VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT on

Operating Systems Lab

Submitted by

Dhravya M(1BM21CS056)

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
May-2023 to July-2023

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 Lab**" carried out by **Dhravya M(1BM21CS056),** 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 academic semester June-2023 to October-2023. The Lab report has been approved as it satisfies the academic requirements in respect of **Operating Systems Lab (22CS4PCOPS)** work prescribed for the said degree.

Dr. Nandini Vineeth
Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course Outcome:

CO1	Apply the different concepts and functionality of operating System
CO2	Apply variousOperating System strategies and teqniques
CO3	Demonstrate the different functionality of Operating Systems
CO4	Conduct Practical Experiment to implement the functionalities of operating Systems

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

- **FCFS**
- **SJF (preemptive & Non- preemptive)**

a)FCFS

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

int main()
{
    int n, i;
    float waitingTime, turnAroundTime;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    float *bt = (float *)malloc(n * sizeof(float));
    float *wt = (float *)malloc(n * sizeof(float));
    float *tt = (float *)malloc(n * sizeof(float));
    printf("Enter the burst times of %d processes: \n-------------------------------------
\n", n);
    for (i = 0; i < n; i++)
    {
        printf("Enter the burst times Process%d:",i+1);
        scanf("%f", &bt[i]);
    }
    printf("\nThe details of the processes are as below:\nProcess\tBurst
Time\tTurn Around Time\tWaiting Time\n");
    for (i = 0; i < n; i++)
    {
        if (i == 0)
        {
            wt[0] = 0;
        }
        else
        {
            wt[i] = bt[i - 1] + wt[i - 1];
        }
        tt[i] = bt[i] + wt[i];
        printf("   %d   \t%f\t\t%f\t\t%f\n", i + 1, bt[i], tt[i], wt[i]);
        waitingTime += wt[i];
        turnAroundTime += tt[i];
    }
```

```
    printf("The average waiting time is: %f", waitingTime/n);
    printf("\nThe average turn around time is: %f", turnAroundTime / n);
    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc "FCFS(CPU scheduling).c"

D:\Codes\c\OS_Lab>.\a.exe
Enter the number of processes: 4
Enter the burst times of 4 processes:
------------------------
Enter the burst times Process1:4
Enter the burst times Process2:5
Enter the burst times Process3:2
Enter the burst times Process4:7

The details of the processes are as below:
Process Burst Time      Turn Around Time        Waiting Time
   1    4.000000                4.000000                0.000000
   2    5.000000                9.000000                4.000000
   3    2.000000                11.000000               9.000000
   4    7.000000                18.000000               11.000000
The average waiting time is: 6.000000
The average turn around time is: 10.500000
```

b) SJF (Non-Preemptive)

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

int main()
{
    int n,i,j,index;
    float WT,TurnAroundTime,temp;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    float *bt = (float *)malloc(n * sizeof(float));
    float *wt = (float *)malloc(n * sizeof(float));
    float *tt = (float *)malloc(n * sizeof(float));
     printf("Enter the burst times of %d processes: \n-----------------------------------
\n", n);
    for (i = 0; i < n; i++)
    {
        printf("Enter the burst times Process%d:",i+1);
        scanf("%f", &bt[i]);
    }

    for(i = 0; i < n-1; i++){
        // index = i;
        for(j=0; j < n-i-1; j++){

            if(bt[j]>bt[j+1]){
                temp = bt[j];
                bt[j] = bt[j+1];
                bt[j+1] = temp;
            }
        }
    }

 printf("\nThe details of the processes are as below:\nProcess\tBurst Time\tTurn
Around Time\tWaiting Time\n");
    for (i = 0; i < n; i++)
    {
        if (i == 0)
        {
            wt[0] = 0;
        }
        else
        {
```

```
            wt[i] = bt[i - 1] + wt[i - 1];
        }
        tt[i] = bt[i] + wt[i];
        printf("   %d   \t%f\t\t%f\t   %f\n", i + 1, bt[i], tt[i], wt[i]);
        WT = WT + wt[i];
        TurnAroundTime = TurnAroundTime + tt[i];
    }
    printf("The average waiting time is: %f", WT/n);
    printf("\nThe average turn around time is: %f", TurnAroundTime/n);
    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc "SJF(Non-Premptive).c"

D:\Codes\c\OS_Lab>.\a.exe
Enter the number of processes: 4
Enter the burst times of 4 processes:
------------------------
Enter the burst times Process1:4
Enter the burst times Process2:5
Enter the burst times Process3:2
Enter the burst times Process4:7

The details of the processes are as below:
Process Burst Time      Turn Around Time        Waiting Time
   1    2.000000                2.000000           0.000000
   2    4.000000                6.000000           2.000000
   3    5.000000                11.000000          6.000000
   4    7.000000                18.000000          11.000000
The average waiting time is: 4.750000
The average turn around time is: 9.250000
```

c) SJF (Pre-Emptive)

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

struct Process
{
    int pid;
    int bt;
    int art;
};

void findWaitingTime(struct Process proc[], int n, int wt[])
{
    int rt[n];
    for (int i = 0; i < n; i++)
    {
        rt[i] = proc[i].bt;
    }

    int complete = 0, t = 0, minm = 99999;
    int shortest = 0, finish_time;
    bool check = false;

    while (complete != n)
    {

        for (int j = 0; j < n; j++)
        {
            if ((proc[j].art <= t) &&
                (rt[j] < minm) && rt[j] > 0)
            {
                minm = rt[j];
                shortest = j;
                check = true;
            }
        }

        if (check == false)
        {
            t++;
            continue;
        }

        rt[shortest]--;
```

```
        minm = rt[shortest];

        if (minm == 0)
            minm = 99999;

        if (rt[shortest] == 0)
        {

            complete++;
            check = false;
            finish_time = t + 1;
            wt[shortest] = finish_time - proc[shortest].bt - proc[shortest].art;
            if (wt[shortest] < 0)
                wt[shortest] = 0;
        }
        t++;
    }
}

void findTurnAroundTime(struct Process proc[], int n, int wt[], int tat[])
{
    for (int i = 0; i < n; i++)
    {
        tat[i] = proc[i].bt + wt[i];
    }
}

void findavgTime(struct Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    findWaitingTime(proc, n, wt);
    findTurnAroundTime(proc, n, wt, tat);
    printf("Processes\tBurst time\tWaiting time\tTurn around time\n");

    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        printf(" %d\t\t%d\t\t%d\t\t%d\t\t\n", proc[i].pid, proc[i].bt, wt[i],
tat[i]);
    }

    printf("Average waiting time = %f", (float)total_wt / (float)n);
    printf("\nAverage turn around time = %f", (float)total_tat / (float)n);
}
```

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

    printf("Enter the burst times of %d processes: \n-----------------------------------
\n", n);
    for (int i = 0; i < n; i++)
    {
         printf("Enter the burst times and Arrival time Process%d:",i+1);
        scanf("%d %d", &proc[i].bt, &proc[i].art);
        proc[i].pid = i + 1;
    }

    findavgTime(proc, n);
    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc "SJF(premptive).c"

D:\Codes\c\OS_Lab>.\a.exe
Enter the number of processes: 4
Enter the burst times of 4 processes:
-------------------------
Enter the burst times and Arrival time Process1:4 0
Enter the burst times and Arrival time Process2:3 0
Enter the burst times and Arrival time Process3:5 1
Enter the burst times and Arrival time Process4:6 2
Processes       Burst time      Waiting time    Turn around time
 1              4               3               7
 2              3               0               3
 3              5               6               11
 4              6               10              16
Average waiting time = 4.750000
Average turn around time = 9.250000
```

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

- Priority

- Round Robin

Priority

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

struct process {
    int proc_id;
    int bt;
    int priority;
    int wt;
    int tat;
};

void find_wt(struct process[], int, int[]);
void find_tat(struct process[], int, int[], int[]);

void find_average_time(struct process[], int);

void priority_scheduling(struct process[], int);

int main()
{
    int n, i;
    struct process proc[10];

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

for(i = 0; i< n; i++)
    {
printf("\nEnter the process ID: ");
scanf("%d", &proc[i].proc_id);

printf("Enter the burst time: ");
scanf("%d", &proc[i].bt);

printf("Enter the priority: ");
scanf("%d", &proc[i].priority);
```

```
    }
priority_scheduling(proc, n);
    return 0;
}

void find_wt(struct process proc[], int n, int wt[])
{
    int i;
    wt[0] = 0;
for(i = 1; i< n; i++)
    {
    wt[i] = proc[i - 1].bt + wt[i - 1];
    }
}

void find_tat(struct process proc[], int n, int wt[], int tat[])
{
    int i;
for(i = 0; i< n; i++)
    {
tat[i] = proc[i].bt + wt[i];
    }
}

void find_average_time(struct process proc[], int n)
{
    int wt[10], tat[10], total_wt = 0, total_tat = 0, i;

find_wt(proc, n, wt);
find_tat(proc, n, wt, tat);

printf("\nProcess ID\tBurst Time\tPriority\tWaiting Time\tTurnaround Time");
for(i = 0; i< n; i++)
    {
total_wt = total_wt + wt[i];
total_tat = total_tat + tat[i];
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", proc[i].proc_id, proc[i].bt,
proc[i].priority, wt[i], tat[i]);
    }
printf("\n\nAverage Waiting Time = %f", (float)total_wt/n);
printf("\nAverage Turnaround Time = %f\n", (float)total_tat/n);
}
void priority_scheduling(struct process proc[], int n)
{
int i, j, pos;
```

```
struct process temp;
for(i = 0; i< n; i++)
{
    pos = i;
for(j = i + 1; j < n; j++)
    {
        if(proc[j].priority< proc[pos].priority)
            pos = j;
    }

    temp = proc[i];
    proc[i] = proc[pos];
    proc[pos] = temp;
}
find_average_time(proc, n);
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc Priority_Scheduling.c

D:\Codes\c\OS_Lab>.\a.exe
Enter the number of processes: 4

Enter the process ID: 1
Enter the burst time: 4
Enter the priority: 2

Enter the process ID: 3
Enter the burst time: 5
Enter the priority: 3

Enter the process ID: 2
Enter the burst time: 6
Enter the priority: 4

Enter the process ID: 4
Enter the burst time: 6
Enter the priority: 1

Process ID      Burst Time      Priority        Waiting Time    Turnaround Time
4               6               1               0               6
1               4               2               6               10
3               5               3               10              15
2               6               4               15              21

Average Waiting Time = 7.750000
Average Turnaround Time = 13.000000
```

Round Robin

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

struct process {
    int proc_id;
    int bt;
    int priority;
    int wt;
    int tat;
};

void find_wt(struct process[], int, int[]);
void find_tat(struct process[], int, int[], int[]);

void find_average_time(struct process[], int);

void priority_scheduling(struct process[], int);

int main()
{
    int n, i;
    struct process proc[10];

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

for(i = 0; i< n; i++)
    {
printf("\nEnter the process ID: ");
scanf("%d", &proc[i].proc_id);

printf("Enter the burst time: ");
scanf("%d", &proc[i].bt);

printf("Enter the priority: ");
scanf("%d", &proc[i].priority);
    }

priority_scheduling(proc, n);
    return 0;
}
```

```
void find_wt(struct process proc[], int n, int wt[])
{
    int i;
    wt[0] = 0;

for(i = 1; i< n; i++)
    {
    wt[i] = proc[i - 1].bt + wt[i - 1];
    }
}

void find_tat(struct process proc[], int n, int wt[], int tat[])
{
    int i;
for(i = 0; i< n; i++)
    {
tat[i] = proc[i].bt + wt[i];
    }
}

void find_average_time(struct process proc[], int n)
{
    int wt[10], tat[10], total_wt = 0, total_tat = 0, i;

find_wt(proc, n, wt);
find_tat(proc, n, wt, tat);

printf("\nProcess ID\tBurst Time\tPriority\tWaiting Time\tTurnaround Time");

for(i = 0; i< n; i++)
    {
total_wt = total_wt + wt[i];
total_tat = total_tat + tat[i];
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", proc[i].proc_id, proc[i].bt,
proc[i].priority, wt[i], tat[i]);
    }

printf("\n\nAverage Waiting Time = %f", (float)total_wt/n);
printf("\nAverage Turnaround Time = %f\n", (float)total_tat/n);
}

void priority_scheduling(struct process proc[], int n)
{
int i, j, pos;
struct process temp;
```

```
for(i = 0; i< n; i++)
{
    pos = i;
for(j = i + 1; j < n; j++)
    {
        if(proc[j].priority< proc[pos].priority)
            pos = j;
    }

    temp = proc[i];
    proc[i] = proc[pos];
    proc[pos] = temp;
}

find_average_time(proc, n);
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc RoundRobin_Scheduling.c

D:\Codes\c\OS_Lab>.\a.exe
Enter the number of processes: 4

Enter the process ID: 1
Enter the burst time: 3
Enter the priority: 3

Enter the process ID: 2
Enter the burst time: 6
Enter the priority: 4

Enter the process ID: 3
Enter the burst time: 2
Enter the priority: 1

Enter the process ID: 4
Enter the burst time: 8
Enter the priority: 1

Process ID      Burst Time      Priority        Waiting Time    Turnaround Time
3               2               1               0               2
4               8               1               2               10
1               3               3               10              13
2               6               4               13              19

Average Waiting Time = 6.250000
Average Turnaround Time = 11.000000
```

Q. Write a 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. Use FCFS scheduling for the processes in each queue.

```
#include<stdio.h>
void swap(int *i,int *j)
{
    int temp=*i;
    *i=*j;
    *j=temp;
}
int main()
{
    int i, k, n;
    printf("Enter the number of processes:");
    scanf("%d",&n);
    int pid[n],bt[n], su[n], wt[n],tat[n];
    for(i=0;i<n;i++)
    {
        pid[i] = i;
        printf("Enter the Burst Time of Process%d:", i);
        scanf("%d",&bt[i]);
        printf("System/User Process (0/1) ? ");
        scanf("%d", &su[i]);
    }
    for(i=0;i<n;i++)
        for(k=i+1;k<n;k++)
            if(su[i] > su[k])
            {
            swap(&pid[i],&pid[k]);
            swap(&bt[i],&bt[k]);
            swap(&su[i],&su[k]);
            }
    float wtTotal= wt[0] = 0;
    float tatTotal= tat[0] = bt[0];
    for(i=1;i<n;i++)
    {
        wt[i] = wt[i-1] + bt[i-1];
        tat[i] = tat[i-1] + bt[i];
        wtTotal = wtTotal + wt[i];
        tatTotal = tatTotal + tat[i];
    }
```

```
    printf("\nPROCESS\t\t SYSTEM/USER PROCESS \tBURST TIME\tWAITING
TIME\tTURNAROUND TIME");
    for(i=0;i<n;i++)
        printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d
",pid[i],su[i],bt[i],wt[i],tat[i]);
    printf("\nAverage Waiting Time is --- %f",wtTotal/n);
    printf("\nAverage Turnaround Time is --- %f",tatTotal/n);
    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc Multi_Level_Queue_Scheduling.c

D:\Codes\c\OS_Lab>.\a.exe
Enter the number of processes:4
Enter the Burst Time of Process1:5
System/User Process (0/1) ? 0
Enter the Burst Time of Process2:4
System/User Process (0/1) ? 1
Enter the Burst Time of Process3:5
System/User Process (0/1) ? 1
Enter the Burst Time of Process4:3
System/User Process (0/1) ? 1

PROCESS          SYSTEM/USER PROCESS    BURST TIME      WAITING TIME    TURNAROUND TIME
0                0              5               0               5
1                1              4               5               9
2                1              5               9               14
3                1              3               14              17
Average Waiting Time is --- 7.000000
Average Turnaround Time is --- 11.250000
```

Q. Write a C program to simulate Real-Time CPU scheduling algorithms:

a) **Rate Monotonic**
b) **Earliest Deadline First**
c) **Proportional scheduling**

Rate Monotonic

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


void swap(int *i,int *j)
{
    int temp=*i;
    *i=*j;
    *j=temp;
}


int main()
{
    int i,temp,n;
    float wtavg,tatavg;
    printf("Enter number of processes: ");

    scanf("%d", &n);

    int pid[n],bt[n],su[n],wt[n],tat[n];

    for (i = 0; i < n; i++)
    {
        pid[i] = i;
        printf("\nEnter the burst time of Process %d :",i+1);
        scanf("%d",&bt[i]);

        printf("For a System Process(0) Else if its a User Process(1):");
        scanf("%d",&su[i]);
    }

    wtavg = wt[0] = 0;
    tatavg = tat[0] = bt[0];
```

```
    for(int i=0;i<n-1;i++)
    {
      for(int j=i+1;j<n;j++)
        {
            if(su[i]>su[j])
            {
                swap(&pid[i],&pid[j]);
                swap(&bt[i],&bt[j]);
                swap(&su[i],&su[j]);
            }
        }
    }

    for(i=1;i<n;i++)
    {
        wt[i] = wt[i-1] + bt[i-1];
        tat[i] = tat[i-1] + bt[i];
        wtavg += wt[i];
        tatavg += tat[i];
    }


    printf("\nProcess-ID \t System/User Process \t\t Burst Time \t\t Waiting Time
\t\t TAT ");

    for(int i =0;i<n;i++){
        printf("\n%d \t\t\t %d \t\t\t %d \t\t\t %d \t\t\t
%d",pid[i]+1,su[i],bt[i],wt[i],tat[i]);
    }


    printf("\nAverage Waiting Time:%0.3f",wtavg/n);
    printf("\nAverage TurnAroundTime:%0.3f",1.0*tatavg/n);
    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc Rate_Monotonic_Scheduling.c

D:\Codes\c\OS_Lab>.\a.exe
Enter number of processes: 4

Enter the burst time of Process 1 :4
For a System Process(0) Else if its a User Process(1):0

Enter the burst time of Process 2 :6
For a System Process(0) Else if its a User Process(1):1

Enter the burst time of Process 3 :3
For a System Process(0) Else if its a User Process(1):0

Enter the burst time of Process 4 :8
For a System Process(0) Else if its a User Process(1):0

Process-ID       System/User Process            Burst Time             Waiting Time           TAT
1                       0                       4                      0                      4
3                       0                       3                      4                      7
4                       0                       8                      7                      15
2                       1                       6                      15                     21
Average Waiting Time:6.500
Average TurnAroundTime:11.750
D:\Codes\c\OS_Lab>
```

Earliest Deadline First

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

#define arrival 0
#define execution   1
#define deadline 2
#define period 3
#define abs_arrival 4
#define execution_copy  5
#define abs_deadline 6
typedef struct
{
    int T[7],instance,alive;

}task;

#define IDLE_TASK_ID 1023
#define ALL 1
#define CURRENT 0

void get_tasks(task *t1,int n);
int hyperperiod_calc(task *t1,int n);
float cpu_util(task *t1,int n);
int gcd(int a, int b);
int lcm(int *a, int n);
int sp_interrupt(task *t1,int tmr,int n);
int min(task *t1,int n,int p);
void update_abs_arrival(task *t1,int n,int k,int all);
void update_abs_deadline(task *t1,int n,int all);
void copy_execution_time(task *t1,int n,int all);
int timer = 0;

int main()
{
    task *t;
    int n, hyper_period, active_task_id;
    float cpu_utilization;
    printf("Enter number of tasks:");
    scanf("%d", &n);
    t = (task*)malloc(n * sizeof(task));
    get_tasks(t, n);
```

```
    cpu_utilization = cpu_util(t, n);
    printf("CPU Utilization %f\n", cpu_utilization);

    if (cpu_utilization < 1)
        printf("Tasks can be scheduled\n");
    else
        printf("Schedule is not feasible\n");

    hyper_period = hyperperiod_calc(t, n);
    copy_execution_time(t, n, ALL);
    update_abs_arrival(t, n, 0, ALL);
    update_abs_deadline(t, n, ALL);

    while (timer <= hyper_period)
    {
        if (sp_interrupt(t, timer, n))
        {
            active_task_id = min(t, n, abs_deadline);
        }

        if (active_task_id == IDLE_TASK_ID)
        {
            printf("%d  Idle\n", timer);
        }

        if (active_task_id != IDLE_TASK_ID)
        {

            if (t[active_task_id].T[execution_copy] != 0)
            {
                t[active_task_id].T[execution_copy]--;
                printf("%d  Task %d\n", timer, active_task_id + 1);
            }
            if (t[active_task_id].T[execution_copy] == 0)
            {
                t[active_task_id].instance++;
                t[active_task_id].alive = 0;
                copy_execution_time(t, active_task_id, CURRENT);
                update_abs_arrival(t, active_task_id, t[active_task_id].instance,
CURRENT);
                update_abs_deadline(t, active_task_id, CURRENT);
                active_task_id = min(t, n, abs_deadline);
            }
        }
        ++timer;
```

```
    }
    free(t);
    return 0;
}
void get_tasks(task *t1, int n)
{
    int i = 0;
    while (i < n)
    {
        printf("Enter Task %d parameters\n", i + 1);
        printf("Arrival time: ");
        scanf("%d", &t1->T[arrival]);
        printf("Execution time: ");
        scanf("%d", &t1->T[execution]);
        printf("Deadline time: ");
        scanf("%d", &t1->T[deadline]);
        printf("Period: ");
        scanf("%d", &t1->T[period]);
        t1->T[abs_arrival] = 0;
        t1->T[execution_copy] = 0;
        t1->T[abs_deadline] = 0;
        t1->instance = 0;
        t1->alive = 0;
        t1++;
        i++;
    }
}

int hyperperiod_calc(task *t1, int n)
{
    int i = 0, ht, a[10];
    while (i < n)
    {
        a[i] = t1->T[period];
        t1++;
        i++;
    }
    ht = lcm(a, n);
    return ht;
}

int gcd(int a, int b)
{
    if (b == 0)
        return a;
```

```
    else
        return gcd(b, a % b);
}

int lcm(int *a, int n)
{
    int res = 1, i;
    for (i = 0; i < n; i++)
    {
        res = res * a[i] / gcd(res, a[i]);
    }
    return res;
}

int sp_interrupt(task *t1, int tmr, int n)
{
    int i = 0, n1 = 0, a = 0;
    task *t1_copy;
    t1_copy = t1;
    while (i < n)
    {
        if (tmr == t1->T[abs_arrival])
        {
            t1->alive = 1;
            a++;
        }
        t1++;
        i++;
    }
    t1 = t1_copy;
    i = 0;

    while (i < n)
    {
        if (t1->alive == 0)
            n1++;
        t1++;
        i++;
    }
    if (n1 == n || a != 0)
    {
        return 1;
    }
    return 0;
}
```

```
void update_abs_deadline(task *t1, int n, int all)
{
    int i = 0;
    if (all)
    {
        while (i < n)
        {
            t1->T[abs_deadline] = t1->T[deadline] + t1->T[abs_arrival];
            t1++;
            i++;
        }
    }
    else
    {
        t1 += n;
        t1->T[abs_deadline] = t1->T[deadline] + t1->T[abs_arrival];
    }
}

void update_abs_arrival(task *t1, int n, int k, int all)
{
    int i = 0;
    if (all)
    {
        while (i < n)
        {
            t1->T[abs_arrival] = t1->T[arrival] + k * (t1->T[period]);
            t1++;
            i++;
        }
    }
    else
    {
        t1 += n;
        t1->T[abs_arrival] = t1->T[arrival] + k * (t1->T[period]);
    }
}

void copy_execution_time(task *t1, int n, int all)
{
    int i = 0;
    if (all)
    {
        while (i < n)
```

```
        {
            t1->T[execution_copy] = t1->T[execution];
            t1++;
            i++;
        }
    }
    else
    {
        t1 += n;
        t1->T[execution_copy] = t1->T[execution];
    }
}

int min(task *t1, int n, int p)
{
    int i = 0, min = 0x7FFF, task_id = IDLE_TASK_ID;
    while (i < n)
    {
        if (min > t1->T[p] && t1->alive == 1)
        {
            min = t1->T[p];
            task_id = i;
        }
        t1++;
        i++;
    }
    return task_id;
}

float cpu_util(task *t1, int n)
{
    int i = 0;
    float cu = 0;
    while (i < n)
    {
        cu = cu + (float)t1->T[execution] / (float)t1->T[deadline];
        t1++;
        i++;
    }
    return cu;
}
```

Output:

```
D:\Codes\c\OS_Lab>.\a.exe
Enter number of tasks:3
Enter Task 1 parameters
Arrival time: 0
Execution time: 1
Deadline time: 4
Period: 4
Enter Task 2 parameters
Arrival time: 0
Execution time: 2
Deadline time: 6
Period: 6
Enter Task 3 parameters
Arrival time: 0
Execution time: 3
Deadline time: 8
Period: 8
CPU Utilization 0.958333
 Tasks can be scheduled
 0  Task 1
 1  Task 2
 2  Task 2
 3  Task 3
 4  Task 1
 5  Task 3
 6  Task 3
 7  Task 2
 8  Task 1
 9  Task 2
 10  Task 3
 11  Task 3
 12  Task 1
 13  Task 3
 14  Task 2
 15  Task 2
 16  Task 1
 17  Task 3
 18  Task 2
 19  Task 2
 20  Task 1
 21  Task 3
 22  Task 3
 23  Idle
 24  Task 1
```

Proportional Scheduling

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

int main() {

    srand(time(0));
    int numbers[5];
    int i;

    for (i = 0; i < 5; i++) {
        numbers[i] = rand() % 10 + 1;
    }

    printf("Initial Numbers: ");
    for (i = 0; i < 5; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n");

    while (1) {

        int all_zero = 1;
        for (i = 0; i < 5; i++) {
            if (numbers[i] > 0) {
                all_zero = 0;
                break;
            }
        }

        if (all_zero) {
            break;
        }

        int selected_index;
        do {
            selected_index = rand() % 5;
        } while (numbers[selected_index] == 0);

        numbers[selected_index]--;
        printf("Decrementing number at index %d: ", selected_index);
        for (i = 0; i < 5; i++) {
```

```
            printf("%d ", numbers[i]);
        }
        printf("\n");
    }

    printf("All numbers reached 0.\n");

    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc Priority_Scheduling.c

D:\Codes\c\OS_Lab>.\a.exe
Enter the number of processes: 3

Enter the process ID: 1
Enter the burst time: 3
Enter the priority: 2

Enter the process ID: 2
Enter the burst time: 4
Enter the priority: 3

Enter the process ID: 3
Enter the burst time: 1
Enter the priority: 1

Process ID      Burst Time      Priority        Waiting Time    Turnaround Time
3               1               1               0               1
1               3               2               1               4
2               4               3               4               8

Average Waiting Time = 1.666667
Average Turnaround Time = 4.333333
```

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

```c
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdlib.h>
#include <windows.h>
#include <time.h>

pthread_mutex_t mutex;
sem_t empty, full;
int in=0, out=0, buffer[5];

void *producer(void *pno){
    for(int i=0;i<5;i++){
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        int x = rand()%100;
        buffer[in]=x;
        in = (in+1)%5;
        printf("Producer %d has put %d in buffer\n",*((int*)pno), x);
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
}

void *consumer(void* cno){
    for(int i=0;i<5;i++){
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int x = buffer[out];
        out = (out+1)%5;
        printf("Comsumer %d has consumed %d\n",*((int*)cno), x);
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
}

void main(){
    pthread_t prod[5], con[5];
    sem_init(&empty,0,10);
    sem_init(&full,0,0);
    pthread_mutex_init(&mutex,NULL);
```

```
    int a[] = {1,2,3,4,5};

    for(int i=0;i<5;i++){
        pthread_create(&prod[i],NULL,(void*)producer, (void*)&a[i]);
        pthread_create(&con[i],NULL,(void*)consumer, (void*)&a[i]);
    }

    for(int i=0;i<5;i++){
        pthread_join(prod[i],NULL);
        pthread_join(con[i],NULL);
    }
    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc Producer_Consumer.c

D:\Codes\c\OS_Lab>.\a.exe
Producer 2 has put 41 in buffer
Producer 2 has put 67 in buffer
Producer 2 has put 34 in buffer
Producer 4 has put 41 in buffer
Producer 4 has put 67 in buffer
Producer 4 has put 34 in buffer
Comsumer 1 has consumed 34
Comsumer 1 has consumed 67
Comsumer 2 has consumed 34
Comsumer 3 has consumed 41
Producer 2 has put 0 in buffer
Producer 2 has put 69 in buffer
Comsumer 2 has consumed 67
Producer 5 has put 41 in buffer
Producer 5 has put 67 in buffer
Producer 5 has put 34 in buffer
Producer 5 has put 0 in buffer
Producer 3 has put 41 in buffer
Comsumer 1 has consumed 34
Comsumer 1 has consumed 0
Comsumer 1 has consumed 41
Comsumer 3 has consumed 41
Comsumer 3 has consumed 67
Comsumer 2 has consumed 34
```

```
Producer 4 has put 0 in buffer
Comsumer 4 has consumed 0
Producer 5 has put 69 in buffer
Comsumer 5 has consumed 41
Producer 3 has put 67 in buffer
Producer 1 has put 41 in buffer
Comsumer 3 has consumed 0
Producer 4 has put 69 in buffer
Comsumer 3 has consumed 69
Producer 3 has put 34 in buffer
Producer 3 has put 0 in buffer
Producer 3 has put 69 in buffer
Comsumer 5 has consumed 69
Comsumer 5 has consumed 41
Comsumer 5 has consumed 69
Comsumer 5 has consumed 34
Comsumer 2 has consumed 0
Comsumer 4 has consumed 69
Producer 1 has put 67 in buffer
Producer 1 has put 34 in buffer
Producer 1 has put 0 in buffer
Producer 1 has put 69 in buffer
Comsumer 4 has consumed 67
Comsumer 4 has consumed 34
Comsumer 4 has consumed 0
Comsumer 2 has consumed 69
```

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

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.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 that eating
        state[phnum] = EATING;

        sleep(2);

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

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

        // sem_post(&S[phnum]) has no effect
        // during takefork
        // used to wake up hungry philosophers
        // during putfork
        sem_post(&S[phnum]);
    }
}
```

```c
// take up chopsticks
void take_fork(int phnum)
{

    sem_wait(&mutex);

    // state that hungry
    state[phnum] = HUNGRY;

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

    // eat if neighbours are not eating
    test(phnum);

    sem_post(&mutex);

    // if unable to eat wait to be signalled
    sem_wait(&S[phnum]);

    sleep(1);
}

// put down chopsticks
void put_fork(int phnum)
{

    sem_wait(&mutex);

    // state that thinking
    state[phnum] = THINKING;

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

    test(LEFT);
    test(RIGHT);

    sem_post(&mutex);
}

void* philosopher(void* num)
{

    while (1) {
```

```
        int* i = num;

        sleep(1);

        take_fork(*i);

        sleep(0);

        put_fork(*i);
    }
}

int main()
{

    int i;
    pthread_t thread_id[N];

    // initialize the semaphores
    sem_init(&mutex, 0, 1);

    for (i = 0; i < N; i++)

        sem_init(&S[i], 0, 0);

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

        // create philosopher processes
        pthread_create(&thread_id[i], NULL,
                       philosopher, &phil[i]);

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

    for (i = 0; i < N; i++)

        pthread_join(thread_id[i], NULL);
}
```

Output:

```
D:\Codes\c\OS_Lab>.\a.exe
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 5 is Hungry
Philosopher 4 is Hungry
Philosopher 3 is Hungry
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 3 is Hungry
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
```

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

```c
#include <stdlib.h>
#include <stdio.h>
int main()
{
    int n, m, i, j, k;

    printf("Enter the no of Process and Resources:");
    scanf("%d %d",&n,&m);

    int *avail = (int*)malloc(m*sizeof(int));

    printf("Enter the available Resources:");

    for(i=0;i<m;i++){
        scanf("%d",&avail[i]);
    }


    int **alloc = (int**)malloc(n*sizeof(int*));
    printf("Enter the allocation matrix:");
     for(i=0;i<n;i++){
        alloc[i] = (int*)malloc(m*sizeof(int));

        for(int j=0;j<m;j++){
            scanf("%d",&alloc[i][j]);
        }
    }

    int **max = (int**)malloc(n*sizeof(int*));


     printf("Enter the Max matrix:");
     for(i=0;i<n;i++){
        max[i] = (int*)malloc(m*sizeof(int));

        for(int j=0;j<m;j++){
            scanf("%d",&max[i][j]);
        }
    }

    int f[n], ans[n], ind = 0;
```

```
for (k = 0; k < n; k++) {
    f[k] = 0;
}

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


int y = 0;
for (k = 0; k < n; k++) {
    for (i = 0; i < n; i++) {
        if (f[i] == 0) {

            int flag = 0;
            for (j = 0; j < m; j++) {
                if (need[i][j] > avail[j]){
                    flag = 1;
                    break;
                }
            }

            if (flag == 0) {
                ans[ind++] = i;
                for (y = 0; y < m; y++)
                    avail[y] += alloc[i][y];
                f[i] = 1;
            }
        }
    }
}

int flag = 1;

for(int i=0;i<n;i++)
{
if(f[i]==0)
{
    flag=0;
    printf("The following system is not safe");
    break;
}
```

```
    }

    if(flag==1)
    {
    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);
    }
    return (0);
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc Bankers_algorithm.c

D:\Codes\c\OS_Lab>.\a.exe
Enter the no of Process and Resources:5 3
Enter the available Resources:3 3 2
Enter the allocation matrix:0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the Max matrix:7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2
```

Q. Write a C program to simulate deadlock detection.

```c
#include <stdio.h>
static int mark[20];
int i, j, np, nr;

int main()
{
  int alloc[10][10], request[10][10], avail[10], r[10], w[10];

  printf("\nEnter the no of the process: ");
  scanf("%d", &np);
  printf("\nEnter the no of resources: ");
  scanf("%d", &nr);
  for (i = 0; i < nr; i++)
  {
    printf("\nTotal Amount of the Resource R % d: ", i + 1);
    scanf("%d", &r[i]);
  }
  printf("\nEnter the request matrix:");

  for (i = 0; i < np; i++)
    for (j = 0; j < nr; j++)
      scanf("%d", &request[i][j]);

  printf("\nEnter the allocation matrix:");
  for (i = 0; i < np; i++)
    for (j = 0; j < nr; j++)
      scanf("%d", &alloc[i][j]);
  /*Available Resource calculation*/
  for (j = 0; j < nr; j++)
  {
    avail[j] = r[j];
    for (i = 0; i < np; i++)
    {
      avail[j] -= alloc[i][j];
    }
  }

  // marking processes with zero allocation

  for (i = 0; i < np; i++)
  {
    int count = 0;
    for (j = 0; j < nr; j++)
```

```
  {
    if (alloc[i][j] == 0)
      count++;
    else
      break;
  }
  if (count == nr)
    mark[i] = 1;
}
// initialize W with avail

for (j = 0; j < nr; j++)
  w[j] = avail[j];

// mark processes with request less than or equal to W
for (i = 0; i < np; i++)
{
  int canbeprocessed = 0;
  if (mark[i] != 1)
  {
    for (j = 0; j < nr; j++)
    {
      if (request[i][j] <= w[j])
        canbeprocessed = 1;
      else
      {
        canbeprocessed = 0;
        break;
      }
    }
    if (canbeprocessed)
    {
      mark[i] = 1;

      for (j = 0; j < nr; j++)
        w[j] += alloc[i][j];
    }
  }
}

// checking for unmarked processes
int deadlock = 0;
for (i = 0; i < np; i++)
  if (mark[i] != 1)
    deadlock = 1;
```

```
  if (deadlock)
    printf("\n Deadlock detected");
  else
    printf("\n No Deadlock possible");
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc Deadlock_Detection.c

D:\Codes\c\OS_Lab>.\a.exe

Enter the no of the process: 4

Enter the no of resources: 2

Total Amount of the Resource R  1: 1

Total Amount of the Resource R  2: 2

Enter the request matrix:4 1
2 1
1 1
0 1

Enter the allocation matrix:2 1
2 1
2 1
2 1

 Deadlock detected
```

Q. Write a C program to simulate the following contiguous memory allocation techniques

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

Worst fit

```c
#include <stdio.h>
#include <string.h>
void worstFit(int blockSize[], int m, int processSize[], int n)
{
    int allocation[n];
    memset(allocation, -1, sizeof(allocation));

    for (int i = 0; i < n; i++)
    {
        int wstIdx = -1;
        for (int j = 0; j < m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                if (wstIdx == -1)
                    wstIdx = j;
                else if (blockSize[wstIdx] < blockSize[j])
                    wstIdx = j;
            }
        }

        if (wstIdx != -1)
        {
            allocation[i] = wstIdx;
            blockSize[wstIdx] -= processSize[i];
        }
    }

    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < n; i++)
    {
        printf("   %d\t\t%d\t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d", allocation[i] + 1);
        else
            printf("Not Allocated");
```

```
        printf("\n");
    }
}

int main()
{
    printf("Enter the number of blocks: ");
    int m;
    scanf("%d", &m);
    int blockSize[m];
    printf("Enter the block sizes: ");
    for (int i = 0; i < m; i++)
        scanf("%d", &blockSize[i]);

    printf("Enter the number of processes: ");
    int n;
    scanf("%d", &n);
    int processSize[n];
    printf("Enter the process sizes: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &processSize[i]);

    worstFit(blockSize, m, processSize, n);

    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>.\a.exe
Enter the number of blocks: 4
Enter the block sizes: 4
^C
D:\Codes\c\OS_Lab>.\a.exe
Enter the number of blocks: 4
Enter the block sizes: 4 3 2 3
Enter the number of processes: 2
Enter the process sizes: 4 3 2 1

Process No.     Process Size    Block no.
   1            4               1
   2            3               2
```

Best Fit

```c
#include <stdio.h>

void bestFit(int blockSize[], int m, int processSize[], int n)
{
    int allocation[n];

    for (int i = 0; i < n; i++)
        allocation[i] = -1;

    for (int i = 0; i < n; i++)
    {
        int bestIdx = -1;

        for (int j = 0; j < m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                if (bestIdx == -1)
                    bestIdx = j;
                else if (blockSize[bestIdx] > blockSize[j])
                    bestIdx = j;
            }
        }

        if (bestIdx != -1)
        {
            allocation[i] = bestIdx;
            blockSize[bestIdx] -= processSize[i];
        }
    }

    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < n; i++)
    {
        printf(" %d \t\t %d \t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
        printf("\n");
    }
}
```

```c
int main()
{
    printf("Enter the number of blocks: ");
    int m;
    scanf("%d", &m);
    int blockSize[m];
    printf("Enter the block sizes: ");
    for (int i = 0; i < m; i++)
        scanf("%d", &blockSize[i]);

    printf("Enter the number of processes: ");
    int n;
    scanf("%d", &n);
    int processSize[n];
    printf("Enter the process sizes: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &processSize[i]);

    bestFit(blockSize, m, processSize, n);

    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>.\a.exe
Enter the number of blocks: 4
Enter the block sizes: 5 2 4 7
Enter the number of processes: 7
Enter the process sizes: 3 1 2 1 4 6 5

Process No.     Process Size    Block no.
 1               3              3

 2               1              3

 3               2              2

 4               1              1

 5               4              1

 6               6              4

 7               5              Not Allocated
```

First Fit

```c
#include <stdio.h>

void firstFit(int blockSize[], int m, int processSize[], int n)
{
    int i, j;
    int allocation[n];

    for (i = 0; i < n; i++)
    {
        allocation[i] = -1;
    }

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                allocation[i] = j;
                blockSize[j] -= processSize[i];
                break;
            }
        }
    }

    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < n; i++)
    {
        printf(" %i\t\t\t", i + 1);
        printf("%i\t\t\t\t", processSize[i]);
        if (allocation[i] != -1)
            printf("%i", allocation[i] + 1);
        else
            printf("Not Allocated");
        printf("\n");
    }
}

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

```
    int blockSize[m];

    printf("Enter the block sizes: ");
    for (int i = 0; i < m; i++)
        scanf("%d", &blockSize[i]);

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

    firstFit(blockSize, m, processSize, n);

    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>.\a.exe
Enter the number of blocks: 4
Enter the block sizes: 3 2 4 1
Enter the number of processes: 4
2 3 1 4

Process No.     Process Size    Block no.
 1                      2                       1
 2                      3                       3
 3                      1                       1
 4                      4                       Not Allocated
```

Q. Write a program to simulate paging technique of memory management.

```c
#include <stdio.h>

int main(void)
{
    int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;

    printf("Enter the memory size : ");
    scanf("%d", &ms);

    printf("Enter the page size : ");
    scanf("%d", &ps);

    nop = ms / ps;
    printf("The no. of pages available in memory are : %d ", nop);

    printf("Enter number of processes : ");
    scanf("%d", &np);
    int s[np], fno[np][20];
    rempages = nop;
    for (i = 1; i <= np; i++)

    {
        printf("\nEnter no. of pages required for p[%d] : ", i);
        scanf("%d", &s[i]);

        if (s[i] > rempages)
        {
            printf("\nMemory is full!");
            break;
        }
        rempages = rempages - s[i];

        printf("\nEnter pagetable for p[%d] :  ", i);
        for (j = 0; j < s[i]; j++)
            scanf("%d", &fno[i][j]);
    }

    printf("\nEnter Logical Address to find Physical Address : ");
    printf("Enter process no. and pagenumber and offset : ");

    scanf("%d %d %d", &x, &y, &offset);
```

```
    if (x > np || y >= s[i] || offset >= ps)
        printf("\nInvalid Process or Page Number or offset!");

    else
    {
        pa = fno[x][y] * ps + offset;
        printf("\nThe Physical Address is : %d", pa);
    }
}
```

Output:

```
D:\Codes\c\OS_Lab>.\a.exe
Enter the memory size : 30
Enter the page size : 5
The no. of pages available in memory are : 6 Enter number of processes : 3

Enter no. of pages required for p[1] : 3

Enter pagetable for p[1] :  1 2 3

Enter no. of pages required for p[2] : 2

Enter pagetable for p[2] :  1 2

Enter no. of pages required for p[3] : 1

Enter pagetable for p[3] :  1

Enter Logical Address to find Physical Address
Enter process no. and pagenumber and offset : 1 2 1

The Physical Address is : 16
```

Q. Write a C program to simulate the following Page Replacement algorithms

a) FIFO

b) LRU

c) Optimal

FIFO

```c
#include <stdio.h>

#define FRAME_SIZE 3

int findPageInFrames(int frames[], int page, int frameCount)
{
    for (int i = 0; i < frameCount; i++)
    {
        if (frames[i] == page)
        {
            return 1;
        }
    }
    return 0;
}

int main()
{
    int referenceString[] = {2, 3, 4, 2, 1, 3, 7, 5, 4, 3};
    int referenceLength = sizeof(referenceString) / sizeof(referenceString[0]);
    int frames[FRAME_SIZE] = {-1};
    int frameIndex = 0;

    int pageFaults = 0;

    for (int i = 0; i < referenceLength; i++)
    {
        int currentPage = referenceString[i];

        if (!findPageInFrames(frames, currentPage, FRAME_SIZE))
        {

            frames[frameIndex] = currentPage;
            frameIndex = (frameIndex + 1) % FRAME_SIZE;
            pageFaults++;
```

```
        }

        printf("Frames: ");
        for (int j = 0; j < FRAME_SIZE; j++)
        {
            if (frames[j] != -1)
            {
                printf("%d ", frames[j]);
            }
            else
            {
                printf("- ");
            }
        }
        printf("\n");
    }

    printf("Total Page Faults: %d\n", pageFaults);

    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc Page-Replacement-FIFO.c

D:\Codes\c\OS_Lab>.\a.exe
Frames: 2 0 0
Frames: 2 3 0
Frames: 2 3 4
Frames: 2 3 4
Frames: 1 3 4
Frames: 1 3 4
Frames: 1 7 4
Frames: 1 7 5
Frames: 4 7 5
Frames: 4 3 5
Total Page Faults: 8
```

LRU

```c
#include <stdio.h>

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

int main(void)
{
    int no_of_frames, no_of_pages, counter = 0, flag1, flag2, i, j, pos, faults =
0;
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);
    int frames[no_of_frames];

    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);
    int pages[no_of_pages];

    int time[no_of_frames];
    printf("Enter reference string: ");
    for (i = 0; i < no_of_pages; ++i)
    {
        scanf("%d", &pages[i]);
    }

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

    for (i = 0; i < no_of_pages; ++i)
    {
        flag1 = flag2 = 0;
```

```
        for (j = 0; j < no_of_frames; ++j)
        {
            if (frames[j] == pages[i])
            {
                counter++;
                time[j] = counter;
                flag1 = flag2 = 1;
                break;
            }
        }

        if (flag1 == 0)
        {
            for (j = 0; j < no_of_frames; ++j)
            {
                if (frames[j] == -1)
                {
                    counter++;
                    faults++;
                    frames[j] = pages[i];
                    time[j] = counter;
                    flag2 = 1;
                    break;
                }
            }
        }
        if (flag2 == 0)
        {
            pos = findLRU(time, no_of_frames);
            counter++;
            faults++;
            frames[pos] = pages[i];
            time[pos] = counter;
        }
        printf("\n");

        for (j = 0; j < no_of_frames; ++j)
        {
            printf("%d\t", frames[j]);
        }
    }
    printf("\n\nTotal Page Faults = %d", faults);
}
```

Output:

```
D:\Codes\c\OS_Lab>.\a.exe
Enter number of frames: 3
Enter number of pages: 6
Enter reference string: 5 7 5 6 7 3

5       -1      -1
5       7       -1
5       7       -1
5       7       6
5       7       6
3       7       6

Total Page Faults = 4
```

Optimal

```c
#include <stdio.h>

int main(void)
{
    int no_of_frames, no_of_pages, temp[10], flag1, flag2, flag3, i, j, k, pos,
max, faults = 0;
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);
    int frames[no_of_frames];

    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);
    int pages[no_of_pages];

    printf("Enter page reference string: ");

    for (i = 0; i < no_of_pages; ++i)
    {
        scanf("%d", &pages[i]);
    }
```

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

    for (i = 0; i < no_of_pages; ++i)
    {
        flag1 = flag2 = 0;

        for (j = 0; j < no_of_frames; ++j)
        {
            if (frames[j] == pages[i])
            {
                flag1 = flag2 = 1;
                break;
            }
        }

        if (flag1 == 0)
        {
            for (j = 0; j < no_of_frames; ++j)
            {
                if (frames[j] == -1)
                {
                    faults++;
                    frames[j] = pages[i];
                    flag2 = 1;
                    break;
                }
            }
        }

        if (flag2 == 0)
        {
            flag3 = 0;

            for (j = 0; j < no_of_frames; ++j)
            {
                temp[j] = -1;

                for (k = i + 1; k < no_of_pages; ++k)
                {
                    if (frames[j] == pages[k])
                    {
                        temp[j] = k;
```

```
                    break;
                }
            }
        }

        for (j = 0; j < no_of_frames; ++j)
        {
            if (temp[j] == -1)
            {
                pos = j;
                flag3 = 1;
                break;
            }
        }

        if (flag3 == 0)
        {
            max = temp[0];
            pos = 0;

            for (j = 1; j < no_of_frames; ++j)
            {
                if (temp[j] > max)
                {
                    max = temp[j];
                    pos = j;
                }
            }
        }
        frames[pos] = pages[i];
        faults++;
    }
    printf("\n");

    for (j = 0; j < no_of_frames; ++j)
    {
        if (frames[j] == -1)
            printf("-\t");
        else
            printf("%d\t", frames[j]);
    }
}

printf("\n\nTotal Page Faults = %d", faults);
```

Output:

```
D:\Codes\c\OS_Lab>gcc Page-Replacement-Optimal.c

D:\Codes\c\OS_Lab>.\a.exe
Enter number of frames: 3
Enter number of pages: 6
Enter page reference string: 5 7 5 6 7 3

5       -       -
5       7       -
5       7       -
5       7       6
5       7       6
3       7       6

Total Page Faults = 4
```

Q. Write a C program to simulate the disk scheduling algorithms

a) FCFS

b) SCAN

c) C-SCAN

FCFS

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

int size = 8;

void FCFS(int arr[],int head) {
    int seek_count = 0;
    int cur_track, distance;

    for(int i=0;i<size;i++) {
        cur_track = arr[i];

        distance = fabs(head - cur_track);

        seek_count += distance;

        head = cur_track;
    }

    printf("Total number of seek operations: %d\n",seek_count);

    printf("Seek Sequence is\n");

    for (int i = 0; i < size; i++) {
        printf("%d\n",arr[i]);
    }
}

int main() {
    int size;
    printf("Enter the size of req array: ");
    scanf("%d", &size);

    int* arr = (int*)malloc(sizeof(int)*size);

    printf("Enter the elements: ");
```

```
    for(int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    int head = 50;

    FCFS(arr,head);

    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>.\a.exe
Enter the size of req array: 3
Enter the elements: 3 2 1
Total number of seek operations: 129543247
Seek Sequence is
3
2
1
527
0
0
64798577
55009
```

SCAN

```c
#include <stdio.h>
int absoluteValue(int);

void main()
{
    int queue[25], n, headposition, i, j, k, seek = 0, maxrange,
                                            difference, temp, queue1[20],
queue2[20], temp1 = 0, temp2 = 0;
    float averageSeekTime;

    printf("Enter the maximum range of Disk: ");
    scanf("%d", &maxrange);

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

    printf("Enter the initial head position: ");
    scanf("%d", &headposition);

    printf("Enter the disk positions to be read(queue): ");
    for (i = 1; i <= n; i++)
    {
        scanf("%d", &temp);

        if (temp > headposition)
        {
            queue1[temp1] = temp;
            temp1++;
        }
        else
        {
            queue2[temp2] = temp;
            temp2++;
        }
    }

    for (i = 0; i < temp1 - 1; i++)
    {
        for (j = i + 1; j < temp1; j++)
        {
            if (queue1[i] > queue1[j])
            {
                temp = queue1[i];
```

```
                queue1[i] = queue1[j];
                queue1[j] = temp;
            }
        }
    }

    for (i = 0; i < temp2 - 1; i++)
    {
        for (j = i + 1; j < temp2; j++)
        {
            if (queue2[i] < queue2[j])
            {
                temp = queue2[i];
                queue2[i] = queue2[j];
                queue2[j] = temp;
            }
        }
    }

    for (i = 1, j = 0; j < temp1; i++, j++)
    {
        queue[i] = queue1[j];
    }

    queue[i] = maxrange;

    for (i = temp1 + 2, j = 0; j < temp2; i++, j++)
    {
        queue[i] = queue2[j];
    }

    queue[i] = 0;

    queue[0] = headposition;

    for (j = 0; j <= n; j++)
    {
        difference = absoluteValue(queue[j + 1] - queue[j]);

        seek = seek + difference;

        printf("Disk head moves from position %d to %d with Seek %d \n",
               queue[j], queue[j + 1], difference);
    }
```

```
    averageSeekTime = seek / (float)n;

    printf("Total Seek Time= %d\n", seek);
    printf("Average Seek Time= %f\n", averageSeekTime);
}

int absoluteValue(int x)
{
    if (x > 0)
    {
        return x;
    }
    else
    {
        return x * -1;
    }
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc Scan_Disc_scheduling.c

D:\Codes\c\OS_Lab>.\a.exe
Enter the maximum range of Disk: 10000
Enter the number of queue requests: 3
Enter the initial head position: 1000
Enter the disk positions to be read(queue): 1002
1020
1090
Disk head moves from position 1000 to 1002 with Seek 2
Disk head moves from position 1002 to 1020 with Seek 18
Disk head moves from position 1020 to 1090 with Seek 70
Disk head moves from position 1090 to 10000 with Seek 8910
Total Seek Time= 9000
Average Seek Time= 3000.000000
```

C-SCAN

```c
#include <stdio.h>
int absoluteValue(int);

void main()
{
    int queue[25], n, headposition, i, j, k, seek = 0, maxrange,
                                            difference, temp, queue1[20],
queue2[20], temp1 = 0, temp2 = 0;
    float averageSeekTime;

    printf("Enter the maximum range of Disk: ");
    scanf("%d", &maxrange);

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

    printf("Enter the initial head position: ");
    scanf("%d", &headposition);

    printf("Enter the disk positions to be read(queue): ");
    for (i = 1; i <= n; i++)
    {
        scanf("%d", &temp);

        if (temp > headposition)
        {
            queue1[temp1] = temp;
            temp1++;
        }
        else
        {
            queue2[temp2] = temp;
            temp2++;
        }
    }

    for (i = 0; i < temp1 - 1; i++)
    {
        for (j = i + 1; j < temp1; j++)
        {
            if (queue1[i] > queue1[j])
            {
                temp = queue1[i];
```

```
                queue1[i] = queue1[j];
                queue1[j] = temp;
            }
        }
    }

    for (i = 0; i < temp2 - 1; i++)
    {
        for (j = i + 1; j < temp2; j++)
        {
            if (queue2[i] < queue2[j])
            {
                temp = queue2[i];
                queue2[i] = queue2[j];
                queue2[j] = temp;
            }
        }
    }

    for (i = 1, j = 0; j < temp1; i++, j++)
    {
        queue[i] = queue1[j];
    }

    queue[i] = maxrange;

    for (i = temp1 + 2, j = 0; j < temp2; i++, j++)
    {
        queue[i] = queue2[j];
    }

    queue[i] = 0;

    queue[0] = headposition;

    for (j = 0; j <= n; j++)
    {
        difference = absoluteValue(queue[j + 1] - queue[j]);

        seek = seek + difference;

        printf("Disk head moves from position %d to %d with Seek %d \n",
               queue[j], queue[j + 1], difference);
    }
```

```
    averageSeekTime = seek / (float)n;

    printf("Total Seek Time= %d\n", seek);
    printf("Average Seek Time= %f\n", averageSeekTime);
}

int absoluteValue(int x)
{
    if (x > 0)
    {
        return x;
    }
    else
    {
        return x * -1;
    }
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc C-scan_Disc_Scheduling.c

D:\Codes\c\OS_Lab>.\a.exe
Enter the maximum range of Disk: 10000
Enter the number of queue requests: 3
Enter the initial head position: 8000
Enter the disk positions to be read(queue): 9000
9500
8500
Disk head moves from position 8000 to 8500 with Seek 500
Disk head moves from position 8500 to 9000 with Seek 500
Disk head moves from position 9000 to 9500 with Seek 500
Disk head moves from position 9500 to 10000 with Seek 500
Total Seek Time= 2000
Average Seek Time= 666.666687
```

Q. Write a C program to simulate the disk scheduling algorithms

a) SSTF

b) LOOK

c) C-LOOK

SSTF

```c
#include <stdio.h>
int absoluteValue(int);

void main()
{
    int queue[25], n, headposition, i, j, k, seek = 0, maxrange,
                                                difference, temp, queue1[20],
queue2[20], temp1 = 0, temp2 = 0;
    float averageSeekTime;

    printf("Enter the maximum range of Disk: ");
    scanf("%d", &maxrange);

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

    printf("Enter the initial head position: ");
    scanf("%d", &headposition);

    printf("Enter the disk positions to be read(queue): ");
    for (i = 1; i <= n; i++)
    {
        scanf("%d", &temp);

        if (temp > headposition)
        {
            queue1[temp1] = temp;
            temp1++;
        }
        else
        {
            queue2[temp2] = temp;
            temp2++;
        }
    }

    for (i = 0; i < temp1 - 1; i++)
```

```
    {
        for (j = i + 1; j < temp1; j++)
        {
            if (queue1[i] > queue1[j])
            {
                temp = queue1[i];
                queue1[i] = queue1[j];
                queue1[j] = temp;
            }
        }
    }

    for (i = 0; i < temp2 - 1; i++)
    {
        for (j = i + 1; j < temp2; j++)
        {
            if (queue2[i] < queue2[j])
            {
                temp = queue2[i];
                queue2[i] = queue2[j];
                queue2[j] = temp;
            }
        }
    }

    for (i = 1, j = 0; j < temp1; i++, j++)
    {
        queue[i] = queue1[j];
    }

    queue[i] = maxrange;

    for (i = temp1 + 2, j = 0; j < temp2; i++, j++)
    {
        queue[i] = queue2[j];
    }

    queue[i] = 0;

    queue[0] = headposition;

    for (j = 0; j <= n; j++)
    {
        difference = absoluteValue(queue[j + 1] - queue[j]);
```

```
        seek = seek + difference;

        printf("Disk head moves from position %d to %d with Seek %d \n",
               queue[j], queue[j + 1], difference);
    }

    averageSeekTime = seek / (float)n;

    printf("Total Seek Time= %d\n", seek);
    printf("Average Seek Time= %f\n", averageSeekTime);
}

int absoluteValue(int x)
{
    if (x > 0)
    {
        return x;
    }
    else
    {
        return x * -1;
    }
}
```

Output:

```
D:\Codes\c\OS_Lab>.\a.exe
Enter the maximum range of Disk: 10000
Enter the number of queue requests: 4
Enter the initial head position: 9001
Enter the disk positions to be read(queue): 7500
9003
9500
9801
Disk head moves from position 9001 to 9003 with Seek 2
Disk head moves from position 9003 to 9500 with Seek 497
Disk head moves from position 9500 to 9801 with Seek 301
Disk head moves from position 9801 to 10000 with Seek 199
Disk head moves from position 10000 to 7500 with Seek 2500
Total Seek Time= 3499
Average Seek Time= 874.750000
```

LOOK

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);
    printf("Enter total disk size\n");
    scanf("%d",&size);
    printf("Enter the head movement direction for high 1 and for low 0\n");
    scanf("%d",&move);

    // logic for look disk scheduling

        /*logic for sort the request array */
    for(i=0;i<n;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(RQ[j]>RQ[j+1])
            {
                int temp;
                temp=RQ[j];
                RQ[j]=RQ[j+1];
                RQ[j+1]=temp;
            }

        }
    }

    int index;
    for(i=0;i<n;i++)
    {
        if(initial<RQ[i])
        {
            index=i;
            break;
        }
```

```
    }

    // if movement is towards high value
    if(move==1)
    {
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }

        for(i=index-1;i>=0;i--)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }

        for(i=index;i<n;i++)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>.\a.exe
Enter the number of Requests
5
Enter the Requests sequence
1 51 201
401
831
Enter initial head position
97
Enter total disk size
1000
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 926
```

C-LOOK

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);
    printf("Enter total disk size\n");
    scanf("%d",&size);
    printf("Enter the head movement direction for high 1 and for low 0\n");
    scanf("%d",&move);

    // logic for C-look disk scheduling

        /*logic for sort the request array */
    for(i=0;i<n;i++)
    {
        for( j=0;j<n-i-1;j++)
```

```
        {
            if(RQ[j]>RQ[j+1])
            {
                int temp;
                temp=RQ[j];
                RQ[j]=RQ[j+1];
                RQ[j+1]=temp;
            }

        }
    }

    int index;
    for(i=0;i<n;i++)
    {
        if(initial<RQ[i])
        {
            index=i;
            break;
        }
    }

    // if movement is towards high value
    if(move==1)
    {
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }

        for( i=0;i<index;i++)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
```

```
        }

        for(i=n-1;i>=index;i--)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

Output:

```
D:\Codes\c\OS_Lab>gcc C_Look_Disc_Scheduling.c

D:\Codes\c\OS_Lab>.\a.exe
Enter the number of Requests
5
Enter the Requests sequence
97 201 501 802 1
Enter initial head position
500
Enter total disk size
1000
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 1303
```