VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

OPERATING SYSTEMS

Submitted by

Varshitha B (1WA23CS035)

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 Varshitha B (1WA23CS035), 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

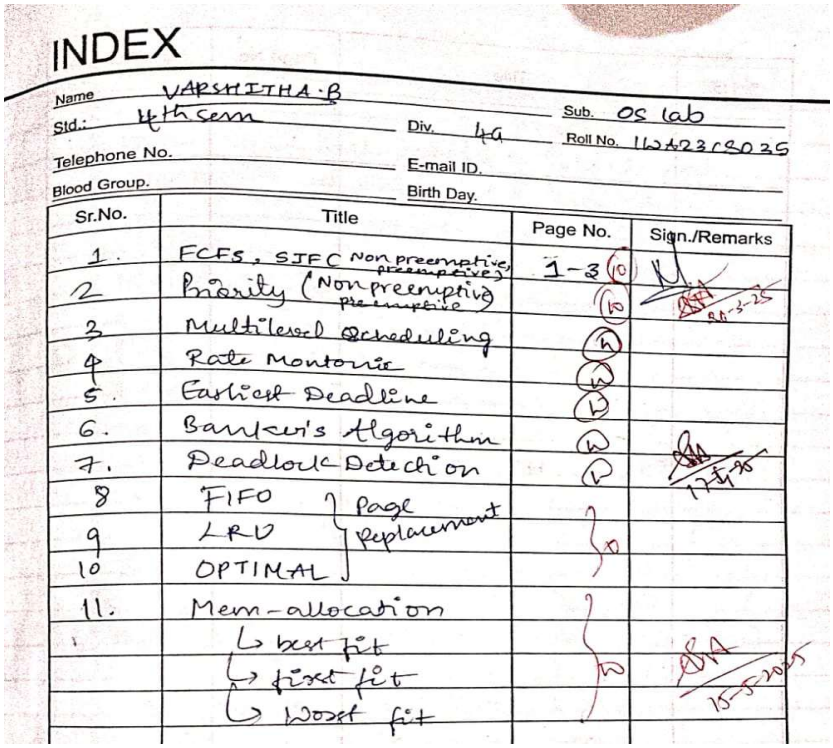
Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

10.	Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) Optimal	69-80

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.



INDEX

Name VARSHITHA.B
Std.: 4th sem Div. 4G Sub. OS lab Roll No. 1BM23CS035
Telephone No. E-mail ID.
Blood Group. Birth Day.

Sr.No.	Title	Page No.	Sign./Remarks
1.	FCFS, SJFC Non preemptive, preemptive	1-3	
2	Priority (Non preemptive, preemptive)		
3	Multilevel Scheduling		
4	Rate Monotonic		
5.	Earliest Deadline		
6.	Banker's Algorithm		
7.	Deadlock Detection		
8	FIFO (Page Replacement)		
9	LRU		
10	OPTIMAL		
11.	Mem-allocation		
	↳ best fit		
	↳ first fit		
	↳ Worst fit		

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:

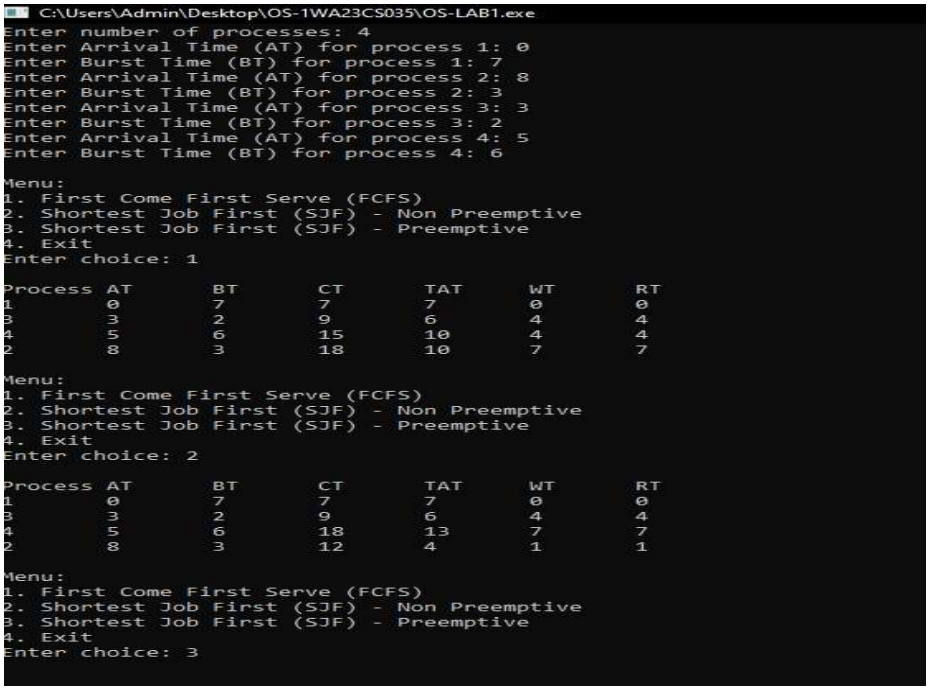
```
#include <stdio.h>
#include <stdlib.h>
struct process {
   int id, AT, BT, CT, TAT, WT, RT, remaining_BT;
   int completed;
};
void sort_by_AT(struct process p[], int n) {
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (p[i].AT > p[j].AT) {
            struct 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:



SURYA Gold

Date _____ Page 1

1 Write a C program to simulate the following CPU scheduling to find turnaround time and waiting time

i) FCFS
ii) SJF → preemptive / nonpreemptive

i) FCFS

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

(SJF non preemptive)

```
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);
```

SURYA Gold

Date _____ Page _____

```
  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 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);
  }
}
```

SURYA Gold

Date _____ Page 2

iv)

```
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[shorted].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++;
}
}
}
```

SURYA Gold

Date _____ Page _____

iii)

```
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].remainingBT--;
```

```
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 ++;
}
}
}
}
```

Output:-

```
Enter number of processes: 3
Enter Arrival Time (AT) for process 1 : 0
Enter Burst Time (BT) for process 1 : 7
Enter Arrival Time (AT) for process 2 : 0
Enter Burst Time (BT) for process 2 : 3
Enter Arrival Time (AT) for process 3 : 0
Enter Burst Time (BT) for process 3 : 4

Menu :
1. First Come First serve (FCFS)
2. Shortest JOP First (SJF) - Non preemptive
3. Shortest Job First (SJF) - preemptive
4. Exit
Enter Choice : 1
```

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

```
Menu :
1
2
3
4
Enter choice : 2
```

Process	AT	BT	CT	TAT	WT	RT
1	0	7	14	14	7	7
2	0	3	3	3	0	0
3	0	4	7	7	3	3

```
Menu :
1
2
3
4
Enter choice : 3
```

Process	AT	BT	CT	TAT	WT	RT
1	0	7	14	14	7	7
2	0	3	3	3	0	0
3	0	4	7	7	3	3

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)

Code:

```
//Priority scheduling
#include <stdio.h>

#define MAX 10

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

// Function for Non-Preemptive Priority Scheduling
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 RT is not yet calculated, calculate it
```

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

        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++;
    }
}

// Function for Preemptive Priority Scheduling
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 RT is not yet calculated, calculate it
        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++;
        }
    }
}

// Function to display the results of processes
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\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;

    // Asking the user for the number of processes
```

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

// Getting arrival times, burst times, and priorities for each process
for (int i = 0; i < n; i++) {
    p[i].pid = i + 1; // Process ID starts from 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;  // Initialize remaining burst time
    p[i].is_completed = 0;        // Mark process as incomplete
    p[i].rt = -1;                 // Response time will be calculated later
}

// Menu to choose the scheduling method
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:

```
C:\Users\BMSCE\Desktop\OS\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): 3

Enter Arrival Time, Burst Time, and Priority for Process 3:
Arrival Time: 2
Burst Time: 1
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): 5

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
3. Exit
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       3               15      14      11      11
3       2       1       4               12      10      9       9
4       3       5       5               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
3. Exit
Enter your choice: 2
Preemptive Scheduling Completed!

PID     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
```

using Priority Scheduling.

Higher the number - Higher the Priority

SURYA Gold

Date 13/08/25

```
#include<stdio.h> #define MAX 10
typedef struct
{
int pid, at, bt, pt, remaining_bt, ct, tat,
    wt, rt, is_completed, st;
} Process;                    // st = start time

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;
p[selected].rt = time - p[selected].at;
// Response Time = Start Time - Arrival Time
}
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;
}
```

SURYA Gold
Date Page

```
if (p[selected].st == -1)
{
p[selected].st = time;
p[selected].st = time - p[selected].at;
}
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\t
       TAT\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
       \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);
```

SURYA Gold
Date Page

```
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);
}
```

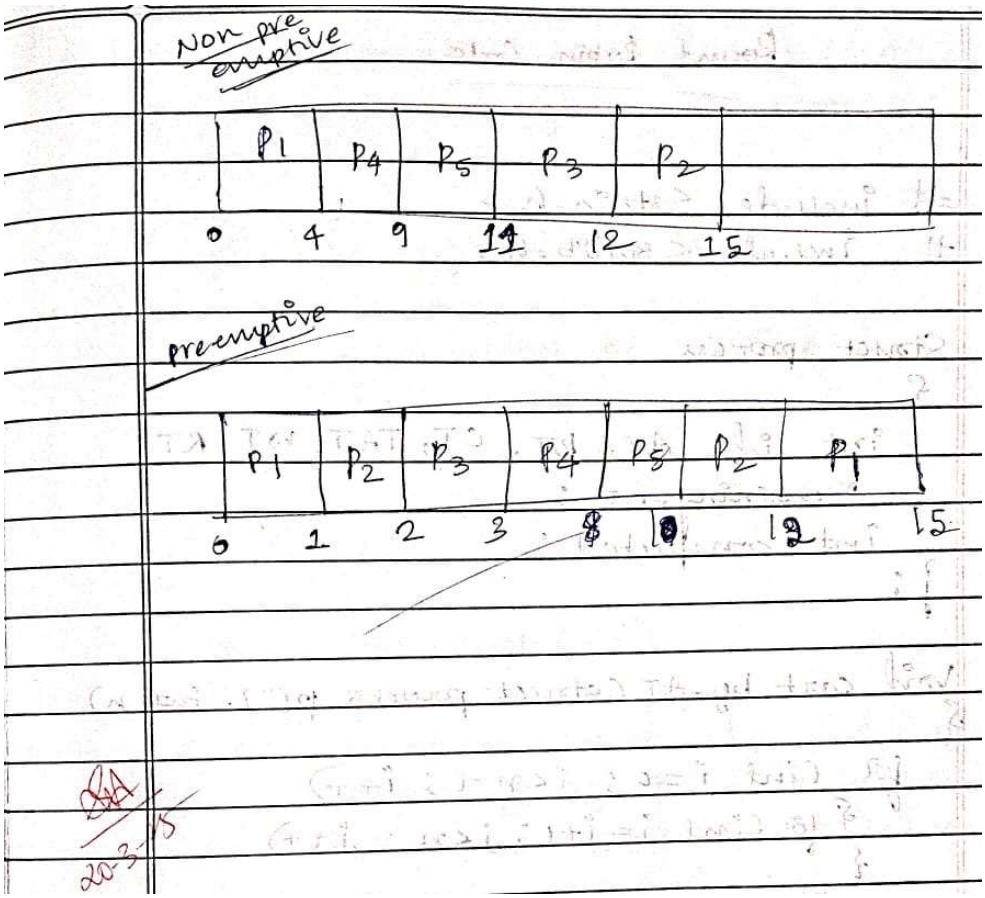
output:

Non pre emptive scheduling

Process	AT	BT	P	CT	TAT	WT	RT
P_1	0	4	2	4	4	0	0
P_2	1	3	3	15	14	11	11
P_3	2	1	4	12	10	9	9
P_4	3	5	5	9	6	1	1
P_5	4	2	5	11	7	5	5

pre emptive scheduling

Process	AT	BT	P	CT	TAT	WT	RT
P_1	0	4	2	15	15	11	0
P_2	1	3	3	12	11	8	11
P_3	2	1	4	3	1	0	9
P_4	3	5	5	8	5	0	1
P_5	4	2	5	10	6	4	5



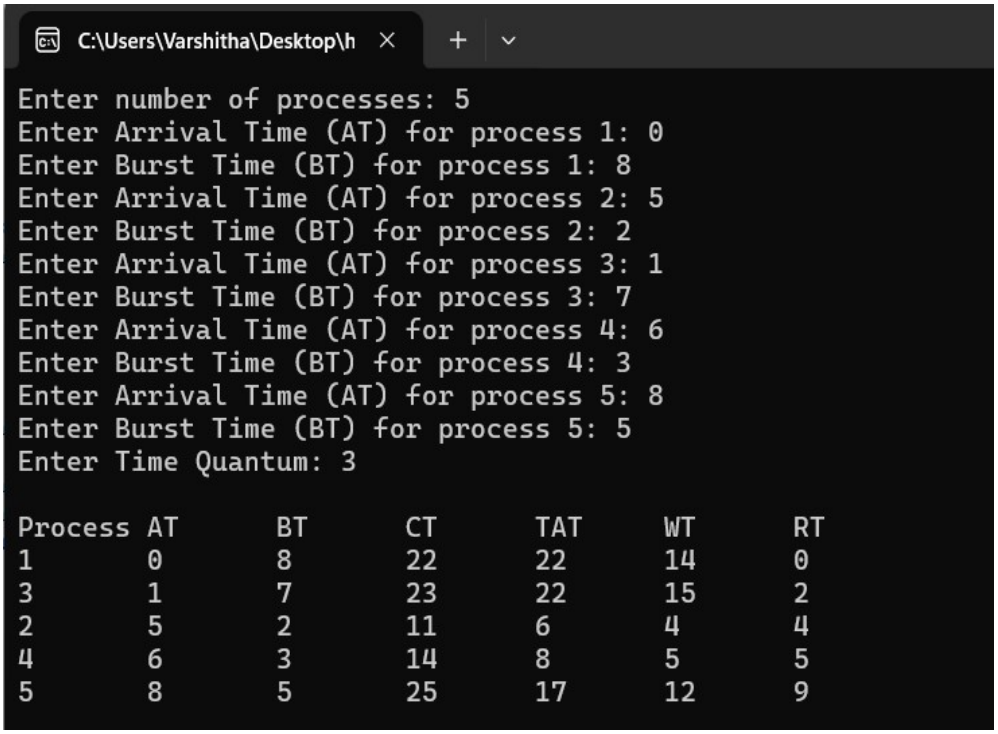

```
//round robin code
 #include <stdio.h>
#include <stdlib.h>
struct process {
    int id, AT, BT, CT, TAT, WT, RT, remaining_BT;
    int completed;
};
void sort_by_AT(struct process p[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (p[i].AT > p[j].AT) {
                struct process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
```

```
        }
    }
}
void calculate_RR(struct process p[], int n, int time_quantum) {
    sort_by_AT(p, n);
    int currentTime = 0, completed = 0;
    int queue[n], front = 0, rear = 0;
    int visited[n];
    for (int i = 0; i < n; i++) {
        p[i].remaining_BT = p[i].BT;
        visited[i] = 0;
    }
    queue[rear++] = 0;
    visited[0] = 1;

    while (completed < n) {
        int idx = queue[front++];
        if (p[idx].remaining_BT == p[idx].BT)
            p[idx].RT = currentTime - p[idx].AT;
        if (p[idx].remaining_BT > time_quantum) {
            currentTime += time_quantum;
            p[idx].remaining_BT -= time_quantum;
        } else {
            currentTime += p[idx].remaining_BT;
            p[idx].remaining_BT = 0;
            p[idx].CT = currentTime;
            p[idx].TAT = p[idx].CT - p[idx].AT;
            p[idx].WT = p[idx].TAT - p[idx].BT;
            completed++;
        }
        for (int i = 0; i < n; i++) {
```

```
            if (!visited[i] && p[i].AT <= currentTime && p[i].remaining_BT > 0) {
                queue[rear++] = i;
                visited[i] = 1;
            }
        }
        if (p[idx].remaining_BT > 0)
            queue[rear++] = idx;
    }
}
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, time_quantum;
    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;
    }
    printf("Enter Time Quantum: ");
    scanf("%d", &time_quantum);
```

```
    calculate_RR(p, n, time_quantum);
    display(p, n);
    return 0;
}
```

Output:



Round Robin 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_RR (struct process p[],
  int n, int time_quantum)
{
  sort_by_AT (p, n);
  int currentTime = 0, completed = 0;
  int queue[n], front = 0, rear = 0;
  int visited[n];

  for (int i=0; i<n; i++)
    {
      p[i].remaining_BT = p[i].BT;
      visited[i] = 0;
    }
  queue[rear++] = 0; // Enqueue
  visited[0] = 1;

  while (completed < n)
  {
    int idx = queue[front++]; // Dequeue
    if(p[idx].remaining_BT == p[idx].BT)
      p[idx].RT = currentTime - p[idx].AT;

    if (p[idx].remaining_BT > time_quantum)
      {
        currentTime += time_quantum;
        p[idx].remaining_BT -= time_quantum;
      }
    else
    {
      currentTime += p[idx].remaining_BT;
      p[idx].remaining_BT = 0;
      p[idx].CT = currentTime;
      p[idx].TAT = p[idx].CT - p[idx].AT;
      p[idx].WT = p[idx].TAT - p[idx].BT;
      completed++;
    }
```

```
for (int i=0 ; i<n ; i++)
{
  if (!visited[i] && p[i].AT <= currentTime &&
      p[i].remaining_BT > 0)
  {
    queue[rear++] = i;  // enqueue
    visited[i] = 1;
  }
}
if (p[idx].remaining_BT > 0)
  queue[rear++] = idx;
}

void display(struct process p[], int n)
{
printf("\n Process \t AT \t BT \t CT \t TAT \t WT \t
        RT \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);
}
}
```

```
Enter the no of processes: 5
Enter Arrival Time (AT) for process 1: 0
Enter Burst Time (BT) for process 1: 8
Enter Arrival Time (AT) for process 2: 5
Enter Burst Time (BT) for process 2: 7
Enter Arrival Time (AT) for process 3: 1
Enter Burst Time (BT) for process 3: 7
Enter Arrival Time (AT) for process 4: 6
Enter Burst Time (BT) for process 4: 3
Enter Arrival Time (AT) for process 5: 8
Enter Burst Time (BT) for process 5: 5
Enter time Quantum: 3
```

Process	AT	BT	CT	TAT	WT	RT
P1	0	8	22	22	14	0
P2	5	7	23	18	11	4
P3	1	7	21	20	13	2
P4	6	3	14	8	5	5
P5	8	5	25	17	12	9

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.

Code:

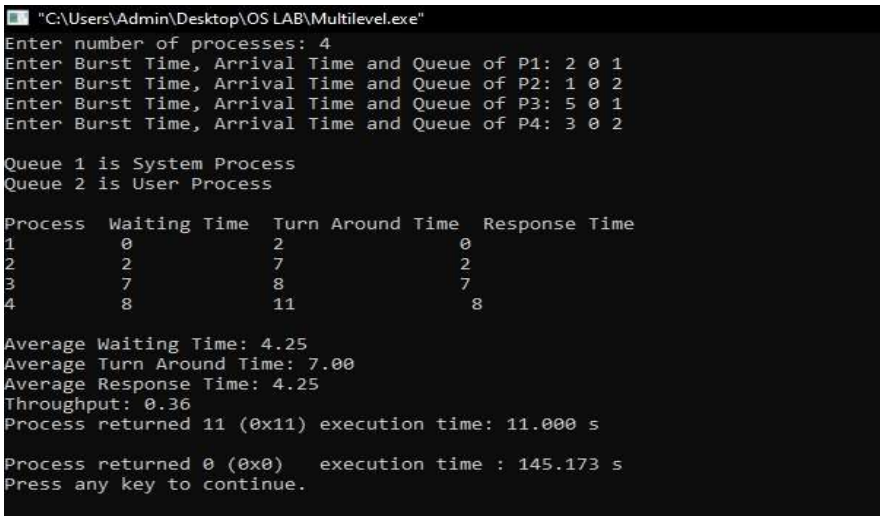
```
#include <stdio.h>

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

    return 0;
}
```

Output:



Multiqueue Scheduling

```
#include <stdio.h>
#define MAX_PROCESS 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 (process[i].remaining_time >
            time_quantum)
      {
        *time += time_quantum;
        process[i].remaining_time -= time_
                                    quantum;
      }
```

SURYA Gold
Date ____ Page ____

```
      else
      {
        *time += process[i].remaining_time;
        processes[i].waiting_time = *time -
            process[i].arrival_time - process[i].burst_
                                    time;
        process[i].turnaround_time = *time -
            process[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 = process[i].arrival_time;
    }
    processes[i].waiting_time = *time -
        process[i].arrival_time;
    processes[i].turnaround_time =
        process[i].waiting_time + process[i].burst_
                                    time;
    processes[i].response_time = process[i].
                                    waiting_time;
    *time += process[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
,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]
}}
```

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  waiting time  turnaround time  Response time
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
```

Program -4

Question:

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

a) Rate- Monotonic
b) Earliest-deadline First

a)

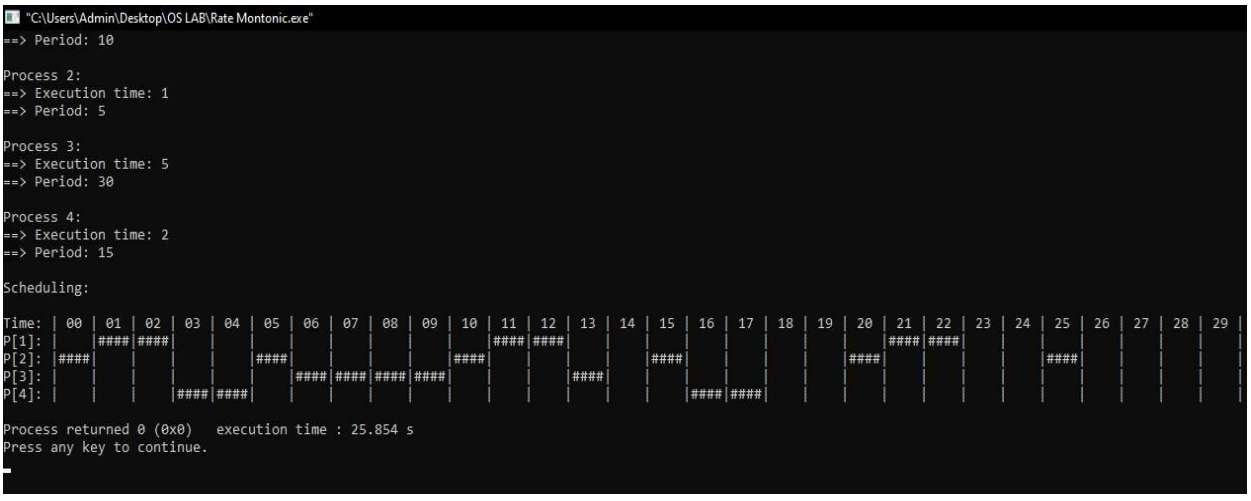
Code:

```
// 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:



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 no of process
        (maximum %d): ", MAX_PROCESS);
scanf("%d", &num_of_process);

if(num_of_process < 1)
  {
    exit(0);
  }
for(int i=0; i<num_process; i++)
{
printf("\n Process %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("\n Scheduling :\n\n");
printf("Time:");
for(int i=0; i<cycles; i++)
  {
    if(i<10)
      printf("| 0%d ", i);
    else
      printf("| %d ", i);
  }
```

SURYA Gold

Date Page

simplify

```
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("In Given 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;
      }
    }
  }
  printfschedule(process_list, time);
}
```

SURYA Gold
Date Page

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

Output - Enter the total number of processes (maximum
70) : 4

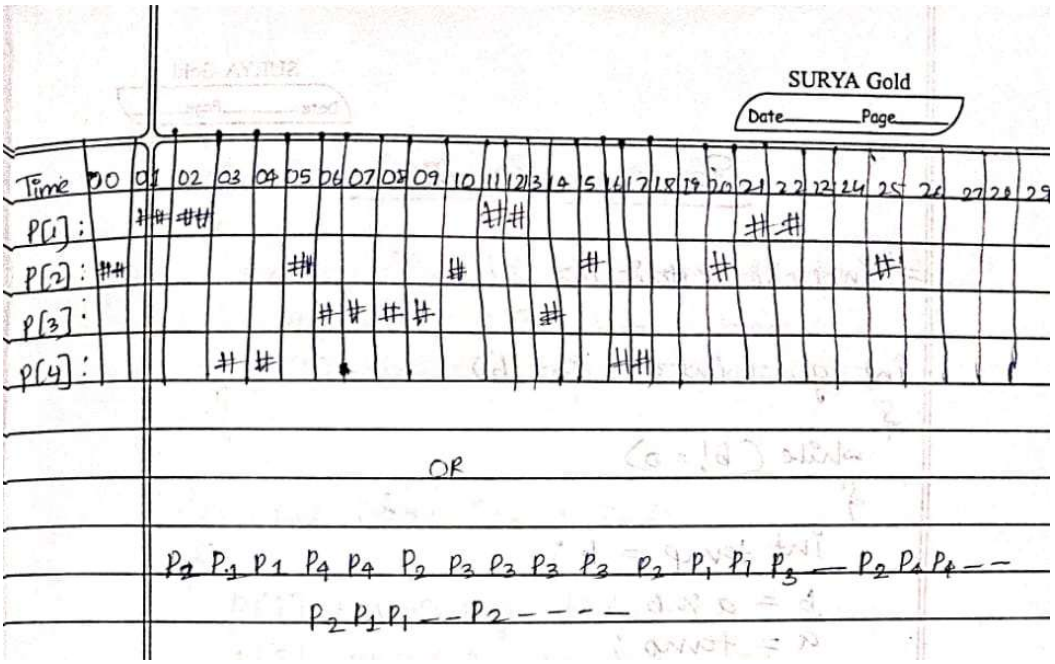
Process 1 :
=> Execution time :
=> Period : 10

Process 2 :
=> Execution time : 1
=> Period : 5

Process 3
=> Execution time : 5
=> Period : 10

Process 4
=> Execution time : 2
=> Period : 15

Scheduling :



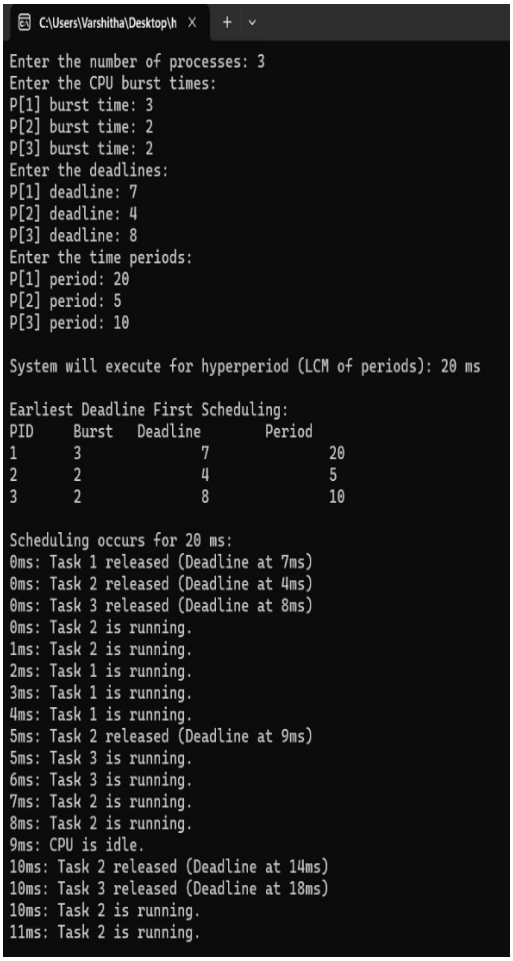
b)

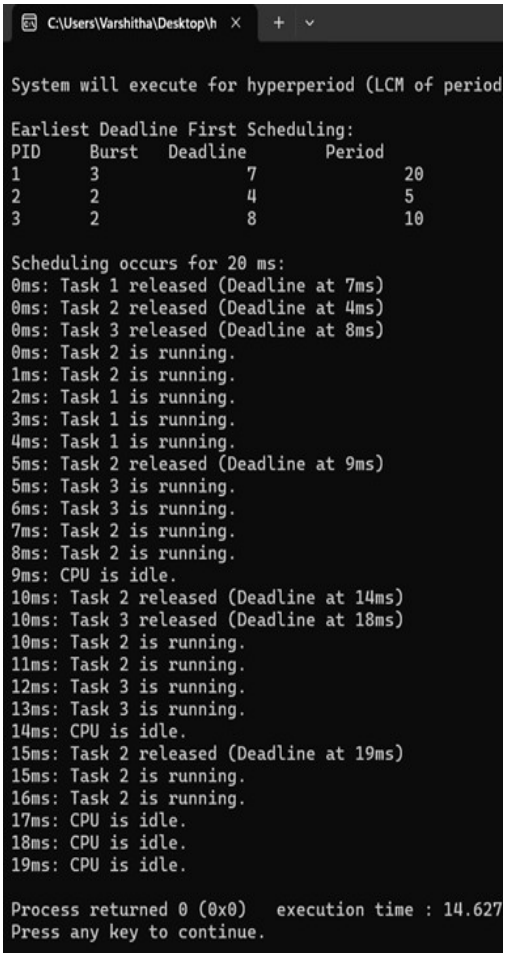
```
// Earliest Deadline
#include <stdio.h>

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;
   int remaining_time;
   int absolute_deadline;
};

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

```
    }

    // Initialization
    for (int i = 0; i < n; i++) {
        p[i].remaining_time = 0;
        p[i].absolute_deadline = -1;
    }

    printf("\nScheduling occurs for %d ms:\n", time_limit);

    for (int time = 0; time < time_limit; time++) {
        // Release new jobs if period matches
        for (int i = 0; i < n; i++) {
            if (time % p[i].period == 0) {
                p[i].remaining_time = p[i].burst_time;
                p[i].absolute_deadline = time + p[i].deadline;
                printf("%dms: Task %d released (Deadline at %dms)\n", time, p[i].id,
p[i].absolute_deadline);
            }
        }
        int earliest = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].remaining_time > 0) {
                if (earliest == -1 || p[i].absolute_deadline < p[earliest].absolute_deadline) {
                    earliest = i;
                }
            }
        }
        if (earliest != -1) {
            printf("%dms: Task %d is running.\n", time, p[earliest].id);
            p[earliest].remaining_time--;
        } else {
            printf("%dms: CPU is idle.\n", 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++) {
```

```
        printf("P[%d] burst time: ", i + 1);
        scanf("%d", &processes[i].burst_time);
        processes[i].id = i + 1;
    }

    printf("Enter the deadlines:\n");
    for (int i = 0; i < n; i++) {
        printf("P[%d] deadline: ", i + 1);
        scanf("%d", &processes[i].deadline);
    }

    printf("Enter the time periods:\n");
    for (int i = 0; i < n; i++) {
        printf("P[%d] period: ", i + 1);
        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\Varshitha\Desktop\h
Enter the number of processes: 3
Enter the CPU burst times:
P[1] burst time: 3
P[2] burst time: 2
P[3] burst time: 2
Enter the deadlines:
P[1] deadline: 7
P[2] deadline: 4
P[3] deadline: 8
Enter the time periods:
P[1] period: 20
P[2] period: 5
P[3] period: 10
System will execute for hyperperiod (LCM of periods): 20 ms
Earliest Deadline First Scheduling:
PID Burst Deadline Period
1 3 7 20
2 2 4 5
3 2 8 10
Scheduling occurs for 20 ms:
0ms: Task 1 released (Deadline at 7ms)
0ms: Task 2 released (Deadline at 4ms)
0ms: Task 3 released (Deadline at 8ms)
0ms: Task 2 is running.
1ms: Task 2 is running.
2ms: Task 1 is running.
3ms: Task 1 is running.
4ms: Task 1 is running.
5ms: Task 2 released (Deadline at 9ms)
5ms: Task 3 is running.
6ms: Task 3 is running.
7ms: Task 2 is running.
8ms: Task 2 is running.
9ms: CPU is idle.
10ms: Task 2 released (Deadline at 14ms)
10ms: Task 3 released (Deadline at 18ms)
10ms: Task 2 is running.
11ms: Task 2 is running.


C:\Users\Varshitha\Desktop\h
System will execute for hyperperiod (LCM of period
Earliest Deadline First Scheduling:
PID Burst Deadline Period
1 3 7 20
2 2 4 5
3 2 8 10
Scheduling occurs for 20 ms:
0ms: Task 1 released (Deadline at 7ms)
0ms: Task 2 released (Deadline at 4ms)
0ms: Task 3 released (Deadline at 8ms)
0ms: Task 2 is running.
1ms: Task 2 is running.
2ms: Task 1 is running.
3ms: Task 1 is running.
4ms: Task 1 is running.
5ms: Task 2 released (Deadline at 9ms)
5ms: Task 3 is running.
6ms: Task 3 is running.
7ms: Task 2 is running.
8ms: Task 2 is running.
9ms: CPU is idle.
10ms: Task 2 released (Deadline at 14ms)
10ms: Task 3 released (Deadline at 18ms)
10ms: Task 2 is running.
11ms: Task 2 is running.
12ms: Task 3 is running.
13ms: Task 3 is running.
14ms: CPU is idle.
15ms: Task 2 released (Deadline at 19ms)
15ms: Task 2 is running.
16ms: Task 2 is running.
17ms: CPU is idle.
18ms: CPU is idle.
19ms: CPU is idle.
Process returned 0 (0x0) execution time : 14.627
Press any key to continue.

SURYA Gold

Date _____ Page _____

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;
  int remaining_time;
  int absolute_deadline;
};

void earliest_deadline_first(Struct
  Process p[], int n, int time_limit)
{
  printf("\n Earliest Deadline First
    Scheduling:\n");
  printf("PID\t Burst\t Deadline\t
    Period\n");
```

SURYA Gold

Date _____ Page _____

```
for (int i=0; i<n; i++)
{
  printf("%d\t %d\t\t %d\t\t %d\n",
    p[i].id, p[i].burst_time,
    p[i].deadline, p[i].period);
}

for (int i=0; i<n; i++)
{
  p[i].remaining_time = 0;
  p[i].absolute_deadline = -1;
}

printf("\n Scheduling occurs for %d
  ms:\n", time_limit);

for (int time=0; time<time_limit;
    time++)
{
  for (int i=0; i<n; i++)
  {
  if (time % p[i].period == 0)
{
    p[i].remaining_time = p[i].burst_time;
    p[i].absolute_deadline = time +
                p[i].deadline;
    printf("%d ms: Task %d released
      (Deadline at %dms)\n", time,
      p[i].id, p[i].absolute_deadline);
  }
}
int earliest = -1;
for (int i=0; i<n; i++)
```

```
if (p[i].remaining_time > 0)
{
    if (earliest == -1 || p[i].absolute_
        deadline < p[earliest].
        absolute_deadline)
    {
        earliest = i;
    }
}
if (earliest != -1)
{
    printf("%d ms: Task %d is
        running.\n", time,
        p[earliest].id);
    p[earliest].remaining_time--;
}
else {
    printf("%d ms: CPU is idle.\n", time);
}
}

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

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

printf("Enter the deadlines:\n");
for(int i=0; i<n; i++)
{
printf("P[%d] deadline:", i+1);
scanf("%d", &processes[i].deadline);
}

printf("Enter the time periods:\n");
for(int i=0; i<n; i++)
{
printf("P[%d] period:", i+1);
scanf("%d", &processes[i].period);
}

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

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

earliest_deadline_first(processes, n,
    hyperperiod);
return 0;
```

SURYA Gold
Date Page

Output:

Enter the number of processes: 3
Enter the CPU burst times:
P[1] burst time: 3
P[2] burst time: 2
P[3] burst time: 2
Enter the deadlines:
P[1] deadline: 7
P[2] deadline: 4
P[3] deadline: 8
Enter the time periods:
P[1] period: 20
P[2] period: 5
P[3] period: 10

System will execute for hyperperiod (LCM of periods): 20 ms

Earliest Deadline First Scheduling:

PID	Burst	Deadline	Period
1	3	7	20
2	2	4	5
3	2	8	10

Scheduling occurs for 20 ms:

0ms: Task 1 released (Deadline at 7ms)
0ms: Task 2 released (Deadline at 4ms)
0ms: Task 3 released (Deadline at 8ms)
0ms: Task 2 is running
1ms: Task 2 is running

SURYA Gold
Date Page

2ms: Task 2 is running
3ms: Task 1 is running
4ms: Task 1 is running
5ms: Task 2 released (Deadline at 9ms)
5ms: Task 3 is running
6ms: Task 3 is running
7ms: Task 2 is running
8ms: Task 2 is running
9ms: CPU is idle
10ms: Task 2 released (Deadline at 14ms)
10ms: Task 3 released (Deadline at 18ms)
10ms: Task 2 is running
11ms: Task 2 is running
12ms: Task 3 is running
13ms: Task 3 is running
14ms: CPU is idle
15ms: Task 2 released (Deadline at 19ms)
15ms: Task 2 is running
16ms: Task 2 is running
17ms: CPU is idle
18ms: CPU is idle
19ms: CPU is idle

Program -5

Question:

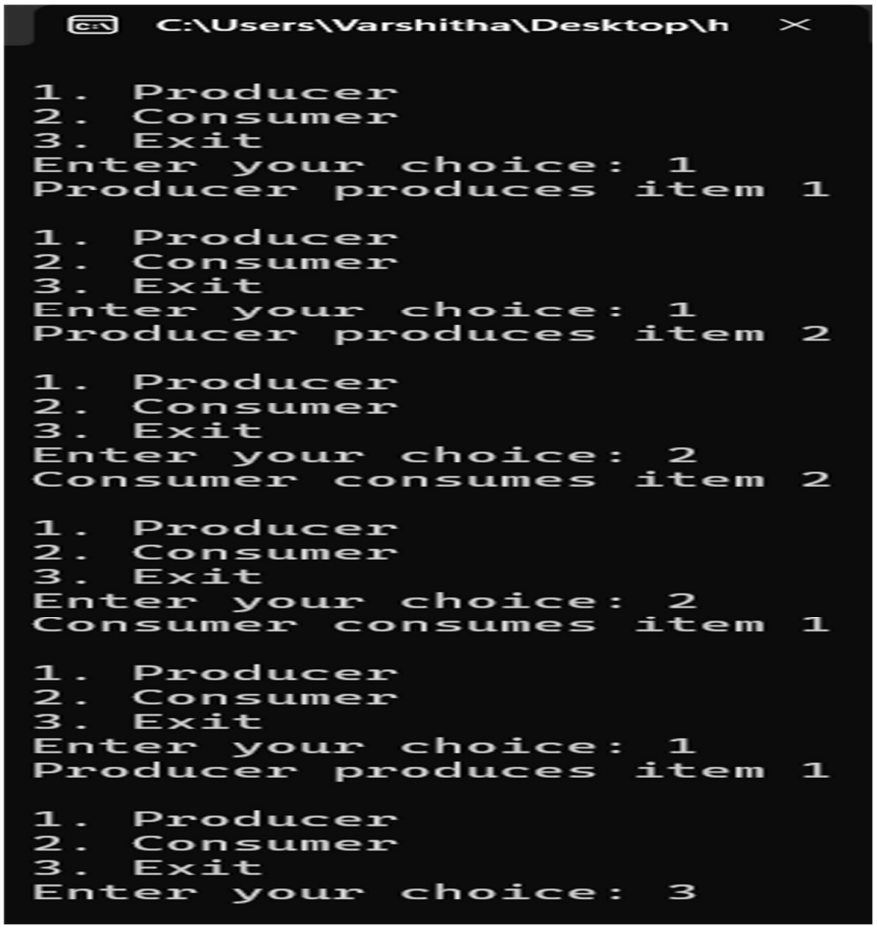
Write a C program to simulate producer-consumer problem using semaphores.

Code:

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 3;
int x = 0;
int wait(int s) {
   return (--s);
}
int signal(int s) {
   return (++s);
}
void producer() {
   mutex = wait(mutex);
   full = signal(full);
   empty = wait(empty);
   x++;
   printf("Producer produces item %d\n", x);
   mutex = signal(mutex);
}
void consumer() {
   mutex = wait(mutex);
   full = wait(full);
   empty = signal(empty);
   printf("Consumer consumes item %d\n", x);
   x--;
   mutex = signal(mutex);
}
int main() {
   int choice;
```

```
    while (1) {
        printf("\n1. Producer\n2. Consumer\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            if ((mutex == 1) && (empty != 0))
                producer();
            else
                printf("Buffer is full!\n");
            break;

        case 2:
            if ((mutex == 1) && (full != 0))
                consumer();
            else
                printf("Buffer is empty!\n");
            break;
        case 3:
            exit(0);
        default:
            printf("Invalid choice!\n");
        }
    }
    return 0;
}
```

Output:


C:\Users\Varshitha\Desktop\h
1. Producer
2. Consumer
3. Exit
Enter your choice: 1
Producer produces item 1
1. Producer
2. Consumer
3. Exit
Enter your choice: 1
Producer produces item 2
1. Producer
2. Consumer
3. Exit
Enter your choice: 2
Consumer consumes item 2
1. Producer
2. Consumer
3. Exit
Enter your choice: 2
Consumer consumes item 1
1. Producer
2. Consumer
3. Exit
Enter your choice: 1
Producer produces item 1
1. Producer
2. Consumer
3. Exit
Enter your choice: 3

SURYA Gold

Date _____ Page _____

Producer & Consumer

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 3;
int x = 0;
int wait (int s)
  { return (--s);
  }
int signal (int s){
    return (++s);
  }
void producer (){
  mutex = wait (mutex);
  full = signal (full);
  empty = wait (empty);
  x++;
  printf ("Producer produces item %d \n", x);
  mutex = signal (mutex);
}

void consumer ()
{
mutex = wait (mutex);
full = wait (full);  // Decrement
empty = signal (empty); // Increment
printf (" consumer consumes item %d \n"
          x);
x--;
mutex = signal (mutex);
}
```

SURYA Gold

Date _____ Page _____

```
int main ()
{
  int choice;
  while (1)
  {
  printf ("\n1. Producer \n2. Consumer
        \n3. Exit\n");
  printf (" Enter your choice :");
  scanf ("%d", &choice);
  switch (choice)
  {
  case 1 :
   if (mutex == 1) && (empty != 0))
        producer ();
   else
      printf (" Buffer is full!\n");
   break;

  Case 2 :
   if ((mutex == 1) && (full != 0))
     consumer ();
   else
     printf ("Buffer is empty!\n");
   break;

  Case 3 :
    exit (0);

  default :
    printf (" Invalid choice!\n");
  }
 }
 return 0;
}
```

SURY

Date

1. Producer
2. Consumer
3. Exit
Enter your choice : 1
Producer produces item 1

1. Producer
2. Consumer
3. Exit
Enter your choice : 1
Producer produces item 2

1. Producer
2. Consumer
3. Exit
Enter your choice : 2
Consumer consumes item 2

1. Producer
2. Consumer
3. Exit
Enter your choice : 2
Consumer consumes item 1

1. Producer
2. Consumer
3. Exit
Enter your choice : 3

Program -6

Question

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

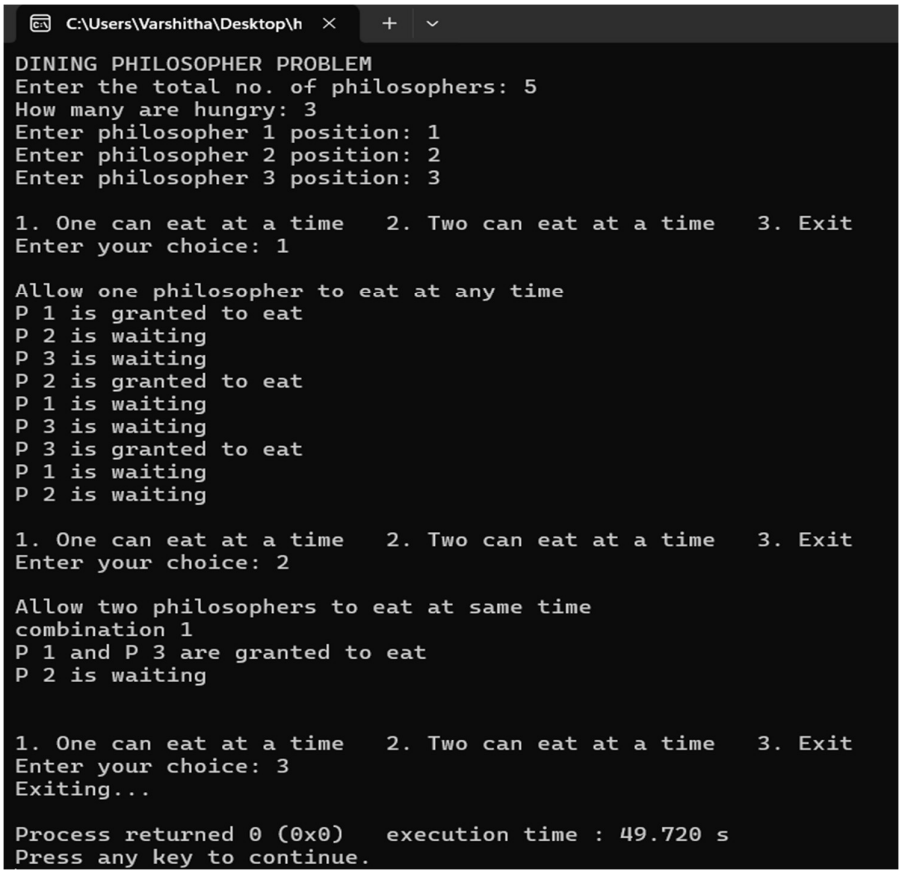
Code

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
int totalPhilosophers;
int hungry[MAX];
int areNeighbors(int a, int b) {
   return (abs(a - b) == 1 || abs(a - b) == totalPhilosophers - 1);}
void option1(int count) {
   printf("\nAllow one philosopher to eat at any time\n");
   for (int i = 0; i < count; i++) {
      printf("P %d is granted to eat\n", hungry[i]);
      for (int j = 0; j < count; j++) {
         if (j != i) {
            printf("P %d is waiting\n", hungry[j]);
         }
      }
   }
}
void option2(int count) {
   printf("\nAllow two philosophers to eat at same time\n");
   int combination = 1;
   for (int i = 0; i < count; i++) {
      for (int j = i + 1; j < count; j++) {
         if (!areNeighbors(hungry[i], hungry[j])) {
            printf("combination %d\n", combination++);
            printf("P %d and P %d are granted to eat\n", hungry[i], hungry[j]);
            for (int k = 0; k < count; k++) {
```

```
                if (k != i && k != j) {
                    printf("P %d is waiting\n", hungry[k]);
                }
            }
            printf("\n");
        }
    }
    if (combination == 1) {
        printf("No combinations found where two non-neighbor philosophers can eat.\n");
    }
}
int main() {
    int hungryCount;
    printf("DINING PHILOSOPHER PROBLEM\n");
    printf("Enter the total no. of philosophers: ");
    scanf("%d", &totalPhilosophers);
    printf("How many are hungry: ");
    scanf("%d", &hungryCount);
    for (int i = 0; i < hungryCount; i++) {
        printf("Enter philosopher %d position: ", i + 1);
        scanf("%d", &hungry[i]);
    }
    int choice;
    do {
        printf("\n1. One can eat at a time   2. Two can eat at a time   3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            option1(hungryCount);
            break;
        case 2:
```

```
            option2(hungryCount);
            break;
        case 3:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 3);
    return 0;
}
```

Output:



Dining Philosopher

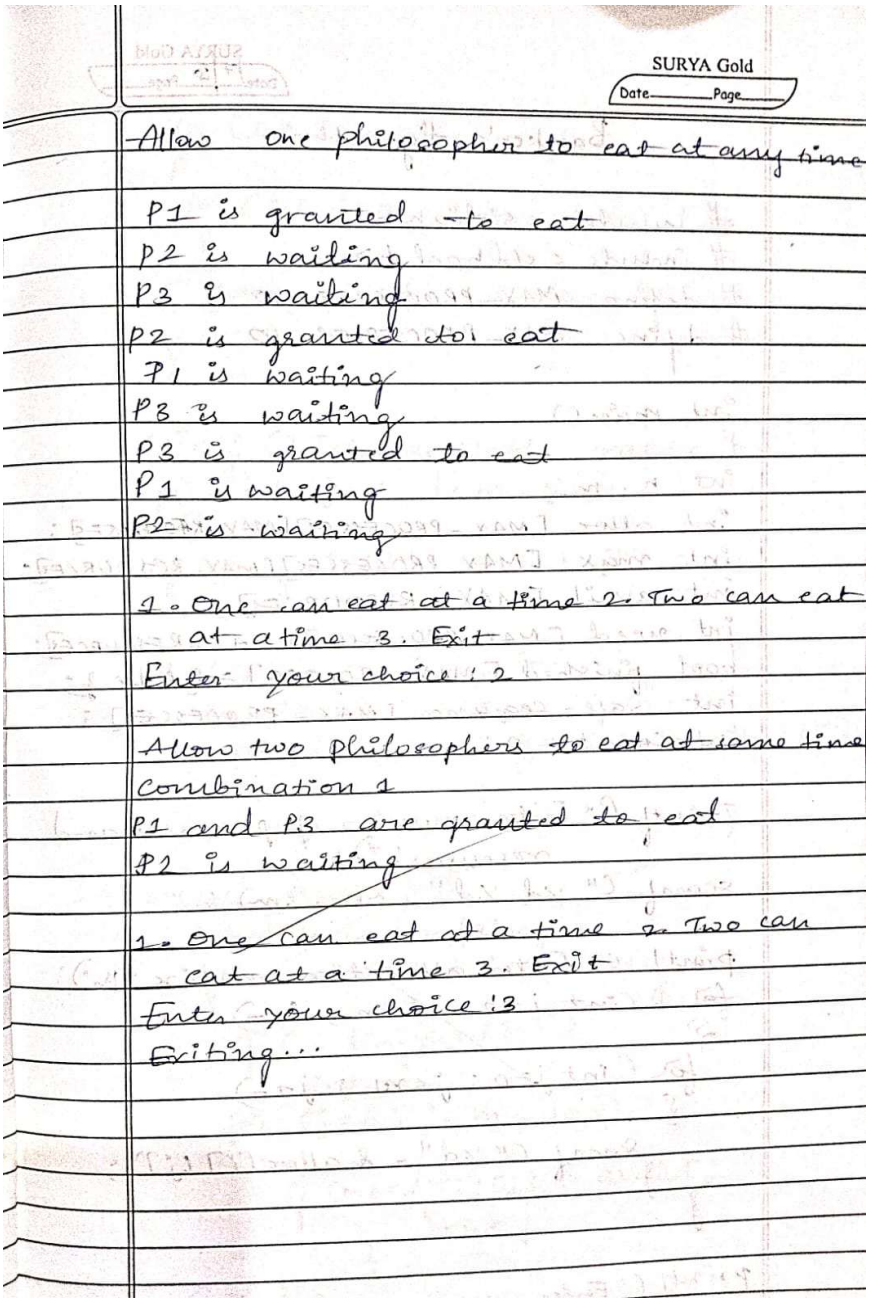
```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10

int totalPhilosophers;
int hungry[MAX];

int areNeighbors(int a, int b)
{
    return (abs(a-b) == 1 || abs(a-b) ==
            totalPhilosophers-1);
}

void option1(int count)
{
    printf("\nAllow one philosopher to
            eat at any time\n");
    for(int i=0; i<count; i++)
    {
        printf("P %d is granted to eat\n",
                hungry[i]);
        for(int j=0; j<count; j++)
        {
            if(j!=i)
            {
                printf("P %d is waiting\n",
                        hungry[j]);
            }
        }
    }
}
```

```
void option2(int count)
{
    printf("\nAllow two philosophers to
            eat at same time\n");
    int combination=1;
    for(int i=0; i<count; i++)
    {
        for(int j=i+1; j<count; j++)
        {
            if(!areNeighbors(hungry[i],
                             hungry[j]))
            {
                printf("Combination %d\n", combination++);
                printf("P %d and P %d are granted
                        to eat\n", hungry[i],
                        hungry[j]);
                for(int k=0; k<count; k++)
                {
                    if(k!=i && k!=j)
                    {
                        printf("P %d is waiting\n",
                                hungry[k]);
                    }
                }
                printf("\n");
            }
        }
    }
    if(combination==1)
    {
        printf("No combinations found where
```

```
        two non-neighbour philosophers
            can eat.\n");
    }
}

int main()
{
int hungryCount;
printf("Dining philosopher problem\n");
printf("Enter the total no of philosophers:");
scanf("%d", &hungryCount);

for(
printf("How many are hungry:");
scanf("%d", &hungryCount);

for(int i=0; i<hungryCount; i++)
{
printf("Enter philosopher %d position:",
        i+1);
scanf("%d", &hungry[i]);
}

int choice;
do
{
printf("\n1. One can eat at a time
        2. Two can eat at a time
        3. Exit\n");
printf("Enter your choice:");
scanf("%d", &choice);
```

```
switch(choice)
{
case 1:
    option1(hungryCount);
    break;
case 2:
    option2(hungryCount);
    break;
case 3:
    printf("Exiting...\n");
    break;
default:
    printf("Invalid choice!\n");
}
while(choice!=3);
return 0;
}
```

```
Dining philosopher problem
Enter the total no of philosophers: 5
How many are hungry: 3
Enter philosopher 1 position: 1
Enter philosopher 2 position: 2
Enter philosopher 3 position: 3

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

Enter your choice: 1
```


SURYA Gold
Date Page

Allow one philosopher to eat at any time

P1 is granted to eat
P2 is waiting
P3 is waiting
P2 is granted to eat
P1 is waiting
P3 is waiting
P3 is granted to eat
P1 is waiting
P2 is waiting

1. One can eat at a time 2. Two can eat at a time 3. Exit
Enter your choice : 2

Allow two philosophers to eat at same time
Combination 1
P1 and P3 are granted to eat
P2 is waiting

1. One can eat at a time 2. Two can eat at a time 3. Exit
Enter your choice : 3
Exiting...

Program -7

Question

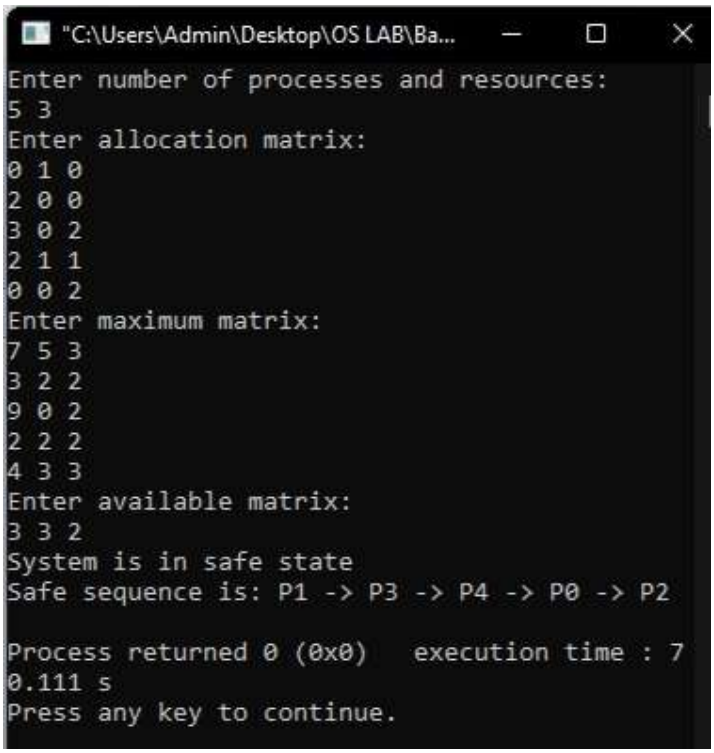
Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

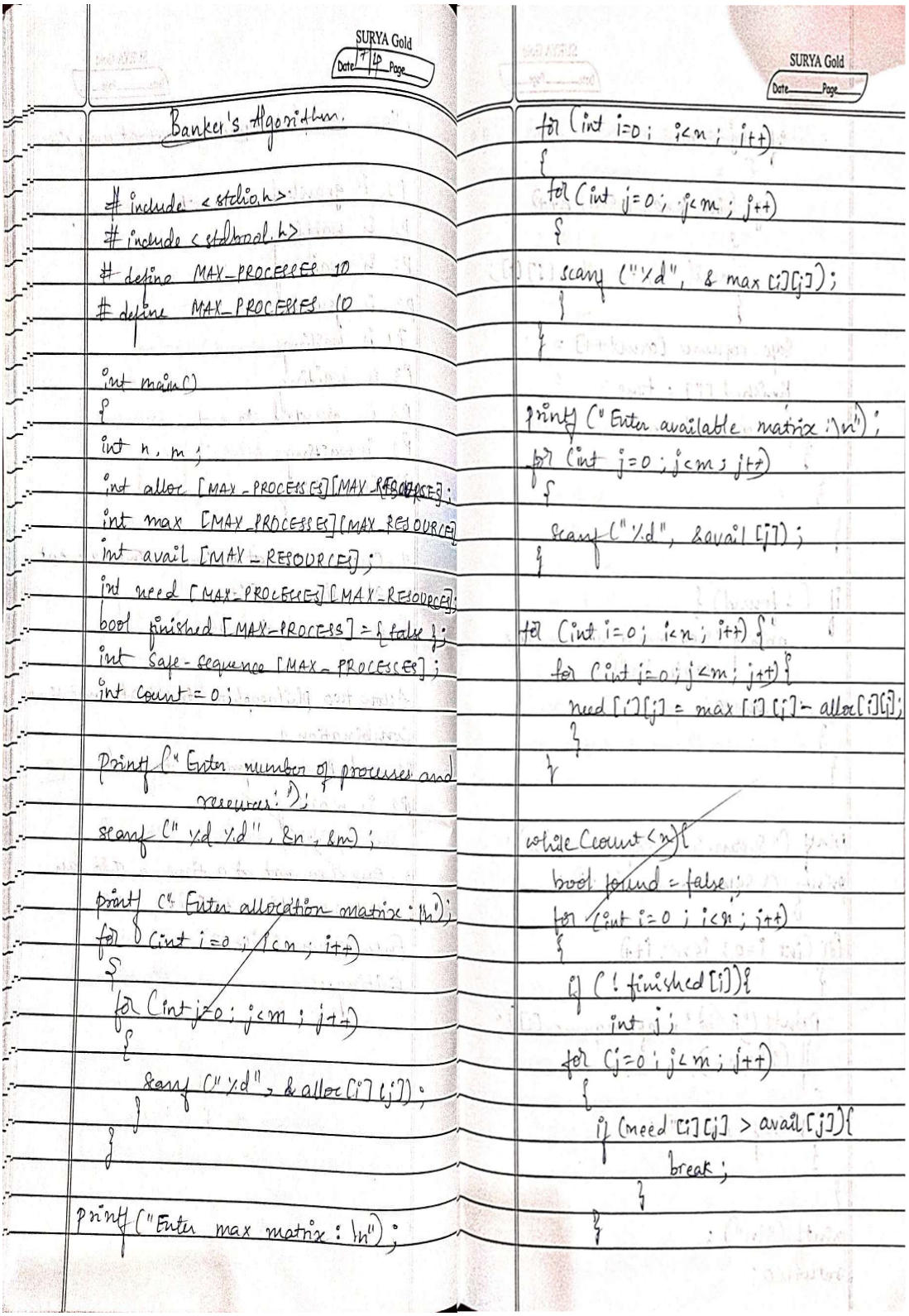
Code:

```
#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.



Banker's Algorithm.

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int main()
{
int n, m;
int alloc [MAX_PROCESSES][MAX_RESOURCES];
int max [MAX_PROCESSES][MAX_RESOURCES];
int avail [MAX_RESOURCES];
int need [MAX_PROCESSES][MAX_RESOURCES];
bool finished [MAX_PROCESS] = {false};
int Safe_Sequence [MAX_PROCESSES];
int count = 0;

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

printf("Enter allocation matrix:\n");
for (int i=0; i<n; i++)
{
  for (int j=0; j<m; j++)
  {
    scanf("%d", &alloc[i][j]);
  }
}

printf("Enter max matrix:\n");
for (int i=0; i<n; i++)
{
  for (int j=0; j<m; j++)
  {
    scanf("%d", &max[i][j]);
  }
}

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

for (int i=0; i<n; i++) {
  for (int j=0; j<m; j++) {
    need[i][j] = max[i][j] - alloc[i][j];
  }
}

while (count < n){
  bool found = false;
  for (int i=0; i<n; i++)
  {
    if (!finished[i]){
      int j;
      for (j=0; j<m; j++)
      {
        if (need[i][j] > avail[j]){
          break;
        }
      }
```

```
            if (j == m)
            {
                for (int k=0; k<m; k++)
                {
                    avail[k] += alloc[i][k];
                }
                safe_sequence[count++] = i;
                finished[i] = true;
                found = true;
            }
        }
    }

    if (!found) {
        printf("System is not in safe state.\n");
        return 0;
    }
}

printf("System is in safe state.\n");
printf("Safe sequence is: ");

for (int i=0; i<n; i++)
{
    printf("P%d", safe_sequence[i]);
    if (i != n-1)
    {
        printf("->");
    }
}
printf("\n");
return 0;
}
```

```
o/p: Enter number of process 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: P1 -> P3 -> P4 -> P0 -> P2
```

Program -8

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;
         }
      }
      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:

```
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 : 42.819 s
Press any key to continue.
```

SURYA Gold
Date Page

Deadlock Detection

```
#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", &alloc[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;
```

SURYA Gold
Date Page

```
    for (j=0; j<m; j++)
    {
      if (alloc[i][j] != 0) {
        is_zero = false;
        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 deadlock state.\n");
else
  printf ("System is not in a deadlock state.\n");
return 0;
}
```

Op:

```
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
0 0 2
2 2 2
4 3 3

Enter available ~~vector~~ matrix:
3 3 2

Process 1 can finish
Process 3 can finish
Process 4 can finish
Process 2 can finish
Process 0 can finish
System is ~~not~~ 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

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;
```

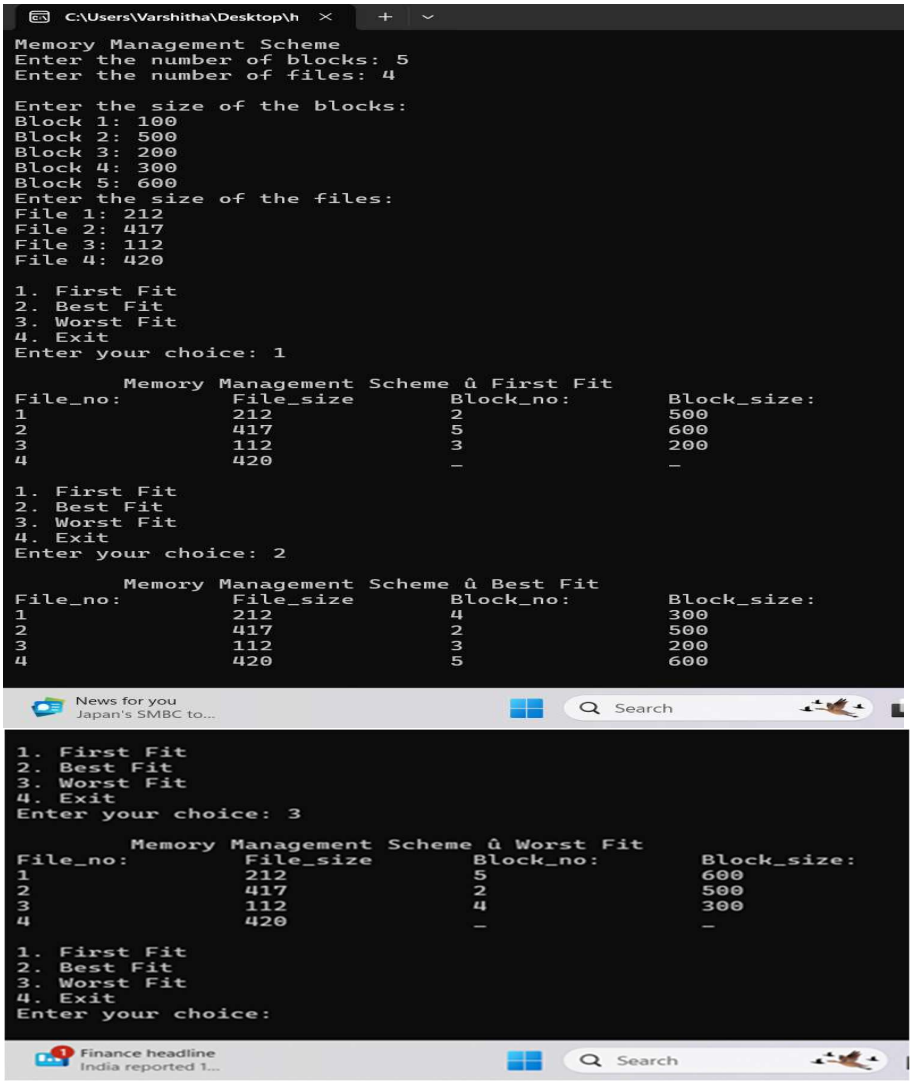
```
            printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, j + 1, blocks[j].size);
            break;
        }
    }
    if (files[i].block_no == -1) {
        printf("%d\t\t%d\t\t_\t\t_\n", i + 1, files[i].size);
    }
  }
}

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

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

```
    printf("\n\tMemory Management Scheme □ Worst Fit\n");
    printf("File_no:\tFile_size\tBlock_no:\tBlock_size:\n");

    for (int i = 0; i < n_files; i++) {
        int worstIdx = -1;
        for (int j = 0; j < n_blocks; j++) {
            if (!blocks[j].allocated && blocks[j].size >= files[i].size) {
                if (worstIdx == -1 || blocks[j].size > blocks[worstIdx].size) {
                    worstIdx = j;
                }
            }
        }
        if (worstIdx != -1) {
            blocks[worstIdx].allocated = 1;
            files[i].block_no = worstIdx + 1;
            printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, worstIdx + 1,
blocks[worstIdx].size);
        } else {
            printf("%d\t\t%d\t\t_\t\t_\n", i + 1, files[i].size);
        }
    }
}

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

    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); // Reset block allocation before each strategy

   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:



SURYA Gold

Memory Allocation

```
#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:\t
        Block_size\n");
```

SURYA Gold

```
  for(int i=0; i<n_files; i++)
  {
    files[i].block_no = -1; //not allocated
    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;
        printf("%d\t\t%d\t\t%d\t\t%d\n",
            i+1, files[i].size, j+1,
            blocks[j].size);
        break;
      }
    }
    if (files[i].block_no == -1)
    {
      printf("%d\t\t%d\t\t-\t\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:\t
        Block_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\t%d\t\t%d\t\t%d\t\t\t-\t\t-\n",
i+1, files[i].size, bestIdx+1,
blocks[bestIdx].size);
}
else
{
printf("%d\t\t%d\t\t-\t\t-\t\t-\n",
i+1, files[i].size);
}
}
}

void worstFit (struct Block blocks[],
int n_blocks, struct File files[],
int n_files)
{
for (int i=0; i<n_files; i++){
int worstIdx = -1;
for (int j=0; j<n_blocks; j++){
if (!blocks[j].allocated && blocks[j].size >=
files[i].size){
if (worstIdx == -1 || blocks[j].size > blocks[
worstIdx].size){
worstIdx = j;
}
}
}
if (worstIdx != -1)
{
blocks[worstIdx].allocated = 1;
files[i].block_no = worstIdx + 1;
printf("%d\t\t%d\t\t%d\t\t%d\t\t\t-\n",
i+1, files[i].size, worstIdx+1,
blocks[worstIdx].size);
}else
{
printf("%d\t\t%d\t\t-\t\t-\n", i+1,
files[i].size);
}
}
}
```

Op:-

```
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
```

SURYA Gold
Date Page

Enter the size of the files :
File 1 : 212
File 2 : 417
File 3 : 112
File 4 : 420

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

Memory management scheme
First fit

Fileno	File-size	Block-no	Block-size
1	212	2	500
2	417	5	600
3	112	3	200
4	420	—	—

Best fit

1	212	4	300
2	417	2	500
3	112	3	200
4	420	5	600

Worst fit

1	212	5	600
2	417	2	500
3	112	4	800
4	420	—	—

Program -10

Question

Write a C program to simulate page replacement algorithms
a) FIFO
b) LRU
c) Optimal

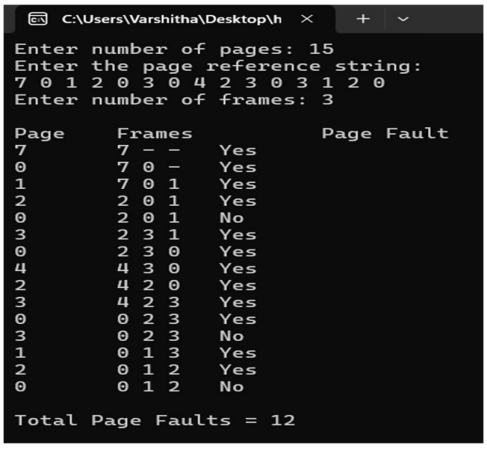
a)

Code

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

```
            count++;
            printf("%d\t", pages[i]);
            for(k = 0; k < frames; k++) {
               if(frame[k] != -1)
                  printf("%d ", frame[k]);
               else
                  printf("- ");
            }
            printf("\tYes\n");
         } else {
            printf("%d\t", pages[i]);
            for(k = 0; k < frames; k++) {
               if(frame[k] != -1)
                  printf("%d ", frame[k]);
               else
                  printf("- ");
            }
            printf("\tNo\n");
         }
      }
      printf("\nTotal Page Faults = %d\n", count);
      return 0;
   }
```

Output:



SURYA Gold
Date Page

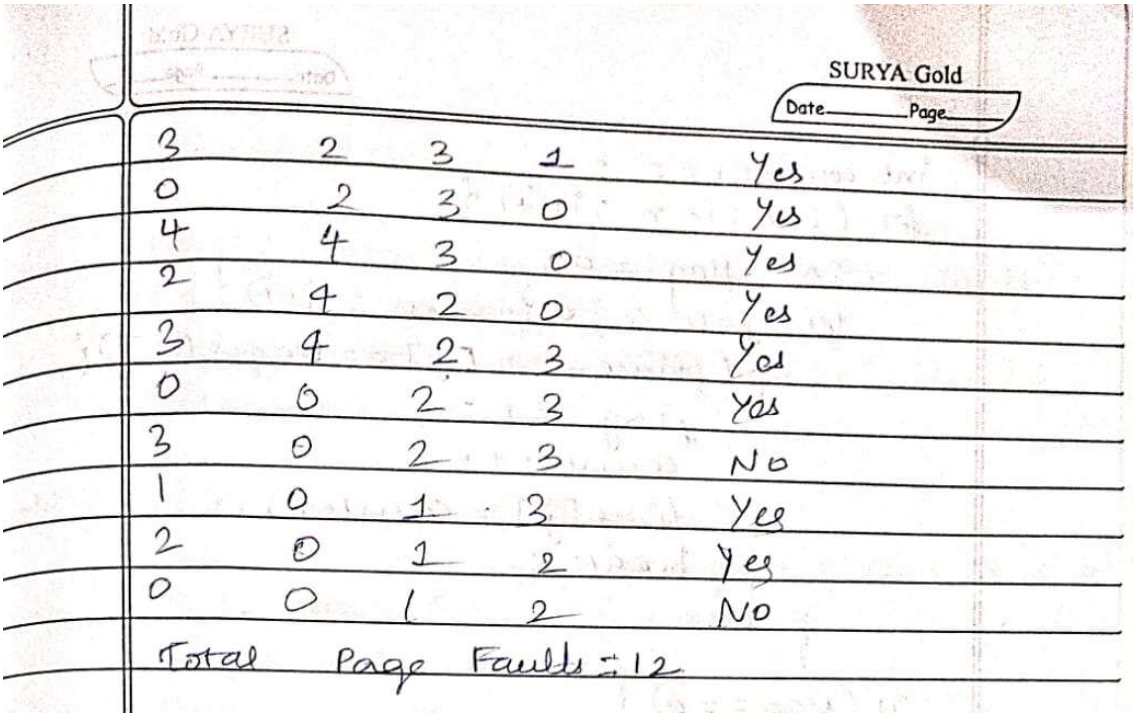
Page Replacement Algorithm

FIFO

```
#include <stdio.h>
int main() {
int frames, pages[50], n, frame[10], i, j, k,
    avail, count = 0;
printf("Enter number of pages : ");
scanf("%d", &n);
Printf("Enter the ordinal page reference string : \n");
for(i=0; i<n; i++)
   scanf("%d", &pages[i]);
printf("Enter number of frames:");
Scanf("%d", &frames);
for(i=0; i<frames; i++)
   frames[i] = -1;
printf("\nPage\t Frames \t\t Page Fault\n");
j=0;
for(i=0; i<n; i++) {
   avail=0;
   for(k=0; k<frames; k++) {
   if(frame[k] == pages[i]) {
      avail=1;
      break;
   }
   }
   if(avail==0) {
      frame[j] = pages[i];
      j=(j+1) % frames;
      count++;
      printf("%d\t", pages[i]);
   for(k=0; k<frames; k++) {
      if(frame[k] != -1)
         printf("%d", frame[k]);
      else
         printf("-");
   }
      printf("\t Yes\n");
   }else{
   printf("%d\t", pages[i]);
   for(k=0; k<frames; k++) {
      if(frame[k] != -1)
         printf("%d", frame[k]);
      else
         printf("-");
   }
   printf("\t No\n");
   }
}
printf("\n Total Page Faults = %d\n",
      count);
return 0;
}
```

Op - Enter number of pages : 15
Enter the page reference string :
7 0 1 2 0 3 0 4 2 3 0 3 1 2 0
Enter the number of frames : 3

Page	Frames	Page Fault
7	7 - -	Yes
0	7 0 -	Yes
1	7 0 1	Yes
2	2 0 1	Yes
0	2 0 1	No



b)

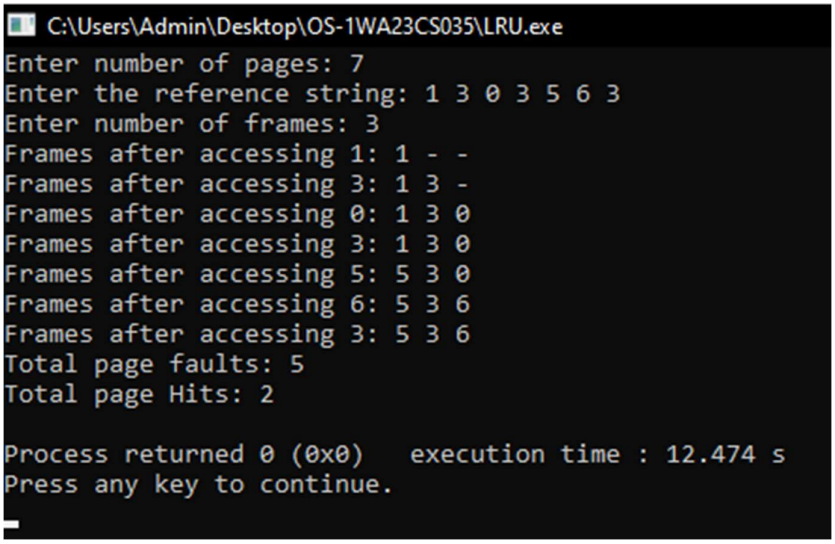
Code

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

int main() {
    int n, frames, i, j, k, faults = 0;
    printf("Enter number of pages: ");
    scanf("%d", &n);
    int pages[n];
    printf("Enter the reference string: ");
    for(i = 0; i < n; i++)
        scanf("%d", &pages[i]);

    printf("Enter number of frames: ");
    scanf("%d", &frames);
```

```
int frame_arr[frames];
int time[frames]; // To track the usage time
for(i = 0; i < frames; i++) {
   frame_arr[i] = -1;
   time[i] = 0;
}

int counter = 0;
for(i = 0; i < n; i++) {
   int flag = 0;
   for(j = 0; j < frames; j++) {
      if(frame_arr[j] == pages[i]) {
         flag = 1;
         counter++;
         time[j] = counter; // Update the usage time
                        break;
      }
   }

   if(flag == 0) { // Page fault
      faults++;
      int min_time = time[0], min_pos = 0;
      for(k = 1; k < frames; k++) {
         if(time[k] < min_time) {
            min_time = time[k];
            min_pos = k;
         }
      }
      frame_arr[min_pos] = pages[i]; // Replace the least recently used
      counter++;
      time[min_pos] = counter; // Update the usage time for the new page
   }
```

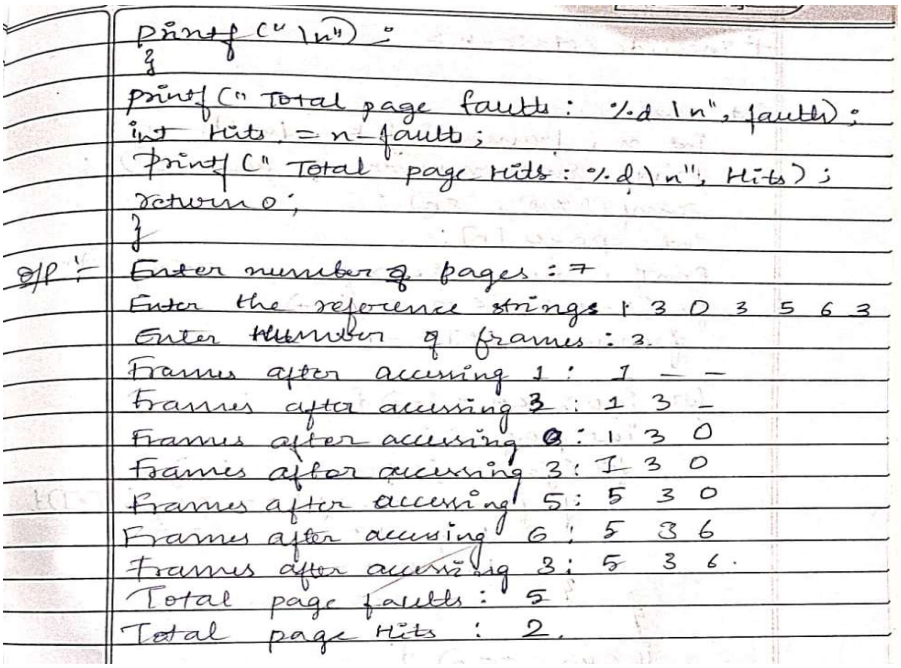
```
        printf("Frames after accessing %d: ", pages[i]);
        for(j = 0; j < frames; j++) {
            if(frame_arr[j] == -1)
                printf("- ");
            else
                printf("%d ", frame_arr[j]);
        }
        printf("\n");
    }

    printf("Total page faults: %d\n", faults);
    int Hits = n-faults;
    printf("Total page Hits: %d\n",Hits);
    return 0;
}
```

Output:



LRU:

```
#include <stdio.h>
int main(){
    int n, frames, i, j, k, faults = 0;
    printf("Enter number of pages: ");
    scanf("%d", &n);
    int pages[n];
    printf("Enter the reference string: ");
    for(i=0; i<n; i++)
        scanf("%d", &pages[i]);
    printf("Enter number of frames: ");
    scanf("%d", &frames);

    int frame_arr[frames];
    int time[frame];
    for(i=0; i<frames; i++){
        frames_arr[i] = -1;
        time[i] = 0;
    }
    int counter = 0;
    for(i=0; i<n; i++){
        int flag = 0;
        for(j=0; j<frames; i++){
            if(frame_arr[j] == pages[i]){
                flag = 1;
                counter++;
                time[j] = counter;
                break;
            }
        }
        if(flag == 0){
            faults++;
            int min_time = time[0], min_pos = 0;
            for(k=1; k<frames; k++){
                if(time[k] < min_time){
                    min_time = time[k];
                    min_pos = k;
                }
            }
            frame_arr[min_pos] = pages[i];
            counter++;
            time[min_pos] = counter;
        }
        printf("Frames after accessing %d:",
                pages[i]);
        for(j=0; j<frames; j++){
            if(frame_arr[j] == -1)
                printf("-");
            else
                printf("%d", frame_arr[j]);
        }
    }
```



c)

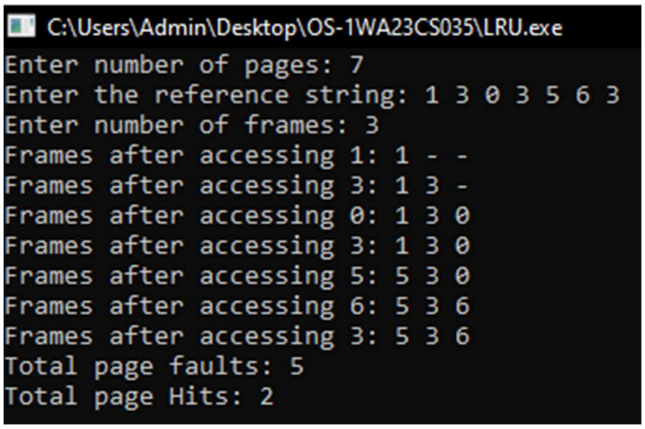
Code

```
//OPTIMAL
#include <stdio.h>
int main() {
    int n, frames, i, j, k, faults = 0;
    printf("Enter number of pages: ");
    scanf("%d", &n);
    int pages[n];
    printf("Enter the reference string: ");
    for(i = 0; i < n; i++)
        scanf("%d", &pages[i]);

    printf("Enter number of frames: ");
    scanf("%d", &frames);
    int frame_arr[frames];
    for(i = 0; i < frames; i++)
        frame_arr[i] = -1;
    for(i = 0; i < n; i++) {
        int flag = 0;
```

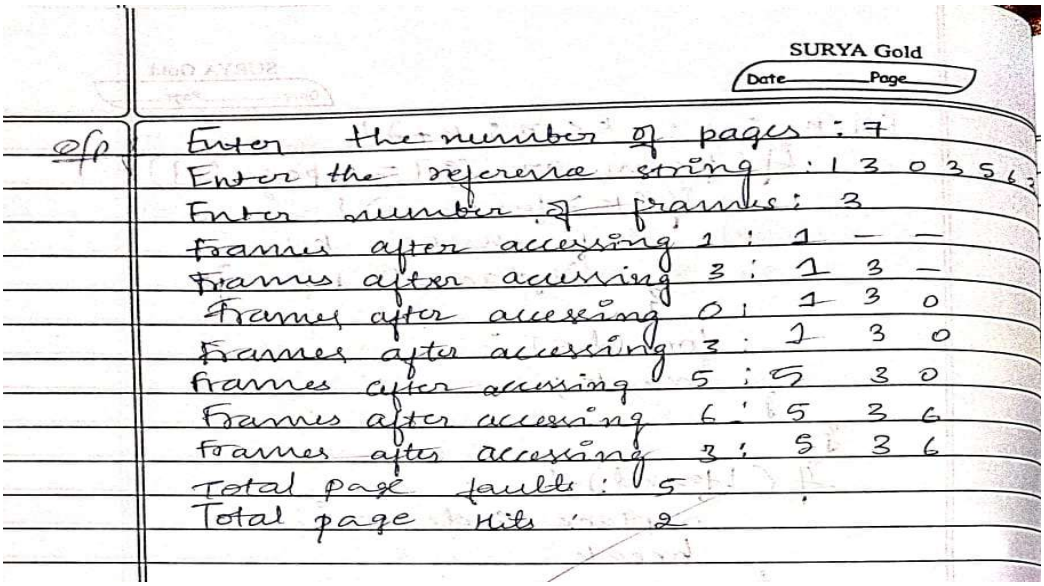
```
for(j = 0; j < frames; j++) {
    if(frame_arr[j] == pages[i]) {
        flag = 1;
        break;
    }
}
if(flag == 0) { // Page fault
    faults++;
    int pos = -1;
    for(j = 0; j < frames; j++) {
        if(frame_arr[j] == -1) {
            pos = j;
            break;
        }
    }
    if(pos == -1) { // Need to replace a page
        int farthest = i, replace_index = 0;
        for(j = 0; j < frames; j++) {
            int found = 0;
            for(k = i + 1; k < n; k++) {
                if(frame_arr[j] == pages[k]) {
                    if(k > farthest) {
                        farthest = k;
                        replace_index = j;
                    }
                    found = 1;
                    break;
                }
            }
            if(!found) {
                replace_index = j;
                break;
            }
        }
```

```
            pos = replace_index;
        }
        frame_arr[pos] = pages[i];
    }
    printf("Frames after accessing %d: ", pages[i]);
    for(j = 0; j < frames; j++) {
        if(frame_arr[j] == -1)
            printf("_ ");
        else
            printf("%d ", frame_arr[j]);
    }
    printf("\n");
  }
  printf("Total page faults: %d\n", faults);
  int Hits = n-faults;
  printf("Total page Hits: %d\n",Hits);
  return 0;
}
```

Output:



```
#include <stdio.h>

int main() {
    int n, frames, i, j, k, faults = 0;
    printf("Enter number of pages: ");
    scanf("%d", &n);
    int pages[n];
    printf("Enter the reference string: ");
    for(i=0; i<n; i++)
        frame_arr[i] = -1;

    for(i=0; i<n; i++) {
        int flag = 0;
        for(j=0; j<frames; j++) {
            if(frame_arr[j] == pages[i]) {
                flag = 1;
                break;
            }
        }
        if(flag == 0) {
            faults++;
            int pos = -1;
            for(j=0; j<frames; j++) {
                if(frame_arr[j] == -1) {
                    pos = j;
                    break;
                }
            }

            if(pos == -1) {
                int farthest = i, replace_index = 0;
                for(j=0; j<frames; j++) {
                    int found = 0;
                    for(k=i+1; k<n; k++) {
                        if(frame_arr[j] == pages[k]) {
                            if(k > farthest) {
                                farthest = k;
                                replace_index = j;
                            }
                            found = 1;
                            break;
                        }
                    }
                    if(!found) {
                        replace_index = j;
                        break;
                    }
                }
                pos = replace_index;
            }
            frame_arr[pos] = pages[i];
        }
        printf("Frames after accessing %d:", pages[i]);
        for(j=0; j<frames; j++) {
            if(frame_arr[j] == -1)
                printf("-");
            else
                printf("%d", frame_arr[j]);
        }
        printf("\n");
    }
    printf("Total page faults: %d\n", faults);
    int hits = n - faults;
    printf("Total page hits: %d\n", hits);
    return 0;
}
```


SURYA Gold
Date Page
o/p
Enter the number of pages : 7
Enter the reference string : 1 3 0 3 5 6 3
Enter number of frames : 3
Frames after accessing 1 : 1 — —
Frames after accessing 3 : 1 3 —
Frames after accessing 0 : 1 3 0
Frames after accessing 3 : 1 3 0
Frames after accessing 5 : 5 3 0
Frames after accessing 6 : 5 3 6
Frames after accessing 3 : 5 3 6
Total page faults : 5
Total page Hits : 2