VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

OPERATING SYSTEMS

Submitted by

NAVANITH KRISHNA R (1BM22CS172)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Apr-2024 to Aug-2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "OPERATING SYSTEMS – 23CS4PCOPS" carried out by **NAVANITH KRISHNA R (1BM22CS172),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **OPERATING SYSTEMS - (23CS4PCOPS)** work prescribed for the said degree.

Dr. Sunayana S
Asst. Professor
Department of CSE
BMSCE, Bengaluru

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

Table Of Contents

Course Outcomes

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

CO2: Analyze various Operating system strategies and techniques.

CO3: Demonstrate the different functionalities of Operating System.

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

Lab Program – 1

Question:

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 (Non-preemptive)

Code:

(a) FCFS

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

struct P{
    int id;
    int bt;
    int at;
    int wt;
    int tat;
    int rt;
    int st;
    int et;
    int v;
};

void main(){
    int n,ct=0;
    float awt=0,atat=0,art=0,tp;
    printf("Enter number of processes: ");
    scanf("%d",&n);
    struct P p[n];
    struct P temp;
    for(int i=0;i<n;i++){
        printf("Enter Process %d ID: ",i+1);
```

```
    scanf("%d",&p[i].id);
    printf("Enter Burst Time and Arrival Time %d: ",i+1);
    scanf("%d %d",&p[i].bt,&p[i].at);
}

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

for(int i=0;i<n;i++){
    if(p[i].at<ct){
        p[i].st=ct;
    }
    else{
        p[i].st=p[i].at;
    }
    p[i].et=p[i].st+p[i].bt;
    ct+=p[i].bt;
}
for(int i=0;i<n;i++){
    p[i].tat=p[i].et-p[i].at;
    p[i].wt=p[i].tat-p[i].bt;
    p[i].rt=p[i].st-p[i].at;
}

printf("Process\tWaiting Time\tTurn Around Time\tResponse Time\n");
for(int i=0;i<n;i++){
    printf("%d\t\t%d\t\t%d\t\t%d\n",p[i].id,p[i].wt,p[i].tat,p[i].rt);
    awt+=p[i].wt;
    atat+=p[i].tat;
```

```
        art+=p[i].rt;
    }

    tp=(float)p[n-1].et/n;
    printf("Average Waiting Time: %.2f\n",awt/n);
    printf("Average Turn Around Time: %.2f\n",atat/n);
    printf("Average Response Time: %.2f\n",art/n);
    printf("Throughput: %.2f",tp);

}
```

b) SJF (Non-preemptive)

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

struct P{
    int id;
    int bt;
    int at;
    int wt;
    int tat;
    int rt;
    int st;
    int et;
    int v;
};

void main(){
    int n,ct=0;
    float awt=0,atat=0,art=0,tp;
    printf("Enter number of processes: ");
    scanf("%d",&n);
    struct P p[n];
    struct P temp;
    for(int i=0;i<n;i++){
        p[i].id=i+1;
```

```
    p[i].v=0;
    printf("Enter Burst Time and Arrival Time of P%d: ",i+1);
    scanf("%d %d",&p[i].bt,&p[i].at);
}

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

int cf=p[0].bt, min=INT_MAX,m,count=0;
p[0].v=1;

while(count<n-1){
for(int i=0;i<n;i++){
    if(p[i].at<=cf && p[i].v==0){
        if(p[i].bt<min){
            min=p[i].bt;
            m=i;
        }
    }
}
p[m].v=1;
cf+=p[m].bt;
```

```
    min=INT_MAX;

    temp=p[count+1];
    p[count+1]=p[m];
    p[m]=temp;

    count++;
    }

    printf("Process\tWaiting Time\tTurn Around Time\tResponse Time\n");

    for(int i=0;i<n;i++){
        if(p[i].at<ct){
            p[i].st=ct;
        }
        else{
            p[i].st=p[i].at;
        }
        p[i].et=p[i].st+p[i].bt;
        ct+=p[i].bt;

        p[i].tat=p[i].et-p[i].at;
        p[i].wt=p[i].tat-p[i].bt;
        p[i].rt=p[i].st-p[i].at;

        printf("%d\t\t%d\t\t%d\t\t%d\n",p[i].id,p[i].wt,p[i].tat,p[i].rt);
        awt+=p[i].wt;
        atat+=p[i].tat;
        art+=p[i].rt;
    }
    tp=(float)p[n-1].et/n;
    printf("Average Waiting Time: %.2f\n",awt/n);
    printf("Average Turn Around Time: %.2f\n",atat/n);
    printf("Average Response Time: %.2f\n",art/n);
    printf("Throughput: %.2f",tp);
}
```

Output:

a)

```
Enter number of processes: 4
Enter Process 1 ID: 1
Enter Burst Time and Arrival Time 1: 3 0
Enter Process 2 ID: 2
Enter Burst Time and Arrival Time 2: 5 2
Enter Process 3 ID: 3
Enter Burst Time and Arrival Time 3: 7 3
Enter Process 4 ID: 4
Enter Burst Time and Arrival Time 4: 10 5
Process Waiting Time    Turn Around Time        Response Time
1               0               3               0
2               1               6               1
3               5               12              5
4               10              20              10
Average Waiting Time: 4.00
Average Turn Around Time: 10.25
Average Response Time: 4.00
Throughput: 6.25
```

b)

```
Enter number of processes: 5
Enter Burst Time and Arrival Time of P1: 3 0
Enter Burst Time and Arrival Time of P2: 4 2
Enter Burst Time and Arrival Time of P3: 8 6
Enter Burst Time and Arrival Time of P4: 11 9
Enter Burst Time and Arrival Time of P5: 15 12
Process Waiting Time    Turn Around Time        Response Time
1               0               3               0
2               1               5               1
3               1               9               1
4               6               17              6
5               14              29              14
Average Waiting Time: 4.40
Average Turn Around Time: 12.60
Average Response Time: 4.40
Throughput: 8.20
```

Lab Program - 2

Question:

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

a) SJF (Preemptive)

b) Round Robin Algorithm

Code:

(a) SJF (Preemptive)

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

struct P{
   int id;
   int bt;
   int at;
   int wt;
   int tat;
   int rt;
   int st;
   int et;
   int v;
   int b;
};

void main(){
   int n,ct=0;
   float awt=0,atat=0,art=0,tp;
   printf("Enter number of processes: ");
   scanf("%d",&n);
   struct P p[n];
   struct P temp;
   for(int i=0;i<n;i++){
      p[i].id=i+1;
      p[i].v=0;
      p[i].st=-1;
      printf("Enter Burst Time and Arrival Time of P%d: ",i+1);
      scanf("%d %d",&p[i].bt,&p[i].at);
      p[i].b=p[i].bt;
```

```
}

int cf=0, min=INT_MAX,m=0,count=0;
while(count<n){
   for(int i=0;i<n;i++){
      if(p[i].at<=cf){
         if(p[i].bt==min){
            min=p[i].at>p[m].at?p[i].bt:p[m].bt;
         }
         if(p[i].bt<min && p[i].v!=1){
            min=p[i].bt;
            m=i;
         }
      }
   }
   p[m].bt-=1;

   if(p[m].st<0)
   p[m].st=cf;

   cf+=1;
   if(p[m].bt==0){
      p[m].et=cf;
      count++;
      p[m].v=1;
   }
   min=INT_MAX;

}
printf("Process\tWaiting Time\tTurn Around Time\tResponse Time\n");

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

   p[i].tat=p[i].et-p[i].at;
   p[i].wt=p[i].tat-p[i].b;
   p[i].rt=p[i].st-p[i].at;

   printf("%d\t\t%d\t\t%d\t\t%d\n",p[i].id,p[i].wt,p[i].tat,p[i].rt);
   awt+=p[i].wt;
   atat+=p[i].tat;
```

```
        art+=p[i].rt;
    }

    tp=(float)cf/n;
    printf("Average Waiting Time: %.2f\n",awt/n);
    printf("Average Turn Around Time: %.2f\n",atat/n);
    printf("Average Response Time: %.2f\n",art/n);
    printf("Throughput: %.2f",tp);

}
```

(b) Round Robin

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

#define MAX_PROCESSES 10

struct Process {
    int id;
    int at;
    int bt;
    int wt;
    int tat;
    int st;
    int et;
    int rt;
    int vi;
    int obt;
};

int main() {
```

```
struct Process p[MAX_PROCESSES];
int n;
int total_wt = 0;
int total_tat = 0;
int total_rt = 0;
int total_time = 0;
int tq = 0;

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



for (int i = 0; i < n; i++) {
    p[i].id = i + 1;
    printf("Enter arrival time and burst time for process %d: ", p[i].id);
    scanf("%d %d", &p[i].at, &p[i].bt);
    p[i].obt = p[i].bt;
    p[i].st = -1;
    p[i].et = -1;
    p[i].vi = 0;
}
int count = 0;
int ind = 0;
int curr_time = 0;
```

```
while(1){
    int skipped = 0;
    if(p[ind].at > curr_time){
        ind = (ind + 1) % n;
        continue;
    }
    if(p[ind].st == -1){
        p[ind].st = curr_time;
    }
    if(p[ind].bt > tq){
        p[ind].bt -= tq;
        skipped = tq;
    }
    else if(p[ind].bt > 0){
        skipped = p[ind].bt;
        p[ind].bt = 0;
        p[ind].et = curr_time + skipped;
        total_time = curr_time;
        count++;
    }
    curr_time += skipped;
    ind = (ind + 1) % n;
    if(count == n){
        break;
    }
}
```

```
    for(int i = 0; i < n; i++){
        p[i].tat = p[i].et - p[i].at;
        p[i].wt = p[i].tat - p[i].obt;
        p[i].rt = p[i].st - p[i].at;
        total_wt += p[i].wt;
        total_tat += p[i].tat;
        total_rt += p[i].rt;
    }

    printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\tResponse
Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].id, p[i].at, p[i].obt, p[i].wt, p[i].tat, p[i].rt);
    }

    printf("\nAverage Waiting Time: %.2f\n", (float)total_wt / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_tat / n);
    printf("Throughput: %.2f\n", (float)total_time / n);
    printf("Average Response Time: %.2f\n", (float)total_rt / n);



    return 0;
}
```

Output:

a) **SJF (Pre-emptive)**

```
Enter number of processes: 5
Enter Burst Time and Arrival Time of P1: 6 2
Enter Burst Time and Arrival Time of P2: 2 5
Enter Burst Time and Arrival Time of P3: 8 1
Enter Burst Time and Arrival Time of P4: 3 0
Enter Burst Time and Arrival Time of P5: 4 4
Process Waiting Time    Turn Around Time        Response Time
1               7               13              1
2               0               2               0
3               14              22              14
4               0               3               0
5               2               6               0
Average Waiting Time: 4.60
Average Turn Around Time: 9.20
Average Response Time: 3.00
Throughput: 4.60
```

b) **Round Robin**

```
Enter the number of processes: 6
Enter time quantum: 5
Enter arrival time and burst time for process 1: 0 7
Enter arrival time and burst time for process 2: 1 4
Enter arrival time and burst time for process 3: 2 15
Enter arrival time and burst time for process 4: 3 11
Enter arrival time and burst time for process 5: 4 20
Enter arrival time and burst time for process 6: 9 4

Process Arrival Time    Burst Time      Waiting Time    Turnaround Time Response Time
1       0               7               23              30              0
2       1               4               4               8               4
3       2               15              33              48              7
4       3               11              37              48              11
5       4               20              37              57              15
6       9               4               15              19              15

Average Waiting Time: 24.83
Average Turnaround Time: 35.00
Throughput: 9.33
Average Response Time: 8.67
```

Lab Program - 3

Question:

Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.
Priority (pre-emptive & Non-pre-emptive)

Code:

a) Pre-emptive

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

struct P{
   int id;
   int bt;
   int at;
   int p;
   int wt;
   int tat;
   int rt;
   int st;
   int et;
   int v;
   int b;
};

void main(){
   int n,ct=0;
   float awt=0,atat=0,art=0,tp;
   printf("Enter number of processes: ");
   scanf("%d",&n);
   struct P p[n];
   struct P temp;
```

```
for(int i=0;i<n;i++){
   p[i].id=i+1;
   p[i].v=0;
   p[i].st=-1;
   printf("Enter Burst Time, Arrival Time and Priority of P%d: ",i+1);
   scanf("%d %d %d",&p[i].bt,&p[i].at,&p[i].p);
   p[i].b=p[i].bt;
}

int cf=0, min=INT_MAX,m=0,count=0;
while(count<n){
   for(int i=0;i<n;i++){
      if(p[i].at<=cf){
         if(p[i].p<min && p[i].v!=1){
            min=p[i].p;
            m=i;
         }
      }
   }
   p[m].bt-=1;

   if(p[m].st<0)
   p[m].st=cf;

   cf+=1;
   if(p[m].bt==0){
      p[m].et=cf;
      count++;
      p[m].v=1;
   }
   min=INT_MAX;
```

```
    }
    printf("Process\tWaiting Time\tTurn Around Time\tResponse Time\n");

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

        p[i].tat=p[i].et-p[i].at;
        p[i].wt=p[i].tat-p[i].b;
        p[i].rt=p[i].st-p[i].at;

        printf("%d\t\t%d\t\t%d\t\t%d\n",p[i].id,p[i].wt,p[i].tat,p[i].rt);
        awt+=p[i].wt;
        atat+=p[i].tat;
        art+=p[i].rt;
    }

    tp=(float)cf/n;
    printf("Average Waiting Time: %.2f\n",awt/n);
    printf("Average Turn Around Time: %.2f\n",atat/n);
    printf("Average Response Time: %.2f\n",art/n);
    printf("Throughput: %.2f",tp);
}
```

b) Non-Pre-emptive

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

struct P{
    int id;
    int bt;
    int at;
    int p;
    int wt;
    int tat;
    int rt;
    int st;
    int et;
```

```
    int v;
};

void main(){
    int n,ct=0;
    float awt=0,atat=0,art=0,tp;
    printf("Enter number of processes: ");
    scanf("%d",&n);
    struct P p[n];
    struct P temp;
    for(int i=0;i<n;i++){
        p[i].id=i+1;
        p[i].v=0;
        printf("Enter Burst Time, Arrival Time and Priority of P%d: ",i+1);
        scanf("%d %d %d",&p[i].bt,&p[i].at,&p[i].p);
    }

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

    int cf=p[0].bt, min=INT_MAX,m,count=0;
    p[0].v=1;

    while(count<n-1){
    for(int i=0;i<n;i++){
        if(p[i].at<=cf && p[i].v==0){
```

```
        if(p[i].p<min){
            min=p[i].p;
            m=i;
        }
    }
}
p[m].v=1;
cf+=p[m].bt;
min=INT_MAX;

temp=p[count+1];
p[count+1]=p[m];
p[m]=temp;

count++;
}

printf("\nProcess\tWaiting Time\tTurn Around Time\tResponse Time\n");

for(int i=0;i<n;i++){
    if(p[i].at<ct){
        p[i].st=ct;
    }
    else{
        p[i].st=p[i].at;
    }
    p[i].et=p[i].st+p[i].bt;
    ct+=p[i].bt;

    p[i].tat=p[i].et-p[i].at;
    p[i].wt=p[i].tat-p[i].bt;
    p[i].rt=p[i].st-p[i].at;

    printf("%d\t\t%d\t\t%d\t\t%d\n",p[i].id,p[i].wt,p[i].tat,p[i].rt);
    awt+=p[i].wt;
    atat+=p[i].tat;
    art+=p[i].rt;
}

tp=(float)p[n-1].et/n;
```

```
    printf("Average Waiting Time: %.2f\n",awt/n);
    printf("Average Turn Around Time: %.2f\n",atat/n);
    printf("Average Response Time: %.2f\n",art/n);
    printf("Throughput: %.2f",tp);

}
```

Output:

a) **Pre-emptive**

```
Enter number of processes: 5
Enter Burst Time, Arrival Time and Priority of P1: 3 0 5
Enter Burst Time, Arrival Time and Priority of P2: 2 2 3
Enter Burst Time, Arrival Time and Priority of P3: 5 3 2
Enter Burst Time, Arrival Time and Priority of P4: 4 4 4
Enter Burst Time, Arrival Time and Priority of P5: 1 6 1
Process Waiting Time    Turn Around Time        Response Time
1               12              15              0
2               6               8               0
3               1               6               0
4               6               10              6
5               0               1               0
Average Waiting Time: 5.00
Average Turn Around Time: 8.00
Average Response Time: 1.20
Throughput: 3.00
```

a) **Non- Pre-emptive**

```
Enter number of processes: 5
Enter Burst Time, Arrival Time and Priority of P1: 3 0 5
Enter Burst Time, Arrival Time and Priority of P2: 2 2 3
Enter Burst Time, Arrival Time and Priority of P3: 5 3 2
Enter Burst Time, Arrival Time and Priority of P4: 4 4 4
Enter Burst Time, Arrival Time and Priority of P5: 1 6 1

Process Waiting Time    Turn Around Time        Response Time
1               0               3               0
3               0               5               0
5               2               3               2
2               7               9               7
4               7               11              7
Average Waiting Time: 3.20
Average Turn Around Time: 6.20
Average Response Time: 3.20
Throughput: 3.00
```

Lab Program - 4

Question:

Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

Code:

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

struct P{
    int id;
    int bt;
    int at;
    int q;
    int wt;
    int tat;
    int rt;
    int st;
    int et;
    int v;
};

void main(){
    int n,ct=0;
    float awt=0,atat=0,art=0,tp;
    printf("Queue 1 is system process\nQueue 2 is User Process\n");
    printf("Enter number of processes: ");
    scanf("%d",&n);
    struct P p[n];
    struct P temp;
```

```
for(int i=0;i<n;i++){
   p[i].id=i+1;
   p[i].v=0;
   printf("Enter Burst Time, Arrival Time and Queue of P%d: ",i+1);
   scanf("%d %d %d",&p[i].bt,&p[i].at,&p[i].q);
}

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

int cf=p[0].bt, min=INT_MAX,m,count=0;
p[0].v=1;

while(count<n-1){
for(int i=0;i<n;i++){
   if(p[i].at<=cf && p[i].v==0){
      if(p[i].q<min){
         min=p[i].q;
```

```
            m=i;
        }
    }
}
p[m].v=1;
cf+=p[m].bt;
min=INT_MAX;

temp=p[count+1];
p[count+1]=p[m];
p[m]=temp;

count++;
}

printf("\nProcess\tWaiting Time\tTurn Around Time\tResponse Time\n");

for(int i=0;i<n;i++){
    if(p[i].at<ct){
        p[i].st=ct;
    }
    else{
        p[i].st=p[i].at;
    }
    p[i].et=p[i].st+p[i].bt;
    ct+=p[i].bt;

    p[i].tat=p[i].et-p[i].at;
    p[i].wt=p[i].tat-p[i].bt;
    p[i].rt=p[i].st-p[i].at;

    printf("%d\t\t%d\t\t%d\t\t%d\n",p[i].id,p[i].wt,p[i].tat,p[i].rt);
```

```
        awt+=p[i].wt;
        atat+=p[i].tat;
        art+=p[i].rt;
    }

    tp=(float)p[n-1].et/n;
    printf("Average Waiting Time: %.2f\n",awt/n);
    printf("Average Turn Around Time: %.2f\n",atat/n);
    printf("Average Response Time: %.2f\n",art/n);
    printf("Throughput: %.2f",tp);

}
```

Output:

```
Queue 1 is system process
Queue 2 is User Process
Enter number of processes: 5
Enter Burst Time, Arrival Time and Queue of P1: 4 0 1
Enter Burst Time, Arrival Time and Queue of P2: 2 1 2
Enter Burst Time, Arrival Time and Queue of P3: 3 2 2
Enter Burst Time, Arrival Time and Queue of P4: 2 2 1
Enter Burst Time, Arrival Time and Queue of P5: 5 8 1

Process Waiting Time    Turn Around Time        Response Time
1               0               4               0
4               2               4               2
2               5               7               5
5               0               5               0
3               11              14              11
Average Waiting Time: 3.60
Average Turn Around Time: 6.80
Average Response Time: 3.60
Throughput: 3.20
```

Lab Program - 5

Question:

Write a C program to simulate Real Time CPU Scheduling Algorithms:

a) Rate- Monotonic

b) Earliest Deadline First

c) Proportional Scheduling

Code:

a) Rate-Monotonic

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

struct P{
   int id;
   float et;
   int tp;
   int v;
   int b;
};
int gcd(int a, int b){
   if (b == 0)
      return a;
   return gcd(b, a % b);
}
int findlcm(int arr[], int n)
{
   int ans = arr[0];

   for (int i = 1; i < n; i++)
      ans = (((arr[i] * ans)) /(gcd(arr[i], ans)));

   return ans;
}
```

```
void main(){
    int n,ct=0,f=0;
    float awt=0,atat=0,art=0,tp;
    printf("Enter number of processes: ");
    scanf("%d",&n);
    struct P p[n];
    struct P temp;
    int a[n];
    for(int i=0;i<n;i++){
        p[i].id=i+1;
        p[i].v=0;
        printf("Enter Excecution Time and Time Period of P%d: ",i+1);
        scanf("%f %d",&p[i].et,&p[i].tp);
        p[i].b=p[i].et;
        a[i]=p[i].tp;
    }

    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            if(p[i].tp>p[j].tp){
                temp=p[i];
                p[i]=p[j];
                p[j]=temp;
            }
        }
    }
    int ans=findlcm(a,n);

    for(int i=0;i<ans;i++){
        f=0;
        for(int j = 0;j<n;j++){
            if (i%p[j].tp==0) {
```

```
            p[j].v=0;
            p[j].et=p[j].b;
        }
    }
        for(int j=0;j<n;j++){
            if(p[j].v==0){
                f=1;
                p[j].et-=1;
                printf("%d to %d P%d\n",i,i+1,p[j].id);
                if(p[j].et==0){
                    p[j].v=1;
                }
                break;
            }
        }

        if(f==0){
            printf("%d to %d -\n",i,i+1);
        }
    }
}
```

b) Earliest Deadline First

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

#define MAX_TSKS 10

typedef struct {
    int p;
    int c;
    int d;
    int rt;
```

```
    int nd;
    int id;
} Task;

void Input(Task tsks[], int *n_tsk) {
    printf("Enter number of tasks (max %d): ", MAX_TSKS);
    scanf("%d", n_tsk);

    if (*n_tsk > MAX_TSKS) {
        printf("Number of tasks exceeds the maximum limit of %d.\n", MAX_TSKS);
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < *n_tsk; i++) {
        tsks[i].id = i + 1;
        printf("Enter period (p) of task %d: ", i + 1);
        scanf("%d", &tsks[i].p);
        printf("Enter execution time (c) of task %d: ", i + 1);
        scanf("%d", &tsks[i].c);
        printf("Enter deadline (d) of task %d: ", i + 1);
        scanf("%d", &tsks[i].d);

        tsks[i].rt = tsks[i].c;
        tsks[i].nd = tsks[i].d;
    }
}

void EDF(Task tsks[], int n_tsk, int tf) {
    printf("\nEarliest-Deadline First Scheduling:\n");
    for (int t = 0; t < tf; t++) {
        int s_tsk = -1;
```

```
        for (int i = 0; i < n_tsk; i++) {
            if (t % tsks[i].p == 0) {
                tsks[i].rt = tsks[i].c;
                tsks[i].nd = t + tsks[i].d;
            }
        }

        for (int i = 0; i < n_tsk; i++) {
            if (tsks[i].rt > 0 && (s_tsk == -1 || tsks[i].nd < tsks[s_tsk].nd)) {
                s_tsk = i;
            }
        }

        if (s_tsk != -1) {
            printf("Time %d: Task %d\n", t, tsks[s_tsk].id);
            tsks[s_tsk].rt--;
        } else {
            printf("Time %d: Idle\n", t);
        }
    }
}

int main() {
    Task tsks[MAX_TSKS];
    int n_tsk;
    int tf;

    Input(tsks, &n_tsk);

    printf("Enter time frame for simulation: ");
    scanf("%d", &tf);
```

```
    EDF(tsks, n_tsk, tf);

    return 0;
}
```

c) Proportional Scheduling

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

int main() {
    int n, sOT = 0;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int pid[n];
    int l[n + 1];
    l[0] = 0;

    printf("\nEnter the number of tickets for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("PID%d: ", i + 1);
        scanf("%d", &pid[i]);
        sOT += pid[i];
        l[i + 1] = pid[i];
    }

    int t = 1;
    int sum = sOT;
    for (int i = 0; i < n; i++) {
        printf("Probability of servicing process %d is %d%%\n", i + 1, (pid[i] * 100) / sOT);
    }
```

```
    srand(time(NULL));
    while (sum > 0) {
        int x = rand() % sOT;
        int j;
        for (j = 0; j < n; j++) {
            if (x < l[j + 1]) {
                printf("%d ms: Servicing Ticket of process %d\n", t, j + 1);
                //l[j + 1]--;
                pid[j]--;
                sum--;
                t++;
                break;
            }
        }
    }

    for (int i = 0; i < n; i++) {
        if (pid[i] == 0) {
            printf("PID%d has finished executing\n", i + 1);
        }
    }
    return 0;
}
```

Output:

a) Rate Monotonic

```
Enter number of processes: 3
Enter Excecution Time and Time Period of P1: 3 20
Enter Excecution Time and Time Period of P2: 10 5
Enter Excecution Time and Time Period of P3: 20 2
0 to 1 P3
1 to 2 P3
2 to 3 P3
3 to 4 P3
4 to 5 P3
5 to 6 P3
6 to 7 P3
7 to 8 P3
8 to 9 P3
9 to 10 P3
10 to 11 P3
11 to 12 P3
12 to 13 P3
13 to 14 P3
14 to 15 P3
15 to 16 P3
16 to 17 P3
17 to 18 P3
18 to 19 P3
19 to 20 P3
```

b) Earliest Deadline First

```
Enter number of tasks (max 10): 3
Enter period (p) of task 1: 20
Enter execution time (c) of task 1: 3
Enter deadline (d) of task 1: 7
Enter period (p) of task 2: 5
Enter execution time (c) of task 2: 2
Enter deadline (d) of task 2: 4
Enter period (p) of task 3: 10
Enter execution time (c) of task 3: 2
Enter deadline (d) of task 3: 8
Enter time frame for simulation: 20

Earliest-Deadline First Scheduling:
Time 0: Task 2
Time 1: Task 2
Time 2: Task 1
Time 3: Task 1
Time 4: Task 1
Time 5: Task 3
Time 6: Task 3
Time 7: Task 2
Time 8: Task 2
Time 9: Idle
Time 10: Task 2
Time 11: Task 2
Time 12: Task 3
Time 13: Task 3
Time 14: Idle
Time 15: Task 2
Time 16: Task 2
Time 17: Idle
Time 18: Idle
Time 19: Idle
```

c) Proportional Scheduling

```
Enter the number of processes: 2

Enter the number of tickets for each process:
PID1: 3
PID2: 2
Probability of servicing process 1 is 60%
Probability of servicing process 2 is 40%
1 ms: Servicing Ticket of process 1
2 ms: Servicing Ticket of process 1
3 ms: Servicing Ticket of process 1
4 ms: Servicing Ticket of process 1
5 ms: Servicing Ticket of process 1
```

Lab Program - 6

Question:

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

Code:

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

#define BUF_SIZE 10
#define MAX_ITMS 20

int buf[BUF_SIZE];
int cnt = 0;
int in = 0;
int out = 0;
int prod_cnt = 0;
int cons_cnt = 0;

pthread_mutex_t mtx;
pthread_cond_t cond_prod;
pthread_cond_t cond_cons;

void* prod(void* param) {
   while (1) {
      int item = rand() % 100;

      pthread_mutex_lock(&mtx);

      if (prod_cnt >= MAX_ITMS) {
```

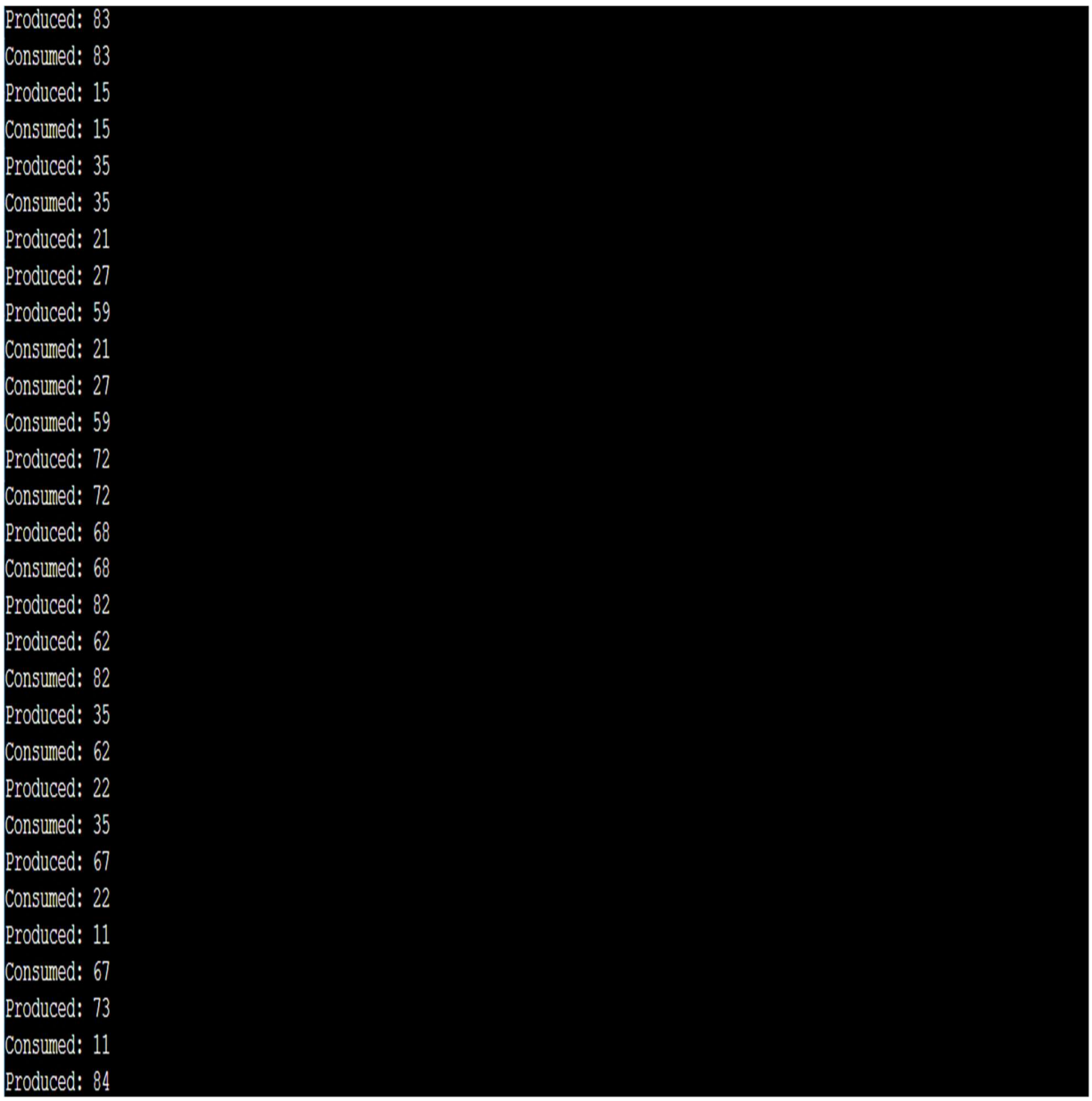
```
            pthread_mutex_unlock(&mtx);
            break;
        }

        while (cnt == BUF_SIZE) {
            pthread_cond_wait(&cond_prod, &mtx);
        }

        buf[in] = item;
        in = (in + 1) % BUF_SIZE;
        cnt++;
        prod_cnt++;
        printf("Produced: %d\n", item);

        pthread_cond_signal(&cond_cons);
        pthread_mutex_unlock(&mtx);

        sleep(rand() % 2);
    }
    return NULL;
}

void* cons(void* param) {
    while (1) {
        pthread_mutex_lock(&mtx);

        if (cons_cnt >= MAX_ITMS) {
            pthread_mutex_unlock(&mtx);
            break;
        }

        while (cnt == 0) {
```

```
            pthread_cond_wait(&cond_cons, &mtx);
        }

        int item = buf[out];
        out = (out + 1) % BUF_SIZE;
        cnt--;
        cons_cnt++;
        printf("Consumed: %d\n", item);

        pthread_cond_signal(&cond_prod);
        pthread_mutex_unlock(&mtx);

        sleep(rand() % 2);
    }
    return NULL;
}

int main() {
    pthread_t tid_prod, tid_cons;

    pthread_mutex_init(&mtx, NULL);
    pthread_cond_init(&cond_prod, NULL);
    pthread_cond_init(&cond_cons, NULL);

    pthread_create(&tid_prod, NULL, prod, NULL);
    pthread_create(&tid_cons, NULL, cons, NULL);

    pthread_join(tid_prod, NULL);
    pthread_join(tid_cons, NULL);

    printf("Production & Consumption complete\n");
```

```
    pthread_mutex_destroy(&mtx);
    pthread_cond_destroy(&cond_prod);
    pthread_cond_destroy(&cond_cons);

    return 0;
}
```

Output:

Producer-Consumer



Lab Program - 7

Question:

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

Code:

```
#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[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]);
    }
}

// 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:

Dining-Philosopher

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

Lab Program - 8

Question:

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

Code:

```
// Banker's Algorithm
#include <stdio.h>
int main()
{
   // P0, P1, P2, P3, P4 are the Process names here

   int n, m, i, j, k;
   n = 5; // Number of processes
   m = 3; // Number of resources
   int alloc[5][3] = { { 0, 1, 0 }, // P0    // Allocation Matrix
                       { 2, 0, 0 }, // P1
                       { 3, 0, 2 }, // P2
                       { 2, 1, 1 }, // P3
                       { 0, 0, 2 } }; // P4

   int max[5][3] = { { 7, 5, 3 }, // P0    // MAX Matrix
                     { 3, 2, 2 }, // P1
                     { 9, 0, 2 }, // P2
                     { 2, 2, 2 }, // P3
                     { 4, 3, 3 } }; // P4

   int avail[3] = { 3, 3, 2 }; // Available Resources

   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 < 5; 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);

    // This code is contributed by Deep Baldha (CandyZack)
}
```

Output:

Bankers Algorithm (Deadlock Avoidance and Detection)

```
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2
```

Lab Program - 9

Question:

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

a) Worst-fit

b) Best-fit

c) First-fit

Code:

```
#include<stdio.h>
#include<limits.h>
struct hole{
   int size;
   int v;
}hole;

struct p{
   int size;
   int d;
}p;

void main(){
   int n1,n2,n,m=-1;

   printf("Enter number of holes: \n");
   scanf("%d",&n1);
   struct hole h[n1];
   printf("Enter hole size: \n");
```

```
for(int i=0;i<n1;i++){
   scanf("%d",&h[i].size);
   h[i].v=-1;
}

printf("Enter number of processes: \n");
scanf("%d",&n2);
struct p p[n2];
printf("Enter processes size: \n");
for(int i=0;i<n2;i++){
   scanf("%d",&p[i]);
   p[i].d=-1;
}

for(int i=0;i<n2;i++){
   for(int j=0;j<n1;j++){
      if(p[i].size<=h[j].size && h[j].v==-1){
         p[i].d=1;
         h[j].v=i;
         break;
      }
   }
}
printf("\nFirst fit \n");
for(int i=0;i<n1;i++){
      if(h[i].v==-1){
         printf(" -- ");
```

```
        }
        else{
        printf(" P%d ",h[i].v+1);
        }
}
printf("\n");
for(int i=0;i<n2;i++){
    if(p[i].d==-1){
        printf("Process P%d(%d) is unfinished\n",(i+1),p[i].size);
    }
}

for(int i=0;i<n1;i++){
    h[i].v=-1;
}
for(int i=0;i<n2;i++){
    p[i].d=-1;
}
printf("\nBest fit \n");
for(int i=0;i<n2;i++){
    n=INT_MAX;
    for(int j=0;j<n1;j++){
        if(p[i].size<=h[j].size && h[j].v==-1 && h[j].size<n){
            m=j;
            n=h[j].size;
        }
    }
```

```
    if(n!=INT_MAX){
        p[i].d=1;
        h[m].v=i;
    }
}

for(int i=0;i<n1;i++){
        if(h[i].v==-1){
            printf(" -- ");
        }
        else{
        printf(" P%d ",h[i].v+1);
        }
}
printf("\n");
for(int i=0;i<n2;i++){
    if(p[i].d==-1){
        printf("Process P%d(%d) is unfinished\n",(i+1),p[i].size);
    }
}

 for(int i=0;i<n1;i++){
    h[i].v=-1;
}
for(int i=0;i<n2;i++){
    p[i].d=-1;
}
```

```
printf("\nWorst fit \n");
for(int i=0;i<n2;i++){
   n=-1;
   for(int j=0;j<n1;j++){
      if(h[j].v==-1 && h[j].size>n){
         m=j;
         n=h[j].size;
      }
   }
   if(n!=-1 && p[i].size<=h[m].size){
      p[i].d=1;
      h[m].v=i;
   }
}

for(int i=0;i<n1;i++){
      if(h[i].v==-1){
         printf(" -- ");
      }
      else{
      printf(" P%d ",h[i].v+1);
      }
}
printf("\n");
for(int i=0;i<n2;i++){
   if(p[i].d==-1){
      printf("Process P%d(%d) is unfinished\n",(i+1),p[i].size);
```

```
    }
  }
}
```

Output:

Memory Allocation (First Fit, Best Fit, Worst Fit)

```
Enter number of holes:
5
Enter hole size:
100 500 200 300 600
Enter number of processes:
4
Enter processes size:
212 417 112 476

First fit
 --  P1  P3  --  P2
Process P4(476) is unfinished

Best fit
 --  P2  P3  P1  P4

Worst fit
 --  P2  --  P3  P1
Process P4(476) is unfinished
```

Lab Program - 10

Question:

Please execute the page Replacement Algorithms: FIFO, OPTIMAL and LRU

Code:

```
#include <stdio.h>


// Function to check if the page is present in the frames
int isPagePresent(int frames[], int n, int page) {
    for (int i = 0; i < n; i++) {
        if (frames[i] == page) {
            return 1;
        }
    }
    return 0;
}


// Function to print the frames
void printFrames(int frames[], int n) {
    for (int i = 0; i < n; i++) {
        if (frames[i] != -1) {
            printf("%d ", frames[i]);
        } else {
            printf("- ");
        }
    }
    printf("\n");
```

```
}

// Function to implement FIFO page replacement
void fifoPageReplacement(int pages[], int numPages, int numFrames) {
    int frames[numFrames];
    int front = 0, pageFaults = 0;

    // Initialize frames
    for (int i = 0; i < numFrames; i++) {
        frames[i] = -1;
    }

    printf("FIFO Replacement\n");
    printf("Reference String\tFrames\n");
    for (int i = 0; i < numPages; i++) {
        printf("%d\t\t", pages[i]);

        if (!isPagePresent(frames, numFrames, pages[i])) {
            frames[front] = pages[i];
            front = (front + 1) % numFrames;
            pageFaults++;
        }
        printFrames(frames, numFrames);
    }

    printf("\nTotal Page Faults: %d\n\n", pageFaults);
}
```

```
// Function to find the page to replace using the Optimal page replacement algorithm
int findOptimalReplacementIndex(int pages[], int numPages, int frames[], int numFrames, int currentIndex) {
    int farthest = currentIndex;
    int index = -1;

    for (int i = 0; i < numFrames; i++) {
        int j;
        for (j = currentIndex; j < numPages; j++) {
            if (frames[i] == pages[j]) {
                if (j > farthest) {
                    farthest = j;
                    index = i;
                }
                break;
            }
        }
        // If the page is not found in future, return this index
        if (j == numPages) {
            return i;
        }
    }

    // If all pages are found in future, return the one with farthest future use
    return (index == -1) ? 0 : index;
}
```

```
// Function to implement Optimal page replacement
void optPageReplacement(int pages[], int numPages, int numFrames) {
    int frames[numFrames];
    int pageFaults = 0;

    // Initialize frames
    for (int i = 0; i < numFrames; i++) {
        frames[i] = -1;
    }

    printf("Optimal Replacement\n");
    printf("Reference String\tFrames\n");
    for (int i = 0; i < numPages; i++) {
        printf("%d\t\t", pages[i]);

        if (!isPagePresent(frames, numFrames, pages[i])) {
            if (isPagePresent(frames, numFrames, -1)) {
                for (int j = 0; j < numFrames; j++) {
                    if (frames[j] == -1) {
                        frames[j] = pages[i];
                        break;
                    }
                }
            } else {
                int index = findOptimalReplacementIndex(pages, numPages, frames, numFrames, i +
1);
```

```
                frames[index] = pages[i];
            }
            pageFaults++;
        }
        printFrames(frames, numFrames);
    }

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

// Function to implement LRU page replacement
void lruPageReplacement(int pages[], int numPages, int numFrames) {
    int frames[numFrames];
    int pageFaults = 0;
    int timestamps[numFrames];

    // Initialize frames and timestamps
    for (int i = 0; i < numFrames; i++) {
        frames[i] = -1;
        timestamps[i] = -1;
    }

    printf("LRU Replacement\n");
    printf("Reference String\tFrames\n");
    for (int i = 0; i < numPages; i++) {
        printf("%d\t\t", pages[i]);
```

```
        if (!isPagePresent(frames, numFrames, pages[i])) {
            int lruIndex = 0;
            for (int j = 1; j < numFrames; j++) {
                if (timestamps[j] < timestamps[lruIndex]) {
                    lruIndex = j;
                }
            }
            frames[lruIndex] = pages[i];
            timestamps[lruIndex] = i;
            pageFaults++;
        } else {
            for (int j = 0; j < numFrames; j++) {
                if (frames[j] == pages[i]) {
                    timestamps[j] = i;
                    break;
                }
            }
        }
        printFrames(frames, numFrames);
    }

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

int main() {
    int numFrames, numPages;
```

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

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

    int pages[numPages];

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

    fifoPageReplacement(pages, numPages, numFrames);
    optPageReplacement(pages, numPages, numFrames);
    lruPageReplacement(pages, numPages, numFrames);

    return 0;
}
```

Output:

Page Replacement (FIFO, OPTIMAL, LRU)

```
Enter the number of frames: 4
Enter the number of pages: 14
Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 3
FIFO Replacement
Reference String       Frames
7              7 - - -
0              7 0 - -
1              7 0 1 -
2              7 0 1 2
0              7 0 1 2
3              3 0 1 2
0              3 0 1 2
4              3 4 1 2
2              3 4 1 2
3              3 4 1 2
0              3 4 0 2
3              3 4 0 2
2              3 4 0 2
3              3 4 0 2

Total Page Faults: 7

Optimal Replacement
Reference String       Frames
7              7 - - -
0              7 0 - -
1              7 0 1 -
2              7 0 1 2
0              7 0 1 2
3              3 0 1 2
0              3 0 1 2
4              3 0 4 2
2              3 0 4 2
3              3 0 4 2
0              3 0 4 2
3              3 0 4 2
2              3 0 4 2
3              3 0 4 2

Total Page Faults: 6

LRU Replacement
Reference String       Frames
7              7 - - -
0              7 0 - -
1              7 0 1 -
2              7 0 1 2
0              7 0 1 2
3              3 0 1 2
0              3 0 1 2
4              3 0 4 2
2              3 0 4 2
3              3 0 4 2
0              3 0 4 2
3              3 0 4 2
2              3 0 4 2
3              3 0 4 2

Total Page Faults: 6
```