VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

OPERATING SYSTEMS

Submitted by

VARSHA VIRUPAKSHA GUJJALA (1WA23CS034)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

Feb-2025 to June-2025

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "OPERATING SYSTEMS – 23CS4PCOPS" carried out by Varsha Virupaksha Gujjala (1WA23CS034), who is Bonafide student of B. M. S. College of Engineering. It is in partial fulfilment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year Feb 2025-June 2025. The Lab report has been approved as it satisfies the academic requirements in respect of a OPERATING SYSTEMS - (23CS4PCOPS) work prescribed for the said degree.

Dr. Seema Patil
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

9.	Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit b) Best-fit c) First-fit	59-65
10.	Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) Optimal	66-78

Course Outcomes

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

INDEX

Name : Varsha V. Gujjalar Class : 6

Section : G Roll No. : 1NA22CS034 Subject : Operating Systems

Sl. No.	Date	Title	Page No.	Teacher's Sign. / Remarks
1.	6/3	FCFS, SJF (preemptive and non-preemptive)	10	
2.	13/3	Priority Scheduling (preemptive and non-preemptive)	10	20/3/25
3	3/4	Multilevel queue (RR & FCFS),		
4		Rate Monotonic, Earliest deadline first	10	
4	16/4	Producer - Consumer, Dining Philosopher	10	
5	17/4	Banker's algorithm, deadlock	10	17-4-25
6	14/5	Memory allocation (Best fit, Worst fit, first fit)	10	
7.	15/5	FIFO, LRU, optimal. (page replacement)	10	

LAB-01

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

(a) FCFS

(b) SJF

1.1.1 Code:

```
#include<stdio.h>
   int n, i, j, pos, temp, choice, Burst_time[20], Waiting_time[20], Turn_around_time[20],
   process[20], total=0;
   float avg_Turn_around_time=0, avg_Waiting_time=0;

int FCFS()
{
   Waiting_time[0]=0;

   for(i=1;i<n;i++)
   {
      Waiting_time[i]=0;
for(j=0;j<i;j++)
         Waiting_time[i]+=Burst_time[j];
   }

   printf("\nProcess\t\tBurst Time\t\tWaiting Time\t\tTurnaround Time");

   for(i=0;i<n;i++)
   {
      Turn_around_time[i]=Burst_time[i]+Waiting_time[i];
avg_Waiting_time+=Waiting_time[i];
avg_Turn_around_time+=Turn_around_time[i];
      printf("\nP[%d]\t\t%d\t\t\t%d\t\t\t\t%d",i+1,Burst_time[i],Waiting_time[i],Turn_around_time[i]);
   }

   avg_Waiting_time =(float)(avg_Waiting_time)/(float)i;
avg_Turn_around_time=(float)(avg_Turn_around_time)/(float)i;
printf("\nAverage Waiting Time:%.2f",avg_Waiting_time);
   printf("\nAverage Turnaround Time:%.2f\n",avg_Turn_around_time);

   return 0;
}

int SJF()
{
   //sorting
for(i=0;i<n;i++)
   {       pos=i;
for(j=i+1;j<n;j++)
{
```

```
            if(Burst_time[j]<Burst_time[pos])
pos=j;
        }

        temp=Burst_time[i];
        Burst_time[i]=Burst_time[pos];
        Burst_time[pos]=temp;

        temp=process[i];
process[i]=process[pos];
        process[pos]=temp;
    }
        Waiting_time[0]=0;


    for(i=1;i<n;i++)
    {
        Waiting_time[i]=0;

        for(j=0;j<i;j++)
            Waiting_time[i]+=Burst_time[j];

        total+=Waiting_time[i];
    }

    avg_Waiting_time=(float)total/n;
total=0;

    printf("\nProcess\t\tBurst Time\t\tWaiting Time\t\tTurnaround Time");

    for(i=0;i<n;i++)
    {
        Turn_around_time[i]=Burst_time[i]+Waiting_time[i];
total+=Turn_around_time[i];

printf("\nP[%d]\t\t%d\t\t\t%d\t\t\t\t%d",process[i],Burst_time[i],Waiting_time[i],Turn_around_time[i]);
    }

    avg_Turn_around_time=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_Waiting_time);
    printf("\nAverage Turnaround Time=%f\n",avg_Turn_around_time);
}

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

    printf("\nEnter Burst Time:\n");
for(i=0;i<n;i++)
    {
        printf("P[%d]:",i+1);
scanf("%d",&Burst_time[i]);
```

```
        process[i]=i+1;
    }

    while(1)
    {   printf("\n-----MAIN MENU-----\n");         printf("1.
FCFS Scheduling\n2. SJF Scheduling\n");
        printf("\nEnter your choice:");
scanf("%d", &choice);
        switch(choice)
        {
            case 1: FCFS();
break;

            case 2: SJF();
            break;

            default: printf("Invalid Input!!!");
        }
    }
    return 0;
}
```

Output:

```
Enter the total number of processes:4

Enter Burst Time:
P[1]:1
P[2]:2
P[3]:3
P[4]:2

-----MAIN MENU-----
1. FCFS Scheduling
2. SJF Scheduling

Enter your choice:1

Process         Burst Time              Waiting Time            Turnaround Time
P[1]            1                       0                               1
P[2]            2                       1                               3
P[3]            3                       3                               6
P[4]            2                       6                               8
Average Waiting Time:2.50
Average Turnaround Time:4.50

-----MAIN MENU-----
1. FCFS Scheduling
2. SJF Scheduling

Enter your choice:2

Process         Burst Time              Waiting Time            Turnaround Time
P[1]            1                       0                               1
P[2]            2                       1                               3
P[4]            2                       3                               5
P[3]            3                       5                               8

Average Waiting Time=2.250000
Average Turnaround Time=4.250000
```

6/3/25

LAB-1

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

→ FCFS → SJF → (preemptive & non-preemptive)

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

struct p{
int pid; int at; int bt; int ct; int tat; int wt;
int rt; // remaining time
};

void fcfs(struct p* pro, int n){
int curTime = 0;
for(int i=0; i<n; i++){
if(curTime < pro[i].at)
    curTime = pro[i].at;
pro[i].ct = curTime + pro[i].bt;
pro[i].tat = pro[i].ct - pro[i].at;
pro[i].wt = pro[i].tat - pro[i].bt;
curTime = pro[i].ct;
}
}

void SJFnp(struct p* pro, int n){
int curTime = 0;
int completed[n];
for(int i=0; i<n; i++)
    completed[i] = 0;
for(int i=0; i<n; i++){
int shortest = -1;
int min_bt = 1000000;
for(int j=0; j<n; j++){
if(!completed[j] && pro[j].at <= curTime && pro[j].bt <
min_bt){
min_bt = pro[j].bt;
shortest = j;}
}

if(shortest == -1){
curTime++; i--;
continue;}
```

```
pro[shortest].ct = curTime + pro[shortest].bt;
pro[shortest].tat = pro[shortest].ct - pro[shortest].at;
pro[shortest].wt = pro[shortest].tat - pro[shortest].bt;
completed[shortest] = 1;
curTime = pro[shortest].ct;
}}

void SJFp(struct p* pro, int n){
int curTime = 0;
int completed = 0;
for(int i = 0; i<n; i++)
    pro[i].rt = pro[i].bt;
while(completed != n){
int shortest = -1;
int min_rt = 1000000;
for(int j = 0; j<n; j++){
if(pro[j].at <= curTime && pro[j].rt < min_rt && pro[j].rt>0){
min_rt = pro[j].rt;
shortest = j;
}}
if(shortest == -1){
curTime++;
continue;
}
pro[shortest].rt--;
curTime++;
if(pro[shortest].rt == 0){
completed++;
pro[shortest].ct = curTime;
pro[shortest].tat = pro[shortest].ct - pro[shortest].at;
pro[shortest].wt = pro[shortest].tat - pro[shortest].bt;
}
}}

void display(struct p* pro, int n){
printf("PID\tAT\tBT\tCT\tTAT\tWT\n");
for(int i = 0; i<n; i++){
printf("%d\t%d\t%d\t%d\t%d\t%d\n", pro[i].pid,
pro[i].at, pro[i].bt, pro[i].ct, pro[i].tat, pro[i].wt);
}}
```

```
int main(){
int n;
printf("Enter number of processes: ");
scanf("%d", &n);
struct p* pro = (struct p*) malloc(n * sizeof(struct p));
if(pro == NULL)
for(int i=0; i<n; i++){
pro[i].pid = i+1;
printf("Enter Arrival Time and Burst time for Process %d
(AT BT): ", pro[i].pid);
scanf("%d %d", &pro[i].at, &pro[i].bt);
}

for(int i=0; i<n-1; i++){
for(int j=0; j<n-1; j++){
if(pro[j].at > pro[j+1].at){
struct p temp = pro[j];
pro[j] = pro[j+1];
pro[j+1] = temp;
}
}}

fcfs(pro, n);
printf("\nFCFS Scheduling:\n");
display(pro, n);

SJFnp(pro, n);
printf("\n SJF Non-Preemptive Scheduling:\n");
display(pro, n);

SJFp(pro, n);
printf("\n SJF Preemptive Scheduling:\n");
display(pro, n);

free(pro);
return 0;
}
```

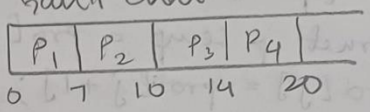
Output:

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

FCFS Scheduling:

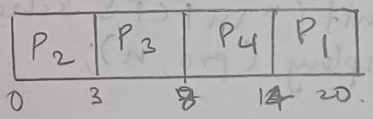
PID	AT	BT	CT	TAT	WT
1	0	7	7	7	0
2	0	3	10	10	7
3	0	4	14	14	10
4	0	6	20	20	14

Gantt Chart



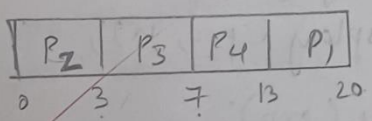
SJF Non-Preemptive Scheduling:

PID	AT	BT	CT	TAT	WT
1	0	7	20	20	13
2	0	3	3	3	0
3	0	4	7	7	3
4	0	6	14	14	8



SJF Preemptive Scheduling:

PID	AT	BT	CT	TAT	WT
1	0	7	20	20	13
2	0	3	3	3	0
3	0	4	7	7	3
4	0	6	13	13	7



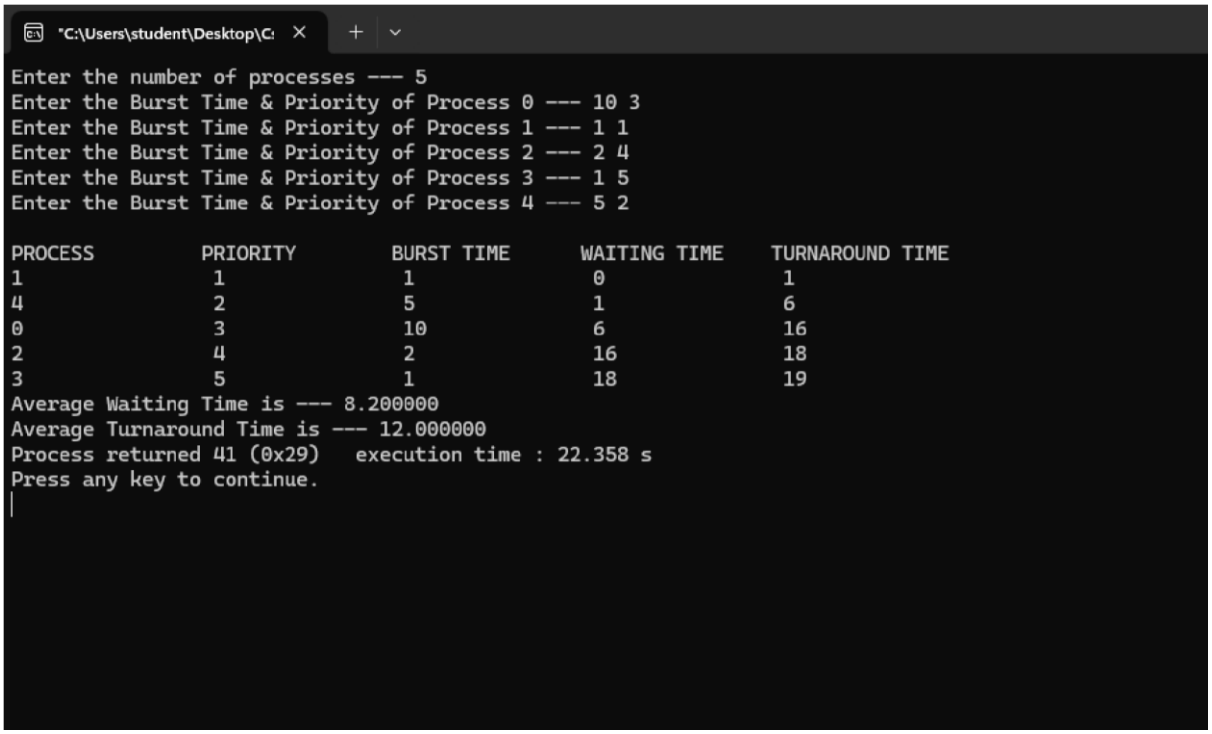
LAB-02

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

```
#include<stdio.h>

void main()

{

int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;

float awt=0,att=0,temp=0;

printf("Enter the no of processes -- ");

scanf("%d",&n); for(i=0;i<n;i++)

{

printf("\nEnter Burst Time for process %d -- ", i+1);

scanf("%d",&bu[i]); ct[i]=bu[i];

}

printf("\nEnter the size of time slice -- ");

scanf("%d",&t); max=bu[0];

for(i=1;i<n;i++)  if(max<bu[i])   max=bu[i];

 for(j=0;j<(max/t)+1;j++){

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

if(bu[i]!=0){ if(bu[i]<=t) {

tat[i]=temp+bu[i];

temp=temp+bu[i];

bu[i]=0;

}

else {

 bu[i]=bu[i]-t;
```

```
 temp=temp+t;
}}}}
for(i=0;i<n;i++) {
wa[i]=tat[i]-ct[i];
att+=tat[i];
awt+=wa[i];}

printf("\nThe Average Turnaround time is -- %f",att/n);  printf("\nThe
Average Waiting time is -- %f ",awt/n);

printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
for(i=0;i<n;i++) printf("\t%d \t %d \t\t %d \t\t %d
\n",i+1,ct[i],wa[i],tat[i]);
```

OUTPUT:

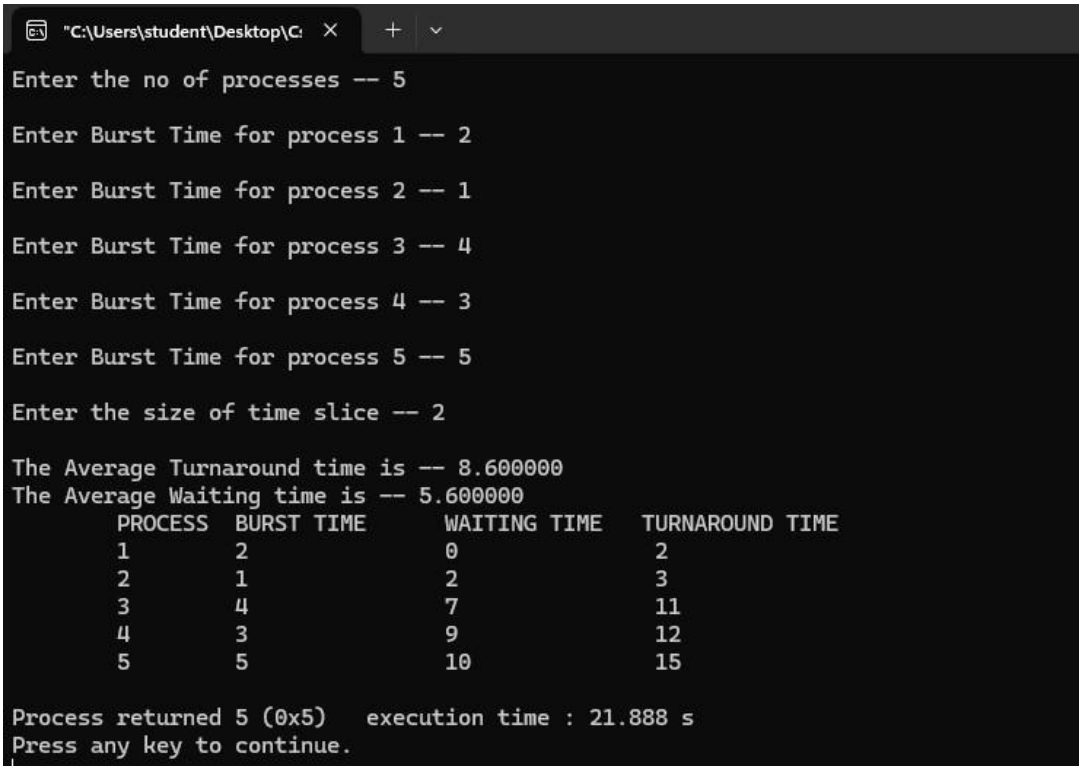


d)PRIORITY:

```
#include<stdio.h> void
main()

{
int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp; float wtavg, tatavg;
printf("Enter the number of processes --- "); scanf("%d",&n); for(i=0;i<n;i++){

p[i] = i;
printf("Enter the Burst Time & Priority of Process %d --- ",i); scanf("%d%d",&bt[i],
&pri[i]);

}
for(i=0;i<n;i++){
for(k=i+1;k<n;k++) {

 if(pri[i] > pri[k])
 { temp=p[i]; p[i]=p[k];
p[k]=temp; temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;
temp=pri[i];
pri[i]=pri[k];
pri[k]=temp;

}}}
wtavg = wt[0] = 0;
tatavg = 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];

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}
printf("\nPROCESS\t\tPRIORITY\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

",p[i],pri[i],bt[i],wt[i],tat[i]); printf("\nAverage Waiting Time is ---

%f",wtavg/n); printf("\nAverage Turnaround Time is --- %f",tatavg/n);

}
```

OUTPUT:



LAB-2 & 3

20/3/24 | 13/3/24

1) Write a C program to stimulate the following CPU scheduling algorithm to find turnaround time and waiting time

→ Priority scheduling (Preemptive and Non preemptive)

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

typedef struct {
 int pid, at, bt, pt, tat, rem_bt, ct, wt, rt, st
 is_completed ;
} Process;
void NPpriority (Process p[], int n){
  int time =0, completed = 0;
  while (completed <n){
   int hp = 9999, selected = -1;
   for(int i=0; i<n; i++){
     if (p[i].at <= time && !p[i].is_completed &&
     p[i].pt <hp) {
       hp = p[i].pt;
        selected = i; }
     }
    if (selected ==-1){
     time++; continue; }
    if (p[selected].rt==-1){
     p[selected].st = time;
     p[selected].rt = time- p[selected]at;}
    time += p[selected].bt;
    p[selected].ct = time;
    p[selected].tat = p[selected].ct- p[selected].at;
    p[selected].wt = p[selected].tat - p[selected].bt;
    p[selected]. is_completed = 1;
    completed ++; }
  }

void Ppriority (Process p[], int n) {
int time = 0; completed =0;
 while (completed < n) {
   int hp = 9999, selected =-1;
```

```
for(int i = 0; i<n; i++){
  if(p[i].at <= time && p[i].rem_bt >0 && p[i].pt<hp)
  { hp = p[i].pt;
    selected = i;}
}
if(selected == -1) {
   time++; continue; }
if(p[selected].rt == -1){
   p[selected].st = time;
   p[selected].rt = time - p[selected].at;}
p[selected].rem_bt --;
time ++;
if(p[selected].rem_bt ==0){
   p[selected].ct =time;
   p[selected].tat = p[selected].ct - p[selected].at;
   p[selected].wt = p[selected].tat - p[selected].bt;
   completed ++; }
}

void displayP(Process p[], int n){
 float avg_tat = 0, avg_wt =0, avg_rt =0;
 printf("\nPID\tAT\tBT\tPriority\tCT\tTAT\tWT\tRT\n");
 for(int i =0; i<n; i++){
   printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
   p[i].pid, p[i].at, p[i].bt, p[i].pt, p[i].ct, p[i].tat,
   p[i].wt, p[i].rt);
    avg_tat += p[i].tat;
    avg_wt += p[i].wt;
    avg_rt += p[i].rt;}
    printf("\nAverage TAT : %2f", avg_tat/n);
    printf("\nAverage WT : %2f", avg_wt/n);
    printf("\nAverage RT : %2f\n", avg_rt/n);
    }

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

```
printf("Enter the number of processes: ");
scanf("%d", &n);
print("\n");
for(int i=0; i<n; i++){
  p[i].pid = i+1;
  printf("Enter Arrival time, Burst time and Priority for
  Process %d: ", p[i].pid);
  scanf("%d %d %d", &p[i].at, &p[i].bt, &p[i].pb);
  p[i].rem_bt = p[i].bt;
  p[i].is_completed = 0;
  p[i].rt = 1;
}
while(1){
printf("\n1. Non-Preemptive Priority Scheduling \n");
printf("2. Preemptive Priority Scheduling \n");
printf("3. Exit\n");
printf("Enter your choice: ");
scanf("%d", &ch);
switch(ch){
case 1:
  NPPriority(p, n);
  printf("\n Non-Preemptive Priority Scheduling: ");
  displayP(p, n);
  break;
case 2:
  Ppriority(p, n);
  printf("\n Preemptive Priority Scheduling: ");
  displayP(p, n);
  break;
case 3:
  printf("Exiting....\n");
  return 0;
default:
  printf("Invalid choice! Try again.\n");
}
}
return 0;
}
```

Output:

Enter the number of processes: 5

Enter Arrival time, Burst time and Priority for Process 1:
0 4 2
Enter Arrival time, Burst time and Priority for Process 2:
1 3 3
Enter Arrival time, Burst time and Priority for Process 3:
2 1 4
Enter Arrival time, Burst time and Priority for Process 4:
3 5 5
Enter Arrival time, Burst time and Priority for Process 5:
4 2 5

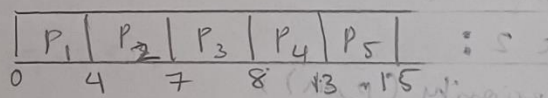
1. Non-Preemptive Priority ~~Program~~ Scheduling
2. Preemptive Priority Scheduling
3. Exit

Enter your choice: 1

Non-Preemptive Scheduling:

PID	AT	BT	Priority	CT	TAT	WT	RT
1	0	4	2	4	4	0	0
2	1	3	3	7	6	3	3
3	2	1	4	8	6	5	5
4	3	5	5	13	10	5	5
5	4	2	5	15	11	9	9

Average TAT: 7.40
Average WT: 4.40
Average RT: 4.40



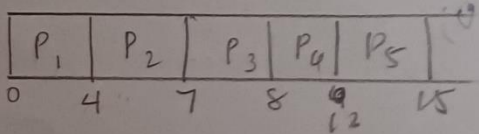
1. Non-Preemptive Priority Scheduling
2. Preemptive Priority Scheduling
3. Exit

Enter your choice: 2

Preemptive Scheduling:

PID	AT	BT	Priority	CT	TAT	WT	RT
1	0	4	2	4	4	0	0
2	1	3	3	7	6	3	3
3	2	1	4	8	6	5	5
4	3	5	5	13	10	5	5
5	4	2	5	15	11	9	9

Average TAT: 7.40
Average WT: 4.40
Average RT: 4.40



LAB- 03

1.MULTILEVEL QUEUE SCHEDULING

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

typedef struct {
   int id;
   int at, bt, rt, ct, wt, tat, queue;
} process;

int total_wt = 0, total_tat = 0;

void schedule(process system[], int s_count, process user[], int u_count, int q) {
   int time = 0, s_ptr = 0, u_ptr = 0, completed = 0;
   int *sys_remaining = malloc(s_count * sizeof(int));
   int *user_remaining = malloc(u_count * sizeof(int));



   for(int i=0; i<s_count; i++) sys_remaining[i] = system[i].bt;
   for(int i=0; i<u_count; i++) user_remaining[i] = user[i].bt;

   while(completed < s_count + u_count) {

      int sys_active = 0;
      for(int i=0; i<s_count; i++) {
         if(system[i].at <= time && sys_remaining[i] > 0) {
            sys_active = 1;
            int exec = (sys_remaining[i] > q) ? q : sys_remaining[i];
            sys_remaining[i] -= exec;
            time += exec;

            if(sys_remaining[i] == 0) {
               system[i].ct = time;
               system[i].tat = time - system[i].at;
               system[i].wt = system[i].tat - system[i].bt;
               total_wt += system[i].wt;
               total_tat += system[i].tat;
               completed++;
```

```
            }
            break;
        }
    }


    if(!sys_active) {
        if(u_ptr < u_count && user[u_ptr].at <= time) {
            int exec_time = user_remaining[u_ptr];
            time += exec_time;
            user[u_ptr].ct = time;
            user[u_ptr].tat = time - user[u_ptr].at;
            user[u_ptr].wt = user[u_ptr].tat - user[u_ptr].bt;
            total_wt += user[u_ptr].wt;
            total_tat += user[u_ptr].tat;
            completed++;
            u_ptr++;
        } else {
            time++;
        }
    }
  }
  free(sys_remaining);
  free(user_remaining);
}

int main() {
  int n, q, s_count = 0, u_count = 0;
  printf("Enter total processes: ");
  scanf("%d", &n);

  process processes[n], system[n], user[n];

  for(int i=0; i<n; i++) {
    printf("Process %d (arrival burst queue): ", i+1);
    scanf("%d %d %d", &processes[i].at, &processes[i].bt, &processes[i].queue);
    processes[i].id = i+1;
    processes[i].rt = processes[i].bt;
```

```
        if(processes[i].queue == 0) system[s_count++] = processes[i];
        else user[u_count++] = processes[i];
    }

    printf("Enter RR quantum: ");
    scanf("%d", &q);


    for(int i=0; i<s_count; i++)
        for(int j=i+1; j<s_count; j++)
            if(system[i].at > system[j].at) {
                process temp = system[i];
                system[i] = system[j];
                system[j] = temp;
            }

    for(int i=0; i<u_count; i++)
        for(int j=i+1; j<u_count; j++)
            if(user[i].at > user[j].at) {
                process temp = user[i];
                user[i] = user[j];
                user[j] = temp;
            }

    schedule(system, s_count, user, u_count, q);
    printf("\nPID\tArrival\tBurst\tCT\tWT\tTAT\tQueue\n");
    for(int i=0; i<s_count; i++)
        printf("%d\t%d\t%d\t%d\t%d\t%d\tRR\n", system[i].id, system[i].at,
               system[i].bt, system[i].ct, system[i].wt, system[i].tat);

    for(int i=0; i<u_count; i++)
        printf("%d\t%d\t%d\t%d\t%d\t%d\tFCFS\n", user[i].id, user[i].at,
               user[i].bt, user[i].ct, user[i].wt, user[i].tat);

    printf("\nAvg WT: %.2f\tAvg TAT: %.2f\n",
         (float)total_wt/n, (float)total_tat/n);
    return 0;
}
```

OUTPUT:

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

Queue 1 is System Process
Queue 2 is User Process

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

Average Waiting Time: 4.25
Average Turn Around Time: 7.00
Average Response Time: 4.25
Throughput: 0.36
Process returned 11 (0x11) execution time: 11.000 s

Process returned 0 (0x0)   execution time : 19.219 s
Press any key to continue.
```

LAB-3

3|4|25
Thursday

1) Write C program to Stimulate multilevele scheduling programming algorithm considering the following scenario. All the processes are divided int 2 categories - system processes & user processes. System processes are to be given higher priority than user processes. Use RR and FCFS scheduling for the processes in each queue.

```
#include <stdio.h>
#define MAX_P 10
#define Q 2

typedef struct {
int bt, at, queue_type, wt, tat, rt, rem_time;
} Process;

void round_robin ( Process proc[], int n, int time_quantum,
int * time){
 int done, i;
do {
done = 1;
for (i=0; i<n; i++){
   if (proc[i].rem_time >0){
      done =0;
      if (proc[i].rem_time > time_quantum){
         *time += time_quantum;
         proc[i].rem_time -= time_quantum; }
      else {
         *time += proc[i].rem_time;
         proc[i].wt = *time - proc[i].at - proc[i].bt;
         proc[i].tat = *time - proc[i].at;
         proc[i].rt = proc[i].wt;
         proc[i].rem_time = 0;
      }
 } }
} while (!done);

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

```
        time += proc[i].bt;
    }
}

int main(){
Process proc[MAX_P], sys_q[MAX_P], user_q[MAX_P];
int n=0, sys_count=0; user_count=0; time=0;
float avg_wt=0, avg_tat=0, avg_rt=0;
printf("Enter number of processes: ");
scanf("%d", &n);
for(int i=0; i<n; i++){
    printf("Enter BT, AT and Queue of P%d: ", i+1);
    scanf("%d %d %d", &proc[i].bt, &proc[i].at, &proc[i].
    queue_type);
    proc[i].rem_time = proc[i].bt;
    if(proc[i].queue_type==1)
        sys_q[sys_count++] = proc[i];
    else
        user_q[user_count++] = proc[i];
}

for(int i=0; i<user_count-1; i++){
    for(int j=0; j<user_count-i-1; j++){
        if(user_q[j].at > user_q[j+1].at){
            Process temp = user_q[j];
            user_q[j] = user_q[j+1];
            user_q[j+1] = temp;
        }
    }
}

printf("\n Queue 1 is system Process \n Queue 2 is user
Process\n");
round_robin(sys_q, sys_count, Q, &time);
fcfs(user_q, user_count, &time);
printf("\n Process  WT  TAT  RT\n");
for(int i=0; i<sys_count; i++){
    avg_wt += sys_q[i].wt;
    avg_tat += sys_q[i].tat;
    avg_rt += sys_q[i].rt;
    printf("%d  %d  %d  %d\n", i+1, sys_q[i].
    sys_q[i].tat, sys_q[i].rt);
```

```
for(int i=0; i<user_count; i++){
    avg_wt += user_q[i].wt;
    avg_tat += user_q[i].tat;
    avg_rt += user_q[i].rt;
    printf("%d  %d  %d  %d\n", i+1+ys_count,
    user_q[i].wt, user_q[i].tat, user_q[i].rt);
}

avg_wt /= n;
avg_tat /= n;
avg_rt /= n;

printf("\n Average WT : %.2f", avg_wt);
printf("\n Average TAT : %.2f", avg_tat);
printf("\n Average RT : %.2f", avg_rt);
printf("\n Process
return 0;
}
```

Output:

```
Enter number of processes : 4
Enter BT, AT and Queue of P1 : 2 0 1
Enter BT, AT and Queue of P2 : 1 0 2
Enter BT, AT and Queue of P3 : 5 0 1
Enter BT, AT and Queue of P4 : 3 0 2
```

Process	WT	TAT	RT
1	0	2	0
2	2	7	2
3	7	8	7
4	8	11	8

```
Average WT : 4.25
Average TAT : 7.00
Average RT : 4.25
```

LAB- 04

1.RATE MONOTONIC and EARLIEST DEADLINE FIRST

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

#define MAX_PROCESS 10

typedef struct {
   int id;     int
burst_time;
float priority;
} Task;

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

void get_process_info(int selected_algo)
{
   printf("Enter total number of processes (maximum %d): ", MAX_PROCESS);
scanf("%d", &num_of_process);     if (num_of_process < 1)
   {
      exit(0);
   }

   for (int i = 0; i < num_of_process; i++)
   {
      printf("\nProcess %d:\n", i + 1);
printf("==> Execution time: ");
scanf("%d", &execution_time[i]);
remain_time[i] = execution_time[i];
if (selected_algo == 2)
      {
         printf("==> Deadline: ");
scanf("%d", &deadline[i]);
      }
      else
      {
```

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

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

int get_observation_time(int selected_algo)
{
   if (selected_algo == 1)
   {
      return max(period[0], period[1], period[2]);
   }
   else if (selected_algo == 2)
   {
      return max(deadline[0], deadline[1], deadline[2]);
   }
}

int uti_time=0; void ut_time(int
selected_algo){
if(selected_algo==1)
 {
     for (int i = 0; i < num_of_process; i++)
   {
    uti_time+=(execution_time[i]/period[i]);
 }
}
   else if(selected_algo==2)
 {
     for (int i = 0; i < num_of_process; i++)
   {
```

```
    uti_time+=(execution_time[i]/deadline[i]);
  }
}
}

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

void rate_monotonic(int time)
{
   int process_list[100] = {0}, min = 999, next_process = 0;
float utilization = 0.0;    for (int i = 0; i < num_of_process;
i++)
   {
      utilization += (float)(execution_time[i] / period[i]);
   }
   int n = num_of_process;    float m
= (n * (pow(2, 1.0 / n) - 1));    if
(utilization > m)
```

```
  {
    printf("\nGiven problem is not schedulable under the said scheduling algorithm.\n");
  }
  for (int i = 0; i < time; i++)
  {
    min = 1000;       for (int j = 0; j <
num_of_process; j++)
    {
      if (remain_time[j] > 0)
      {
        if (min > period[j])
        {
          min = period[j];
          next_process = j;
        }
      }
    }
    if (remain_time[next_process] > 0)
    {
      process_list[i] = next_process + 1;
      remain_time[next_process] -= 1;
    }
    for (int k = 0; k < num_of_process; k++)
    {
      if ((i + 1) % period[k] == 0)
      {
        remain_time[k] = execution_time[k];
        next_process = k;
      }
    }
  }
  printf("Utilisation time %d",utilization);
print_schedule(process_list, time);
}

void earliest_deadline_first(int time){
float utilization = 0;
for (int i = 0; i < num_of_process; i++){
utilization += (1.0*execution_time[i])/deadline[i];
```

```
   }
   int n = num_of_process;

   int process[num_of_process];
int max_deadline, current_process=0, min_deadline,process_list[time];
bool is_ready[num_of_process];

   for(int i=0; i<num_of_process; i++){
      is_ready[i] = true;
process[i] = i+1;
   }

   max_deadline=deadline[0];
for(int i=1; i<num_of_process; i++){
if(deadline[i] > max_deadline)
max_deadline = deadline[i];
   }

   for(int i=0; i<num_of_process; i++){
for(int j=i+1; j<num_of_process; j++){
if(deadline[j] < deadline[i]){
int temp = execution_time[j];
execution_time[j] = execution_time[i];
execution_time[i] = temp;
temp = deadline[j];
deadline[j] = deadline[i];
deadline[i] = temp;
temp = process[j];
process[j] = process[i];
process[i] = temp;
         }
      }
   }

   for(int i=0; i<num_of_process; i++){
remain_time[i] = execution_time[i];
remain_deadline[i] = deadline[i];
   }

   for (int t = 0; t < time; t++){
if(current_process != -1){
```

```
        --execution_time[current_process];
        process_list[t] = process[current_process];
      }
 else
process_list[t] = 0;

      for(int i=0;i<num_of_process;i++){
        --deadline[i];
 if((execution_time[i] == 0) && is_ready[i]){
deadline[i] += remain_deadline[i];
is_ready[i] = false;
        }
        if((deadline[i] <= remain_deadline[i]) && (is_ready[i] == false)){
 execution_time[i] = remain_time[i];
           is_ready[i] = true;
        }
      }

      min_deadline = max_deadline;
current_process = -1;
for(int i=0;i<num_of_process;i++){
if((deadline[i] <= min_deadline) && (execution_time[i] > 0)){
current_process = i;
           min_deadline = deadline[i];
        }
      }
   }
   print_schedule(process_list, time);
}

int main()
{
   int option;
   int observation_time;

   while (1)
   {
   printf("\n1. Rate Monotonic\n2. Earliest Deadline first\n3. Proportional Scheduling\n\nEnter your
choice: ");    scanf("%d", &option);
switch(option)
```

```
   {
      case 1: get_process_info(option);
observation_time = get_observation_time(option);
rate_monotonic(observation_time);
break;
      case 2: get_process_info(option);
            observation_time = get_observation_time(option);
earliest_deadline_first(observation_time);
 break;

      case 3: exit (0);
default: printf("\nInvalid Statement");
   }
   }
   return 0;
}
```

OUTPUT:

```
Enter total number of processes (maximum 10): 3

Process 1:
==> Execution time: 1
==> Period: 4

Process 2:
==> Execution time: 2
==> Period: 6

Process 3:
==> Execution time: 3
==> Period: 12

Given problem is not schedulable under the said scheduling algorithm.

Scheduling:

Time: | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 |
P[1]: |####|    |    |    |####|    |    |    |####|    |    |    |
P[2]: |    |####|####|    |    |    |####|####|    |    |    |    |
P[3]: |    |    |    |####|    |####|    |    |    |####|    |    |
```

```
Enter total number of processes (maximum 10): 3

Process 1:
==> Execution time: 1
==> Deadline: 4

Process 2:
==> Execution time: 2
==> Deadline: 6

Process 3:
==> Execution time: 3
==> Deadline: 5
```

Rate Monotonic AND earliest Deadline 3|4|24

```
#include <stdio.h>
#define max 10
typedef struct{
  int id;
  int bt;
  int period;
  int rem_time;
  int next_dl;
} Process;

void sort_by_period ( Process p[], int n){
 for(int i=0; i<n-1 ; i++){
   for (int j=0; j<n-i-1 ; j++){
     if ( p[j].period > p[j+1].period){
        Process temp = p[j];
        p[j] = p[j+1];
        p[j+1] = temp;
     }
   }
 }
}
int gcd (int a, int b){
   return b==0 ? a: gcd (b, a%b);
}
int LCM (int a, int b){
   return (a*b)/gcd(a,b);
}
int calc_LCM ( Process p[], int n){
  int result = p[0].period;
  for (int i=1; i<n; i++){
    result = LCM ( result, p[i].period );
  }
  return result;
}
double util_factor ( Process p[], int n){
  double sum=0;
  for (int i=0; i<n; i++){
    sum += (double) p[i].bt / p[i].period;
  }
  return sum;
}
```

```
double rms_threshold(int n){
    return n * (pow(2.0, 1.0/n) - 1);
}
void rate_monotonic(Process p[], int n){
 int LCM.period = calc_LCM(p, n);
 printf("LCM = %d \n0\n", LCM_period);
 printf("Rate Monotonic Scheduling: \n");
 printf(PID Burst Period \n");
 for(int i=0; i<n; i++){
  printf("%d %d %d\n"); p[i].id, p[i].bt,
  p[i].period);
 }
 double util = util_factor(p0, n);
 double threshold = rms_threshold(n);
 printf("\n %.6f <= %.6f => %s\n", util,
 threshold, (util <= threshold)? "true": "false");
 if(util > threshold){
  printf("System may not be schedulable \n");
  return;
}
int timeline=0, ex=0;
while(timeline < LCM_period){
 int selected = -1;
 for(int i=0; i<n; i++){
    if(timeline % p[i].period == 0){
      p[i].rem_time = p[i].bt;
    }
    if(p[i].rem_time > 0){
      selected = i;
      break;
    }
   if(selected != -1){
   printf("time %d: Process %d is remaining \n",
   timeline, p[selected].id);
   process p[selected].rem_time--;
    ex++;
   }
   else{
    printf("Time %d : CPU is idle \n", timeline);
  }
  timeline++;
}
```

Output:

Enter the no. of processes = 3
Enter the CPU burst time
3 6 8
Enter the time periods: 2 4 5
LCM = 20

Rate Monotonic Scheduling:

PID	Burst	Period
1	3	2
2	6	4
3	8	5

4.600000 <= 0.7790763 is false
System may not be schedulable.

Earliest deadline first scheduling:

3/4/25

```
#include <stdio.h>
int gcd (int a, int b){
 while (b != 0){
  int temp = b;
  b = a % b;
  a = temp;
 }
 return a;
}
int LCM (int a, int b){
 return (a* b) / gcd(a,b);
}
int LCM (int a, int b){
 return (a * b) / gcd (a,b);
}

 struct process{
   int id, bt, deadline, period;
 };
void earliest_deadline_first (struct Process p[], int n,
int timelimit){
 int time =0;
 printf (" Earliest deadline scheduling ");
 printf (" %. PID \t burst \t deadline \t period \n");
 for(int i=0; i<n, i++){
  printf (" %.d \t %.d \t %.d \t %.d \n",
  p[i].id, p[i].bt, p[i].deadline, p[i].period);
 }
printf("\n Scheduling occurs for %.d ms\n", timelimit);
while (time < time_limit){
 int earliest = -1;
 for (int i=0; i<n; i++){
   if (p[i].bt >0){
     if (earliest == -1 || p[i].deadline < p[earliest].deadline)
      { earliest = i;
   } }
   if (earliest == -1)
      break;
   printf(" %.d ms: Task %.d is running \n", time,
   p [earliest].id);
   p[earliest].bt --;   time ++; } }
```

```
int main(){
  int n;
  printf("Enter the number of processes: ");
  scanf("%d", &n);
  struct Process p[n];
  printf("Enter the CPU burst time:\n");
  for(int i=0; i<n; i++){
    scanf("%d", &p[i].bt);
    p[i].id = i+1;
  }
  printf("Enter the deadline: ");
  for(int i=0; i<n; i++){
    scanf("%d", &p[i].deadline);
  }
  printf("Enter the time period: ");
  for(int i=0; i<n; i++){
    scanf("%d", p[i].period);
  }
  int hyperperiod = p[i].period;
  for(int i=1; i<n; i++){
    hyperperiod = LCM(hyperperiod, p[i].period);
  }
  printf("S/m will execute for hyperperiod (LCM of periods): %d ms\n", hyperperiod);
  earliest_deadline_first(p, n, hyperperiod);
  return 0;
}
```

Output:

Enter the number of processes: 3
Enter the CPU burst time: 2 3 4
Enter the deadline: 1 2 3
S/m will execute for hyper period (LCM of period): 6 ms

PID	Burst	Readline	period
1	2	1	1
2	3	2	2
3	4	3	3

Scheduling occurs for 6 ms:
0 ms : Task 1 is running
1 ms : Task 1 is running

2ms : Task 2 is running
3 ms : Task 2 is running
4 ms : Task 3 is running
5 ms : Task 3 is running

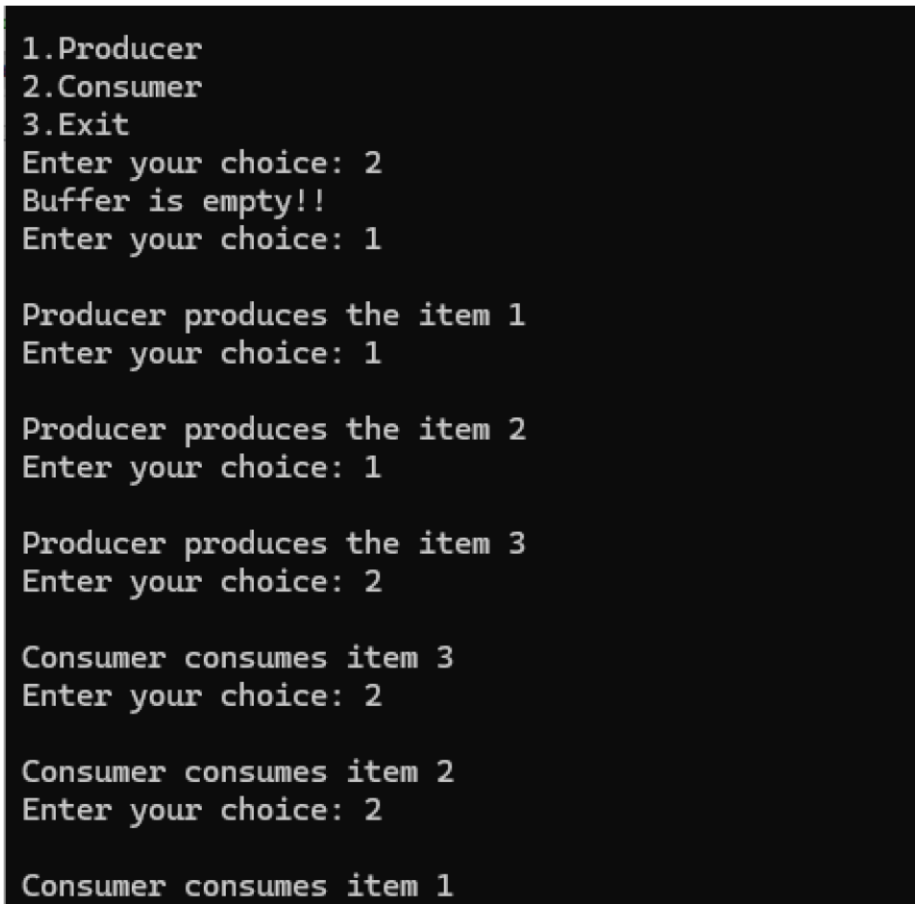
LAB -05

Producer consumer problem

```
#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\n2.Consumer\n3.Exit");
while(1)
   {
      printf("\nEnter your choice: ");
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);
}
```

OUTPUT:



Producer consumer:

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

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

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

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

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

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

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

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

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

int main(){
int p, c, cap, choice;
srand(time(NULL));
printf("Enter the number of Producers: ");
scanf("%d", &p);
printf("Enter the number of Consumers: ");
scanf("%d", &c);
printf("Capacity: Enter buffer capacity: ");
scanf("%d", &cap);
empty = cap;

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

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

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

```
        case 2:
            consumer(2);
            break;
        case 3:
            exit(0);
        default:
            printf("Invalid choice\n");
    }
    return 0;
}
```

Output:

```
Enter the number of Producers : 1
Enter the number of Consumers : 1
Enter the buffer capacity : 1
Successfully created Producer 1
Successfully created Consumer 1

1. Producer
2. Consumer
3. Exit
Enter your choice: 1
Producer 1 produced 39
Buffer : 39

1. Producer
2. Consumer
3. Exit
Enter your choice : 2
Consumer 1 consumed 39
Buffer : 0

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

LAB-06

Dining philosophers problem

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

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (i + 4) % N #define
RIGHT (i + 1) % N
int state[N]; int phil[N]
= {0,1,2,3,4}; sem_t
mutex; sem_t S[N];
void test(int i)
{
if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
{
state[i] = EATING; sleep(2);
printf("Philosopher %d takes fork %d and %d\n", i +1, LEFT +1, i +1);
printf("Philosopher %d is Eating\n", i +1);
sem_post(&S[i]);
}
}
void take_fork(int i)
{
sem_wait(&mutex); state[i] = HUNGRY;
printf("Philosopher %d is Hungry\n",i+1);
test(i);
sem_post(&mutex);
sem_wait(&S[i]);
sleep(1);
}

void put_fork(int i)
{
sem_wait(&mutex); state[i]
= THINKING;
```

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

OUTPUT:

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

Dining Philosopher : 16/04/25

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>

#define N 5
#define think 2
#define hungry 1
#define eating 0
#define L (phnum + 4) % N
#define R (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[L] != eating &&
  state[R] != eating){
      state[phnum] = eating;
      sleep(2);
      printf("Philosopher %d takes fork %d and %d\n",
      phnum+1, L+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] = think;
```

```
printf("Philosopher %d putting fork %d and %d down\n",
phnum+1, L+1, phnum+1);
printf("Philosopher %d is thinking\n", phnum+1);
test(L); test(R);
sem_post(&mutex);
}}

void * philosopher (void *num){
  int * i= num;
  while (1){
     sleep(1);
     take_fork(*i);
     sleep(0);
     put_fork(*i);
  }
}

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

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

  return 0;
}
```

Output:

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

LAB-07

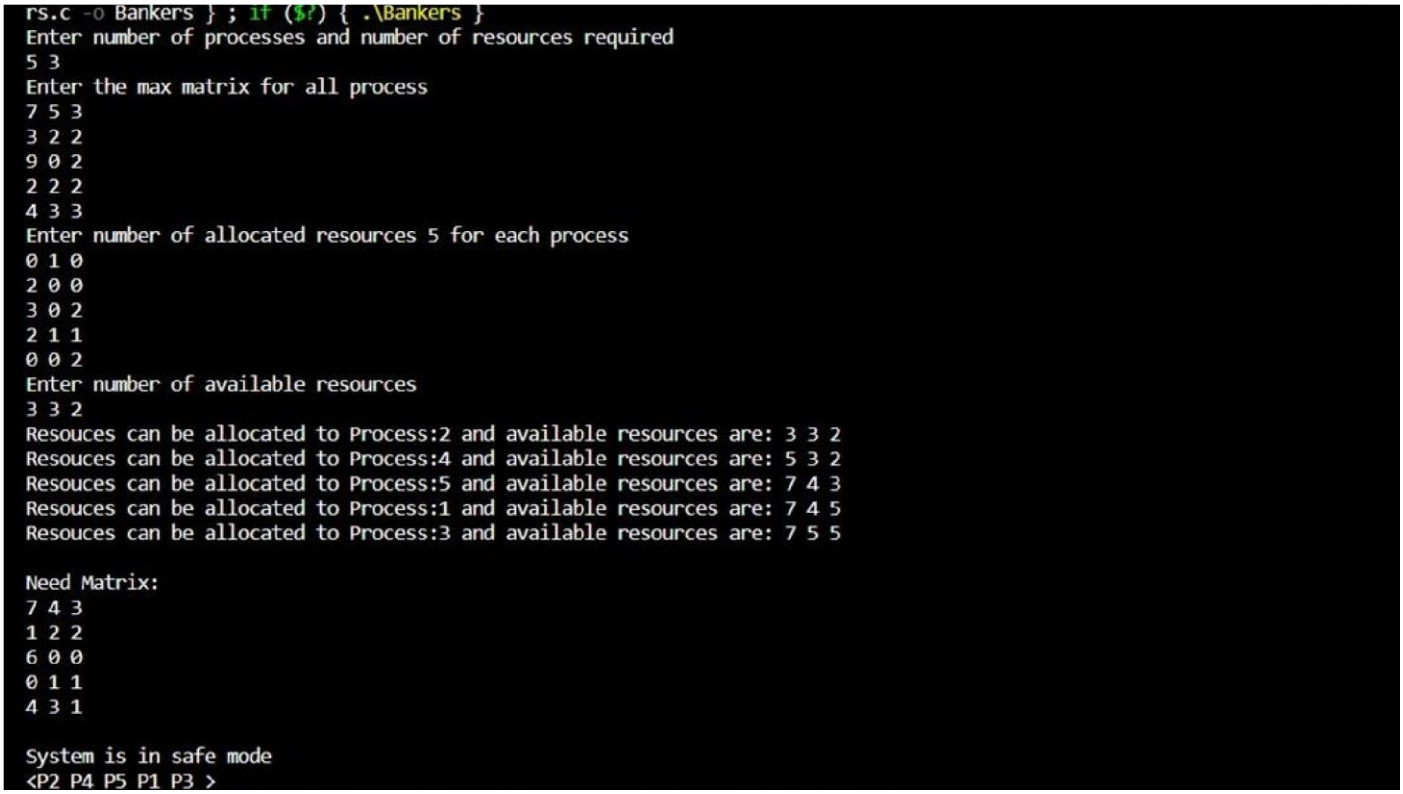
Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

```
#include <stdio.h>

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

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

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

   int available[m];
   printf("Enter the Available Resources:\n");
for (i = 0; i < m; i++)
   {
      scanf("%d", &available[i]);
   }

   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] - allocation[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] > available[j])
                  {
                        flag = 1;
              break;
                   }
               }

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

    int flag = 1;
    for (i = 0; i < n; i++)
    {
        if (f[i] == 0)
        {
            flag = 0;
```

```
            printf("The following system is not safe\n");
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\n", ans[n - 1]);
    }
    return 0;
}
```

Output:



17/4/25
Thursday

Bankers algorithm:

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_P 10
#define MAX_R 10

int main(){
 int n, m;
 int alloc [MAX_P][MAX_R];
 int max [MAX_P][MAX_R];
 int avail [MAX_R];
 int need [MAX_P][MAX_R];
 bool finished [MAX_P] = { false };
 int safe_seq [MAX_P];
 int count = 0;
 printf("Enter number of processes and resources: ");
 scanf("%d %d", &n, &m);
 printf("Enter allocation matrix:\n");
 for (int i=0; i<n; i++) {
    for (int j = 0; j<m ; j++) {
       scanf("%d", &alloc[i][j]);
    }
 }

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

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

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

```
while (count < n) {
  bool found = false;
  for (int i = 0; i < n; i++) {
    if (!finished[i]) {
      int j;
      for (j = 0; j < m; j++) {
        if (need[i][j] > avail[j]) {
          break;
        }
      }
      if (j == m) {
        for (int k = 0; k < m; k++) {
          avail[k] += alloc[i][k];
        }
        safe_seq[count++] = i;
        finished[i] = true;
        found = true;
      }
    }
  }
  if (!found) {
    printf("System is not in safe state.\n");
    return 0;
  }
}
printf("System is in safe state.\n");
printf("Safe sequence is: ");
for (int i = 0; i < n; i++) {
    printf("P%d", safe_seq[i]);
    if (i != n-1) {
        printf(" -> ");
    }
}
printf("\n");
return 0;
}
```

Output:

Enter number of processes and resources : 5 3

Enter allocation matrix :

0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter max matrix :

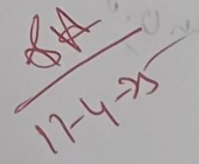
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Enter available matrix :

3 3 2

System is in safe state.

Safe sequence is : P1 -> P3 -> P4 -> P0 -> P2



LAB-08
Write a C program to simulate deadlock detection.

```
#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 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]; }

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

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];
}
}
}
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:

```
Deadlock Detection
Enter the no of Processes: 3
Enter the no of resource instances: 3
Enter the Max Matrix:
3 6 8
4 3 3
3 4 4
Enter the Allocation Matrix:
3 3 3
2 0 4
1 2 4
Enter the available Resources:
1 2 0
Process  Allocation      Max     Available
P1         3 3 3        3 6 8   1 2 0
P2         2 0 4        4 3 3
P3         1 2 4        3 4 4

System is in Deadlock and the Deadlock process are
P0      P1      P2
```

Deadlock

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

#define MAX_PROC 5
#define MAX_RES 3

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

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

```
for(i=0; i<n; i++){
   alloc[i][j] != 0 bool all_zero = true;
   for(j=0; j<m; j++){
     if(alloc[i][j] != 0){
        all_zero = false;
      break; }
   }
   finish[i] = all_zero;
}
bool changed;
```

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

bool deadlock = false;
for(i=0; i<n; i++){
  if (!finish[i]){
     deadlock = true;
     break
  }
}

if (deadlock){
 printf("System is in a deadlock state\n"); }
else {
  printf("System is not in a deadlock state\n");
}
 return 0;
}
```

Output:

```
Enter number of processes and resources:
5 3
Enter allocation matrix:
0 1 0
2 0 0
3 0 2
```

2 1 1
0 0 2

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

Enter available matrix:
3 3 2

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

LAB-09

Write a C program to simulate the following contiguous memory allocation techniques: (a) Worst-fit

(b) Best-fit

(c) First-fit

```
#include
<stdio.h>
#define max 25
void firstFit(int b[], int nb, int f[], int nf);
void worstFit(int b[], int nb, int f[], int nf);
void bestFit(int b[], int nb, int f[], int nf);

int main()
{
   int b[max], f[max], nb, nf;

   printf("Memory Management Schemes\n");

   printf("\nEnter the number of blocks:");
scanf("%d", &nb);

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

   printf("\nEnter the size of the blocks:\n");
   for (int i = 1; i <= nb; i++)
   {
      printf("Block %d:", i);
      scanf("%d", &b[i]);
   }
   printf("\nEnter the size of the files:\n");
   for (int i = 1; i <= nf; i++)
   {
      printf("File %d:", i);
      scanf("%d", &f[i]);
   }
   printf("\nMemory Management Scheme - First Fit");
firstFit(b, nb, f, nf);

   printf("\n\nMemory Management Scheme - Worst Fit");
worstFit(b, nb, f, nf);

   printf("\n\nMemory Management Scheme - Best Fit");
bestFit(b, nb, f, nf);     return 0;
}
void firstFit(int b[], int nb, int f[], int nf)
{
   int bf[max] = {0};
int ff[max] = {0};     int
frag[max], i, j;     for (i
= 1; i <= nf; i++)
```

```
    {
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1 && b[j] >= f[i])
            {
              ff[i] = j;
              bf[j] = 1;
              frag[i] = b[j] - f[i];
break;
            }
        }
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
for (i = 1; i <= nf; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);}

void worstFit(int b[], int nb, int f[], int nf)
{
    int bf[max] = {0};
int ff[max] = {0};
    int frag[max], i, j, temp, highest = 0;

    for (i = 1; i <= nf; i++)
    {
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1)
            {
                temp = b[j] - f[i];
                if (temp >= 0 && highest < temp)
                {
                  ff[i] = j;
                  highest = temp;
                }
            }
        }
        frag[i] = highest;
bf[ff[i]] = 1;
        highest = 0;
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
for (i = 1; i <= nf; i++)
    {
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
}

void bestFit(int b[], int nb, int f[], int nf)
{
```

```
    int bf[max] = {0};
int ff[max] = {0};
    int frag[max], i, j, temp, lowest = 10000;

    for (i = 1; i <= nf; i++)
    {
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1)
            {
                temp = b[j] - f[i];
                if (temp >= 0 && lowest > temp)
                {
                    ff[i] = j;
                    lowest = temp;
                }
            }
        }
        frag[i] = lowest;
bf[ff[i]] = 1;
        lowest = 10000;
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
for (i = 1; i <= nf && ff[i] != 0; i++)
    {
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
}
```

Output:

```
Enter the number of blocks
5
Enter the number of processes
8
Enter the block size
100 500 200 300 600
Enter the process size
212 415 63 124 23 89 73 13

1.First-fit
2.Best-fit
3.Worst-fit
Enter your choice
1

Process No.     Process Size    Block no.
 1                      212     2
 2                      415     5
 3                      63      1
 4                      124     2
 5                      23      1
 6                      89      2
 7                      73      2
 8                      13      1
```

14/5/25
Wednesday

~~8) Write~~

9) Write a C program to simulate contingeous memory allocation techniques
a) Worst fit b) Best fit c) First fit

```
#include <stdio.h>
struct Block {
    int size;
    int allocated;
};
struct File {
  int size;
  int block_no;
};
void resetBlocks ( struct Block block[], int n) {
for (int i=0; i<n; i++){
   blocks[i]. allocated =0; }
}
void firstFit ( struct Block blocks[], int n_blocks,
struct File files[], int nfiles){
  printf ("\n Memory Management scheme -
First fit \n");
printf (" File_no : \t File_size\t Block_no : \t
Block_size : \n");
for (int i=0 ; i<n_files; i++) {
    files[i]. block_no = -1;
for (int j =0 ; j<n_blocks, j++) {
    if ( ! blocks[j]. allocated && blocks[i]. size
    >= files[i]. size){
      files[i]. block_no =j+1;
      blocks[j]. allocated = 1;
      printf ("%d \t\t %d \t\t %d \t\t %d \n", i+1,
      files[i]. size, j+1, blocks[j]. size);
      break;
  } }
if (files[i]. block_no == -1){
   printf ("%d \t\t %d \t\t - \t\t - \n", i+1,
   files[i]. size); }
```

```
}
}
void bestFit(struct Block blocks[], int n_blocks, struct
File files[], int n_files){
 printf("\n\t memory management scheme - Best Fit\n");
 printf("File no:\t File_size\t Block_no:\t Block_size:\n");
 for(int i=0; i<n_files; i++){
   int bestIdx = -1;
   for(int j=0; j<n_blocks; j++){
     if(!blocks[j].allocated && blocks[j].size >= files[i].size){
       if(bestIdx == -1 || blocks[j].size < blocks[bestIdx].size)
       { bestIdx = j; }
     }
   }
   if(bestIdx != -1){
     blocks[bestIdx].allocated = 1;
     files[i].block_no = bestIdx + 1;
     printf("%d\t\t %d\t\t %d\t\t %d\n", i+1,
     files[i].size, bestIdx+1, blocks[bestIdx].size);
   }
   else{
     printf("%d\t\t %d\t\t -\t\t\n", i+1, files[i].size);
   }
 }
}

void worstFit(struct Block blocks[], int n_blocks,
struct File files[], int n_files){
  printf("\n\t memory management scheme - worst
  fit\n");
  printf("File_no:\t File_size\t Block_no:\t Block_size:\n");
  for(int i=0; i<n; i++){
     int worstIdx = -1;
     for(int j=0; j<n_blocks, j++){
       if(!blocks[j].allocated && blocks[j].size >=
       files[i].size){
         if(worstIdx == -1 || blocks[j].size >
           blocks[worstIdx].size){
             worstIdx = j;
     }}
   }}
```

```
if (worstIdx != -1) {
    blocks[worstIdx].allocated = 1;
    files[i].block_no = worstIdx + 1;
    printf("%d\t\t%d\t\t%d\t\t%d\n",
    i+1, files[i].size, worstIdx+1,
    blocks[worstIdx].size);
}
else {
    printf("%d\t\t\t\t%d\t\t\t\n", i+1, files[i].size);
}
}
}
```

Output:

```
Memory Management Scheme
Enter the number of blocks: 5
Enter the number of files: 4
Enter the size of the blocks:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600

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

1) First fit
2) Best fit
3) Worst Fit
Enter your choice: 1

Memory Management Scheme is First fit
```

File_no	File_size	Block_no	Block_size
1	212	2	500
2	417	5	600
3	112	3	200
4	420	-	-

1. First Fit
2. Best fit
3. Worst fit
4- Exit

Enter your choice: 2

Memory Management scheme is Best fit

File_no	File_size	Block_no	Block_size
1	212	4	300
2	417	2	500
3	112	3	200
4	420	5	600

1) First fit
2) Best fit
3) Worst fit
4) Exit

Enter your choice : 3

Memory Management scheme is worst fit

File_no	File_size	Block_no	Block_size
1	212	5	600
2	417	2	500
3	112	4	200
4	420	—	—

LAB-10

Write a C program to simulate page replacement algorithms:

(a) FIFO

(b) LRU

(c) Optimal

```
#include<stdio.h>
int n, f, i, j, k;
int in[100]; int
p[50]; int
hit=0;
int pgfaultcnt=0;

void getData()
{
    printf("\nEnter length of page reference sequence:");
scanf("%d",&n);
    printf("\nEnter the page reference sequence:");
for(i=0; i<n; i++)
   scanf("%d",&in[i]);
  printf("\nEnter no of frames:");
   scanf("%d",&f);
}

void initialize()
{
    pgfaultcnt=0;
for(i=0; i<f; i++)
        p[i]=9999; }

int isHit(int data)
{
hit=0;
    for(j=0; j<f; j++)
    {
        if(p[j]==data)
        {
hit=1;
break;         } }
return hit;
}

int getHitIndex(int data)
{     int hitind;
for(k=0; k<f; k++)
    {
        if(p[k]==data)
        {
            hitind=k;
break;
        }}     return
hitind;
```

```
}

void dispPages()
{
    for (k=0; k<f; k++)
    {
        if(p[k]!=9999)
            printf(" %d",p[k]); }}

void dispPgFaultCnt()
    printf("\nTotal no of page faults:%d",pgfaultcnt);
void fifo() {
    getdata();
initialize();
for(i=0; i<n; i++)
    {
        printf("\nFor %d :",in[i]);
//not a hit
if(isHit(in[i])==0)
        {

            for(k=0; k<f-1; k++)
p[k]=p[k+1];             p[k]=in[i];
pgfaultcnt++;
            dispPages();
        }
else
            printf("No page fault");
    }
    dispPgFaultCnt();
}
void optimal()
{     initialize();
int near[50];
    for(i=0; i<n; i++)
    {
        printf("\nFor %d :",in[i]);
if(isHit(in[i])==0)
        {
            for(j=0; j<f; j++)
            {                int
pg=p[j];                int
found=0;
                for(k=i; k<n; k++)
                {
                    if(pg==in[k])
{                       near[j]=k;
found=1;
break;                  }
else
found=0;                }
```

```
if(!found)
near[j]=9999;
            }
            int max=-9999;
int repindex;
for(j=0; j<nf; j++)            {
if(near[j]>max)
{
max=near[j];
repindex=j;
                }
            }
p[repindex]=in[i];
            pgfaultcnt++;

            dispPages();
        }        else
printf("No page fault");
    }
    dispPgFaultCnt();
}

void lru() {
initialize();

    int least[50];
for(i=0; i<n; i++)
    {

        printf("\nFor %d :",in[i]);

if(isHit(in[i])==0)
        {

            for(j=0; j<nf; j++)
{                int pg=p[j];
int found=0;                for(k=i-
1; k>=0; k--)
                {
                    if(pg==in[k])
                    {
least[j]=k;
found=1;
break;
                    }
else
found=0;
                }
if(!found)
least[j]=-9999;
            }
```

```
            int min=9999;
int repindex;
            for(j=0; j<nf; j++)
            {
                if(least[j]<min)
                {
                    min=least[j];
repindex=j;
                }
            }
            p[repindex]=in[i];
            pgfaultcnt++;

            dispPages();
        }
else
            printf("No page fault!");
    }
    dispPgFaultCnt();
}

int main()
{     int
choice;
    while(1)
    {
        printf("\nPage Replacement Algorithms\n
1.Enter data\n 2.FIFO\n 3.Optimal\n 4.LRU\n 5.Exit\n Enter your choice:");
scanf("%d",&choice);
        switch(choice)
        {
        case 1: getData();
break;
     case 2: fifo();
break;
 case 3: optimal();
break;
   case 4: lru();
break;
   default: return 0;
break;
        }
    } }
```

Output:

(a) Enter Data:

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:1

Enter length of page reference sequence:8

Enter the page reference sequence:2 3 4 2 3 5 6 2

Enter no of frames:3
```

(b) FIFO:

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:2

For 2 : 2
For 3 : 2 3
For 4 : 2 3 4
For 2 :No page fault
For 3 :No page fault
For 5 : 3 4 5
For 6 : 4 5 6
For 2 : 5 6 2
Total no of page faults:6
```

(c) OPTIMAL:

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:3

For 2 : 2
For 3 : 2 3
For 4 : 2 3 4
For 2 :No page fault
For 3 :No page fault
For 5 : 2 5 4
For 6 : 2 6 4
For 2 :No page fault
Total no of page faults:5
```

(d) LRU:

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:4

For 2 : 2
For 3 : 2 3
For 4 : 2 3 4
For 2 :No page fault!
For 3 :No page fault!
For 5 : 2 3 5
For 6 : 6 3 5
For 2 : 6 2 5
Total no of page faults:6
```

15 | 5 | 25
Thursday

Page Replacement (FIFO)

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

```
        else
            printf("- ");
    }
    printf("\t No\n");
  }
 }
 printf("\n Total Page Faults = %d \n", count);
 return 0;
}
```

Output :

Enter the number of pages : 15

Enter the page reference string :

7 0 1 2 0 3 0 4 2 3 0 3 1 2 0

Enter number of frames: 3

Page	Frames	Page Fault
7	7 - -	Yes
0	7 0 -	Yes
1	7 0 1	Yes
2	2 0 1	Yes
0	2 0 1	No
3	2 3 1	Yes
0	2 3 0	Yes
4	4 3 0	Yes
2	4 2 0	Yes
3	4 2 3	Yes
0	0 2 3	Yes
3	0 2 3	No
1	0 1 3	Yes
2	0 1 2	Yes
0	0 1 2	No

Total Page Faults = 12

2) LRU

```
#include<stdio.h>
int main(){
  int n, frames, i, j, k, faults=0;
  printf("Enter number of pages: ");
  scanf("%d", &n);
  int pages[n];
  printf("Enter the reference string: ");
  for(i=0; i<n; i++)
     scanf("%d", &pages[i]);
  printf("Enter number of frames: ");
  scanf("%d", &frames);
  int frame_arr[frames];
  int time[frames];
  for(i=0; i<frames; i++){
     frame_arr[i] = -1;
     time[i] = 0;
  }
  int counter = 0;
  for(i=0; i<n; i++){
     int flag = 0;
     for(j=0; j<frames; j++){
        if(frame_arr[j] == pages[i]){
           flag = 1;
           counter++;
           time[j] = counter;
           break;
     }}
     if(flag==0){
        faults++;
        int min_time = time[0], min_pos = 0;
        for(k=1; k<frames; k++){
           if(time[k] < min_time){
              min_time = time[k];
              min_pos = k;
           }
        }
        frame_arr[min_pos] = pages[i];
        counter++;
        time[min_pos] = counter; }
```

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

=

Output:

```
Enter number of pages: 7
Enter the reference string: 1 3 0 3 5 6 3
Enter number of frames: 3
Frames after accessing 1: 1 - -
Frames after accessing 3: 1 3 -
Frames after accessing 0: 1 3 0
Frames after accessing 3: 1 3 0
Frames after accessing 5: 5 3 0
Frames after accessing 6: 5 3 6
Frames after accessing 3: 5 3 6

Total pages faults : 5
Total page hits : 2
```

3) Optimal

```
#include <stdio.h>

int main() {
 int n, frames, i, j, k, faults = 0;
 printf("Enter number of pages: ");
 scanf("%d", &n);
 int pages[n];
 printf("Enter the ref reference strings: ");
 for (i=0; i<n; i++)
    scanf("%d", &pages[i]);
 printf("Enter number of frames: ");
 scanf("%d", &frames);
 int frame_an[frames];
 for (i=0; i<frames; i++)
    frame_an[i] = -1;
 for (i=0; i<n; i++) {
    int flag = 0;
    for (j=0; j<frames; j++) {
       if (frame_an[j] == pages[i]) {
          flag = 1;
          break;
       }
    }
    if (flag == 0) {
       faults++;
       int pos = -1;
       for (j=0; j<frames; j++) {
          if (frame_an[j] == -1) {
             pos = j;
             break;
          }
       }
       if (pos == -1) {
          int farthest = -1, replace_index = 0;
          for (j=0; j<frames; j++) {
             int found = 0;
             for (k=i+1; k<n; k++) {
                if (frame_an[j] == pages[k]) {
                   if (k > farthest) {
                      farthest = k;
```

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

Output:

```
Enter number of pages: 7
Enter the reference string:
1 3 0 3 5 6 3
Enter number of frames: 3
Frames after accessing 1 : 1 - -
Frames after accessing 3 : 1 3 -
Frames after accessing 0 : 1 3 0
Frames after accessing 3 : 1 3 0
Frames after accessing 5 : 5 3 0
Frames after accessing 6 : 6 3 0
```

Frames after accessing 3 : 0 6 3 0

Total page faults : 5

Total page hits : 2

OK

15-5-2025