

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.



LAB REPORT

on

OPERATING SYSTEMS

Submitted by

Vikas P R (1WA23CS042)

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 Vikas P R (1WA23CS042), 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.

Prof. Sheetal V A
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.	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)	5-18
2.	Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time. → Priority (pre-emptive & Non-pre-emptive) →Round Robin (Experiment with different quantum sizes for RR algorithm)	19-35
3.	Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.	36-41
4.	Write a C program to simulate Real-Time CPU Scheduling algorithms: a) Rate- Monotonic b) Earliest-deadline First c) Proportional scheduling	42-45
5.	Write a C program to simulate producer-consumer problem using semaphores	46-50
6.	Write a C program to simulate the concept of Dining Philosophers problem.	51-55
7.	Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.	56-60
8.	Write a C program to simulate deadlock detection	61-68
9.	Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit b) Best-fit c) First-fit	69-77

10.	<p>a) Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) Optimal</p>	78-87
-----	--	-------

Course Outcomes

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

Program -1

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:

=>FCFS:

```
#include <stdio.h>
struct Process {
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
    int rt;
};
void calculateFCFS(struct Process proc[], int n) {
    int time = 0;
    for (int i = 0; i < n; i++) {
        if (time < proc[i].at) {
            time = proc[i].at;
        }
        proc[i].ct = time + proc[i].bt;
        proc[i].tat = proc[i].ct - proc[i].at;
        proc[i].wt = proc[i].tat - proc[i].bt;
        proc[i].rt = time - proc[i].at;
        time = proc[i].ct;
    }
}
int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
```

```

struct Process proc[n];
printf("Enter Arrival Time and Burst Time for each process:\n");
for (int i = 0; i < n; i++) {
    printf("P%d Arrival Time: ", i + 1);
    scanf("%d", &proc[i].at);
    printf("P%d Burst Time: ", i + 1);
    scanf("%d", &proc[i].bt);
}
calculateFCFS(proc, n);
printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\tRT\n");
for (int i = 0; i < n; i++) {
    printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, proc[i].at, proc[i].bt,
proc[i].ct, proc[i].tat, proc[i].wt, proc[i].rt);
}

// Calculate average WT and TAT
float totalWT = 0, totalTAT = 0;
for (int i = 0; i < n; i++) {
    totalWT += proc[i].wt;
    totalTAT += proc[i].tat;
}
printf("\nAverage Waiting Time: %.2f", totalWT / n);
printf("\nAverage Turnaround Time: %.2f\n", totalTAT / n);
return 0;
}

```

Result:

```
Enter number of processes: 4
Enter Arrival Time and Burst Time for each process:
P1 Arrival Time: 0
P1 Burst Time: 7
P2 Arrival Time: 0
P2 Burst Time: 3
P3 Arrival Time: 0
P3 Burst Time: 4
P4 Arrival Time: 0
P4 Burst Time: 6

Process AT      BT      CT      TAT      WT      RT
P1    0       7     20     20     13     13
P2    0       3      3      3      0      0
P3    0       4      7      7      3      3
P4    0       6     13     13      7      7

Average Waiting Time: 5.75
Average Turnaround Time: 10.75

Process returned 0 (0x0)  execution time : 53.889 s
Press any key to continue.
```

Lab-01

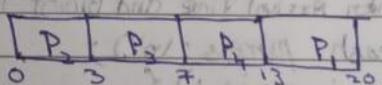
Write a c program to stimulate non pre-emptive CPU scheduling algorithm, to find turnaround time, waiting time

FCFS

Process	AT	BT	CT	TAT	WT	RT
P ₁	0	7	7	7	0	0
P ₂	0	3	10	10	7	7
P ₃	0	4	14	14	10	10
P ₄	0	6	20	20	14	14

Average Waiting Time: 7.75

Average Turnaround Time: 12.75



```

#include <stdio.h>
struct Process {
    int at; // arrivalTime
    int bt; // burstTime
    int ct; // completionTime
    int tat; // TurnAroundTime
    int wt; // WaitingTime
    int rt; // ResponseTime
};

void calculateFCFS (struct Process proc[], int n) {
    int time = 0;
    for (int i = 0; i < n; i++) {
        if (time <= proc[i].at) {
            time = proc[i].at;
        }
        time += proc[i].bt;
        proc[i].ct = time;
        proc[i].tat = proc[i].ct - proc[i].at;
        proc[i].wt = proc[i].tat - proc[i].bt;
        proc[i].rt = 0;
    }
}
  
```

<pre> time = proc[i].at; // i=1 > proc[i].ct = time + proc[i].bt; proc[i].tat = proc[i].ct - proc[i].at; proc[i].wt = proc[i].tat - proc[i].bt; proc[i].rt = time - proc[i].at; time = proc[i].ct; > F F F F F > 1 2 3 4 5 int main () { int n; printf ("Enter number of processes: "); scanf ("%d", &n); struct Process proc[n]; printf ("Enter Arrival Time, and Burst Time for each process: "); for (int i=0; i<n; i++) { printf ("P%d's Arrival Time: ", i+1); scanf ("%d", &proc[i].at); printf ("P%d's Burst Time: ", i+1); scanf ("%d", &proc[i].bt); } calculateFCFS(proc, n); printf ("\nProcess\tAT\tBT\tCT\tWT\tAT"); for (int i=0; i<n; i++) { printf ("\nP%d\t%d\t%d\t%d\t%d\t%d", i+1, proc[i].at, proc[i].bt, proc[i].ct, proc[i].wt, proc[i].tat); } </pre>	<p>11 Calculate avg WT & TAT</p> <pre> float totalWT=0, totalTAT=0; for (int i=0; i<n; i++) { totalWT += proc[i].wt; totalTAT += proc[i].tat; } printf ("\nAverage Waiting Time : %f", totalWT/n); printf ("\nAverage Turnaround Time : %f", totalTAT/n); return 0; </pre>
---	---

```

=>SJF(preemptive):
#include <stdio.h>
#define MAX 10
typedef struct {
    int pid, at, bt, rt, wt, tat, completed;
} Process;

void sjf_preemptive(Process p[], int n) {
    int time = 0, completed = 0, shortest = -1, min_bt = 9999;

    while (completed < n) {
        shortest = -1;
        min_bt = 9999;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && p[i].rt > 0 && p[i].rt < min_bt) {
                min_bt = p[i].rt;
                shortest = i;
            }
        }

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

        p[shortest].rt--;
        time++;

        if (p[shortest].rt == 0) {
            p[shortest].completed = 1;
            completed++;
            p[shortest].tat = time - p[shortest].at;
            p[shortest].wt = p[shortest].tat - p[shortest].bt;
        }
    }
}

```

```

    }

int main() {
    Process p[MAX];
    int n;

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

    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter arrival time and burst time for process %d: ", p[i].pid);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].rt = p[i].bt;
        p[i].completed = 0;
    }

    sjf_preemptive(p, n);

    printf("\nPID\tAT\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].at, p[i].bt, p[i].wt, p[i].tat);

    return 0;
}

```

Result:

```
C:\Users\Admin\Desktop\sjf preemptive.exe"
Enter number of processes: 4
Enter arrival time and burst time for process 1: 0
8
Enter arrival time and burst time for process 2: 1
4
Enter arrival time and burst time for process 3: 2
9
Enter arrival time and burst time for process 4: 3
5

PID      AT       BT       WT       TAT
1        0        8        9        17
2        1        4        0        4
3        2        9       15       24
4        3        5        2        7

Process returned 0 (0x0)   execution time : 35.485 s
Press any key to continue.
```

SJF

pre-emptive

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

```
# define MAX 10
```

```
typedef struct
```

```
int pid, at, bt, st, wt, tat, completed ; } Process ;
```

```
void sjf preemptive (Process p[], int n) {
```

```
int time = 0, completed = 0, shortest = -1, min_bt = 9999 ;
```

```
while (completed < n) {
```

```
shortest = -1 ;
```

```
min_bt = 9999 ;
```

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

```
if (p[i].at <= time && p[i].bt > 0 && p[i].at < min_bt)
```

```
min_bt = p[i].at
```

```
shortest = i ;
```

```
}
```

```
}
```

```
if (shortest == -1) {
```

```
time++ ;
```

```
continue ;
```

```
}
```

```
p[shortest].at-- ;
```

```
time++ ;
```

```
if (p[shortest].at == 0) {
```

```
p[shortest].completed = 1 ;
```

```
completed++ ;
```

```
p[shortest].tat = time - p[shortest].at ;
```

```
p[shortest].wt = p[shortest].tat - p[shortest].bt ;
```

```
>
```

```
>
```

```
>
```

<pre> int main() { Process p[5]; int n; printf("Enter number of processes: "); scanf("%d", &n); for (int i=0; i<n; i++) { p[i].pid = i+1; printf("Enter AT & BT for process %d: ", p[i].pid); scanf("%d%d", &p[i].at, &p[i].bt); p[i].st = p[i].bt; p[i].completed = 0; } SJF_nonpreemptive(p, n); printf("nPID AT BT WT TAT(n)"); for (int i=0; i<n; i++) printf("%d %d %d %d %d\n", p[i].pid, p[i].at, p[i].bt, p[i].wt, p[i].tat); } </pre>	<p>O/P</p> <p>Enter the number of processes: 4</p> <p>Enter the arrival time and burst time process 1: 0 8</p> <p>Enter the arrival time and burst time process 2: 1 4</p> <p>Enter the arrival time and burst time process 3: 2 9</p> <p>Enter the arrival time and burst time process 4: 3 5</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th>P</th><th>P₂</th><th>P₄</th><th>P₁</th><th>P₃</th></tr> <tr> <td>0</td><td>1</td><td>10</td><td>17</td><td>20</td></tr> </table> <p>WAT = waiting time + burst time TAAT = turnaround time = completed time - arrival time WT = waiting time = TAAT - burst time TAT = completed time - arrival time</p>	P	P ₂	P ₄	P ₁	P ₃	0	1	10	17	20
P	P ₂	P ₄	P ₁	P ₃							
0	1	10	17	20							

=>SJF(Non-preemptive):

```

#include <stdio.h>
#include <limits.h>
struct Process {
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
    int rt;
    int pid;
};
void calculateSJF(struct Process proc[], int n) {
    int time = 0;
    int completed = 0;
    int min_index;
    int is_completed[n];
    for (int i = 0; i < n; i++) {
        is_completed[i] = 0;
    }
}

```

```

}

while (completed < n) {
    min_index = -1;
    int min_bt = INT_MAX;
    for (int i = 0; i < n; i++) {
        if (proc[i].at <= time && !is_completed[i] && proc[i].bt < min_bt) {
            min_bt = proc[i].bt;
            min_index = i;
        }
    }
    if (min_index == -1) {
        time++;
    } else {
        proc[min_index].ct = time + proc[min_index].bt;
        proc[min_index].tat = proc[min_index].ct - proc[min_index].at;
        proc[min_index].wt = proc[min_index].tat - proc[min_index].bt;
        proc[min_index].rt = time - proc[min_index].at;
        time = proc[min_index].ct;
        is_completed[min_index] = 1;
        completed++;
    }
}
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process proc[n];
    printf("Enter Arrival Time and Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        proc[i].pid = i + 1;
        printf("P%d Arrival Time: ", i + 1);
        scanf("%d", &proc[i].at);
        printf("P%d Burst Time: ", i + 1);
        scanf("%d", &proc[i].bt);
    }
    calculateSJF(proc, n);
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\tRT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", proc[i].pid, proc[i].at, proc[i].bt, proc[i].ct,
        proc[i].tat, proc[i].wt, proc[i].rt);
    }
}

```

```

    // Calculate average WT and TAT
    float totalWT = 0, totalTAT = 0;
    for (int i = 0; i < n; i++) {
        totalWT += proc[i].wt;
        totalTAT += proc[i].tat;
    }
    printf("\nAverage Waiting Time: %.2f", totalWT / n);
    printf("\nAverage Turnaround Time: %.2f\n", totalTAT / n);
    return 0;
}

```

Result:

```

Enter number of processes: 4
Enter Arrival Time and Burst Time for each process:
P1 Arrival Time: 0
P1 Burst Time: 7
P2 Arrival Time: 0
P2 Burst Time: 3
P3 Arrival Time: 0
P3 Burst Time: 4
P4 Arrival Time: 0
P4 Burst Time: 6

Process AT      BT      CT      TAT      WT      RT
P1      0       7       7       7       0       0
P2      0       3      10      10      7       7
P3      0       4      14      14     10      10
P4      0       6      20      20     14      14

Average Waiting Time: 7.75
Average Turnaround Time: 12.75

Process returned 0 (0x0)  execution time : 15.676 s
Press any key to continue.

```

DATE: _____

PAGE NO.: _____
DATE: _____

Shortest Job First

```

non pre-emptive
#include <stdio.h>
#include <limits.h>

struct Process {
    int at; // Arrival Time
    int bt; // Burst Time
    int ct; // Completion Time
    int tat; // Turnaround Time
    int wt; // Waiting Time
    int rt; // Response Time
    int pid;
};

void calculateSJF(struct Process proc[], int n) {
    int time = 0;
    int completed = 0;
    int min_index;
    int is_completed[n];
    for (int i=0; i<n; i++) {
        is_completed[i] = 0;
    }
    while (completed < n) {
        min_index = -1;
        int min_bt = INT_MAX;
        for (int i=0; i<n; i++) {
            if (proc[i].at <= time && !is_completed[i] && proc[i].bt < min_bt)
                min_bt = proc[i].bt;
            if (min_bt == proc[i].bt) {
                min_index = i;
            }
        }
        if (min_index == -1) {
            time++;
            continue;
        }
        proc[min_index].ct = time + proc[min_index].bt;
        proc[min_index].tat = proc[min_index].ct - proc[min_index].at;
        proc[min_index].wt = proc[min_index].tat - proc[min_index].bt;
        proc[min_index].rt = time - proc[min_index].at;
        time = proc[min_index].ct;
        if (completed < n) {
            completed++;
        }
    }
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process proc[n];
    printf("Enter AT & BT:\n");
    for (int i=0; i<n; i++) {
        proc[i].pid = i+1;
        printf("P%d Arrival Time: ", i+1);
        scanf("%d", &proc[i].at);
        printf("P%d Burst Time: ", i+1);
        scanf("%d", &proc[i].bt);
    }
    calculateSJF(proc, n);
    printf("\nIn Process AT + BT + CT + TA + WT + RT\n");
    for (int i=0; i<n; i++) {
        printf("P%d At %d | T %d | C %d | T %d | W %d | R %d\n",
               proc[i].pid, proc[i].at, proc[i].ct, proc[i].tat,
               proc[i].wt, proc[i].rt);
    }
}

```

PAGE NO.:
DATE:

II Calculate average WT and TAT

```

float totalWT=0, totalTAT=0;
for(int i=0; i<n; i++){
    totalWT+=proc[i].wt;
    totalTAT+=proc[i].tat;
}
printf("\n Average Waiting Time: %f", totalWT/n);
printf("\n Average Turnaround Time: %f\n", totalTAT/n);
return 0;
>

```

O/P

Enter number of processes: 4

Enter AT and BT for each:

P1 AT = 0

P1 BT = 7

P2 AT = 0

P2 BT = 3

P3 AT = 0

P3 BT = 4

P4 AT = 0

P4 BT = 6

Process	AT	BT	TAT	WT	RJ
P1	0	7	20	20	13
P2	0	3	3	0	0
P3	0	4	7	7	3
P4	0	6	13	7	7

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)
- Round Robin (Experiment with different quantum sizes for RR algorithm)

=>Priority(preemptive):

```
#include <stdio.h>
#include <limits.h>
struct Process {
    int pid, at, bt, pr, ct, wt, tat, rt, remaining;
};

void findPreemptivePriorityScheduling(struct Process p[], int n) {
    int completed = 0, time = 0, min_idx = -1;
    float totalWT = 0, totalTAT = 0;
    for (int i = 0; i < n; i++) {
        p[i].remaining = p[i].bt;
        p[i].rt = -1;
    }
    while (completed != n) {
        int min_priority = INT_MAX;
        min_idx = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && p[i].remaining > 0 && p[i].pr < min_priority) {
                min_priority = p[i].pr;
                min_idx = i;
            }
        }
        if (min_idx == -1) {
            time++;
            continue;
        }
        if (p[min_idx].rt == -1) {
```

```

    p[min_idx].rt = time - p[min_idx].at;
}

p[min_idx].remaining--;
time++;

if (p[min_idx].remaining == 0) {
    completed++;
    p[min_idx].ct = time;
    p[min_idx].tat = p[min_idx].ct - p[min_idx].at;
    p[min_idx].wt = p[min_idx].tat - p[min_idx].bt;
    totalWT += p[min_idx].wt;
    totalTAT += p[min_idx].tat;
}
}

printf("PID\tAT\tBT\tPR\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].pr, p[i].ct, p[i].tat, p[i].wt, p[i].rt);
}
printf("\nAverage Turnaround Time: %.2f\n", totalTAT / n);
printf("Average Waiting Time: %.2f\n", totalWT / n);
}

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

```

```
    findPreemptivePriorityScheduling(p, n);
    return 0;
}
```

Result:

```
Enter the number of processes: 5
Enter Arrival Time, Burst Time, and Priority for each process:
Process 1: 0
10
3
Process 2: 0
1
1
Process 3: 0
2
4
Process 4: 0
1
5
Process 5: 0
5
2
PID AT BT PR CT TAT WT RT
1 0 10 3 16 16 6 6
2 0 1 1 1 1 0 0
3 0 2 4 18 18 16 16
4 0 1 5 19 19 18 18
5 0 5 2 6 6 1 1

Average Turnaround Time: 12.00
Average Waiting Time: 8.20
```

Priority pre emptive

```

#include <stdio.h>
#include <limits.h>
struct Process {
    int pid, at, bt, pr, ct, wt, tat, st, remaining;
};

void findPriorityScheduling (struct Process p[], int n) {
    int completed = 0, time = 0, min_idx = -1;
    float totalWT = 0, totalTAT = 0;
    for (int i=0 ; i<n ; i++) {
        p[i].remaining = p[i].bt;
        p[i].st = -1;
    }
    while (completed != n) {
        int min_priority = INT_MAX;
        min_idx = -1;
        for (int i=0 ; i<n ; i++) {
            if ((p[i].at <= time) & (p[i].remaining > 0) & (p[i].pr
                < min_priority)) {
                min_priority = p[i].pr;
                min_idx = i;
            }
        }
        if (min_idx == -1) {
            time++;
            continue;
        }
        p[min_idx].st = time - p[min_idx].at;
        p[min_idx].remaining -= 1;
        time++;
    }
}

```

<pre> if (p[min_idx].remaining == 0) { completed++; p[min_idx].st = time; p[min_idx].tat = p[min_idx].st - p[min_idx].bt; p[min_idx].wt = p[min_idx].st - p[min_idx].bt; totalWT += p[min_idx].wt; totalTAT += p[min_idx].tat; } } void (*P)(int PID, int AT, int BT, int PR, int CT, int TAT, int WT, int RT); for (int i = 0; i < n; i++) { P(&p[i].pid, p[i].at, p[i].bt, p[i].pr, p[i].ct, &p[i].tat, &p[i].wt, &p[i].rt); } printf("Average Turnaround Time: %.2f\n", totalTAT / n); printf("Average Waiting Time: %.2f\n", totalWT / n); </pre>	<p style="text-align: center;">Non Preemptive Priority Scheduling (n)</p> <p>Enter the number of processes: 7</p> <p>Enter Arrival Time, Burst Time, and Priority for each process:</p> <table border="1"> <tbody> <tr><td>Process 1: 0 8 3</td></tr> <tr><td>Process 2: 1 2 4</td></tr> <tr><td>Process 3: 3 4 4</td></tr> <tr><td>Process 4: 4 1 5</td></tr> <tr><td>Process 5: 5 6 2</td></tr> <tr><td>Process 6: 6 5 6</td></tr> <tr><td>Process 7: 7 1 1</td></tr> </tbody> </table> <p>PID AT BT PR CT TAT WT RT</p> <table border="1"> <tbody> <tr><td>1 0 8 3 15 15 7 0</td></tr> <tr><td>2 1 2 4 17 16 14 1</td></tr> <tr><td>3 3 4 4 21 18 14 1</td></tr> <tr><td>4 4 1 5 22 18 17 1</td></tr> <tr><td>5 5 6 2 18 7 1 0</td></tr> <tr><td>6 6 5 6 27 21 16 16</td></tr> <tr><td>7 7 1 1 31 1 0 6</td></tr> </tbody> </table> <p>Average Turnaround Time: 13.57 Average Waiting Time: 9.86</p>	Process 1: 0 8 3	Process 2: 1 2 4	Process 3: 3 4 4	Process 4: 4 1 5	Process 5: 5 6 2	Process 6: 6 5 6	Process 7: 7 1 1	1 0 8 3 15 15 7 0	2 1 2 4 17 16 14 1	3 3 4 4 21 18 14 1	4 4 1 5 22 18 17 1	5 5 6 2 18 7 1 0	6 6 5 6 27 21 16 16	7 7 1 1 31 1 0 6
Process 1: 0 8 3															
Process 2: 1 2 4															
Process 3: 3 4 4															
Process 4: 4 1 5															
Process 5: 5 6 2															
Process 6: 6 5 6															
Process 7: 7 1 1															
1 0 8 3 15 15 7 0															
2 1 2 4 17 16 14 1															
3 3 4 4 21 18 14 1															
4 4 1 5 22 18 17 1															
5 5 6 2 18 7 1 0															
6 6 5 6 27 21 16 16															
7 7 1 1 31 1 0 6															

=>Priority(non-preemptive):

#include <stdio.h>

```

struct Process {
    int pid, at, bt, pr, ct, wt, tat, rt;
    int isCompleted; // Flag to check if process is completed
};

```

```

void sortByArrival(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 findPriorityScheduling(struct Process p[], int n) {
    sortByArrival(p, n);
    int time = 0, completed = 0;
    float totalWT = 0, totalTAT = 0;

    while (completed < n) {
        int idx = -1, highestPriority = 9999;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && p[i].isCompleted == 0) {
                if (p[i].pr < highestPriority) {
                    highestPriority = p[i].pr;
                    idx = i;
                }
            }
        }

        if (idx == -1) {
            time++; // CPU idle
        } else {
            p[idx].rt = time - p[idx].at;
            time += p[idx].bt;
            p[idx].ct = time;
            p[idx].tat = p[idx].ct - p[idx].at;
            p[idx].wt = p[idx].tat - p[idx].bt;
            p[idx].isCompleted = 1;

            totalWT += p[idx].wt;
        }
    }
}

```

```

        totalTAT += p[idx].tat;
        completed++;
    }
}

printf("PID\tAT\tBT\tPR\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].pr, p[i].ct, p[i].tat, p[i].wt, p[i].rt);
}

printf("\nAverage Turnaround Time: %.2f\n", totalTAT / n);
printf("Average Waiting Time: %.2f\n", totalWT / n);
}

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

    printf("Enter Arrival Time, Burst Time, and Priority for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Process %d: ", i + 1);
        scanf("%d %d %d", &p[i].at, &p[i].bt, &p[i].pr);
        p[i].isCompleted = 0;
    }

    findPriorityScheduling(p, n);
    return 0;
}

```

Result:

```
Enter the number of processes: 7
Enter Arrival Time, Burst Time, and Priority for each process:
Process 1: 0
8
3
Process 2: 1
2
4
Process 3: 3
4
4
Process 4: 4
1
5
Process 5: 5
6
2
Process 6: 6
5
6
Process 7: 7
1
1
PID AT BT PR CT TAT WT RT
1 0 8 3 15 15 7 0
2 1 2 4 17 16 14 14
3 3 4 4 21 18 14 14
4 4 1 5 22 18 17 17
5 5 6 2 12 7 1 0
6 6 5 6 27 21 16 16
7 7 1 1 8 1 0 0

Average Turnaround Time: 13.71
Average Waiting Time: 9.86
```

Priority - Non Pre-emptive

```
#include < stdio.h >
Start Process {
    int pid, st, bt, pr, ct, wt, tat, n;
    id is completed;
}
void sortByArrival (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 findPriorityScheduling (Struct Process p[], int n) {
    SortByArrival(p, n);
    int time = 0, completed = 0;
    float totalWT = 0, totalTAT = 0;
    while (completed < n) {
        int id = -1, highestPriority = -999;
```

```
for (int i=0; i<n; i++) {
    if (p[i].at <= time && p[i].isCompleted == 0) {
        if (p[i].pr < highestPriority) {
            highestPriority = p[i].pr;
            id = i;
        }
    }
}
```

```
if (ct[0] == -1) {
    time += 1; //Initial time
} else {
    p[ct].ct = time - p[ct].at;
    time += p[ct].bt;
    p[ct].ct = time;
    p[ct].tat = p[ct].ct - p[ct].at;
    p[ct].wt = p[ct].tat - p[ct].bt;
    p[ct].isCompleted = 1;
    totalWT += p[ct].wt;
    totalTAT += p[ct].tat;
    completed++;
}
```

```
printf ("PID AT BT PR CT TAT WT\n");
for (int i=0; i<n; i++) {
    printf ("%d %d %d %d %d %d %d\n",
           p[i].pid, p[i].at, p[i].bt, p[i].pr, p[i].ct, p[i].tat,
           p[i].wt, p[i].xt);
}
```

```
> - - - - -
float printf ("Average Turnaround Time: %.2f\n", totalTAT/n);
printf ("Average Waiting Time: %.2f\n", totalWT/n);
> - - - - -
int main() {
    int n;
    printf ("Enter the number of processes: ");
    scanf ("%d", &n);
    StartProcess p[n];
    printf ("Enter the Arrival Time, Burst Time, and Priority for each process: ");
    for (int i=0; i<n; i++) {
        p[i].pid = i+1;
```

```

    priority ("Process % d", i + 1);
    scrunf ("% d % d % d", & p[i].at, &p[i].bt, &p[i].rt);
    p[i].isCompleted = 0;
}

```

Find Priority Scheduling (p, n);
return 0;

>

O/P

Enter the number of process 5

Enter Arrival Time, Burst Time, and Priority for each process:

process 1: 0 3 5

process 2: 2 2 3

process 3: 3 5 2

process 4: 4 4 4

process 5: 6 1 1

PID AT BT PR CT TAT WT RT

1 0 3 5 3 3 0 0

2 2 2 3 11 9 7 7

3 3 5 2 8 5 0 0

4 4 4 4 15 11 7 7

5 6 1 1 9 3 2 2

Average Turnaround Time : 6.20

Average Waiting Time : 3.20

```

=>Round Robin
#include <stdio.h>

#define MAX 100

void roundRobin(int n, int at[], int bt[], int quant) {
    int ct[n], tat[n], wt[n], rem_bt[n];
    int queue[MAX], front = 0, rear = 0;
    int time = 0, completed = 0, visited[n];

    for (int i = 0; i < n; i++) {
        rem_bt[i] = bt[i];
        visited[i] = 0;
    }

    queue[rear++] = 0;
    visited[0] = 1;

    while (completed < n) {
        int index = queue[front++];

        if (rem_bt[index] > quant) {
            time += quant;
            rem_bt[index] -= quant;
        } else {
            time += rem_bt[index];

```

```

rem_bt[index] = 0;
ct[index] = time;
completed++;
}

for (int i = 0; i < n; i++) {
    if (at[i] <= time && rem_bt[i] > 0 && !visited[i]) {
        queue[rear++] = i;
        visited[i] = 1;
    }
}

if (rem_bt[index] > 0) {
    queue[rear++] = index;
}

if (front == rear) {
    for (int i = 0; i < n; i++) {
        if (rem_bt[i] > 0) {
            queue[rear++] = i;
            visited[i] = 1;
            break;
        }
    }
}
}

```

```

float total_tat = 0, total_wt = 0;
printf("P#\tAT\tBT\tCT\tTAT\tWT\n");
for (int i = 0; i < n; i++) {
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
    total_tat += tat[i];
    total_wt += wt[i];
    printf("%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], ct[i], tat[i], wt[i]);
}

printf("Average TAT: %.2f\n", total_tat / n);
printf("Average WT: %.2f\n", total_wt / n);
}

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

    int at[n], bt[n];
    for (int i = 0; i < n; i++) {
        printf("Enter AT and BT for process %d: ", i + 1);
        scanf("%d %d", &at[i], &bt[i]);
    }
}

```

```

printf("Enter time quantum: ");
scanf("%d", &quant);

roundRobin(n, at, bt, quant);

return 0;

}

```

Result:

```

Enter number of processes: 5
Enter AT and BT for process 1: 0 8
Enter AT and BT for process 2: 5 2
Enter AT and BT for process 3: 1 7
Enter AT and BT for process 4: 6 3
Enter AT and BT for process 5: 8 5
Enter time quantum: 3
P#      AT      BT      CT      TAT      WT
1        0       8       22      22      14
2        5       2       11       6       4
3        1       7       23      22      15
4        6       3       14      8       5
5        8       5       25      17      12
Average TAT: 15.00
Average WT: 10.00

```

Round Robin

```
#include <stdio.h>
#define MAX 100

void roundRobin (int n, int dt[], int bt[], int quant) {
    int ct[n], tot[n], w[n], rem_bt[n];
    int queue[MAX], front = 0, rear = 0;
    int time = 0, completed = 0, visited[n];
    for (int i = 0; i < n; i++) {
        rem_bt[i] = bt[i];
        visited[i] = 0;
        queue[rear + i] = 0;
        visited[i] = 1;
    }

    while (completed < n) {
        int index = queue[front];
        if (rem_bt[index] > quant) {
            time += quant;
            rem_bt[index] -= quant;
            ct[index] += time;
            completed++;
        } else {
            time += rem_bt[index];
            ct[index] += time;
            completed++;
        }
        for (int i = 0; i < n; i++) {
            if (ct[i] <= time && rem_bt[i] > 0 && visited[i] == 0) {
                queue[rear + i] = i;
                visited[i] = 1;
            }
        }
        if (rem_bt[index] > 0) {
            queue[rear + i] = index;
        }
    }
}

If (front == rear) {
    for (int i = 0; i < n; i++) {
}
```

```
if (rem_bt[i] > 0) {
```

```
    queue[rear + i] = i;
```

```
    visited[i] = 1;
```

```
    break;
```

```
} // if (rem_bt[i] > 0) {
```

```
    if (i < n - 1) {
        tot[i + 1] = tot[i] + time;
        w[i + 1] = tot[i + 1] - ct[i + 1];
    }
}
```

```
float total_tat = 0, total_wt = 0;
printf ("Process ID\tArrival Time\tBurst Time\tQuantum\tCompletion Time\tWaiting Time\n");
for (int i = 0; i < n; i++) {
    tot[i] = ct[i] + w[i];
    total_tat += tot[i];
    total_wt += w[i];
    printf ("%d\t%d\t%d\t%d\t%d\t%.2f\n", i + 1, tot[i], ct[i], quant, tot[i], w[i]);
}
```

```
printf ("Average TAT: %.2f\n", total_tat / n);
printf ("Average WT: %.2f\n", total_wt / n);
```

```
int main() {
```

```
    int n, quant;
```

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

```

```
    scanf ("%d", &n);
```

```
    int dt[n], bt[n];
```

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

```

```
        printf ("Enter AT and BT for process %d: ", i + 1);

```

```
        scanf ("%d %d", &dt[i], &bt[i]);
    }
```

```
    printf ("Enter time quantum: ");

```

```
    scanf ("%d", &quant);

```

```
    roundRobin(n, dt, bt, quant);
    return 0;
}
```

Output:

Enter number of processes: 5

Enter AT and BT for process 1: 0 8

Enter AT and BT for process 2: 5 2

Enter AT and BT for process 3: 1 7

Enter AT and BT for process 4: 6 3

Enter AT and BT for process 5: 8 5

Enter time quantum: 3

P# AT BT CT TAT WT

1 0 8 22 14

2 5 2 11 6 4

3 1 7 23 22 15

4 6 3 14 8 5

5 8 5 25 17 12

Average TAT: 15.00

Average WT: 10.00

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 FCFS scheduling for the processes in each queue.

```
#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;
                    processes[i].turnaround_time = *time - processes[i].arrival_time;
                    processes[i].response_time = processes[i].waiting_time;
                }
            }
        }
    } while (!done);
}
```

```

        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%x) execution time: %.3f s\n", time, time,
(float)time);

return 0;
}

```

Result:

```
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 (0x11) execution time: 11.000 s

Process returned 0 (0x0) execution time : 41.000 s
Press any key to continue.
```

Multilevel queue.c

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

```
#define MAX PROCESSES 10
```

```
#define TIME QUANTUM 2
```

```
typedef struct {
```

```
    int burst_time, arrival_time, queue_type, waiting_time, turn_around_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)
```

```
                    done = 1;
```

```
*time += processes[i].remaining_time;
```

```
processes[i].turnaround_time = *time - processes[i].arrival_time;
```

```
- processes[i].burst_time;
```

```
processes[i].turnaround_time = *time - processes[i].arrival_time;
```

```
processes[i].response_time = processes[i].turnaround_time -
```

```
processes[i].remaining_time;
```

```
} while (!done);
```

```
void ffs (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 type: ");
```

```
        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[user_count] = processes[i];
```

```
> else {
```

```
    user_queue[sys_count] = processes[i];
```

```
>}
```

DATE:		PAGE NO.:	DATE:
<pre> Wait user processes by arrival time for FCFS : 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; } } } </pre>	<pre> printf("In Average Waiting Time : %f", avg_waiting); printf("In Average Turn Around Time : %f", avg turnaround); printf("In Average Response Time : %f", avg response); printf("In Throughput : %f", Throughput); printf("In Process Executed Id (%d / %d) execution time : %f sec", time, time, (float)time); return 0; </pre>	<pre> printf("In Queue [is System Process [Process P is User Process]]"); round robin (System queue, sys count, TIME QUANTUM, time lets user queue, user count, time); printf("In Process Waiting Time, Turn Around Time and Response Time"); 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; </pre>	<p>Output</p> <pre> 6.000000 1.000000 0.000000 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 Arrival Time, turnaround and Queue of P3: 5 0 1 Enter Burst Time, Arrival Time and Queue of P4: 3 0 2 P1 is system process P2 is User Process P3 is User Process P4 is User Process Process Waiting Time Turn Around Time Response Time 1 0 2 0 2 2 7 7 3 7 8 8 4 8 5 5 Average Waiting Time: 4.25 Average Turn Around Time: 7.000000 Average Response Time: 4.250000 Throughput: 0.36 Access scheduled 11 times executing from 11.000000 process released 0 (0x0) execution time: 41.000000 </pre>

Program -4

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

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

Rate- Monotonic :

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

typedef struct {
    int id, burst, period;
} Task;

int gcd(int a, int b) {
    return (b == 0) ? a : gcd(b, a % b);
}

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

int findLCM(Task tasks[], int n) {
    int result = tasks[0].period;
    for (int i = 1; i < n; i++)
        result = lcm(result, tasks[i].period);
    return result;
}
```

```

void rateMonotonic(Task tasks[], int n) {
    float utilization = 0;
    printf("\nRate Monotonic Scheduling:\nPID\tBurst\tPeriod\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\n", tasks[i].id, tasks[i].burst, tasks[i].period);
        utilization += (float)tasks[i].burst / tasks[i].period;
    }

    float bound = n * (pow(2, (1.0 / n)) - 1);
    printf("\nUtilization: %.6f, Bound: %.6f\n", utilization, bound);
    if (utilization <= bound)
        printf("Tasks are Schedulable\n");
    else
        printf("Tasks are NOT Schedulable\n");
}

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

    Task tasks[n];
    printf("Enter the CPU burst times: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &tasks[i].burst);

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

```

```
    tasks[i].id = i + 1;  
}  
  
rateMonotonic(tasks, n);  
return 0;  
}
```

Result:

```
Enter the number of processes: 3  
Enter the CPU burst times: 3 6 8  
Enter the time periods: 3 4 5  
  
Rate Monotonic Scheduling:  
PID Burst Period  
1   3   3  
2   6   4  
3   8   5  
  
Utilization: 4.100000, Bound: 0.779763  
Tasks are NOT Schedulable
```

<p>Rate Monotonic</p> <pre> #include <stdio.h> #include <math.h> typedef struct { int id, burst, period; } Task; int gcd(int a, int b) { return (b == 0) ? a : gcd(b, a % b); } int lcm(int a, int b) { return (a * b) / gcd(a, b); } int findLCM(Task tasks[], int n) { int result = tasks[0].period; for (int i = 1; i < n; i++) { result = lcm(result, tasks[i].period); } return result; } void rateMonotonic(Task tasks[], int n) { float utilization = 0; printf("In Rate Monotonic Scheduling: In PID + Burst + Periodn"); for (int i = 0; i < n; i++) { float bound = (float) tasks[i].burst / tasks[i].period; float utilization = utilization + (float) tasks[i].burst / tasks[i].period; if (utilization > bound) { printf("Tasks are NOT schedulable\n"); return; } } } </pre>	<p>PAGE NO.: DATE:</p> <p>PAGE NO.: DATE:</p> <pre> printf("Tasks are NOT schedulable\n"); } int main() { int n; printf("Enter the number of processes: "); scanf("%d", &n); Task tasks[n]; printf("Enter the CPU burst times: "); for (int i = 0; i < n; i++) { scanf("%d", &tasks[i].burst); } printf("Enter the time periods: "); for (int i = 0; i < n; i++) { scanf("%d", &tasks[i].period); tasks[i].id = i + 1; } rateMonotonic(tasks, n); return 0; } O/P Enter the number of processes: 3 Enter the CPU burst times: 3 6 8 Enter the time periods: 3 4 5 Rate Monotonic Scheduling: PID Burst Period 1 3 3 2 6 4 3 8 5 Utilization: 4.100000, Bound: 0.779763 Tasks are NOT schedulable </pre>
---	--

Earliest-deadline First:

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

```
typedef struct {
    int pid;
    int burst_time;
    int deadline;
    int period;
    int remaining_time;
```

```

} Process;

// Function to compare processes based on their deadlines
int compare_deadlines(const void *a, const void *b) {
    return ((Process *)a)->deadline - ((Process *)b)->deadline;
}

int main() {
    int num_processes;

    printf("Enter the number of processes: ");
    scanf("%d", &num_processes);

    Process processes[num_processes];

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

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

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

    // Sort processes based on deadlines
    qsort(processes, num_processes, sizeof(Process), compare_deadlines);
}

```

```

printf("\nEarliest Deadline Scheduling:\n");
printf("PID\tBurst\tDeadline\tPeriod\n");
for (int i = 0; i < num_processes; i++) {
    printf("%d\t%d\t%d\t%d\n", processes[i].pid, processes[i].burst_time,
processes[i].deadline, processes[i].period);
}

int current_time = 0;
int completed_processes = 0;

printf("\nScheduling occurs for %d ms\n", processes[0].deadline); // Assuming the
first process has the earliest deadline

while (completed_processes < num_processes) {
    for (int i = 0; i < num_processes; i++) {
        if (processes[i].remaining_time > 0) {
            printf("%dms : Task %d is running.\n", current_time, processes[i].pid);
            processes[i].remaining_time--;
            current_time++;

            if (processes[i].remaining_time == 0) {
                completed_processes++;
            }
        }
    }
}

printf("\nProcess returned %d (0x%X)\texecution time : %.3f s\n", current_time,
current_time, (float)current_time / 1000.0);

return 0;
}

```

Result:

```
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

Earliest Deadline Scheduling:
PID      Burst      Deadline      Period
1        2          1            1
2        3          2            2
3        4          3            3

Scheduling occurs for 1 ms
0ms : Task 1 is running.
1ms : Task 2 is running.
2ms : Task 3 is running.
3ms : Task 1 is running.
4ms : Task 2 is running.
5ms : Task 3 is running.
6ms : Task 2 is running.
7ms : Task 3 is running.
8ms : Task 3 is running.

Process returned 9 (0x9)      execution time : 0.009 s

Process returned 0 (0x0)      execution time : 18.448 s
Press any key to continue.
```

Earliest Deadline

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

typedef struct {
    int pid;
    int burst_time;
    int deadline;
    int period;
    int remaining_time;
} Process;
```

```
// Function to compare processes based on their deadlines
int compare_deadlines(const void *a, const void *b) {
    return ((Process *)a)->deadline - ((Process *)b)->deadline;
}

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

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

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

// Sort processes based on deadlines

```
qsort(&processes, num_processes, sizeof(Process), compare_deadlines);
```

```
printf("In Earliest Deadline scheduling: \n");
```

```
printf("PID %d has its deadline %d\n", processes[0].pid,
```

```
processes[0].burst_time, processes[0].deadline);
```

```
int current_time = 0;
int completed_processes = 0;
```

```
printf("In Scheduling occurs for %d ms \n", processes[0].deadline);
// Assigning the first process to the earliest deadline
```

```
while (completed_processes < num_processes) {
    for (int i = 0; i < num_processes; i++) {
```

```
        if (processes[i].remaining_time > 0) {
            printf("Task %d is running \n", current_time + processes[i].burst_time);
            processes[i].remaining_time -= 1;
            current_time += 1;
        }
    }
}
```

```
# (processes[i].remaining_time == 0) &
```

```
    completed_processes++;
```

```
printf("In case returned %d (0x%lx) Iteration time: %d ms \n",
```

49

current-time, current-time, (float) current-time / 1000.0);

return 0;

}

Output:

Enter the numbers of processes : 3

Enter the CPU burst times : 2 3 4

Enter the deadlines : 1 2 3

Enter the time periods : 1 2 3

Earliest Deadline Scheduling.

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

Scheduling occurs for 1ms

0ms: Task 1 is running

1ms: Task 2 is running

2ms: Task 3 is running

3ms: Task 1 is running

4ms: Task 2 is running

5ms: Task 3 is running

6ms: Task 2 is running

7ms: Task 3 is running

8ms: Task 3 is running

Program - 5

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

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
int mutex = 1, full = 0, empty, x = 0;
int *buffer, buffer_size;
int in = 0, out = 0;
int wait(int s) {
    return (--s);
}
int signal(int s) {
    return (++s);
}

void producer(int id) {
    if ((mutex == 1) && (empty != 0)) {
        mutex = wait(mutex);
        full = signal(full);
        empty = wait(empty);
        int item = rand() % 50;
        buffer[in] = item;
        x++;
        printf("Producer %d produced %d\n", id, item);
        printf("Buffer:%d\n", item);
        in = (in + 1) % buffer_size;
        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);
        x--;
    }
}
```

```

printf("Current buffer len: %d\n", x);
out = (out + 1) % buffer_size;
mutex = signal(mutex);
} else {
    printf("Buffer is empty\n");
}
}

int main() {
    int producers, consumers;
    printf("Enter the number of Producers:");
    scanf("%d", &producers);
    printf("Enter the number of Consumers:");
    scanf("%d", &consumers);
    printf("Enter buffer capacity:");
    scanf("%d", &buffer_size);
    buffer = (int *)malloc(sizeof(int) * buffer_size);
    empty = buffer_size;
    for (int i = 1; i <= producers; i++)
        printf("Successfully created producer %d\n", i);
    for (int i = 1; i <= consumers; i++)
        printf("Successfully created consumer %d\n", i);
    srand(time(NULL));
    int iterations = 10;
    for (int i = 0; i < iterations; i++) {
        producer(1);
        sleep(1);
        consumer(2);
        sleep(1);
    }
    free(buffer);
    return 0;
}

```

Result:

```
Enter the number of Producers:1
Enter the number of Consumers:1
Enter buffer capacity:1
Successfully created producer 1
Successfully created consumer 1
Producer 1 produced 24
Buffer:24
Consumer 2 consumed 24
Current buffer len: 0
Producer 1 produced 37
Buffer:37
Consumer 2 consumed 37
Current buffer len: 0
Producer 1 produced 26
Buffer:26
Consumer 2 consumed 26
Current buffer len: 0
Producer 1 produced 46
Buffer:46
Consumer 2 consumed 46
Current buffer len: 0
Producer 1 produced 34
Buffer:34
Consumer 2 consumed 34
Current buffer len: 0
Producer 1 produced 13
Buffer:13
Consumer 2 consumed 13
Current buffer len: 0
Producer 1 produced 22
Buffer:22
Consumer 2 consumed 22
Current buffer len: 0
Producer 1 produced 3
Buffer:3
Consumer 2 consumed 3
Current buffer len: 0
Producer 1 produced 12
Buffer:12
Consumer 2 consumed 12
Current buffer len: 0
Producer 1 produced 14
Buffer:14
Consumer 2 consumed 14
Current buffer len: 0

Process returned 0 (0x0)  execution time : 29.417 s
Press any key to continue.
```

Producer Consumer

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <time.h>

int mvalue = 1, full = 0, empty, x=0;
int *buffer; buffer_size;
int in = 0, out = 0;

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

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

void producer (int id) {
    if (mvalue == 1 && empty != 0) {
        mvalue = wait (mvalue);
        full = signal (full);
        empty = wait (empty);
        int item = rand (0, 50);
        buffer [in] = item;
        in++;
        printf ("Producer %d produced %d \n", id, item);
        printf ("Buffer : %d \n", item);
        m = (in + 1) / buffer_size;
        mvalue = signal (mvalue);
    } else {
        printf ("Buffer is full \n");
    }
}
```

<pre> void consumer (int id) { if (empty == 1) && (full == 0) { motor = wait (motor); full = wait (full); empty = signal (empty); int item = buffer [id]; print ("Consumer %d consumed %d\n"; id, item); sleep (1); cout << "Current buffer len: " << id << endl; out = cout + P::buffer->size; motor = signal (motor); } else { print ("Buffer is empty\n"); } } int main () { int producers, consumers; print ("Enter the number of Producers: "); scanf ("%d", &producers); print ("Enter the number of Consumers: "); scanf ("%d", &consumers); print ("Enter buffer capacity: "); scanf ("%d", &buffer->size); buffer = (int *) malloc (sizeof (int) * buffer->size); empty = buffer->size; for (int i = 1; i <= producers; i++) { printf ("successfully created producer %d (%d)\n"; for (int j = 1; j <= consumers; j++) print ("successfully created consumer %d (%d)\n"; send (j, (NUL)); } } </pre>	<pre> int iterations = 10; for (int i = 0; i < iterations; i++) { producer (1); sleep (1); consumer (2); sleep (1); } free (buffer); return 0; </pre> <p><u>Output</u></p> <p>Enter the number of producers: 1 Enter the number of consumers: 1 Enter buffer capacity: 1 successfully created producer 1 successfully created consumer 1 producer 1 produced 1 consumer 1 consumed 1 current bufferlen: 0 producer 1 produced 2 consumer 1 consumed 2 current bufferlen: 0 producer 1 produced 3 consumer 1 consumed 3 current bufferlen: 0 </p>
--	---

Program -6

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

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>
#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N
int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };
sem_t mutex;
sem_t S[N];
void test(int phnum)
{
    if (state[phnum] == HUNGRY &&
        state[LEFT] != EATING &&
        state[RIGHT] != EATING) {
        // Eating state
        state[phnum] = EATING;
        printf("Philosopher %d takes chopstick %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);
        printf("Philosopher %d is Eating\n", phnum + 1);
        sem_post(&S[phnum]);
    }
}
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 chopstick %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)
{
    while (1) {
        int* i = num;
        sleep(1);
        take_fork(*i);
        sleep(1);
        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;
}

```

Result:

```
Philosopher 5 is thinking
Philosopher 1 takes chopstick 5 and 1
Philosopher 1 is Eating
Philosopher 3 putting chopstick 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 takes chopstick 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 5 is Hungry
Philosopher 1 putting chopstick 5 and 1 down
Philosopher 1 is thinking
Philosopher 2 takes chopstick 1 and 2
Philosopher 2 is Eating
Philosopher 4 putting chopstick 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 takes chopstick 4 and 5
Philosopher 5 is Eating
Philosopher 1 is Hungry
Philosopher 4 is Hungry
Philosopher 5 putting chopstick 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes chopstick 3 and 4
Philosopher 4 is Eating
Philosopher 2 putting chopstick 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes chopstick 5 and 1
Philosopher 1 is Eating
Philosopher 2 is Hungry
Philosopher 5 is Hungry
Philosopher 1 putting chopstick 5 and 1 down
Philosopher 1 is thinking
Philosopher 2 takes chopstick 1 and 2
Philosopher 2 is Eating
Philosopher 4 putting chopstick 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 takes chopstick 4 and 5
Philosopher 5 is Eating
Philosopher 1 is Hungry
Philosopher 4 is Hungry
Philosopher 5 putting chopstick 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes chopstick 3 and 4
Philosopher 4 is Eating
Philosopher 2 putting chopstick 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes chopstick 5 and 1
Philosopher 1 is Eating
Philosopher 2 is Hungry
Philosopher 5 is Hungry
Philosopher 1 putting chopstick 5 and 1 down
Philosopher 1 is thinking
Philosopher 2 takes chopstick 1 and 2
Philosopher 2 is Eating
Philosopher 4 putting chopstick 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 takes chopstick 4 and 5
Philosopher 5 is Eating
Philosopher 1 is Hungry
Philosopher 4 is Hungry
```

<p style="text-align: center;"><small>PAGE NO: _____ DATE: _____</small></p> <p><u>Dining Philosophers' problem</u></p> <pre> #include <phnnum.h> #include <sompheore.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]; int phil[5] = {0,1,2,3,4}; sem_t muter; sem_t sem[N]; void test(int phnum) { if (state[phnum] == HUNGRY) if (state[LEFT(phnum)] == EATING) if (state[RIGHT(phnum)] == EATING) // Hungry state state[phnum] = EATING; } printf("Philosopher %d takes chopstick %d and %d\n", phnum); LEFT += 1, phnum += 1; printf("Philosopher %d is Eating\n", phnum); sem_post(&sem[phnum]); </pre>	<p style="text-align: center;"><small>PAGE NO: _____ DATE: _____</small></p> <p>void take_fork(int phnum)</p> <pre> sem_wait(&muter); state[phnum] = HUNGRY; printf("Philosopher %d is Hungry\n", phnum); sem_post(&sem[phnum]); </pre> <p>test(phnum);</p> <pre> sem_post(&muter); sem_wait(&sem[phnum]); sleep(3); </pre> <p>void put_fork(int phnum)</p> <pre> sem_wait(&muter); state[phnum] = THINKING; printf("Philosopher %d putting chopstick %d and %d down\n", phnum + 1, LEFT + 1, phnum + 1); sem_post(&sem[phnum]); </pre> <p>left(LEFT);</p> <p>test(RIGHT);</p> <p>sem_post(&muter);</p> <pre> void philosopher(int *num) { while(1) { int i = *num; sleep(1); take_fork(i); sleep(2); put_fork(i); } } </pre>
--	---

int main() {
 // code for initializing shared memory
 // ...
 }

int i;
 thread t1 = thread(&function, &i);
 sem_init(&semaphore, 0, 1); // semaphore initialized to 1
 for (i = 0; i < N; i++) {
 sem_wait(&semaphore); // wait for semaphore
 t1.join(); // join with thread t1
 cout << "Value of i is " << i << endl;
 sem_post(&semaphore); // release semaphore
 }
}

function(i) {
 cout << "Value of i is " << i << endl;
 i++;
 }

for (i = 0; i < N; i++) {
 cout << "Value of i is " << i << endl;
 }
}

cout << "Final value of i is " << i << endl;

Output:

- 1. The program has deadlock
- 2. Deadlock occurs because both philosophers are holding chopsticks
- 3. Each philosopher holds two chopsticks
- 4. There are 5 chopsticks in total
- 5. There are 3 eaters
- 6. Each philosopher eats 2 times
- 7. Each philosopher needs 3 chopsticks

 1. Two can eat at a time
 2. Two can eat at a time
 3. Eat

Eat your first 2?
 Allow the philosophers to eat at once
 PS is granted to eat
 PS is waiting
 PS is granted to eat

PS is eating

PS is granted to eat

P2 is waiting

P1 is waiting

1. One can eat at a time
2. Two can eat at a time
3. Both

Take your choice : P

Allow two philosophers to eat at same time
combination 1

PP and PM are granted eat

PS is waiting

Combination 2

PC and PS are granted eat

PM is waiting

Program -7

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

Program -8

Question: Write a C program to simulate deadlock detection

```
#include <stdio.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], req[n][m], avail[m], 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", &req[i][j]);
    printf("Enter available matrix:\n");
    for (i = 0; i < m; i++)
        scanf("%d", &avail[i]);
    for (i = 0; i < n; i++)
        finish[i] = 0;
    int done;
    do {
        done = 0;
        for (i = 0; i < n; i++) {
            if (finish[i] == 0) {
                int canFinish = 1;
                for (j = 0; j < m; j++) {
                    if (req[i][j] > avail[j]) {
                        canFinish = 0;
                        break;
                    }
                }
                if (canFinish) {
                    for (j = 0; j < m; j++)
                        avail[j] += alloc[i][j];
                    finish[i] = 1;
                    done = 1;
                }
            }
        }
    } while (!done);
}
```

```

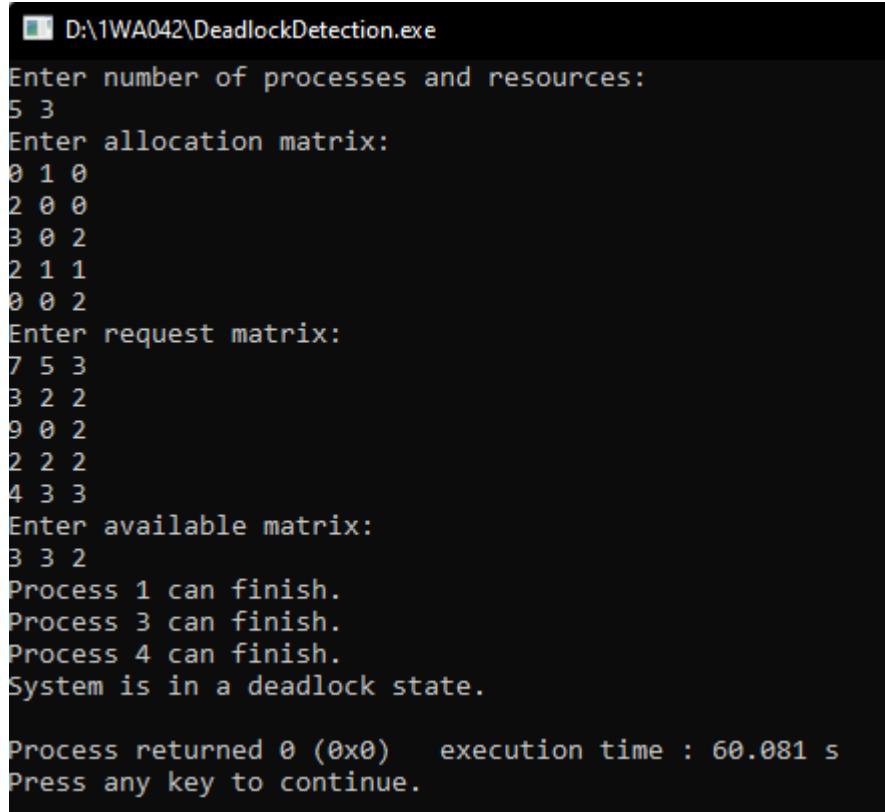
        printf("Process %d can finish.\n", i);
    }
}
}
} while (done);

int deadlock = 0;
for (i = 0; i < n; i++)
    if (finish[i] == 0)
        deadlock = 1;
if (deadlock)
    printf("System is in a deadlock state.\n");
else
    printf("System is not in a deadlock state.\n");

return 0;
}

```

Result:



```

D:\1WA042\DeadlockDetection.exe

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 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 returned 0 (0x0)  execution time : 60.081 s
Press any key to continue.

```

Pankaj's Algorithm

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

```
int main() {
```

```
    int n, m, j, k;
```

```
    printf("Enter number of processes and resources : ");
```

```
    scanf("%d %d", &n, &m);
```

```
    int alloc[n][m], max[n][m], avail[m];
```

```
    int need[n][m], f[m], ans[n];
```

```
    int C[5] = {0};
```

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

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

```
            scanf("%d", &alloc[i][j]);
```

```
    printf("Enter max matrix : ");
```

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

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

```
            scanf("%d", &max[i][j]);
```

```
    printf("Enter available matrix : ");
```

```
    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<n; i++)
```

```
        if (f[i] == 0) {
```

```
            int flag=0;
```

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

```
        if (need[i][j] > avail[j]) {
```

```
            flag = 1;
```

```
            break;
```

```
>
```

```
        if (flag == 0) {
```

```
            ans[i] = j;
```

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

```
                avail[j] += alloc[i][j];
```

```
            f[i] = 1;
```

```
    int safe = 1;
```

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

```
        if (f[i] == 0)
```

```
            safe = 0;
```

```
} // end of for
```

```
printf("System is in safe state. In safe sequence : ");
```

```
for (i=0; i<n-1; i++)
```

```
    printf("P%d -> ", ans[i]);
```

```
    printf("P%d\n", ans[n-1]);
```

```
else {
```

```
    printf("System is not in safe state. ");
```

```
return 0;
```

Output:-

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 max 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: A \rightarrow P₃ \rightarrow P₄ \rightarrow P₀ \rightarrow P₂

Program 8:

Question: Write a C program to simulate deadlock detection

```
#include <stdio.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], req[n][m], avail[m], 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", &req[i][j]);

    printf("Enter available matrix:\n");
    for (i = 0; i < m; i++)
        scanf("%d", &avail[i]);

    for (i = 0; i < n; i++)
        finish[i] = 0;

    int done;
    do {
        done = 0;
        for (i = 0; i < n; i++) {
            if (finish[i] == 0) {
                int canFinish = 1;
                for (j = 0; j < m; j++) {
                    if (req[i][j] > avail[j]) {
                        canFinish = 0;
                        break;
                    }
                }
                if (canFinish) {
                    for (j = 0; j < m; j++)
                        avail[j] += alloc[i][j];
                    finish[i] = 1;
                }
            }
        }
    } while (done == 0);
}
```

```

        done = 1;
        printf("Process %d can finish.\n", i);
    }
}
}

} while (done);

int deadlock = 0;
for (i = 0; i < n; i++)
    if (finish[i] == 0)
        deadlock = 1;

if (deadlock)
    printf("System is in a deadlock state.\n");
else
    printf("System is not in a deadlock state.\n");

return 0;
}

```

Result:

```

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 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.

```

Deadlock Detection

```

#include < stdio.h >
int main () {
    int n, m, j, i, k;
    printf ("Enter number of processes and resources :\n");
    scanf ("%d %d", &n, &m);
    int alloc[n][m], req[n][m], avail[m], 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", &req[i][j]);
        }
    }
    printf ("Enter available matrix :\n");
    for (i=0; i<m; i++) {
        scanf ("%d", &avail[i]);
    }
    for (i=0; i<n; i++) {
        finish[i] = 0;
    }
    int done;
    do {
        done = 0;
        for (i=0; i<n; i++) {
            if (finish[i]==0) {
                int canFinish = 1;
                for (j=0; j<m; j++) {
                    if (req[i][j] > avail[j]) {
                        canFinish = 0;
                    }
                }
                if (canFinish == 1) {
                    finish[i] = 1;
                    for (j=0; j<m; j++) {
                        avail[j] = avail[j] - req[i][j];
                    }
                }
            }
        }
    } while (done != n);
}

```

```

if (CanFinish) {
    for (j = 0; j < m; j++) {
        avail[i][j] = avail[i][j] - need[i][j];
        if (avail[i][j] <= 0) {
            done = 1;
            break;
        }
    }
    if (done == 1)
        printf("Process %d can finish.\n", i);
}
while (done);
int deadlock = 0;
for (i = 0; i < n; i++) {
    if (finish[i] == 0)
        deadlock = 1;
}
if (deadlock)
    printf("System is in a deadlock state.\n");
else
    printf("System is not in a deadlock state.\n");
return 0;

```

Output :-

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 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.

Program 9:

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

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

→Worst Fit

```
#include <stdio.h>

struct Block {
    int block_no;
    int block_size;
    int is_free;
};

struct File {
    int file_no;
    int file_size;
};

void worstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("\nMemory Management Scheme - Worst Fit\n");
    printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragment\n");

    for (int i = 0; i < n_files; i++) {
        int worst_fit_block = -1;
        int max_fragment = -1;

        for (int j = 0; j < n_blocks; j++) {
            if (blocks[j].is_free && blocks[j].block_size >= files[i].file_size) {
                int fragment = blocks[j].block_size - files[i].file_size;
                if (fragment > max_fragment) {
                    max_fragment = fragment;
                    worst_fit_block = j;
                }
            }
        }

        if (worst_fit_block != -1) {
            blocks[worst_fit_block].is_free = 0;
            printf("%d\t%d\t%d\t%d\t%d\n",
                   files[i].file_no,
                   files[i].file_size,
                   blocks[worst_fit_block].block_no,
                   blocks[worst_fit_block].block_size,
                   max_fragment);
        } else {
    }
```

```

        printf("%d\t%d\t\tNot Allocated\n", files[i].file_no, files[i].file_size);
    }
}
}

int main() {
    int n_blocks, n_files;

    printf("Enter the number of blocks: ");
    scanf("%d", &n_blocks);
    struct Block blocks[n_blocks];
    for (int i = 0; i < n_blocks; i++) {
        blocks[i].block_no = i + 1;
        printf("Enter the size of block %d: ", i + 1);
        scanf("%d", &blocks[i].block_size);
        blocks[i].is_free = 1;
    }
    printf("Enter the number of files: ");
    scanf("%d", &n_files);
    struct File files[n_files];
    for (int i = 0; i < n_files; i++) {
        files[i].file_no = i + 1;
        printf("Enter the size of file %d: ", i + 1);
        scanf("%d", &files[i].file_size);
    }
    worstFit(blocks, n_blocks, files, n_files);

    return 0;
}

```

Result:

Memory Management Scheme - Worst Fit				
File_no	File_size	Block_no	Block_size	Fragment
1	212	5	600	388
2	417	2	500	83
3	112	4	300	188
4	426	Not Allocated		

→Best Fit

```
#include <stdio.h>
struct Block {
    int block_no;
    int block_size;
    int is_free;
};
struct File {
    int file_no;
    int file_size;
};
void bestFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("\nMemory Management Scheme - Best Fit\n");
    printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragment\n");

    for (int i = 0; i < n_files; i++) {
        int best_fit_block = -1;
        int min_fragment = 10000; // Large initial value

        for (int j = 0; j < n_blocks; j++) {
            if (blocks[j].is_free && blocks[j].block_size >= files[i].file_size) {
                int fragment = blocks[j].block_size - files[i].file_size;
                if (fragment < min_fragment) {
                    min_fragment = fragment;
                    best_fit_block = j;
                }
            }
        }

        if (best_fit_block != -1) {
            blocks[best_fit_block].is_free = 0;
            printf("%d\t%d\t%d\t%d\t%d\n",
                   files[i].file_no,
                   files[i].file_size,
                   blocks[best_fit_block].block_no,
                   blocks[best_fit_block].block_size,
                   min_fragment);
        } else {
            printf("%d\t%d\tNot Allocated\n", files[i].file_no, files[i].file_size);
        }
    }
}

int main() {
    int n_blocks, n_files;

    printf("Enter the number of blocks: ");
    scanf("%d", &n_blocks);
```

```

struct Block blocks[n_blocks];

for (int i = 0; i < n_blocks; i++) {
    blocks[i].block_no = i + 1;
    printf("Enter the size of block %d: ", i + 1);
    scanf("%d", &blocks[i].block_size);
    blocks[i].is_free = 1;
}

printf("Enter the number of files: ");
scanf("%d", &n_files);

struct File files[n_files];

for (int i = 0; i < n_files; i++) {
    files[i].file_no = i + 1;
    printf("Enter the size of file %d: ", i + 1);
    scanf("%d", &files[i].file_size);
}

bestFit(blocks, n_blocks, files, n_files);

return 0;
}

```

Result:

```

Enter the number of blocks: 5
Enter the size of block 1: 100
Enter the size of block 2: 500
Enter the size of block 3: 200
Enter the size of block 4: 300
Enter the size of block 5: 600
Enter the number of files: 4
Enter the size of file 1: 212
Enter the size of file 2: 417
Enter the size of file 3: 113
Enter the size of file 4: 426

Memory Management Scheme - Best Fit
File_no File_size     Block_no     Block_size   Fragment
1      212           4            300          88
2      417           2            500          83
3      113           3            200          87
4      426           5            600          174

```

→First Fit

```
#include <stdio.h>

struct Block {
    int block_no;
    int block_size;
    int is_free;
};

struct File {
    int file_no;
    int file_size;
};

void firstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("\nMemory Management Scheme - First Fit\n");
    printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragment\n");

    for (int i = 0; i < n_files; i++) {
        int allocated = 0;
        for (int j = 0; j < n_blocks; j++) {
            if (blocks[j].is_free && blocks[j].block_size >= files[i].file_size) {
                int fragment = blocks[j].block_size - files[i].file_size;
                blocks[j].is_free = 0;
                printf("%d\t%d\t%d\t%d\t%d\n",
                    files[i].file_no,
                    files[i].file_size,
                    blocks[j].block_no,
                    blocks[j].block_size,
                    fragment);

                allocated = 1;
                break;
            }
        }
        if (!allocated) {
            printf("%d\t%d\tNot Allocated\n", files[i].file_no, files[i].file_size);
        }
    }
}

int main() {
    int n_blocks, n_files;
    printf("Enter the number of blocks: ");
    scanf("%d", &n_blocks);

    struct Block blocks[n_blocks];

    for (int i = 0; i < n_blocks; i++) {
        blocks[i].block_no = i + 1;
```

```

printf("Enter the size of block %d: ", i + 1);
scanf("%d", &blocks[i].block_size);
blocks[i].is_free = 1;
}
printf("Enter the number of files: ");
scanf("%d", &n_files);
struct File files[n_files];
for (int i = 0; i < n_files; i++) {
    files[i].file_no = i + 1;
    printf("Enter the size of file %d: ", i + 1);
    scanf("%d", &files[i].file_size);
}
firstFit(blocks, n_blocks, files, n_files);
return 0;
}

```

Result:

```

Enter the number of blocks: 5
Enter the size of block 1: 100
Enter the size of block 2: 500
Enter the size of block 3: 200
Enter the size of block 4: 300
Enter the size of block 5: 600
Enter the number of files: 4
Enter the size of file 1: 212
Enter the size of file 2: 417
Enter the size of file 3: 112
Enter the size of file 4: 426

```

Memory Management Scheme - First Fit

File_no	File_size	Block_no	Block_size	Fragment
1	212	2	500	288
2	417	5	600	183
3	112	3	200	88
4	426		Not Allocated	

PAGE NO:
DATE:

> Categorial

First fit

```

#include <stdio.h>
struct Block {
    int block_no;
    int block_size;
    int is_free;
};
struct File {
    int file_no;
    int file_size;
    int file_start;
};
void FirstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("In Memory Management Scheme - First Fit\n");
    printf("File no\tFile size\tBlock size\tFragment\n");
    for (int i = 0; i < n_files; i++) {
        int allocated = 0;
        for (int j = 0; j < n_blocks; j++) {
            if (blocks[j].is_free && blocks[j].block_size >= files[i].file_size) {
                int fragment = blocks[j].block_size - files[i].file_size;
                blocks[j].is_free = 0;
                printf("%d\t%d\t%d\t%d\n", files[i].file_no,
                    files[i].file_size,
                    blocks[j].block_no,
                    blocks[j].block_size - fragment);
                allocated = 1;
            }
        }
        if (!allocated)
            printf("No free block found for file %d\n", files[i].file_no);
    }
}

```

PAGE NO.:
DATE:

PAGE NO.:
DATE:

```

f (Allocated) {
    printf ("x d t d l t t N o t Allocated In", file[1],
           file[2], file[3], file[4]);
}
else {
    printf ("x d t d l t t N o t Allocated In", file[1],
           file[2], file[3], file[4]);
}

void bestfit (char BlockNoes[], int n_blocks, struct
              files[], int n_files) {
    printf ("Memory Management Scheme - Best Fit\n");
    printf ("File no. of File size & Block No's Block size\n");
    for (int i=0; i<n_file; i++) {
        int best_block = -1;
        int min_fragments = 1000; // Large initial value
        for (int j=0; j<n_blocks; j++) {
            if (blocks[j].is_free == 1) {
                if (blocks[j].block_size == file[i].file_size) {
                    if (fragments < n_file - 1) {
                        min_fragments = fragments;
                        best_fit_block = j;
                    }
                }
            }
        }
        if (best_fit_block == -1) {
            printf ("No free block found for file %d", i+1);
            printf ("File no. , file size , blocks[best_fit_block].block_no,
                   blocks[best_fit_block].block_size,
                   min_fragments");
        }
    }
}

```

Output:

Enter the number of blocks : 5

Enter the size of blocks 1 : 100

Enter the size of block 2 : 500

3 : 200

4 : 300

5 : 600

Take the number of file : 4

File no size of file 1 : 214

2 : 417

3 : 112

Memory Manager 4 : 426

selected file

File no	File size	Block no	Block size	fragment
1	212	2	500	288
2	44	5	600	183
3	112	3	200	88
4	426	4	300	148

Memory Management scheme - Best fit

File no	File size	Block no	Block size	fragment
1	212	4	300	88
2	44	2	500	83
3	112	3	200	87
4	426	5	600	148

Memory management Scheme - First fit

File no	File size	Block no	Block size	fragment
1	212	5	600	388
2	44	2	500	83
3	112	4	300	188
4	426	not Allocated		

Program 10:

Question: Write a C program to simulate page replacement algorithms

- d) FIFO
- e) LRU
- f) Optimal

→FIFO

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

Result:

```
Enter the size of the pages:  
7  
Enter the page strings:  
103563  
Enter the no of page frames:  
3  
FIFO Page Faults: 6, Page Hits: 1
```

FIFO

#include < stdio.h >

int search (int page[], int frame[], int n, int size)

```
for (int i=0; i<n; i++) {
    if (frame[i] == -1) {
        return i;
    }
}
```

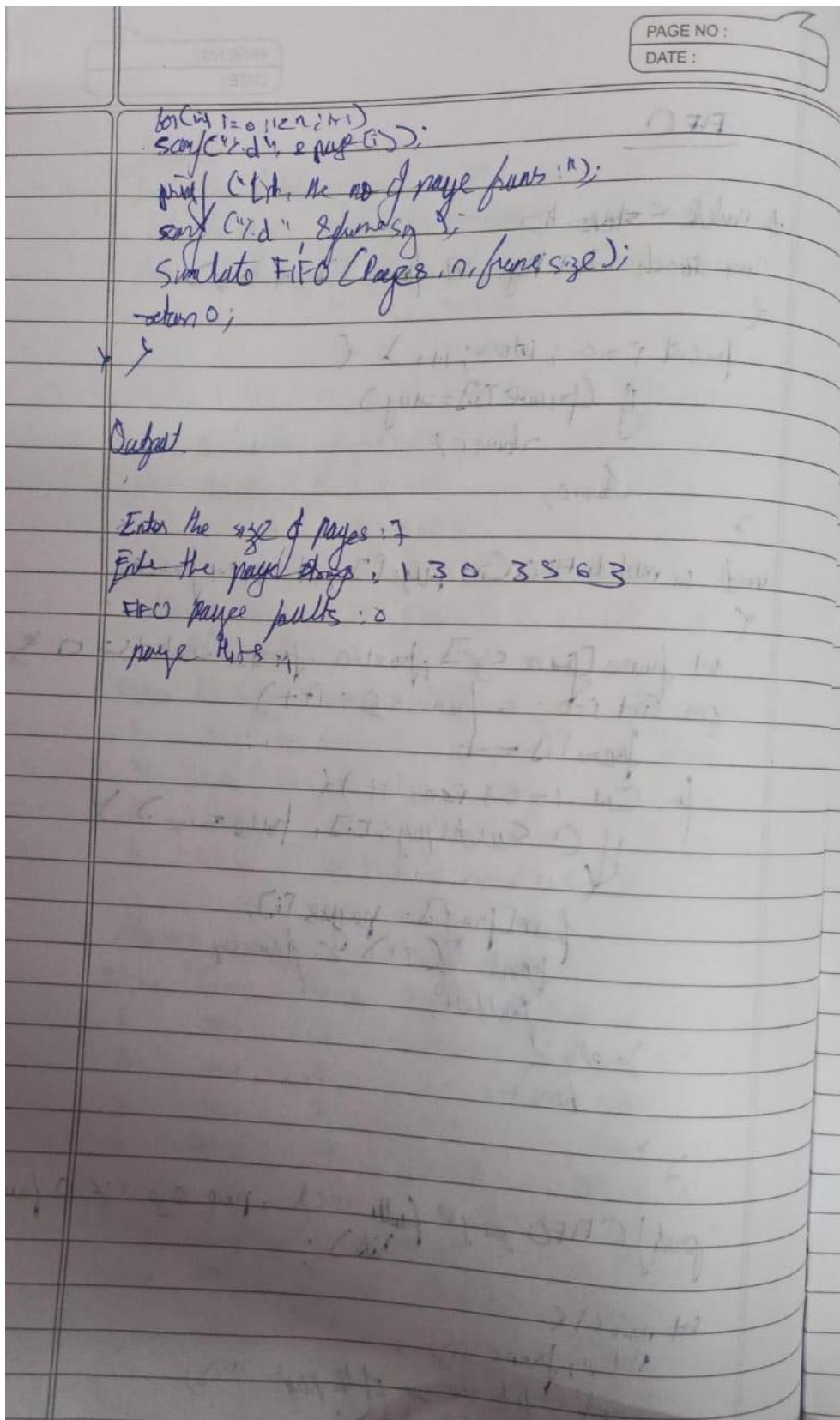
void simulateFIFO (int page[], int n, int frames)

```
int frames [frame size]; front = 0, faults = 0, hots = 0
for (int i=0; i< frame size; i++)
    frames[i] = -1;
for (int i=0; i<n; i++) {
    if (!search (page[i], frames, size)) {
        frames[front] = page[i];
        front = (front + 1) % frames;
        faults++;
    }
}
```

printf ("FIFO page fault : %d , page size : %d faults, %d);

int main ()

```
int n, frame size;
page [ ] ; // Intial address of the pages ...
scanf ("%d %d", &n, &frame size);
int page [n];
printf ("The %d page string (%d);
```



→LRU

```
#include <stdio.h>

int main() {
    int n, frames, i, j, k, page_faults = 0, hits = 0;
    char pages[100];
    int mem[10], used[10];

    printf("Enter the size of the pages:\n");
    scanf("%d", &n);
    printf("Enter the page strings:\n");
    scanf("%s", pages);
    printf("Enter the no of page frames:\n");
    scanf("%d", &frames);

    for (i = 0; i < frames; i++) {
        mem[i] = -1;
        used[i] = -1;
    }

    for (i = 0; i < n; i++) {
        int page = pages[i] - '0';
        int found = 0;

        for (j = 0; j < frames; j++) {
            if (mem[j] == page) {
                hits++;
                used[j] = i;
                found = 1;
                break;
            }
        }

        if (!found) {
            int lru = 0;
            for (j = 1; j < frames; j++) {
                if (used[j] < used[lru]) lru = j;
            }
            mem[lru] = page;
            used[lru] = i;
            page_faults++;
        }
    }

    printf("LU Page Faults: %d, Page Hits: %d\n", page_faults, hits);
    return 0;
}
```

Result:

```
Enter the size of the pages:
```

```
7
```

```
Enter the page strings:
```

```
1303563
```

```
Enter the no of page frames:
```

```
3
```

```
LU Page Faults: 5, Page Hits: 2
```

→Optimal

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

```
int main() {
    int n, frames, i, j, k, page_faults = 0, hits = 0;

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

    char pages[n + 1];
    printf("Enter the page strings:\n");
    scanf("%s", pages);

    printf("Enter the no of page frames:\n");
    scanf("%d", &frames);

    int mem[frames], next_use[frames];
    for (i = 0; i < frames; i++) {
        mem[i] = -1;
    }

    for (i = 0; i < n; i++) {
        int page = pages[i] - '0';
        int found = 0;

        for (j = 0; j < frames; j++) {
            if (mem[j] == page) {
                hits++;
                found = 1;
                break;
            }
        }

        if (!found) {
            if (page_faults < frames) {
                mem[page_faults++] = page;
            } else {
                for (j = 0; j < frames; j++) {
```

```

next_use[j] = -1;
for (k = i + 1; k < n; k++) {
    if (mem[j] == pages[k] - '0') {
        next_use[j] = k;
        break;
    }
}

int farthest = 0;
for (j = 1; j < frames; j++) {
    if (next_use[j] > next_use[farthest]) {
        farthest = j;
    }
}

mem[farthest] = page;
page_faults++;
}
}

printf("Optimal Page Faults: %d, Page Hits: %d\n", page_faults, hits);
return 0;
}

```

Result:

```

Enter the size of the pages:
7
Enter the page strings:
13035631
Enter the no of page frames:
3
Optimal Page Faults: 6, Page Hits: 1

```

LRU and Optimal

include < stdio.h >

include < stdlib.h >

int search (int key, int frames[], int page size)

{ for (int i=0; i<page size; i++)

{

if (frames[i] == key)

return i;

}

> return -1;

>

int findOptimal (int pages[], int frames[], int n, int index,
int page size)

<

int faulted = index - pages - 1;

for (int i=0; i<page size; i++)

{ int j;

for (j=index; j<n; j++)

{ if (frames[j] == pages[j])

if (j > faulted)

return j;

pages[i] = pages[j];

index = j;

,

if (j == n)

return i;

>

PAGE NO. _____
 DATE _____

char (page = 'A' - 'a', frame[0] = 0, i = 0, j = 0, count = 0, hit = 0, len = 0, pos = 0, frame size = 0, frame[1] = 0, frame[2] = 0, frame[3] = 0, frame[4] = 0, frame[5] = 0, frame[6] = 0, frame[7] = 0, frame[8] = 0, frame[9] = 0, frame[10] = 0, frame[11] = 0, frame[12] = 0, frame[13] = 0, frame[14] = 0, frame[15] = 0, frame[16] = 0, frame[17] = 0, frame[18] = 0, frame[19] = 0, frame[20] = 0, frame[21] = 0, frame[22] = 0, frame[23] = 0, frame[24] = 0, frame[25] = 0, frame[26] = 0, frame[27] = 0, frame[28] = 0, frame[29] = 0, frame[30] = 0, frame[31] = 0, frame[32] = 0, frame[33] = 0, frame[34] = 0, frame[35] = 0, frame[36] = 0, frame[37] = 0, frame[38] = 0, frame[39] = 0, frame[40] = 0, frame[41] = 0, frame[42] = 0, frame[43] = 0, frame[44] = 0, frame[45] = 0, frame[46] = 0, frame[47] = 0, frame[48] = 0, frame[49] = 0, frame[50] = 0, frame[51] = 0, frame[52] = 0, frame[53] = 0, frame[54] = 0, frame[55] = 0, frame[56] = 0, frame[57] = 0, frame[58] = 0, frame[59] = 0, frame[60] = 0, frame[61] = 0, frame[62] = 0, frame[63] = 0, frame[64] = 0, frame[65] = 0, frame[66] = 0, frame[67] = 0, frame[68] = 0, frame[69] = 0, frame[70] = 0, frame[71] = 0, frame[72] = 0, frame[73] = 0, frame[74] = 0, frame[75] = 0, frame[76] = 0, frame[77] = 0, frame[78] = 0, frame[79] = 0, frame[80] = 0, frame[81] = 0, frame[82] = 0, frame[83] = 0, frame[84] = 0, frame[85] = 0, frame[86] = 0, frame[87] = 0, frame[88] = 0, frame[89] = 0, frame[90] = 0, frame[91] = 0, frame[92] = 0, frame[93] = 0, frame[94] = 0, frame[95] = 0, frame[96] = 0, frame[97] = 0, frame[98] = 0, frame[99] = 0)

void simulate (char page[], int n, int frame size)

{
 int frame [frame size], count = 0, hit = 0;
 for (int i = 0; i < frame size; i++)
 frame[i] = -1;
 pos (int i = 0, int j = 0, int t = 0)
 if (count (page[t], frame, frame size) == -1)
 {
 int index = -1;
 for (int j = 0; j < frame size; j++)
 if (frame[j] == -1)
 index = j;
 frame [index] = page[t];
 count++;
 }
 else
 {
 int replace index = find (page, frame, n, i + 1);
 frame [replace index] = page[i];
 count++;
 }
 else
 hit++;
 }

cout << "After " << hit << endl;

PAGE NO :
DATE :

printf ("Optimal page faults : %d, page hits %d\n", count, hits);

int main ()

{

int n, framesize;

printf ("Enter the size of the pages : ");

scanf ("%d", & n);

printf ("Enter the page sizes : ");

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

scanf ("%d", & pages[i]);

printf ("Enter the no. of page frames : ");

scanf ("%d", & framesize);

simulatedOptimal (pages, n, framesize);

simulatedLRU (pages, n, framesize);

return 0;

}

Output

Enter the size of pages : 12

Enter the page sizes : 0 12 03 04 23 03

Enter No. of frames : 5

Optimal page Faults : 7, page hits : 5

LRU page Faults : 9, page hits : 5.