
        scanf ("%d", &processes[i].burst_time);
    }

    printf ("Enter the deadlines: ");
    for (int i = 0; i < n; i++) {
        scanf ("%d", &processes[i].deadline);
    }

    printf ("Enter the time periods: ");
    for (int i = 0; i < n; i++) {
        scanf ("%d", &processes[i].period);
    }

    int hyperperiod = lcm (hyperperiod, processes[0].period);
    for (int i = 1; i < n; i++) {
        hyperperiod = lcm (hyperperiod, processes[i].period);
    }

    printf ("In System will execute for hyperperiod
    (LCM of periods): %d ms\n", hyperperiod);
    earliest_deadline_first (processes, n, hyperperiod);
    return 0;
}

```

Handwritten notes from the output window:

- Enter the number of processes: 3
- Enter the CPU burst times: 2 3 4
- Enter the deadlines: 1 2 3
- Enter the time periods: 1 2 3
- System will execute for hyperperiod (LCM of periods): 6 ms
- Earliest Deadline Scheduling:

PID	Burst	Deadline	Period
1	2	1	1
2	3	2	2
3	4	3	3

- Scheduling occurs for 6 ms
- 0ms: Task 1 is running.
- 1ms: Task 1 is running.
- 2ms: Task 2 is running.
- 3ms: Task 2 is running.
- 4ms: Task 2 is running.
- 5ms: Task 3 is running.

Program -4

Question: Write a C program to simulate **producer-consumer problem using semaphores**

Code:

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

int mutex = 1;
int full = 0;
int empty = 3;
int buffer[3];
int in = 0;
int out = 0;

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

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

void display_buffer() {
    printf("Buffer:");
    for (int i = 0; i < full; i++) {
        int index = (out + i) % 3;
        printf("%d ", buffer[index]);
    }
    printf("\n");
}
```

```

void producer(int id) {
    if ((mutex == 1) && (empty != 0)) {
        mutex = wait(mutex);
        full = signal(full);
        empty = wait(empty);

        int item = rand() % 40;
        buffer[in] = item;
        printf("Producer %d produced %d\n", id, item);
        in = (in + 1) % 3;

        display_buffer();
        mutex = signal(mutex);
    } else {
        printf("Buffer is full\n");
    }
}

void consumer(int id) {
    if ((mutex == 1) && (full != 0)) {
        mutex = wait(mutex);
        full = wait(full);
        empty = signal(empty);

        int item = buffer[out];
        printf("Consumer %d consumed %d\n", id, item);
        out = (out + 1) % 3;

        printf("Current buffer len: %d\n", full);
        mutex = signal(mutex);
    } else {
}

```

```

    printf("Buffer is empty\n");
}

}

int main() {
    int p, c, capacity, choice;

    srand(time(NULL));

    printf("Enter the number of Producers:");
    scanf("%d", &p);
    printf("Enter the number of Consumers:");
    scanf("%d", &c);
    printf("Enter buffer capacity:");
    scanf("%d", &capacity);

    empty = capacity;

    for (int i = 1; i <= p; i++) {
        printf("Successfully created producer %d\n", i);
    }
    for (int i = 1; i <= c; i++) {
        printf("Successfully created consumer %d\n", i);
    }

    while (1) {
        printf("\n1. Produce\n2. Consume\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:

```

```

    producer(1);
    break;

case 2:
    consumer(2);
    break;

case 3:
    exit(0);

default:
    printf("Invalid choice\n");
}

}

return 0;
}

```

OUTPUT:

```

C:\Users\Admin\Desktop\OS LAB\Producer Consumer.exe"
Enter the number of Producers:1
Enter the number of consumers:1
Enter buffer capacity:1
Successfully created producer 1
Successfully created consumer 1
1. Produce
2. Consume
3. Exit
Enter your choice: 1
Producer 1 produced 33
Buffer:33
1. Produce
2. Consume
3. Exit
Enter your choice: 2
Consumer 2 consumed 33
Current buffer len: 0
1. Produce
2. Consume
3. Exit
Enter your choice: 1
Producer 1 produced 2
Buffer:2
1. Produce
2. Consume
3. Exit
Enter your choice: 2
Consumer 2 consumed 2
Current buffer len: 0
1. Produce
2. Consume
3. Exit
Enter your choice: 1
Producer 1 produced 13
Buffer:13
1. Produce
2. Consume
3. Exit
Enter your choice: 2
Consumer 2 consumed 13
Current buffer len: 0
1. Produce
2. Consume
3. Exit
Enter your choice: 3
process returned 0 (0x0)  execution time : 209.774 s
Press any key to continue.

```

```

1.1.05
Producer Consumer
#include < stdio.h >
#include < stdlib.h >
#include < time.h >
int mutex = 1;
int full = 0;
int empty = 3;
int buffer[3];
int in = 0;
int out = 0;

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

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

void displayBuffer() {
    printf("Buffer");
    for (int i = 0; i < full; i++) {
        int index = (out + i) % 3;
        printf("%d", buffer[index]);
    }
    printf("\n");
}

void producer(int id) {
    if (mutex == 1 && (empty == 0)) {
        mutex = wait(mutex);
        full = signal(full);
        empty = wait(empty);
        int item = rand() % 40;
        buffer[in] = item;
        printf("Producer %d produced %d\n", id, item);
        in = (in + 1) % 3;
    }
}

```

```

displayBuffer();
mutex = signal(mutex);
}
else {
    printf("Buffer is full\n");
}

void consumer(int id) {
    if (mutex == 1 && (full == 0)) {
        mutex = wait(mutex);
        full = wait(full);
        empty = signal(empty);
        int item = buffer[out];
        printf("Consumer %d consumed %d\n", id, item);
        out = (out + 1) % 3;
        printf("Current buffer len: %d\n", full);
        mutex = signal(mutex);
    }
    else {
        printf("Buffer is empty\n");
    }
}

int main() {
    int p, c, capacity, choice;
    srand(time(NULL));
    printf("Enter the number of Producers:");
    scanf("%d", &p);
    printf("Enter the number of Consumers:");
    scanf("%d", &c);
    printf("Enter buffer capacity:");
    scanf("%d", &capacity);
    empty = capacity;
    for (int i = 1; i <= p; i++) {
        printf("Successfully created producer %d\n", i);
    }
}

```

```

for (int i = 1; i <= c; i++) {
    printf("Successfully created consumer %d\n");
}

while(1) {
    printf("1. Produce\n2. Consume\n3. Exit\n");
    printf("Enter your choice : ");
    scanf("%d", &choice);
    switch(choice) {
        case 1: producer(1);
        break;
        case 2: consumer(2);
        break;
        case 3: exit(0);
        default: printf("Invalid choice\n");
    }
}

Output
Enter the number of Producers: 1
Enter the number of Consumers: 1
Enter buffer capacity: 1
Successfully created producer 1
Successfully created consumer 1
1. Produce
2. Consume
3. Exit
Enter your choice: 1
Producer 1 produced 33
Buffer: 33

```

```

1. Produce
2. Consume
3. Exit
Enter your choice: 2
Consumer 2 consumed 33
Current buffer len: 0
1. Produce
2. Consume
3. Exit
Enter your choice: 1
Producer 1 produced 12
Buffer: 12
1. Produce
2. Consume
3. Exit
Enter your choice: 2
Consumer 2 consumed 12
Current buffer len: 0

```

Date: / /

```

1. Produce
2. Consume
3. Exit
Enter your choice: 3

```

Question: Write a C program to simulate the concept of **Dining Philosophers** problem.

Code:

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>
#define N 5 // Number of philosophers
#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];

// Function to test if a philosopher can start eating
void test(int phnum)
{
    if (state[phnum] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
    {
        state[phnum] = EATING;
        sleep(2); // Simulate time spent eating
        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]); // Signal that the philosopher can start eating
    }
}

void take_fork(int phnum)
{
```

```

sem_wait(&mutex);
state[phnum] = HUNGRY;
printf("Philosopher %d is Hungry\n", phnum + 1);

test(phnum);
sem_post(&mutex);
sem_wait(&S[phnum]);
sleep(1);

}

void put_fork(int phnum)
{
    sem_wait(&mutex);
    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 = num;
    while (1) {
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
    }
}

```

```
}
```

```
int main()
{
    int i;
    pthread_t thread_id[N];
    sem_init(&mutex, 0, 1);
    for (i = 0; i < N; i++) {
        sem_init(&S[i], 0, 0);
    }
    for (i = 0; i < N; i++) {
        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;
}
```

OUTPUT:

```

Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
^ [Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
^ [Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 4 is Hungry
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 4 is Hungry
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4

```

15/1/25

Dining Philosopher

Date: / /

papergrid

```

#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 phid[N] = {0, 1, 2, 3, 4};
sem_t mutex;
sem_t S[N];

void test (int phnum)
{
    if (state[phnum] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != 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);
        sem_post(&S[phnum]);
    }
}

void take_fork(int phnum)
{
    sem_wait(&mutex);
    state[phnum] = HUNGRY;
    printf("Philosopher %d is Hungry\n", phnum);
    test(phnum);
    sem_post(&mutex);
    sem_wait(&S[phnum]);
    sleep(1);
}

3

```

Date: / /

```

void put_fork(int phnum)
{
    sem_wait(&mutex);
    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 = num;
    while(1)
    {
        sleep(1);
        take_fork(i);
        sleep(0);
        put_fork(*i);
    }
}

int main()
{
    int i;
    pthread_t thread_id[N];
    sem_init(&mutex, 0, 1);
    for(i=0;i<N;i++)
        sem_init(&S[i], 0, 0);

    for(i=0;i<N;i++)
    {
        pthread_create(&thread_id[i], NULL,
                      philosopher, (void*)i);
        printf("Philosopher %d is thinking\n", i);
    }
}

```

Date: / /

papergrid

```

for (i=0; i<N; i++) {
    pthread_join(thread_id[i], NULL);
}
return 0;
}

Output: (infinite loop)
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 is putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 3
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 is putting fork 2 and 4 down
Philosopher 4 is thinking
^ [Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
^ [Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
^ [Philosopher 5 takes fork 4 and 5

```

continues ..

Program -5

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

Code:

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 10

int main() {
    int n, m, i, j, k;
    int alloc[MAX][MAX], max[MAX][MAX], avail[MAX], need[MAX][MAX];
    int finish[MAX] = {0}, safeSeq[MAX], work[MAX];
    printf("Enter number of processes and resources:\n");
    scanf("%d %d", &n, &m);
    printf("Enter allocation matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);
    printf("Enter maximum matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            scanf("%d", &max[i][j]);
    printf("Enter available matrix:\n");
    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];
    for (i = 0; i < m; i)
        work[i] = avail[i];
```

```

int count = 0;
bool found;
while (count < n) {
    found = false;
    for (i = 0; i < n; i++) {
        if (!finish[i]) {
            for (j = 0; j < m; j++) {
                if (need[i][j] > work[j])
                    break;
            }
            if (j == m) {
                for (k = 0; k < m; k++)
                    work[k] += alloc[i][k];
                safeSeq[count++] = i;
                finish[i] = 1;
                found = true;
            }
        }
    }
    if (!found)
        break;
}
if (count == n) {
    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");
} else {
    printf("System is in unsafe state\n");
}

return 0;
}

```

OUTPUT:

```

C:\Users\Admin\Desktop\OS LAB\Ba... - □ ×
Enter number of processes and resources:
5 3
Enter allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter maximum matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter available matrix:
3 3 2
System is in safe state
Safe sequence is: P1 -> P3 -> P4 -> P0 -> P2
Process returned 0 (0x0) execution time : 7
0.111 s
Press any key to continue.

```

Banks's Algorithm

```

#include <stdio.h>
#include <conio.h>
#define MAX 10
int main ()
{
    int n, m, i, j, k;
    int alloc[MAX][MAX], max[MAX][MAX];
    int avail[MAX][MAX];
    int finish[MAX][MAX], safeSeq[MAX];
    int work[10];
    int count = 0;
    int found = false;
    int i1, i2, i3;
    int i4, i5, i6, i7;
    int i8, i9, i10, i11;
    int i12, i13, i14, i15;
    int i16, i17, i18, i19;
    int i20, i21, i22, i23;
    int i24, i25, i26, i27;
    int i28, i29, i30, i31;
    int i32, i33, i34, i35;
    int i36, i37, i38, i39;
    int i40, i41, i42, i43;
    int i44, i45, i46, i47;
    int i48, i49, i50, i51;
    int i52, i53, i54, i55;
    int i56, i57, i58, i59;
    int i60, i61, i62, i63;
    int i64, i65, i66, i67;
    int i68, i69, i70, i71;
    int i72, i73, i74, i75;
    int i76, i77, i78, i79;
    int i80, i81, i82, i83;
    int i84, i85, i86, i87;
    int i88, i89, i90, i91;
    int i92, i93, i94, i95;
    int i96, i97, i98, i99;
    int i100, i101, i102, i103;
    int i104, i105, i106, i107;
    int i108, i109, i110, i111;
    int i112, i113, i114, i115;
    int i116, i117, i118, i119;
    int i120, i121, i122, i123;
    int i124, i125, i126, i127;
    int i128, i129, i130, i131;
    int i132, i133, i134, i135;
    int i136, i137, i138, i139;
    int i140, i141, i142, i143;
    int i144, i145, i146, i147;
    int i148, i149, i150, i151;
    int i152, i153, i154, i155;
    int i156, i157, i158, i159;
    int i160, i161, i162, i163;
    int i164, i165, i166, i167;
    int i168, i169, i170, i171;
    int i172, i173, i174, i175;
    int i176, i177, i178, i179;
    int i180, i181, i182, i183;
    int i184, i185, i186, i187;
    int i188, i189, i190, i191;
    int i192, i193, i194, i195;
    int i196, i197, i198, i199;
    int i200, i201, i202, i203;
    int i204, i205, i206, i207;
    int i208, i209, i210, i211;
    int i212, i213, i214, i215;
    int i216, i217, i218, i219;
    int i220, i221, i222, i223;
    int i224, i225, i226, i227;
    int i228, i229, i230, i231;
    int i232, i233, i234, i235;
    int i236, i237, i238, i239;
    int i240, i241, i242, i243;
    int i244, i245, i246, i247;
    int i248, i249, i250, i251;
    int i252, i253, i254, i255;
    int i256, i257, i258, i259;
    int i260, i261, i262, i263;
    int i264, i265, i266, i267;
    int i268, i269, i270, i271;
    int i272, i273, i274, i275;
    int i276, i277, i278, i279;
    int i280, i281, i282, i283;
    int i284, i285, i286, i287;
    int i288, i289, i290, i291;
    int i292, i293, i294, i295;
    int i296, i297, i298, i299;
    int i2900, i2901, i2902, i2903;
    int i2904, i2905, i2906, i2907;
    int i2908, i2909, i2910, i2911;
    int i2912, i2913, i2914, i2915;
    int i2916, i2917, i2918, i2919;
    int i2920, i2921, i2922, i2923;
    int i2924, i2925, i2926, i2927;
    int i2928, i2929, i2930, i2931;
    int i2932, i2933, i2934, i2935;
    int i2936, i2937, i2938, i2939;
    int i2940, i2941, i2942, i2943;
    int i2944, i2945, i2946, i2947;
    int i2948, i2949, i2950, i2951;
    int i2952, i2953, i2954, i2955;
    int i2956, i2957, i2958, i2959;
    int i2960, i2961, i2962, i2963;
    int i2964, i2965, i2966, i2967;
    int i2968, i2969, i2970, i2971;
    int i2972, i2973, i2974, i2975;
    int i2976, i2977, i2978, i2979;
    int i2980, i2981, i2982, i2983;
    int i2984, i2985, i2986, i2987;
    int i2988, i2989, i2990, i2991;
    int i2992, i2993, i2994, i2995;
    int i2996, i2997, i2998, i2999;
    int i29900, i29901, i29902, i29903;
    int i29904, i29905, i29906, i29907;
    int i29908, i29909, i29910, i29911;
    int i29912, i29913, i29914, i29915;
    int i29916, i29917, i29918, i29919;
    int i29920, i29921, i29922, i29923;
    int i29924, i29925, i29926, i29927;
    int i29928, i29929, i29930, i29931;
    int i29932, i29933, i29934, i29935;
    int i29936, i29937, i29938, i29939;
    int i29940, i29941, i29942, i29943;
    int i29944, i29945, i29946, i29947;
    int i29948, i29949, i29950, i29951;
    int i29952, i29953, i29954, i29955;
    int i29956, i29957, i29958, i29959;
    int i29960, i29961, i29962, i29963;
    int i29964, i29965, i29966, i29967;
    int i29968, i29969, i29970, i29971;
    int i29972, i29973, i29974, i29975;
    int i29976, i29977, i29978, i29979;
    int i29980, i29981, i29982, i29983;
    int i29984, i29985, i29986, i29987;
    int i29988, i29989, i29990, i29991;
    int i29992, i29993, i29994, i29995;
    int i29996, i29997, i29998, i29999;
    int i299900, i299901, i299902, i299903;
    int i299904, i299905, i299906, i299907;
    int i299908, i299909, i299910, i299911;
    int i299912, i299913, i299914, i299915;
    int i299916, i299917, i299918, i299919;
    int i299920, i299921, i299922, i299923;
    int i299924, i299925, i299926, i299927;
    int i299928, i299929, i299930, i299931;
    int i299932, i299933, i299934, i299935;
    int i299936, i299937, i299938, i299939;
    int i299940, i299941, i299942, i299943;
    int i299944, i299945, i299946, i299947;
    int i299948, i299949, i299950, i299951;
    int i299952, i299953, i299954, i299955;
    int i299956, i299957, i299958, i299959;
    int i299960, i299961, i299962, i299963;
    int i299964, i299965, i299966, i299967;
    int i299968, i299969, i299970, i299971;
    int i299972, i299973, i299974, i299975;
    int i299976, i299977, i299978, i299979;
    int i299980, i299981, i299982, i299983;
    int i299984, i299985, i299986, i299987;
    int i299988, i299989, i299990, i299991;
    int i299992, i299993, i299994, i299995;
    int i299996, i299997, i299998, i299999;
    int i2999000, i2999001, i2999002, i2999003;
    int i2999004, i2999005, i2999006, i2999007;
    int i2999008, i2999009, i2999010, i2999011;
    int i2999012, i2999013, i2999014, i2999015;
    int i2999016, i2999017, i2999018, i2999019;
    int i2999020, i2999021, i2999022, i2999023;
    int i2999024, i2999025, i2999026, i2999027;
    int i2999028, i2999029, i2999030, i2999031;
    int i2999032, i2999033, i2999034, i2999035;
    int i2999036, i2999037, i2999038, i2999039;
    int i2999040, i2999041, i2999042, i2999043;
    int i2999044, i2999045, i2999046, i2999047;
    int i2999048, i2999049, i2999050, i2999051;
    int i2999052, i2999053, i2999054, i2999055;
    int i2999056, i2999057, i2999058, i2999059;
    int i2999060, i2999061, i2999062, i2999063;
    int i2999064, i2999065, i2999066, i2999067;
    int i2999068, i2999069, i2999070, i2999071;
    int i2999072, i2999073, i2999074, i2999075;
    int i2999076, i2999077, i2999078, i2999079;
    int i2999080, i2999081, i2999082, i2999083;
    int i2999084, i2999085, i2999086, i2999087;
    int i2999088, i2999089, i2999090, i2999091;
    int i2999092, i2999093, i2999094, i2999095;
    int i2999096, i2999097, i2999098, i2999099;
    int i29990000, i29990001, i29990002, i29990003;
    int i29990004, i29990005, i29990006, i29990007;
    int i29990008, i29990009, i29990010, i29990011;
    int i29990012, i29990013, i29990014, i29990015;
    int i29990016, i29990017, i29990018, i29990019;
    int i29990020, i29990021, i29990022, i29990023;
    int i29990024, i29990025, i29990026, i29990027;
    int i29990028, i29990029, i29990030, i29990031;
    int i29990032, i29990033, i29990034, i29990035;
    int i29990036, i29990037, i29990038, i29990039;
    int i29990040, i29990041, i29990042, i29990043;
    int i29990044, i29990045, i29990046, i29990047;
    int i29990048, i29990049, i29990050, i29990051;
    int i29990052, i29990053, i29990054, i29990055;
    int i29990056, i29990057, i29990058, i29990059;
    int i29990060, i29990061, i29990062, i29990063;
    int i29990064, i29990065, i29990066, i29990067;
    int i29990068, i29990069, i29990070, i29990071;
    int i29990072, i29990073, i29990074, i29990075;
    int i29990076, i29990077, i29990078, i29990079;
    int i29990080, i29990081, i29990082, i29990083;
    int i29990084, i29990085, i29990086, i29990087;
    int i29990088, i29990089, i29990090, i29990091;
    int i29990092, i29990093, i29990094, i29990095;
    int i29990096, i29990097, i29990098, i29990099;
    int i299900000, i299900001, i299900002, i299900003;
    int i299900004, i299900005, i299900006, i299900007;
    int i299900008, i299900009, i299900010, i299900011;
    int i299900012, i299900013, i299900014, i299900015;
    int i299900016, i299900017, i299900018, i299900019;
    int i299900020, i299900021, i299900022, i299900023;
    int i299900024, i299900025, i299900026, i299900027;
    int i299900028, i299900029, i299900030, i299900031;
    int i299900032, i299900033, i299900034, i299900035;
    int i299900036, i299900037, i299900038, i299900039;
    int i299900040, i299900041, i299900042, i299900043;
    int i299900044, i299900045, i299900046, i299900047;
    int i299900048, i299900049, i299900050, i299900051;
    int i299900052, i299900053, i299900054, i299900055;
    int i299900056, i299900057, i299900058, i299900059;
    int i299900060, i299900061, i299900062, i299900063;
    int i299900064, i299900065, i299900066, i299900067;
    int i299900068, i299900069, i299900070, i299900071;
    int i299900072, i299900073, i299900074, i299900075;
    int i299900076, i299900077, i299900078, i299900079;
    int i299900080, i299900081, i299900082, i299900083;
    int i299900084, i299900085, i299900086, i299900087;
    int i299900088, i299900089, i299900090, i299900091;
    int i299900092, i299900093, i299900094, i299900095;
    int i299900096, i299900097, i299900098, i299900099;
    int i2999000000, i2999000001, i2999000002, i2999000003;
    int i2999000004, i2999000005, i2999000006, i2999000007;
    int i2999000008, i2999000009, i2999000010, i2999000011;
    int i2999000012, i2999000013, i2999000014, i2999000015;
    int i2999000016, i2999000017, i2999000018, i2999000019;
    int i2999000020, i2999000021, i2999000022, i2999000023;
    int i2999000024, i2999000025, i2999000026, i2999000027;
    int i2999000028, i2999000029, i2999000030, i2999000031;
    int i2999000032, i2999000033, i2999000034, i2999000035;
    int i2999000036, i2999000037, i2999000038, i2999000039;
    int i2999000040, i2999000041, i2999000042, i2999000043;
    int i2999000044, i2999000045, i2999000046, i2999000047;
    int i2999000048, i2999000049, i2999000050, i2999000051;
    int i2999000052, i2999000053, i2999000054, i2999000055;
    int i2999000056, i2999000057, i2999000058, i2999000059;
    int i2999000060, i2999000061, i2999000062, i2999000063;
    int i2999000064, i2999000065, i2999000066, i2999000067;
    int i2999000068, i2999000069, i2999000070, i2999000071;
    int i2999000072, i2999000073, i2999000074, i2999000075;
    int i2999000076, i2999000077, i2999000078, i2999000079;
    int i2999000080, i2999000081, i2999000082, i2999000083;
    int i2999000084, i2999000085, i2999000086, i2999000087;
    int i2999000088, i2999000089, i2999000090, i2999000091;
    int i2999000092, i2999000093, i2999000094, i2999000095;
    int i2999000096, i2999000097, i2999000098, i2999000099;
    int i29990000000, i29990000001, i29990000002, i29990000003;
    int i29990000004, i29990000005, i29990000006, i29990000007;
    int i29990000008, i29990000009, i29990000010, i29990000011;
    int i29990000012, i29990000013, i29990000014, i29990000015;
    int i29990000016, i29990000017, i29990000018, i29990000019;
    int i29990000020, i29990000021, i29990000022, i29990000023;
    int i29990000024, i29990000025, i29990000026, i29990000027;
    int i29990000028, i29990000029, i29990000030, i29990000031;
    int i29990000032, i29990000033, i29990000034, i29990000035;
    int i29990000036, i29990000037, i29990000038, i29990000039;
    int i29990000040, i29990000041, i29990000042, i29990000043;
    int i29990000044, i29990000045, i29990000046, i29990000047;
    int i29990000048, i29990000049, i29990000050, i29990000051;
    int i29990000052, i29990000053, i29990000054, i29990000055;
    int i29990000056, i29990000057, i29990000058, i29990000059;
    int i29990000060, i29990000061, i29990000062, i29990000063;
    int i29990000064, i29990000065, i29990000066, i29990000067;
    int i29990000068, i29990000069, i29990000070, i29990000071;
    int i29990000072, i29990000073, i29990000074, i29990000075;
    int i29990000076, i29990000077, i29990000078, i29990000079;
    int i29990000080, i29990000081, i29990000082, i29990000083;
    int i29990000084, i29990000085, i29990000086, i29990000087;
    int i29990000088, i29990000089, i29990000090, i29990000091;
    int i29990000092, i29990000093, i29990000094, i29990000095;
    int i29990000096, i29990000097, i29990000098, i29990000099;
    int i299900000000, i299900000001, i299900000002, i299900000003;
    int i299900000004, i299900000005, i299900000006, i299900000007;
    int i299900000008, i299900000009, i299900000010, i299900000011;
    int i299900000012, i299900000013, i299900000014, i299900000015;
    int i299900000016, i299900000017, i299900000018, i299900000019;
    int i299900000020, i299900000021, i299900000022, i299900000023;
    int i299900000024, i299900000025, i299900000026, i299900000027;
    int i299900000028, i299900000029, i299900000030, i299900000031;
    int i299900000032, i299900000033, i299900000034, i299900000035;
    int i299900000036, i299900000037, i299900000038, i299900000039;
    int i299900000040, i299900000041, i299900000042, i299900000043;
    int i299900000044, i299900000045, i299900000046, i299900000047;
    int i299900000048, i299900000049, i299900000050, i299900000051;
    int i299900000052, i299900000053, i299900000054, i299900000055;
    int i299900000056, i299900000057, i299900000058, i299900000059;
    int i299900000060, i299900000061, i299900000062, i299900000063;
    int i299900000064, i299900000065, i299900000066, i299900000067;
    int i299900000068, i299900000069, i299900000070, i299900000071;
    int i299900000072, i299900000073, i299900000074, i299900000075;
    int i299900000076, i299900000077, i299900000078, i299900000079;
    int i299900000080, i299900000081, i299900000082, i29990000
```

Question: Write a C program to simulate 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 alloc[n][m], request[n][m], avail[m];
    bool finish[n];
    printf("Enter allocation matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            scanf("%d", &alloc[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", &avail[i]);
    for (i = 0; i < n; i++) {
        bool is_zero = true;
        for (j = 0; j < m; j++) {
            if (alloc[i][j] != 0) {
                is_zero = false;
                break;
            }
        }
        if (is_zero)
            finish[i] = true;
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            if (alloc[i][j] > request[i][j]) {
                request[i][j] = alloc[i][j];
                alloc[i][j] = 0;
            } else {
                request[i][j] -= alloc[i][j];
                alloc[i][j] = 0;
            }
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            if (request[i][j] > avail[j]) {
                request[i][j] = avail[j];
                avail[j] = 0;
            } else {
                avail[j] -= request[i][j];
                request[i][j] = 0;
            }
        }
    }
    for (i = 0; i < n; i++) {
        if (finish[i])
            continue;
        for (j = 0; j < m; j++) {
            if (request[i][j] > 0) {
                printf("Process %d is in deadlock\n", i + 1);
                break;
            }
        }
    }
}
```

```

}

finish[i] = is_zero;

}

bool changed;

do {

changed = false;

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

if (!finish[i]) {

bool can_finish = true;

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

if (request[i][j] > avail[j]) {

can_finish = false;

break;

}

}

if (can_finish) {

for (k = 0; k < m; k++)

avail[k] += alloc[i][k];

finish[i] = true;

changed = true;

printf("Process %d can finish.\n", i);

}

}

}

}

} while (changed);

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:



```

"C:\Users\Admin\Desktop\OS LAB\Deadlock detection.exe"
Enter number of processes and resources:
4
Enter allocation matrix:
0 1 0
2 0 0
0 0 0
2 1 1
0 0 2
Enter request matrix:
1 0 0
3 2 2
0 0 2
2 0 0
4 3 3
Enter available matrix:
3 1 1
Process 1 can finish.
Process 3 can finish.
Process 4 can finish.
system is in a deadlock state.

Process returned 0 (0x0)  execution time : 67.276 s
Press any key to continue.

```

17/4/25

Deadlock Detection Algorithm

papergrid
Date: / /

```

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

int main() {
    int n, m, i, j, k;
    printf("Enter number of processes and resources: \n");
    scanf("%d %d", &n, &m);
    int alloc[n][m], request[n][m], avail[m];
    bool finish[n];
    printf("Enter allocation matrix: \n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            scanf("%d", &alloc[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", &avail[i]);
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            if (alloc[i][j] != 0) {
                is_zero = false;
                break;
            }
        }
        if (!is_zero) {
            finish[i] = true;
        }
    }
    bool changed;
    do {
        changed = false;
        for (i = 0; i < n; i++) {
            if (!finish[i]) {

```

papergrid
Date: / /

```

            bool can_finish = true;
            for (j = 0; j < m; j++) {
                if (request[i][j] > avail[j]) {
                    can_finish = false;
                    break;
                }
            }
            if (can_finish) {
                for (k = 0; k < m; k++) {
                    avail[k] += alloc[i][k];
                }
                finish[i] = true;
                changed = true;
                printf("Process %d can finish.\n", i);
            }
        }
    } while (changed);
    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:

Enter the number of processes and resources:
5 3

Enter allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter request matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Enter available matrix:
3 3 2

Process 1 can finish.
Process 3 can finish.
Process 4 can finish.
System is in a deadlock state.

Process 1 can finish.
Process 3 can finish.
Process 4 can finish.
System is in a deadlock state.

Process 1 can finish.
Process 3 can finish.
Process 4 can finish.
System is in a deadlock state.

Process 1 can finish.
Process 3 can finish.
Process 4 can finish.
System is in a deadlock state.

Process 1 can finish.
Process 3 can finish.
Process 4 can finish.
System is in a deadlock state.

Process 1 can finish.
Process 3 can finish.
Process 4 can finish.
System is in a deadlock state.

Program -6

Question: Write a C program to simulate the following contiguous memory allocation techniques

- a) Worst-fit
- b) Best-fit
- c) First-fit

Code:

```
#include <stdio.h>

struct Block {
    int size;
    int allocated;
};

struct File {
    int size;
    int block_no;
};

void resetBlocks(struct Block blocks[], int n) {
    for (int i = 0; i < n; i++) {
        blocks[i].allocated = 0;
    }
}

void firstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("\n\tMemory Management Scheme – First Fit\n");
    printf("File_no:\tFile_size\tBlock_no:\tBlock_size:\n");
    for (int i = 0; i < n_files; i++) {
        files[i].block_no = -1;
        for (int j = 0; j < n_blocks; j++) {
            if (!blocks[j].allocated && blocks[j].size >= files[i].size) {
                files[i].block_no = j + 1;
                blocks[j].allocated = 1;
            }
        }
    }
}
```

```

blocks[j].allocated = 1;

printf("%d\t%d\t%d\t%d\n", i + 1, files[i].size, j + 1, blocks[j].size);
break;

}

}

if (files[i].block_no == -1) {

printf("%d\t%d\t%d\t_\n", i + 1, files[i].size);

}

}

}

void bestFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {

printf("\n\tMemory Management Scheme – Best Fit\n");

printf("File_no:\tFile_size\tBlock_no:\tBlock_size:\n");

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

int bestIdx = -1;

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

if (!blocks[j].allocated && blocks[j].size >= files[i].size) {

if (bestIdx == -1 || blocks[j].size < blocks[bestIdx].size) {

bestIdx = j;

}

}

}

if (bestIdx != -1) {

blocks[bestIdx].allocated = 1;

files[i].block_no = bestIdx + 1;

printf("%d\t%d\t%d\t%d\n", i + 1, files[i].size, bestIdx + 1, blocks[bestIdx].size);

} else {

printf("%d\t%d\t_\n", i + 1, files[i].size);
}
}
}

```



```

printf("Memory Management Scheme\n");
printf("Enter the number of blocks: ");
scanf("%d", &n_blocks);
printf("Enter the number of files: ");
scanf("%d", &n_files);
struct Block blocks[n_blocks];
struct File files[n_files];
printf("\nEnter the size of the blocks:\n");
for (int i = 0; i < n_blocks; i++) {
    printf("Block %d: ", i + 1);
    scanf("%d", &blocks[i].size);
    blocks[i].allocated = 0;
}
printf("Enter the size of the files:\n");
for (int i = 0; i < n_files; i++) {
    printf("File %d: ", i + 1);
    scanf("%d", &files[i].size);
}
do {
    printf("\n1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    resetBlocks(blocks, n_blocks);
    switch (choice) {
        case 1:
            firstFit(blocks, n_blocks, files, n_files);
            break;
        case 2:

```

```

        bestFit(blocks, n_blocks, files, n_files);

        break;

    case 3:

        worstFit(blocks, n_blocks, files, n_files);

        break;

    case 4:

        printf("\nExiting...\n");

        break;

    default:

        printf("Invalid choice.\n");

    }

} while (choice != 4);

return 0;
}

```

OUTPUT:

```

Memory Management Scheme
Enter the number of blocks: 5
Enter the number of files: 4
Enter the size of the blocks:
Block 1: 100
Block 2: 500
Block 3: 300
Block 4: 200
Block 5: 600
Enter the size of the files:
File 1: 212
File 2: 417
File 3: 112
File 4: 426
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 1

Memory Management Scheme 0 First Fit
File_no: File_size Block_no: Block_size:
1 212 2 500
2 417 5 600
3 112 3 200
4 426 - -
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 2

Memory Management Scheme 0 Best Fit
File_no: File_size Block_no: Block_size:
1 212 4 500
2 417 2 500
3 112 3 200
4 426 5 600
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 3

Memory Management Scheme 0 Worst Fit
File_no: File_size Block_no: Block_size:
1 212 5 600
2 417 2 500
3 112 4 300
4 426 - -
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 4

Exiting...

```

Date: / /

Memory Allocation Techniques

```

1/1/15
#include <stdio.h>
struct Block {
    int size;
    int allocated;
};

struct File {
    int size;
    int block_no;
};

void resetBlocks(struct Block blocks[], int n_blocks);
void firstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files);
void bestFit(struct Block blocks[], int n_blocks, struct File files[], int n_files);
void worstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files);

int main() {
    printf("1. First Fit\n");
    printf("2. Best Fit\n");
    printf("3. Worst Fit\n");
    printf("Enter your choice: ");
    int choice;
    scanf("%d", &choice);

    if (choice == 1) {
        firstFit(blocks, n_blocks, files, n_files);
    } else if (choice == 2) {
        bestFit(blocks, n_blocks, files, n_files);
    } else if (choice == 3) {
        worstFit(blocks, n_blocks, files, n_files);
    } else {
        printf("Invalid choice\n");
    }
}

```

Date: / /

```

if (files[i].block_no == -1) {
    printf("%d %d %d %d %d %d %d\n", i+1, files[i].size);
}

void bestFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("1. Best Fit\n");
    printf("File no. File size Block no. Block size\n");
    for (int i=0; i<n_files; i++) {
        for (int j=0; j<n_blocks; j++) {
            if (!blocks[j].allocated && blocks[j].size >= files[i].size) {
                blocks[j].allocated = 1;
                blocks[j].block_no = i+1;
                files[i].block_no = j+1;
                files[i].size -= blocks[j].size;
                blocks[j].size -= files[i].size;
                break;
            }
        }
    }
}

void firstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("1. First Fit\n");
    for (int i=0; i<n_files; i++) {
        for (int j=0; j<n_blocks; j++) {
            if (!blocks[j].allocated && blocks[j].size >= files[i].size) {
                blocks[j].allocated = 1;
                blocks[j].block_no = i+1;
                files[i].block_no = j+1;
                files[i].size -= blocks[j].size;
                blocks[j].size -= files[i].size;
                break;
            }
        }
    }
}

void worstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("1. Worst Fit\n");
    for (int i=0; i<n_files; i++) {
        for (int j=0; j<n_blocks; j++) {
            if (!blocks[j].allocated && blocks[j].size >= files[i].size) {
                blocks[j].allocated = 1;
                blocks[j].block_no = i+1;
                files[i].block_no = j+1;
                files[i].size -= blocks[j].size;
                blocks[j].size -= files[i].size;
                break;
            }
        }
    }
}

```

Date: / /

```

void worstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("1. Worst Fit\n");
    printf("File no. File size Block no. Block size\n");
    for (int i=0; i<n_files; i++) {
        for (int j=0; j<n_blocks; j++) {
            if (!blocks[j].allocated && blocks[j].size >= files[i].size) {
                blocks[j].allocated = 1;
                blocks[j].block_no = i+1;
                files[i].block_no = j+1;
                files[i].size -= blocks[j].size;
                blocks[j].size -= files[i].size;
                break;
            }
        }
    }
}

int main() {
    int n_blocks, n_files, choice;
    ...
}

```

Date: / /

```

printf("Memory Management Scheme\n");
printf("Enter the number of blocks: ");
scanf("%d", &n_blocks);
printf("Enter the number of files: ");
scanf("%d", &n_files);
struct Block blocks[n_blocks];
struct File files[n_files];
printf("Enter the size of the blocks: ");
for (int i=0; i<n_blocks; i++) {
    printf("%d ", blocks[i].size);
}
printf("\n");
do {
    printf("1. First Fit\n");
    printf("2. Best Fit\n");
    printf("3. Worst Fit\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    int choice;
    scanf("%d", &choice);

    switch(choice) {
        case 1:
            firstFit(blocks, n_blocks, files, n_files);
            break;
        case 2:
            bestFit(blocks, n_blocks, files, n_files);
            break;
        case 3:
            worstFit(blocks, n_blocks, files, n_files);
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice\n");
    }
} while(choice != 4);

```

Date: / /

Output:

```

Memory Management Scheme
Enter the number of blocks: 5
Enter the number of files: 4
Enter the size of the blocks:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600
Enter the size of the files:
File 1: 212
File 2: 417
File 3: 112
File 4: 426
Enter your choice: 1
Memory Management Scheme - First Fit
File no. File size Block no. Block size
1 212 2 500
2 417 5 600
3 112 3 200
4 426 - -

```

Output:

```

Memory Management Scheme - Best Fit
File no. File size Block no. Block size
1 212 1 500
2 417 2 600
3 112 3 200
4 426 4 200

```

Output:

```

Memory Management Scheme - Worst Fit
File no. File size Block no. Block size
1 212 5 600
2 417 2 500
3 112 1 200
4 426 - -

```

Date: / /

Memory Management Scheme - Best Fit

File no.	File size	Block no.	Block size
1	212	1	500
2	417	2	600
3	112	3	200
4	426	4	200

Memory Management Scheme - Worst Fit

File no.	File size	Block no.	Block size
1	212	5	600
2	417	2	500
3	112	1	200
4	426	-	-

Memory Management Scheme - First Fit

File no.	File size	Block no.	Block size
1	212	2	500
2	417	5	600
3	112	3	200
4	426	-	-

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 1
Exiting..

Question:

Code:

a) FIFO

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

int search(int key, int frame[], int frames) {
    for (int i = 0; i < frames; i++) {
        if (frame[i] == key)
            return 1;
    }
    return 0;
}

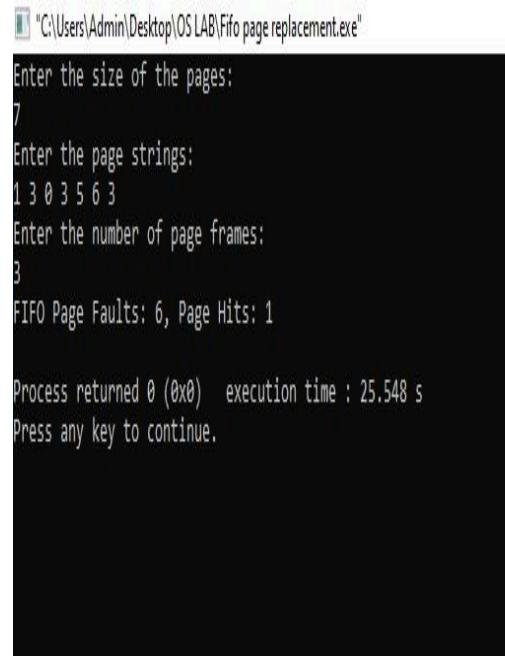
void fifo(int pages[], int n, int frames) {
    int *frame = (int *)malloc(frames * sizeof(int));
    int page_faults = 0, page_hits = 0, index = 0;
    for (int i = 0; i < frames; i++)
        frame[i] = -1;
    for (int i = 0; i < n; i++) {
        if (!search(pages[i], frame, frames)) {
            frame[index] = pages[i];
            index = (index + 1) % frames;
            page_faults++;
        } else {
            page_hits++;
        }
    }
    printf("FIFO Page Faults: %d, Page Hits: %d\n", page_faults, page_hits);
    free(frame);
}
```

```
}
```

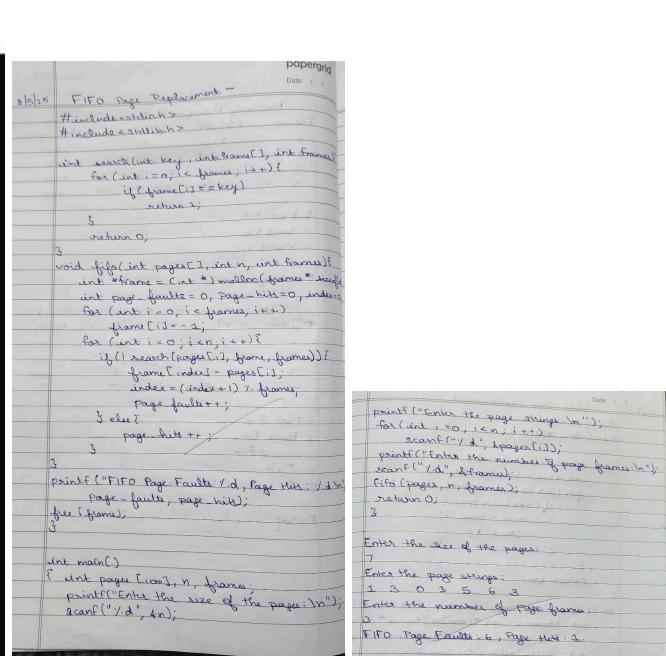
```
int pages[100], n, frames;  
printf("Enter the size of the pages:\n");  
scanf("%d", &n);  
printf("Enter the page strings:\n");  
for (int i = 0; i < n; i++)  
    scanf("%d", &pages[i]);  
printf("Enter the number of page frames:\n");  
scanf("%d", &frames);  
fifo(pages, n, frames);  
return 0;
```

```
}
```

OUTPUT:



```
"C:\Users\Admin\Desktop\OS LAB\Fifo page replacement.exe"  
Enter the size of the pages:  
7  
Enter the page strings:  
1 3 0 3 5 6 3  
Enter the number of page frames:  
3  
FIFO Page Faults: 6, Page Hits: 1  
  
Process returned 0 (0x0) execution time : 25.548 s  
Press any key to continue.
```



```
FIFO Page Replacement -  
#include <cs50.h>  
#include <math.h>  
  
int search(int key, int frame[], int frames)  
{  
    for (int i = 0; i < frames; i++) {  
        if (frame[i] == key)  
            return i;  
    }  
    return -1;  
}  
  
void fifo(int pages[], int n, int frames)  
{  
    int *frame = (int *) malloc(frames * sizeof(int));  
    int page_faults = 0, page_hits = 0, index;  
    for (int i = 0; i < n; i++) {  
        frame[i % frames] = pages[i];  
        if (search(pages[i], frame, frames) == -1) {  
            page_faults++;  
            frame[index % frames] = pages[i];  
            index = (index + 1) % frames;  
            page_faults++;  
        } else {  
            page_hits++;  
        }  
    }  
    printf("FIFO Page Faults: %d, Page Hits: %d\n", page_faults, page_hits);  
    free(frame);  
}  
  
int main()  
{  
    int pages[100], n, frames;  
    printf("Enter the size of the pages:\n");  
    scanf("%d", &n);  
}
```

b) LRU

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

int findLRU(int time[], int frames) {
    int min = time[0], pos = 0;
    for (int i = 1; i < frames; i++) {
        if (time[i] < min) {
            min = time[i];
            pos = i;
        }
    }
    return pos;
}

void lru(int pages[], int n, int frames) {
    int *frame = (int *)malloc(frames * sizeof(int));
    int *recent = (int *)malloc(frames * sizeof(int));
    int page_faults = 0, page_hits = 0;
    int time = 0;
    for (int i = 0; i < frames; i++) {
        frame[i] = -1;
        recent[i] = -1;
    }
    for (int i = 0; i < n; i++) {
        int page = pages[i];
        int hit = 0;
        for (int j = 0; j < frames; j++) {
            if (frame[j] == page) {
                hit = 1;
                break;
            }
        }
        if (hit == 0) {
            int min_recent = recent[0];
            int min_index = 0;
            for (int k = 1; k < frames; k++) {
                if (recent[k] < min_recent) {
                    min_recent = recent[k];
                    min_index = k;
                }
            }
            frame[min_index] = page;
            recent[min_index] = time;
            page_faults++;
        } else {
            recent[page] = time;
        }
        time++;
    }
}
```

```

page_hits++;
recent[j] = time;
break;
}
}

if (!hit) {
    int pos = -1;
    for (int j = 0; j < frames; j++) {
        if (frame[j] == -1) {
            pos = j;
            break;
        }
    }
    if (pos == -1)
        pos = findLRU(recent, frames);
    frame[pos] = page;
    recent[pos] = time;
    page_faults++;
}
time++;
}

printf("LRU Page Faults: %d, Page Hits: %d\n", page_faults, page_hits);
free(frame);
free(recent);
}

int main() {
    int pages[100], n, frames;

```

```

printf("Enter the size of the pages:\n");
scanf("%d", &n);

printf("Enter the page strings:\n");
for (int i = 0; i < n; i++)
    scanf("%d", &pages[i]);

printf("Enter the number of page frames:\n");
scanf("%d", &frames);
lru(pages, n, frames);
return 0;
}

```

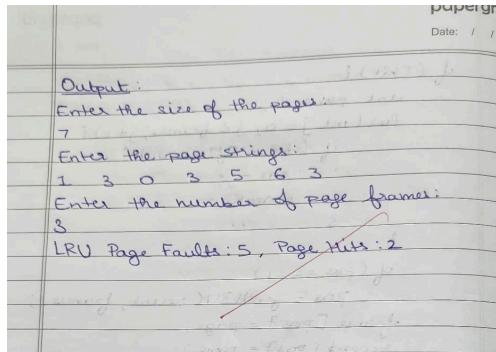
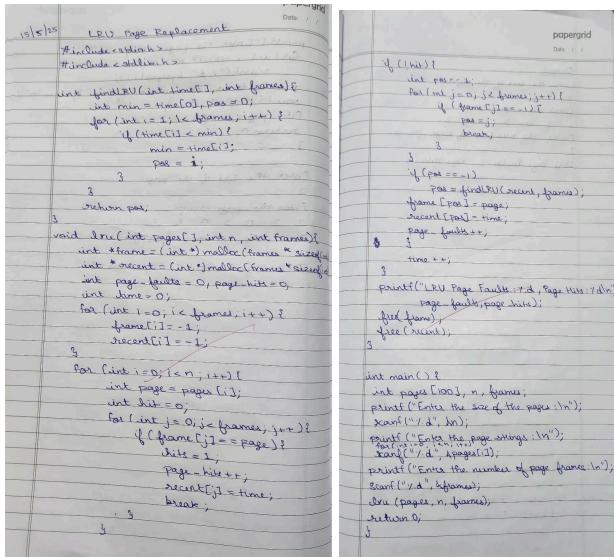
OUTPUT:

```

C:\Users\Admin\Desktop\OS LAB\LRU page replacement.exe
Enter the size of the pages:
7
Enter the page strings:
1 3 0 3 5 6 3
Enter the number of page frames:
3
LRU Page Faults: 5, Page Hits: 2

Process returned 0 (0x0) execution time : 11.216 s
Press any key to continue.

```



c)Optimal

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

int search(int key, int frame[], int frames) {
    for (int i = 0; i < frames; i++) {
        if (frame[i] == key) {
            return 1;
        }
    }
    return 0;
}

int findOptimal(int pages[], int frame[], int frames, int index, int n) {
    int farthest = index, pos = -1, i, j;
    for (i = 0; i < frames; i++) {
        for (j = index; j < n; j++) {
            if (frame[i] == pages[j]) {
                if (j > farthest) {
                    farthest = j;
                    pos = i;
                }
            }
            break;
        }
        if (j == n) {
            return i;
        }
    }
    return (pos == -1) ? 0 : pos;
}
```

```

}

void optimal(int pages[], int n, int frames) {
    int *frame = (int *)malloc(frames * sizeof(int));
    int page_faults = 0, page_hits = 0;
    for (int i = 0; i < frames; i++) {
        frame[i] = -1;
    }
    for (int i = 0; i < n; i++) {
        int page = pages[i];
        if (search(page, frame, frames)) {
            page_hits++;
        } else {
            int pos = -1;
            for (int j = 0; j < frames; j++) {
                if (frame[j] == -1) {
                    pos = j;
                    break;
                }
            }
            if (pos == -1) {
                pos = findOptimal(pages, frame, frames, i + 1, n);
            }
            frame[pos] = page;
            page_faults++;
        }
    }
    printf("Optimal Page Faults: %d, Page Hits: %d\n", page_faults, page_hits);
    free(frame);
}

```

```
}

int main() {
    int pages[100], n, frames;
    printf("Enter the size of the pages:\n");
    scanf("%d", &n);
    printf("Enter the page strings:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &pages[i]);
    }
    printf("Enter the number of page frames:\n");
    scanf("%d", &frames);
    optimal(pages, n, frames);
    return 0;
}
```

OUTPUT:

C:\Users\Admin\Desktop\OS LAB\optimal page replacement.exe"

Enter the size of the pages:

7

Enter the page strings:

1 3 0 3 5 6 3

Enter the number of page frames:

3

Optimal Page Faults: 5, Page Hits: 2

Process returned 0 (0x0) execution time : 13.863 s

Press any key to continue.

```

Optimal Page Replacement
# include <conio.h>
# include <stdlib.h>
int search(int page, int frame[], int frames);
for (int i=0; i<frames; i++) {
    if (frame[i]==page) {
        return i;
    }
}
int findOptimal(int page[], int frame[], int frames,
    int index, int n) {
    int firstset = index, pos=-1, j;
    for (j=0; j<frames; j++) {
        if (frame[j]==index) {
            if (frame[j]>firstset) {
                firstset = j;
            }
            pos = j;
        }
    }
    if (pos == -1) {
        pos = findOptimal (page, frame, index+1, n);
    }
    return pos;
}
int main() {
    int pages[7], n, frames;
    printf("Enter the size of pages\n");
    scanf("%d", &n);
    printf("Enter the page strings\n");
    scanf("%s", pages);
    printf("Enter the number of page frames\n");
    scanf("%d", &frames);
    int page_faults = 0, page_hits = 0;
    for (int i=0; i<n; i++) {
        int page = pages[i];
        if (search(page, frame, frames) == -1) {
            page_faults++;
            frame[findOptimal(pages, frame, i, frames)] = page;
            page_hits++;
        }
    }
    printf("Optimal Page Faults: %d, Page Hits: %d\n",
        page_faults, page_hits);
    getch();
}

```

```

Optimal Page Replacement
# include <conio.h>
# include <stdlib.h>
int search(int page[], int frame[], int frames);
for (int i=0; i<frames; i++) {
    if (frame[i]==page) {
        return i;
    }
}
int findOptimal(int page[], int frame[], int frames,
    int index, int n) {
    int firstset = index, pos=-1, j;
    for (j=0; j<frames; j++) {
        if (frame[j]==index) {
            if (frame[j]>firstset) {
                firstset = j;
            }
            pos = j;
        }
    }
    if (pos == -1) {
        pos = findOptimal (page, frame, index+1, n);
    }
    return pos;
}
int main() {
    int pages[7], n, frames;
    printf("Enter the size of pages\n");
    scanf("%d", &n);
    printf("Enter the page strings\n");
    scanf("%s", pages);
    printf("Enter the number of page frames\n");
    scanf("%d", &frames);
    int page_faults = 0, page_hits = 0;
    for (int i=0; i<n; i++) {
        int page = pages[i];
        if (search(page, frame, frames) == -1) {
            page_faults++;
            frame[findOptimal(pages, frame, i, frames)] = page;
            page_hits++;
        }
    }
    printf("Optimal Page Faults: %d, Page Hits: %d\n",
        page_faults, page_hits);
    getch();
}

```

```

Date: / /
point("Enter the number of page frames\n");
scanf("%d", &frames);
optional(pages, n, frames);
return 0;
}

Output:
Enter the size of pages:
7
Enter the number of page strings.
1 3 0 3 5 6 3
Enter the number of page frames:
3
Optimal Page Faults: 5, Page Hits: 2

```

15/5/2024