VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

OPERATING SYSTEMS

Submitted by

Tissa Maria (1BM22CS309)

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
Apr-2024 to Aug-2024

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 **Tissa Maria(1BM22CS309),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024. 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.

Prameetha Pai	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Table Of Contents

Course Outcomes

CO1: Apply the different concepts and functionalities of Operating System.

CO2: Analyse various Operating system strategies and techniques.

CO3: Demonstrate the different functionalities of Operating System.

CO4: Conduct practical experiments to implement the functionalities of Operating system

Program 1

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

1)FCFS

2)SJF (pre-emptive & Non-preemptive)

```
//fcfs

#include <stdio.h>


struct Process {

    int pid;

    int arrival_time;

    int burst_time;

    int completion_time;

    int turnaround_time;

    int waiting_time;

};


void findCompletionTime(struct Process proc[], int n) {

    proc[0].completion_time = proc[0].arrival_time + proc[0].burst_time;

    proc[0].turnaround_time = proc[0].completion_time - proc[0].arrival_time;

    proc[0].waiting_time = proc[0].turnaround_time - proc[0].burst_time;

    for (int i = 1; i < n; i++) {

        if (proc[i].arrival_time > proc[i - 1].completion_time) {

            proc[i].completion_time = proc[i].arrival_time + proc[i].burst_time;

        } else {
```

```
            proc[i].completion_time = proc[i - 1].completion_time + proc[i].burst_time;
        }
        proc[i].turnaround_time = proc[i].completion_time - proc[i].arrival_time;
        proc[i].waiting_time = proc[i].turnaround_time - proc[i].burst_time;
    }
}



void printProcesses(struct Process proc[], int n) {
    printf("PID\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting
Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
               proc[i].pid, proc[i].arrival_time, proc[i].burst_time,
               proc[i].completion_time, proc[i].turnaround_time, proc[i].waiting_time);
    }
}

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

    struct Process proc[n];
```

```
    for (int i = 0; i < n; i++) {
        proc[i].pid = i + 1;
        printf("Enter arrival time and burst time for process %d: ", i + 1);
        scanf("%d %d", &proc[i].arrival_time, &proc[i].burst_time);
    }

    findCompletionTime(proc, n);
    printProcesses(proc, n);
    return 0;
}
```

```
PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc fcfs.c -o fcfs } ; if ($?) { .\fcfs }
Enter the number of processes: 2
Enter arrival time and burst time for process 1: 0
2
Enter arrival time and burst time for process 2: 2
4
PID     Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time
1       0               2               2               2               0
2       2               4               6               4               0
PS C:\TISSA\OS 2023-24>
```

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

struct Process {
    int pid;
    int arrival_time;
    int burst_time;
    int completion_time;
```

```
    int turnaround_time;
    int waiting_time;
};

void findCompletionTime(struct Process proc[], int n) {
    int current_time = 0;
    int completed = 0;
    int is_completed[n];
    for (int i = 0; i < n; i++) {
        is_completed[i] = 0;
    }

    while (completed != n) {
        int min_index = -1;
        int min_burst = 1000000;
        for (int i = 0; i < n; i++) {
            if (proc[i].arrival_time <= current_time && is_completed[i] == 0) {
                if (proc[i].burst_time < min_burst) {
                    min_burst = proc[i].burst_time;
                    min_index = i;
                }
            }
        }
```

```
        if (min_index == -1) {
            current_time++;
        } else {
            proc[min_index].completion_time = current_time + proc[min_index].burst_time;
            current_time += proc[min_index].burst_time;
            proc[min_index].turnaround_time = proc[min_index].completion_time -
proc[min_index].arrival_time;
            proc[min_index].waiting_time = proc[min_index].turnaround_time -
proc[min_index].burst_time;
            is_completed[min_index] = 1;
            completed++;
        }
    }
}


void printProcesses(struct Process proc[], int n) {
    printf("PID\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting
Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
               proc[i].pid, proc[i].arrival_time, proc[i].burst_time,
               proc[i].completion_time, proc[i].turnaround_time, proc[i].waiting_time);
    }
}


int main() {
```

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

    struct Process proc[n];

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

    findCompletionTime(proc, n);

    printProcesses(proc, n);

    return 0;
}
```

```
PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc sjf.c -o sjf } ; if ($?) { .\sjf }
Enter the number of processes: 4
Enter arrival time and burst time for process 1: 0
7
Enter arrival time and burst time for process 2: 0
3
Enter arrival time and burst time for process 3: 0
4
Enter arrival time and burst time for process 4: 0
6
PID     Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time
1       0               7               20              20              13
2       0               3               3               3               0
3       0               4               7               7               3
4       0               6               13              13              7
PS C:\TISSA\OS 2023-24>
```

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


struct Process {
    int pid;
    int arrival_time;
    int burst_time;
    int remaining_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
};


void findCompletionTime(struct Process proc[], int n) {
    int current_time = 0;
```

```
int completed = 0;
int min_index;
int min_remaining;

while (completed != n) {
    min_index = -1;
    min_remaining = 1000000;

    for (int i = 0; i < n; i++) {
        if (proc[i].arrival_time <= current_time && proc[i].remaining_time > 0) {
            if (proc[i].remaining_time < min_remaining) {
                min_remaining = proc[i].remaining_time;
                min_index = i;
            }
        }
    }

    if (min_index == -1) {
        current_time++;
    } else {
        proc[min_index].remaining_time--;
        current_time++;

        if (proc[min_index].remaining_time == 0) {
```

```
            proc[min_index].completion_time = current_time;

            proc[min_index].turnaround_time = proc[min_index].completion_time -
proc[min_index].arrival_time;

            proc[min_index].waiting_time = proc[min_index].turnaround_time -
proc[min_index].burst_time;

            completed++;

         }

      }

   }

}


void printProcesses(struct Process proc[], int n) {

   printf("PID\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting
Time\n");

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

      printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",

            proc[i].pid, proc[i].arrival_time, proc[i].burst_time,

            proc[i].completion_time, proc[i].turnaround_time, proc[i].waiting_time);

   }

}


int main() {

   int n;

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

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

```
    struct Process proc[n];

    for (int i = 0; i < n; i++) {
        proc[i].pid = i + 1;
        printf("Enter arrival time and burst time for process %d: ", i + 1);
        scanf("%d %d", &proc[i].arrival_time, &proc[i].burst_time);
        proc[i].remaining_time = proc[i].burst_time;
    }

    findCompletionTime(proc, n);

    printProcesses(proc, n);

    return 0;
}
```

```
PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc srtf.c -o srtf } ; if ($?) { .\srtf }
Enter the number of processes: 6
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
2
Enter arrival time and burst time for process 4: 3
1
Enter arrival time and burst time for process 5: 4
3
Enter arrival time and burst time for process 6: 5
2
PID     Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time
1       0               8               20              20              12
2       1               4               10              9               5
3       2               2               4               2               0
4       3               1               5               2               1
5       4               3               13              9               6
6       5               2               7               2               0
PS C:\TISSA\OS 2023-24>
```

Program 2 :

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

1) **Priority (pre-emptive & Non-pre-emptive)**
2) **Round Robin (Experiment with different quantum sizes for RR algorithm)**

```
//priority

#include <stdio.h>

#define MAX(a,b) ((a)>(b)?(a):(b))



void priorityNonPreemptive(int processes[], int n, int burst_time[], int priority[], int
arrival_time[]) {
```

```c
    int waiting_time[n], turnaround_time[n];

    waiting_time[0] = MAX(0, arrival_time[0]);

    for (int i = 1; i < n; i++) {

        waiting_time[i] = MAX(0, waiting_time[i-1] + burst_time[i-1] - arrival_time[i]);

    }

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

        turnaround_time[i] = waiting_time[i] + burst_time[i];

    }



    printf("\nNon-Preemptive Priority Scheduling:\n");

    printf("Process\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");

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

        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], arrival_time[i], burst_time[i],
priority[i], waiting_time[i], turnaround_time[i]);

    }

}



void priorityPreemptive(int processes[], int n, int burst_time[], int priority[], int
arrival_time[]) {

    int remaining_time[n], waiting_time[n], turnaround_time[n], completed = 0,
current_time = 0;

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

```
        remaining_time[i] = burst_time[i];
        waiting_time[i] = 0;
    }
    while (completed != n) {
        int selected_process = -1;
        int lowest_priority = 1000000; // higher the value lower the priority
        for (int i = 0; i < n; i++) {
            if (remaining_time[i] > 0 && priority[i] < lowest_priority && current_time >=
arrival_time[i]) {
                lowest_priority = priority[i];
                selected_process = i;
            }
        }
        if (selected_process == -1) {
            current_time++;
            continue;
        }
        remaining_time[selected_process]--;
        current_time++;
        if (remaining_time[selected_process] == 0) {
            completed++;
```

```
            turnaround_time[selected_process] = current_time -
arrival_time[selected_process];

        }

    }

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

        waiting_time[i] = turnaround_time[i] - burst_time[i];

    }


    printf("\nPreemptive Priority Scheduling:\n");

    printf("Process\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");

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

        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], arrival_time[i], burst_time[i],
priority[i], waiting_time[i], turnaround_time[i]);

    }

}


int main() {

    int n;

    printf("Enter number of processes: ");

    scanf("%d", &n);

    int processes[n], burst_time[n], arrival_time[n], priority[n];

    printf("Enter arrival time, burst time, and priority for each process:\n");
```

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

        printf("Process %d: ", i + 1);

        scanf("%d%d%d", &arrival_time[i], &burst_time[i], &priority[i]);

        processes[i] = i + 1;

    }

    priorityNonPreemptive(processes, n, burst_time, priority, arrival_time);

    priorityPreemptive(processes, n, burst_time, priority, arrival_time);

    return 0;

}
```

```
PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc PriorityScheduling.c -o PriorityScheduling } ; if ($?) { .\PrioritySchedu
ling }
Enter number of processes: 3
Enter arrival time, burst time, and priority for each process:
Process 1: 0
5
2
Process 2: 2
3
1
Process 3: 4
1
3

Non-Preemptive Priority Scheduling:
Process Arrival Time    Burst Time      Priority        Waiting Time    Turnaround Time
1       0               5               2               0               5
2       2               3               1               3               6
3       4               1               3               2               3

Preemptive Priority Scheduling:
Process Arrival Time    Burst Time      Priority        Waiting Time    Turnaround Time
1       0               5               2               3               8
2       2               3               1               0               3
3       4               1               3               4               5
PS C:\TISSA\OS 2023-24>
```

```
//Round Robin

#include<stdio.h>
```

```
void RoundRobin(int processes[],int n,int burst_time[],int arrival_time[],int
timeQuantum){
    int remaining_time[n];
    int waiting_time[n];
    int turnaround_time[n];
    int completion_time[n];
    int current_time=0;
    for(int i=0;i<n;i++){
        remaining_time[i]=burst_time[i];
        waiting_time[i]=0;
    }
    while(1){
        int allDone=1;
        for(int i=0;i<n;i++){
            if(remaining_time[i]>0){
                allDone=0;
                if(remaining_time[i]<=timeQuantum){
                    current_time+=remaining_time[i];
                    turnaround_time[i]=current_time-arrival_time[i];
                    completion_time[i]=current_time;
                    remaining_time[i]=0;
                }else{
```

```
                current_time+=timeQuantum;
                remaining_time[i]-=timeQuantum;
            }
        }
    }
    if(allDone){
        break;
    }
}
    for(int i=0;i<n;i++){
        waiting_time[i]=turnaround_time[i]-burst_time[i];
    }
    printf("Process\tArrival Time\tBurst Time\tCompletion Time\tWaiting
Time\tTurnaround Time\n");
    for(int i=0;i<n;i++){

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",processes[i],arrival_time[i],burst_time[i],compl
etion_time[i],waiting_time[i],turnaround_time[i]);


    }


}
```

```
int main(){
    int n;
    printf("Enter number of processes:");
    scanf("%d",&n);
    int timeQuantum;
    printf("Enter Time Quantum");
    scanf("%d",&timeQuantum);
    int processes[n],burst_time[n],arrival_time[n],priority[n];
    printf("Enter arrival time, burst time for each process\n");
    for(int i=0;i<n;i++){
        printf("Process %d ",i+1);
        scanf("%d%d",&arrival_time[i],&burst_time[i]);
        processes[i]=i+1;


    }
    RoundRobin(processes,n,burst_time,arrival_time,timeQuantum);
    return 0;
}
```

```
PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc roundRobin.c -o roundRobin } ; if ($?) { .\roundRobin }
Enter number of processes:4
Enter Time Quantum2
Enter arrival time, burst time for each process
Process 1 0
10
Process 2 1
5
Process 3 2
8
Process 4 3
12
Process Arrival Time    Burst Time      Completion Time Waiting Time    Turnaround Time
1               0               10              31              21              31
2               1               5               19              13              18
3               2               8               27              17              25
4               3               12              35              20              32
PS C:\TISSA\OS 2023-24> 
```

Program 3:

Write a C program to simulate a 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.

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define MAX 100


typedef struct {

    int pid;

    char type[10];

    int arrival_time;

    int burst_time;

    int completion_time;
```

```
    int turnaround_time;
    int waiting_time;
} Processes;

void sortProcessByArrivalTime(Processes *queue, int count) {
    for (int i = 0; i <= count - 1; i++) {
        for (int j = 0; j < count - i - 1; j++) {
            if (queue[j].arrival_time > queue[j + 1].arrival_time) {
                Processes temp = queue[j];
                queue[j] = queue[j + 1];
                queue[j + 1] = temp;
            }
        }
    }
}

void calculateTime(Processes *queue, int count, int *currentTime) {
    for (int i = 0; i < count; i++) {
        if (*currentTime < queue[i].arrival_time) {
            *currentTime = queue[i].arrival_time;
        }
        queue[i].completion_time = *currentTime + queue[i].burst_time;
        queue[i].turnaround_time = queue[i].completion_time - queue[i].arrival_time;
        queue[i].waiting_time = queue[i].turnaround_time - queue[i].burst_time;
```

```
        *currentTime = queue[i].completion_time;
    }
}

void simulateMultiLevelQueueing(Processes *process, int n) {
    Processes systemQueue[MAX], userQueue[MAX];
    int systemCount = 0, userCount = 0;

    for (int i = 0; i < n; i++) {
        if (strcmp(process[i].type, "system") == 0) {
            systemQueue[systemCount++] = process[i];
        } else if (strcmp(process[i].type, "user") == 0) {
            userQueue[userCount++] = process[i];
        }
    }

    sortProcessByArrivalTime(systemQueue, systemCount);
    sortProcessByArrivalTime(userQueue, userCount);

    int currentTime = 0;
    calculateTime(systemQueue, systemCount, &currentTime);
    calculateTime(userQueue, userCount, &currentTime);

    printf("PID\tType\tArrival Time\tBurst Time\tCompletion Time\tTurnaround
Time\tWaiting Time\n");
```

```
    float totalTurnaroundTime = 0;

    float totalWaitingTime = 0;


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

        totalTurnaroundTime += systemQueue[i].turnaround_time;

        totalWaitingTime += systemQueue[i].waiting_time;

        printf("%d\t%s\t%d\t%d\t%d\t%d\t%d\n", systemQueue[i].pid, systemQueue[i].type,
systemQueue[i].arrival_time, systemQueue[i].burst_time,
systemQueue[i].completion_time, systemQueue[i].turnaround_time,
systemQueue[i].waiting_time);

    }


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

        totalTurnaroundTime += userQueue[i].turnaround_time;

        totalWaitingTime += userQueue[i].waiting_time;

        printf("%d\t%s\t%d\t%d\t%d\t%d\t%d\n", userQueue[i].pid, userQueue[i].type,
userQueue[i].arrival_time, userQueue[i].burst_time, userQueue[i].completion_time,
userQueue[i].turnaround_time, userQueue[i].waiting_time);

    }


    int totalProcesses = systemCount + userCount;

    printf("Average Turnaround Time: %f\n", totalTurnaroundTime / totalProcesses);

    printf("Average Waiting Time: %f\n", totalWaitingTime / totalProcesses);

}
```

```
int main() {
    Processes process[MAX];
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("Process ID: ");
        scanf("%d", &process[i].pid);
        printf("Process Type (system/user): ");
        scanf("%s", process[i].type);
        printf("Process Arrival Time: ");
        scanf("%d", &process[i].arrival_time);
        printf("Process Burst Time: ");
        scanf("%d", &process[i].burst_time);
    }

    simulateMultiLevelQueueing(process, n);
    return 0;
}
```

```
PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc multilevelQueueing.c -o multilevelQueueing } ; if ($?) { .\multilevelQueu
eing }
Enter the number of processes: 4
Process ID: 1
Process Type (system/user): system
Process Arrival Time: 0
Process Burst Time: 5
Process ID: 2
Process Type (system/user): user
Process Arrival Time: 1
Process Burst Time: 3
Process ID: 3
Process Type (system/user): system
Process Arrival Time: 2
Process Burst Time: 2
Process ID: 4
Process Type (system/user): user
Process Arrival Time: 3
Process Burst Time: 1
PID     Type    Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time
1       system  0       5               5               5               0
3       system  2       2               7               5               3
2       user    1       3               10              9               6
4       user    3       1               11              8               7
Average Turnaround Time: 6.750000
Average Waiting Time: 4.000000
PS C:\TISSA\OS 2023-24>
```

Program 4:

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

a) Rate- Monotonic b) Earliest-deadline First

```
//rate monotonic

#include <stdio.h>

#include <stdlib.h>

#include <math.h>



#define MAX_TASKS 10



typedef struct {

    int id;
```

```
    int period;

    int execution_time;

    int remaining_time;

} Task;


int gcd(int a, int b) {

    if (b == 0) {

        return a;

    }

    return gcd(b, a % b);

}


int lcm(int a, int b) {

    return (a * b) / gcd(a, b);

}


int calculateLCM(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, int simulationTime) {
    printf("RMS\n");
    int time = 0;
    while (time < simulationTime) {
        int min_period = 9999;
        int index = -1;

        for (int i = 0; i < n; i++) {
            if (time % tasks[i].period == 0) {
                tasks[i].remaining_time = tasks[i].execution_time;
            }
            if (tasks[i].remaining_time > 0 && tasks[i].period < min_period) {
                min_period = tasks[i].period;
                index = i;
            }
        }

        if (index != -1) {
```

```
            printf("Time %d : Task %d\n", time, tasks[index].id);
            tasks[index].remaining_time--;
        } else {
            printf("Time %d : Idle\n", time);
        }
        time++;
    }
    printf("\n");
}

int main() {
    Task tasks[MAX_TASKS];
    int n;

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

    if (n > MAX_TASKS) {
        printf("Error: Number of tasks exceeds the maximum allowed (%d).\n",
MAX_TASKS);
        return 1;
    }
```

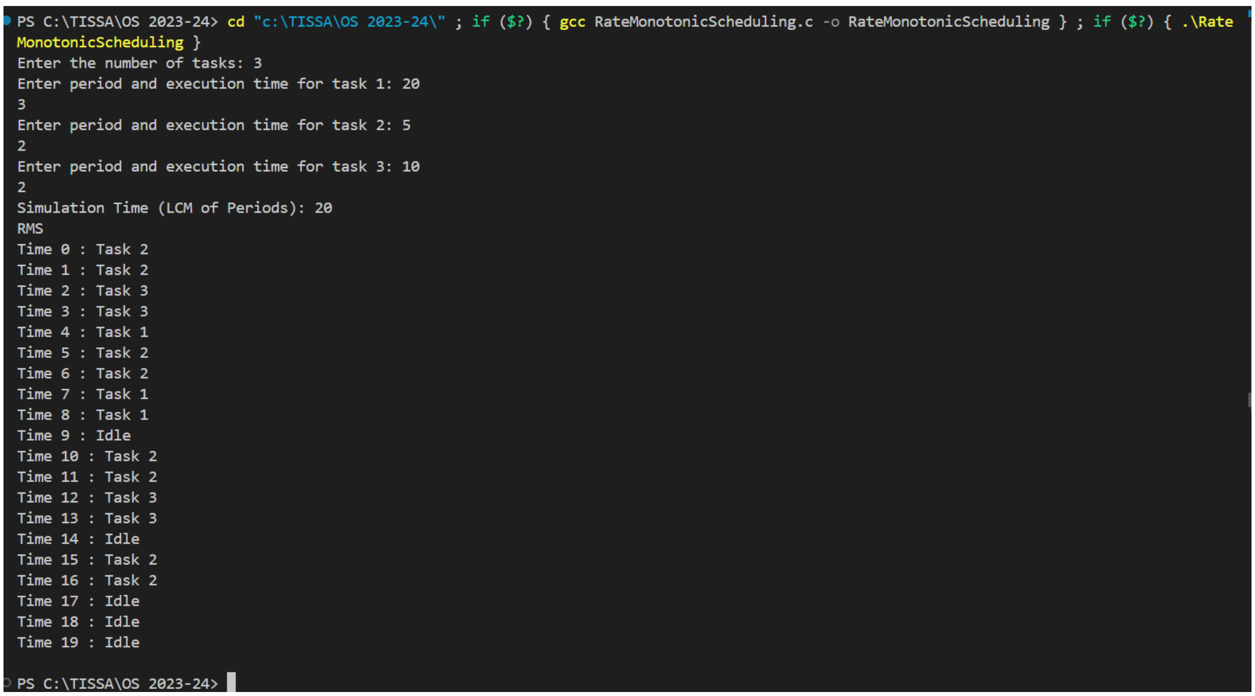
```
    double utilization = 0.0;
    for (int i = 0; i < n; i++) {
        tasks[i].id = i + 1;
        printf("Enter period and execution time for task %d: ", i + 1);
        scanf("%d %d", &tasks[i].period, &tasks[i].execution_time);
        tasks[i].remaining_time = 0;
        utilization += (double)tasks[i].execution_time / tasks[i].period;
    }

    double threshold = n * (pow(2, 1.0 / n) - 1);
    if (utilization > threshold) {
        printf("Error: The set of tasks is not schedulable under RMS (Utilization >
%f%%).\n", threshold * 100);
        return 1;
    }

    int simulation_time = calculateLCM(tasks, n);
    printf("Simulation Time (LCM of Periods): %d\n", simulation_time);

    rateMonotonic(tasks, n, simulation_time);
```

```
    return 0;
}
```



```
PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc RateMonotonicScheduling.c -o RateMonotonicScheduling } ; if ($?) { .\Rate
MonotonicScheduling }
Enter the number of tasks: 3
Enter period and execution time for task 1: 20
3
Enter period and execution time for task 2: 5
2
Enter period and execution time for task 3: 10
2
Simulation Time (LCM of Periods): 20
RMS
Time 0 : Task 2
Time 1 : Task 2
Time 2 : Task 3
Time 3 : Task 3
Time 4 : Task 1
Time 5 : Task 2
Time 6 : Task 2
Time 7 : Task 1
Time 8 : Task 1
Time 9 : Idle
Time 10 : Task 2
Time 11 : Task 2
Time 12 : Task 3
Time 13 : Task 3
Time 14 : Idle
Time 15 : Task 2
Time 16 : Task 2
Time 17 : Idle
Time 18 : Idle
Time 19 : Idle

PS C:\TISSA\OS 2023-24>
```

```
//earliest deadline first
#include <stdio.h>
#include <stdlib.h>

#define MAX_TASKS 10

typedef struct {
    int id;
    int period;
```

```
    int execution_time;

    int remaining_time;

    int deadline;

    int next_deadline;

} Task;


void earliestDeadline(Task tasks[], int n, int totalTime) {

    printf("EDF\n");

    Task *current_task = NULL;

    int time = 0;


    while (time < totalTime) {

        // Update tasks at the beginning of each period

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

            if (time % tasks[i].period == 0) {

                tasks[i].remaining_time = tasks[i].execution_time;

                tasks[i].next_deadline = time + tasks[i].deadline;

            }

        }


        // Find the task with the earliest deadline

        Task *earliest_task = NULL;

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

            if (tasks[i].remaining_time > 0) {
```

```
            if (earliest_task == NULL || tasks[i].next_deadline <
earliest_task->next_deadline) {

                earliest_task = &tasks[i];

            }

        }

    }


    // Execute the task with the earliest deadline

    if (earliest_task != NULL) {

        earliest_task->remaining_time--;

        printf("Time %d: Executing task %d\n", time, earliest_task->id);

    } else {

        printf("Time %d: Idle\n", time);

    }


    // Increment time

    time++;

  }

}


int main() {

  Task tasks[MAX_TASKS];

  int n, totalTime;


  printf("Enter the number of tasks (max %d): ", MAX_TASKS);
```

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

    for (int i = 0; i < n; i++) {
        printf("Enter details for task %d\n", i + 1);
        tasks[i].id = i + 1;
        printf("Period: ");
        scanf("%d", &tasks[i].period);
        printf("Execution time: ");
        scanf("%d", &tasks[i].execution_time);
        printf("Deadline: ");
        scanf("%d", &tasks[i].deadline);
        tasks[i].remaining_time = 0;
        tasks[i].next_deadline = 0;
    }

    printf("Enter the total simulation time: ");
    scanf("%d", &totalTime);

    earliestDeadline(tasks, n, totalTime);

    return 0;
}
```

```
PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc earliestDeadline.c -o earliestDeadline } ; if ($?) { .\earliestDea
dline }
Enter the number of tasks (max 10): 3
Enter details for task 1
Period: 20
Execution time: 3
Deadline: 7
Enter details for task 2
Period: 5
Execution time: 2
Deadline: 4
Enter details for task 3
Period: 10
Execution time: 2
Deadline: 8
Enter the total simulation time: 20
EDF
Time 0: Executing task 2
Time 1: Executing task 2
Time 2: Executing task 1
Time 3: Executing task 1
Time 4: Executing task 1
Time 5: Executing task 3
Time 6: Executing task 3
Time 7: Executing task 2
Time 8: Executing task 2
Time 9: Idle
Time 10: Executing task 2
Time 11: Executing task 2
Time 12: Executing task 3
Time 13: Executing task 3
Time 14: Idle
Time 15: Executing task 2
```

Program 5:

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

```
#include<stdio.h>

#include <stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()

{

    int n;

    void producer();

    void consumer();

    int wait(int);

    int signal(int);
```

```
    printf("\n1. Producer\t2.Consumer\t3.Exit\n");
    while(1){
        printf("\nEnter Choice\n");
        scanf("%d",&n);
        switch(n){
            case 1 : if((mutex==1) && (empty!=0)) producer();
                    else printf("Buffer is full");
                    break;
            case 2 : if((mutex==1)&&(full!=0)) consumer();
                    else printf("Buffer is empty");
                    break;
            case 3 : exit(0);
                    break;
        }
    }
    return 0;
}
int wait(int s){
    return --s;
}
int signal(int s){
    return ++s;
}
void producer(){
```

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

```
PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc ProducerConsumer.c -o ProducerConsumer } ; if ($?) { .\ProducerConsumer }


1. Producer     2.Consumer      3.Exit

Enter Choice
1

Producer produces the item 1
Enter Choice
1

Producer produces the item 2
Enter Choice
1

Producer produces the item 3
Enter Choice
1
Buffer is full
Enter Choice
2

Consumer consumes item 3
Enter Choice
2

Consumer consumes item 2
Enter Choice
2

Consumer consumes item 1
Enter Choice
```

Program 6:

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

```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.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) {

       state[phnum] = EATING;

       sleep(2);

       printf("Philosopher %d takes fork %d and %d\n", phnum + 1, LEFT + 1, phnum +
1);

       printf("Philosopher %d is eating\n", phnum + 1);

       sem_post(&S[phnum]);

   }

}


void take_fork(int phnum) {

   sem_wait(&mutex);

   state[phnum] = HUNGRY;

   printf("Philosopher %d is hungry\n", phnum + 1);

   test(phnum);

   sem_post(&mutex);
```

```
    sem_wait(&S[phnum]);

    sleep(1);

}


void put_fork(int phnum) {

    sem_wait(&mutex);

    state[phnum] = THINKING;

    printf("Philosopher %d putting fork %d and %d down\n", phnum + 1, LEFT + 1,
phnum + 1);

    printf("Philosopher %d is thinking\n", phnum + 1);

    test(LEFT);

    test(RIGHT);

    sem_post(&mutex);

}


void* philosopher(void* num) {

    while (1) {

        int* i = num;

        sleep(1);

        take_fork(*i);

        sleep(0);

        put_fork(*i);

    }

}
```

```
int main() {
    int i;
    pthread_t thread_id[N];
    sem_init(&mutex, 0, 1);
    for (i = 0; i < N; i++) {
        sem_init(&S[i], 0, 0);
    }
    for (i = 0; i < N; i++) {
        pthread_create(&thread_id[i], NULL, philosopher, &phil[i]);
        printf("Philosopher %d is thinking\n", i + 1);
    }
    for (i = 0; i < N; i++) {
        pthread_join(thread_id[i], NULL);
    }
    sem_destroy(&mutex);
    for (i = 0; i < N; i++) {
        sem_destroy(&S[i]);
    }

    return 0;
}
```

```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is hungry
Philosopher 2 is hungry
Philosopher 3 is hungry
Philosopher 4 is hungry
Philosopher 5 is hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is eating
Philosopher 1 takes fork 5 and 1
Philosopher 1 is eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is eating
Philosopher 5 is hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
```

Program 7:

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

```
#include <stdio.h>

#include <stdbool.h>


#define P 5 // processes
```

```
#define R 3 // resources

int allocation[P][R] = {
    {0, 1, 0}, // P0
    {2, 0, 0},
    {3, 0, 2},
    {2, 1, 1},
    {0, 0, 2}
};

int maximum[P][R] = {
    {7, 5, 3},
    {3, 2, 2},
    {9, 0, 2},
    {2, 2, 2},
    {4, 3, 3}
};

int available[R] = {3, 3, 2};

bool isSafeSequence() {
    int work[R];
    for (int i = 0; i < R; i++)
        work[i] = available[i];
```

```
bool finish[P] = {0};
int safeSeq[P];
int count = 0;

while (count < P) {
    bool found = false;
    for (int p = 0; p < P; p++) {
        if (!finish[p]) {
            int j;
            for (j = 0; j < R; j++)
                if (allocation[p][j] + work[j] < maximum[p][j])
                    break;

            if (j == R) {
                for (int k = 0; k < R; k++)
                    work[k] += allocation[p][k];

                safeSeq[count++] = p;
                finish[p] = true;
                found = true;
            }
        }
    }
```

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

    printf("System is in a safe state.\nSafe sequence is: ");
    for (int i = 0; i < P; i++)
        printf("%d ", safeSeq[i]);
    printf("\n");
    return true;
}

int main() {
    isSafeSequence();
    return 0;
}
```

```
PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc BankerAlgo.c -o BankerAlgo } ; if ($?) { .\BankerAlgo }
System is in a safe state.
Safe sequence is: 1 3 4 0 2
PS C:\TISSA\OS 2023-24>
```

Program 8:

Write a C program to simulate deadlock detection

```
#include <stdio.h>


#define MAX_PROCESSES 5

#define MAX_RESOURCES 3


int allocated[MAX_PROCESSES][MAX_RESOURCES];

int requested[MAX_PROCESSES][MAX_RESOURCES];

int available[MAX_RESOURCES];

int work[MAX_RESOURCES];

int finish[MAX_PROCESSES];


void initialize()

{

    // Initialize allocated and requested matrices

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

    {

        printf("Enter allocated resources for process P%d:\n", i);

        for (int j = 0; j < MAX_RESOURCES; j++)

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


        printf("Enter requested resources for process P%d:\n", i);

        for (int j = 0; j < MAX_RESOURCES; j++)

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

```
        finish[i] = 0; // Process is not finished yet


    }
}

int checkSafety()
{
    for (int i = 0; i < MAX_RESOURCES; i++)
        work[i] = available[i];

    int count = 0;
    while (count < MAX_PROCESSES)
    {
        int found = 0;
        for (int i = 0; i < MAX_PROCESSES; i++)
        {
            if (!finish[i])
            {
                int j;
                for (j = 0; j < MAX_RESOURCES; j++)
                {
                    if (requested[i][j] > work[j])
```

```
                break;
            }
            if (j == MAX_RESOURCES)
            {
                for (int k = 0; k < MAX_RESOURCES; k++)
                    work[k] += allocated[i][k];
                finish[i] = 1;
                found = 1;
                count++;
            }
        }
    }
    if (!found)
        break;
  }

  return count == MAX_PROCESSES;
}

int main()
{
  initialize();

  // Assume available resources are initially zero
```

```
    for (int i = 0; i < MAX_RESOURCES; i++)
        available[i] = 0;

    if (checkSafety())
        printf("System is in safe state.\n");
    else
        printf("System is in unsafe state.\n");

    return 0;
}
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   GITLENS

PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc deadlockDetection.c -o deadlockDetection } ; if ($?) { .\deadlockDetectio
n }
Enter allocated resources for process P0:
0 1 0
Enter requested resources for process P0:
0 0 0
Enter allocated resources for process P1:
2 0 0
Enter requested resources for process P1:
2 0 2
Enter allocated resources for process P2:
3 0 3
Enter requested resources for process P2:
0 0 0
Enter allocated resources for process P3:
2 1 1
Enter requested resources for process P3:
0 0 2
Enter allocated resources for process P4:
0 0 2
Enter requested resources for process P4:
0 0 2
System is in safe state.
PS C:\TISSA\OS 2023-24>
```

Program 9:

Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit b) Best-fit c) First-fit

```
#include <stdio.h>

#include <stdlib.h>


#define MAX 25


int frag[MAX], b[MAX], f[MAX], nf, nb;

int bf[MAX], ff[MAX];


void firstfit() {

    int i, j;


    for (i = 1; i <= nf; i++) {

        for (j = 1; j <= nb; j++) {

            if (bf[j] == 0 && b[j] >= f[i]) {

                ff[i] = j;

                frag[i] = b[j] - f[i];

                bf[j] = 1;

                break;

            }

        }

    }


    printf("\nFile Size:\tBlock Size:");
```

```
    for (i = 1; i <= nf; i++) {
        if (ff[i] != 0)
            printf("\n%d\t\t%d\t\t", f[i], b[ff[i]]);
        else
            printf("\n%d\t\tNot Allocated", f[i]);
    }
    printf("\n");
}

void bestfit() {
    int i, j, bestIdx;

    for (i = 1; i <= nf; i++) {
        bestIdx = -1;
        for (j = 1; j <= nb; j++) {
            if (bf[j] == 0 && b[j] >= f[i]) {
                if (bestIdx == -1 || b[j] < b[bestIdx]) {
                    bestIdx = j;
                }
            }
        }

        if (bestIdx != -1) {
            ff[i] = bestIdx;
```

```
            frag[i] = b[bestIdx] - f[i];
            bf[bestIdx] = 1;
        }
    }

    printf("\nFile Size:\tBlock Size:");
    for (i = 1; i <= nf; i++) {
        if (ff[i] != 0)
            printf("\n%d\t\t%d\t\t", f[i], b[ff[i]]);
        else
            printf("\n%d\t\tNot Allocated", f[i]);
    }
    printf("\n");
}

void worstfit() {
    int i, j, worstIdx;

    for (i = 1; i <= nf; i++) {
        worstIdx = -1;
        for (j = 1; j <= nb; j++) {
            if (bf[j] == 0 && b[j] >= f[i]) {
                if (worstIdx == -1 || b[j] > b[worstIdx]) {
                    worstIdx = j;
```

```
            }
        }
    }

    if (worstIdx != -1) {
        ff[i] = worstIdx;
        frag[i] = b[worstIdx] - f[i];
        bf[worstIdx] = 1;
    }
}

printf("\nFile Size:\tBlock Size:");
for (i = 1; i <= nf; i++) {
    if (ff[i] != 0)
        printf("\n%d\t\t%d\t\t", f[i], b[ff[i]]);
    else
        printf("\n%d\t\tNot Allocated", f[i]);
}
printf("\n");
}

int main() {
    int c;
```

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

printf("Enter the size of the blocks:\n");
for (int i = 1; i <= nb; i++) {
    printf("Block %d: ", i);
    scanf("%d", &b[i]);
    bf[i] = 0; // initialize
}

printf("Enter the size of the files:\n");
for (int i = 1; i <= nf; i++) {
    printf("File %d: ", i);
    scanf("%d", &f[i]);
}

while (1) {
    printf("\n1. First Fit 2. Best Fit 3. Worst Fit 4. Exit");
    printf("\nEnter choice: ");
    scanf("%d", &c);
    switch (c) {
        case 1:
```

```
            firstfit();
            break;
        case 2:
            bestfit();
            break;
        case 3:
            worstfit();
            break;
        case 4:
            return 0;
        default:
            printf("Invalid choice\n");
        }

        // Reset for next
        for (int i = 1; i <= nb; i++) bf[i] = 0;
        for (int i = 1; i <= nf; i++) ff[i] = 0;
    }
    return 0;
}
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   GITLENS

PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc bestworstfirstfit.c -o bestworstfirstfit } ; if ($?) { .\bestworstfirstfi
t }
Enter the number of blocks: 5
Enter the number of files: 4
Enter the size of the blocks:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600
Enter the size of the files:
File 1: 212
File 2: 417
File 3: 112
File 4: 426

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

File Size:      Block Size:
212             500
417             600
112             200
426             Not Allocated

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

File Size:      Block Size:
212             300
417             500
112             200
426             600
```

Program 10:

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

```
#include <stdio.h>

#include <stdlib.h>



#define MAX 50



void fifo(int pages[], int n, int capacity) {

    int frames[MAX], page_faults = 0, next = 0;

    for (int i = 0; i < capacity; i++)
```

```
        frames[i] = -1;

    printf("\nFIFO Page Replacement:\n");

    for (int i = 0; i < n; i++) {
        int found = 0;
        for (int j = 0; j < capacity; j++) {
            if (frames[j] == pages[i]) {
                found = 1;
                break;
            }
        }
        if (!found) {
            frames[next] = pages[i];
            next = (next + 1) % capacity;
            page_faults++;
        }

        printf("\nFrames: ");
        for (int j = 0; j < capacity; j++)
            printf("%d ", frames[j]);
    }
    printf("\nTotal Page Faults: %d\n", page_faults);
}
```

```
void lru(int pages[], int n, int capacity) {
    int frames[MAX], page_faults = 0, time[MAX], counter = 0;
    for (int i = 0; i < capacity; i++) {
        frames[i] = -1;
        time[i] = 0;
    }

    printf("\nLRU Page Replacement:\n");

    for (int i = 0; i < n; i++) {
        int found = 0, lru_index = 0, min_time = counter;
        for (int j = 0; j < capacity; j++) {
            if (frames[j] == pages[i]) {
                found = 1;
                time[j] = counter++;
                break;
            }
        }
        if (!found) {
            for (int j = 0; j < capacity; j++) {
                if (frames[j] == -1) {
                    lru_index = j;
                    break;
```

```
            }
            if (time[j] < min_time) {
                min_time = time[j];
                lru_index = j;
            }
        }
        frames[lru_index] = pages[i];
        time[lru_index] = counter++;
        page_faults++;
    }

    printf("\nFrames: ");
    for (int j = 0; j < capacity; j++)
        printf("%d ", frames[j]);
  }
  printf("\nTotal Page Faults: %d\n", page_faults);
}

void optimal(int pages[], int n, int capacity) {
  int frames[MAX], page_faults = 0;
  for (int i = 0; i < capacity; i++)
    frames[i] = -1;

  printf("\nOptimal Page Replacement:\n");
```

```
for (int i = 0; i < n; i++) {
    int found = 0;
    for (int j = 0; j < capacity; j++) {
        if (frames[j] == pages[i]) {
            found = 1;
            break;
        }
    }
    if (!found) {
        int pos = -1, farthest = i + 1;
        for (int j = 0; j < capacity; j++) {
            int k;
            for (k = i + 1; k < n; k++) {
                if (frames[j] == pages[k]) {
                    if (k > farthest) {
                        farthest = k;
                        pos = j;
                    }
                    break;
                }
            }
            if (k == n) {
                pos = j;
```

```
                break;
            }
        }
        if (pos == -1)
            pos = 0;
        frames[pos] = pages[i];
        page_faults++;
    }

    printf("\nFrames: ");
    for (int j = 0; j < capacity; j++)
        printf("%d ", frames[j]);
    }
    printf("\nTotal Page Faults: %d\n", page_faults);
}

int main() {
    int pages[MAX], n, capacity = 0;

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

    if (n <= 0) {
        printf("The number of pages should be greater than zero.\n");
```

```
        return 1;
    }

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

    printf("Enter the capacity of frames: ");
    scanf("%d", &capacity);

    if (capacity <= 0) {
        printf("The capacity of frames should be greater than zero.\n");
        return 1;
    }

    fifo(pages, n, capacity);
    lru(pages, n, capacity);
    optimal(pages, n, capacity);

    return 0;
}
```

```
PS C:\TISSA\OS 2023-24> cd "c:\TISSA\OS 2023-24\" ; if ($?) { gcc pageReplacementLRUFIFO.c -o pageReplacementLRUFIFO } ; if ($?) { .
\pageReplacementLRUFIFO }
Enter the number of pages: 20
Enter the pages: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the capacity of frames: 3

FIFO Page Replacement:

Frames: 7 -1 -1
Frames: 7 0 -1
Frames: 7 0 1
Frames: 2 0 1
Frames: 2 0 1
Frames: 2 3 1
Frames: 2 3 0
Frames: 4 3 0
Frames: 4 2 0
Frames: 4 2 3
Frames: 0 2 3
Frames: 0 2 3
Frames: 0 2 3
Frames: 0 1 3
Frames: 0 1 2
Frames: 0 1 2
Frames: 0 1 2
Frames: 7 1 2
Frames: 7 0 2
Frames: 7 0 1
Total Page Faults: 15

LRU Page Replacement:

Frames: 7 -1 -1
```

```
LRU Page Replacement:

Frames: 7 -1 -1
Frames: 7 0 -1
Frames: 7 0 1
Frames: 2 0 1
Frames: 2 0 1
Frames: 2 0 3
Frames: 2 0 3
Frames: 4 0 3
Frames: 4 0 2
Frames: 4 3 2
Frames: 0 3 2
Frames: 0 3 2
Frames: 0 3 2
Frames: 1 3 2
Frames: 1 3 2
Frames: 1 0 2
Frames: 1 0 2
Frames: 1 0 7
Frames: 1 0 7
Frames: 1 0 7
Total Page Faults: 12

Optimal Page Replacement:

Frames: 7 -1 -1
Frames: 7 0 -1
Frames: 7 0 1
Frames: 2 0 1
Frames: 2 0 1
Frames: 2 0 3
```

```
Total Page Faults: 12

Optimal Page Replacement:

Frames: 7 -1 -1
Frames: 7 0 -1
Frames: 7 0 1
Frames: 2 0 1
Frames: 2 0 1
Frames: 2 0 3
Frames: 2 0 3
Frames: 2 4 3
Frames: 2 4 3
Frames: 2 4 3
Frames: 2 0 3
Frames: 2 0 3
Frames: 2 0 3
Frames: 2 0 1
Frames: 2 0 1
Frames: 2 0 1
Frames: 2 0 1
Frames: 7 0 1
Frames: 7 0 1
Frames: 7 0 1
Total Page Faults: 9
PS C:\TISSA\OS 2023-24>
```