VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

OPERATING SYSTEMS

Submitted by

PANNAGA R BHAT (1BM22CS189)

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 **Pannaga R Bhat (1BM22CS189),** 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. Asha G.R.
Associate Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Course Outcome

C01	Apply the different concepts and functionalities of Operating System
C02	Analyze 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

Program -1

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

→FCFS
→ SJF (pre-emptive & Non-preemptive)

Code: First Come First Serve

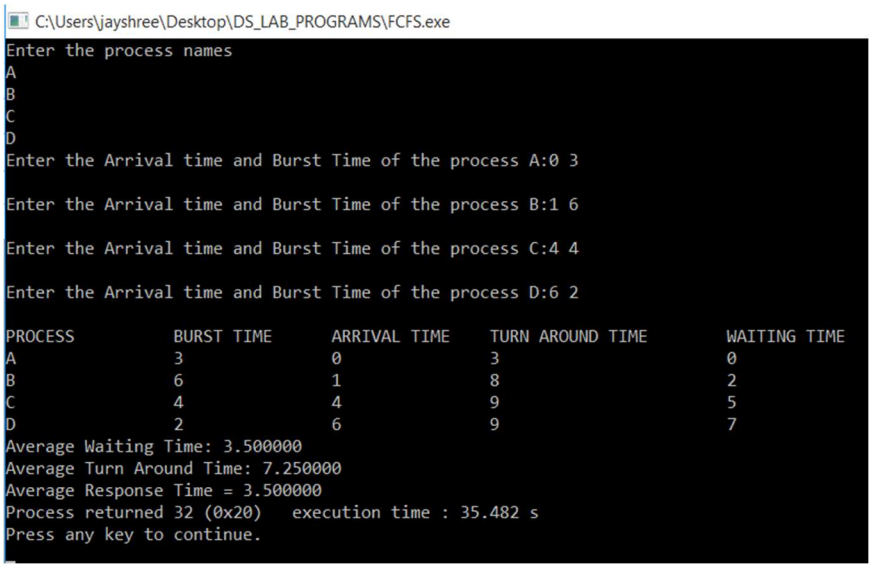
```
#include<stdio.h>
# define n 4
struct gt
{
   char p[2];
   int a;
   int bt;
   int c;
   int wt;
   int tat;
   int start_time;
```

```
    int end_time
}ob[n];
void main()
{
    printf("Enter the process names\n");
    for(int i=0;i<n;i++)
        scanf("%s",&ob[i].p);
    for(int i=0;i<n;i++)
    {
        printf("Enter the Arrival time and Burst Time of the process %s:",ob[i].p);
        scanf("%d %d",&ob[i].a, &ob[i].bt);
        printf("\n");
    }
    // Sorted arrival time
    for(int i=0;i<n;i++)
    {
        for(int j = i+1;j<n;j++)
        {
            if(ob[i].a > ob[j].a)
            {
                struct gt temp = ob[i];
                ob[i] = ob[j];
                ob[j] = temp;
            }
        }
    }
```

```
// Gantt Chart with start time and end time
int s = 0;
for(int i=0;i<n;i++)
{
   ob[i].start_time = (ob[i].a > s) ? ob[i].a : s;
   ob[i].end_time = (ob[i].start_time) + ob[i].bt;
   s = ob[i].end_time;
}
// Calculation
float s_tat = 0;
float s_wt = 0;
for(int i=0;i<n;i++)
{
   ob[i].tat = ob[i].end_time - ob[i].a;
}
for(int i=0;i<n;i++)
{
   ob[i].wt = ob[i].tat - ob[i].bt;
}
for(int i=0;i<n;i++)
{
   s_tat += ob[i].tat;
   s_wt += ob[i].wt;
}
float avg_tat = s_tat / (float)n;
float avg_wt = s_wt / (float)n;
```

```
printf("PROCESS \t BURST TIME \t ARRIVAL TIME \t TURN AROUND TIME \t WAITING TIME \n");
   for(int i = 0;i<n;i++)
   {

      printf("%s\t\t %d \t\t %d \t\t %d \t\t\t %d \n", ob[i].p, ob[i].bt, ob[i].a, ob[i].tat, ob[i].wt);
   }

printf("Average Waiting Time: %f\n",avg_wt);
printf("Average Turn Around Time: %f\n",avg_tat);

float s_rt = 0;
for(int i=0;i<n;i++)
{
   s_rt = s_rt +(ob[i].start_time - ob[i].a);
}
printf("Average Response Time = %f", (s_rt/n));

}
```

Result:



Code: Shortest Job First (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(){
```

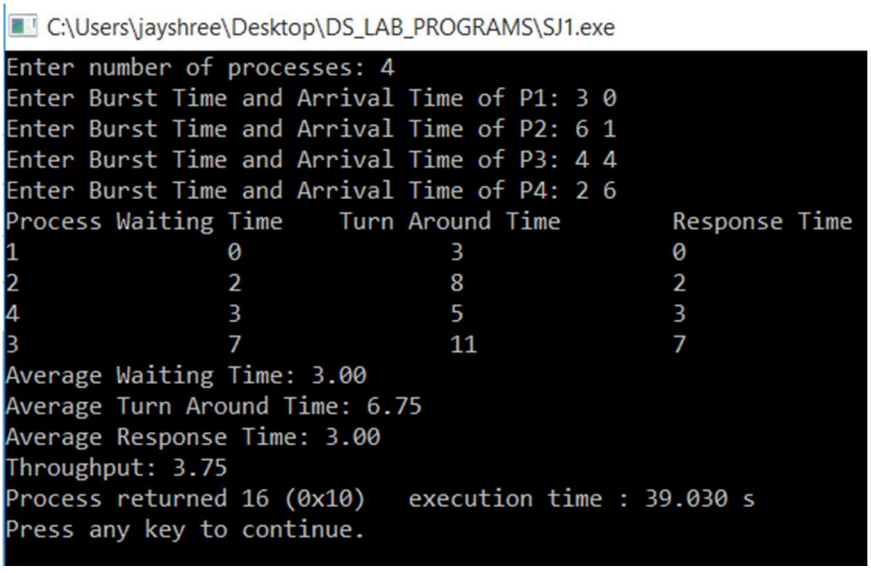
```
int n,ct=0;
float awt=0,atat=0,art=0,tp;
printf("Enter number of processes: ");
scanf("%d",&n);

struct P p[n];
struct P temp;

for(int i=0;i<n;i++){
    p[i].id=i+1;

    // Visited array = 0
    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);
}

// Sorting based on arrival time, if multiple process arrives at same time, sort based on Burst Time.
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;
        }
        }
    }
}

// Catching first process's Burst Time
int cf=p[0].bt, min=INT_MAX, m, count=0;
p[0].v=1;

// iterating through remaining processes
while(count < n-1){
for(int i=0;i<n;i++){

    // seeing if other process are arrived before the Burst Time and not visited processes
    if(p[i].at<=cf && p[i].v==0){

        // Catching minimum Burst Time and it's index.
        if(p[i].bt<min){
            min=p[i].bt;
            m=i;
        }
    }
}
// making the catched process as visited
```

```
p[m].v=1;
// adding burst time to end time of the process
cf+=p[m].bt;
// re-initializing
min=INT_MAX;
// Swapping between the processes, by next process
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);
}
```

Result:



Code: Shortest Job First (Pre-emptive)

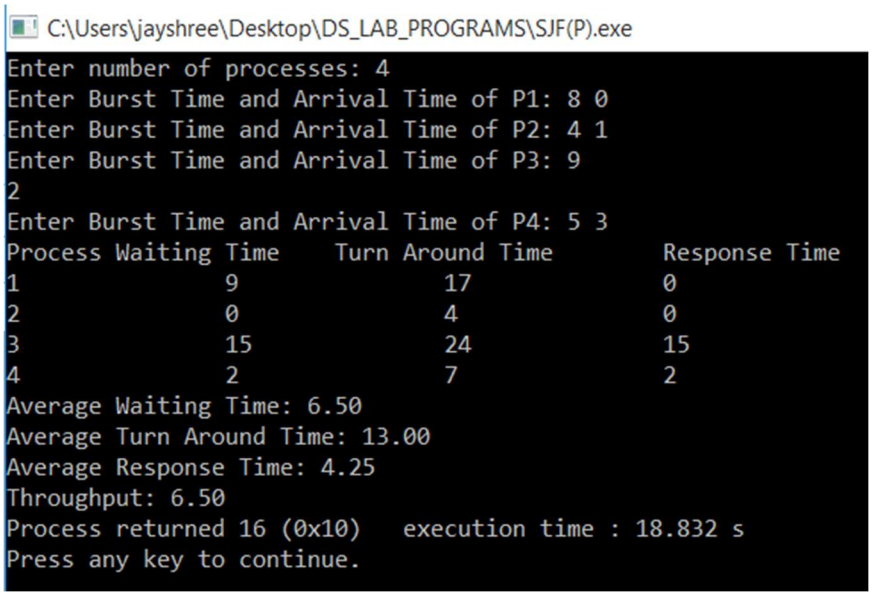
```
#include <stdio.h>
```

```
#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);
}
```

Result:



Program - 2

Question: Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time. → Priority (pre-emptive & Non-pre-emptive) →Round Robin (Experiment with different quantum sizes for RR algorithm)

Code: Priority Scheduling (pre-emptive)

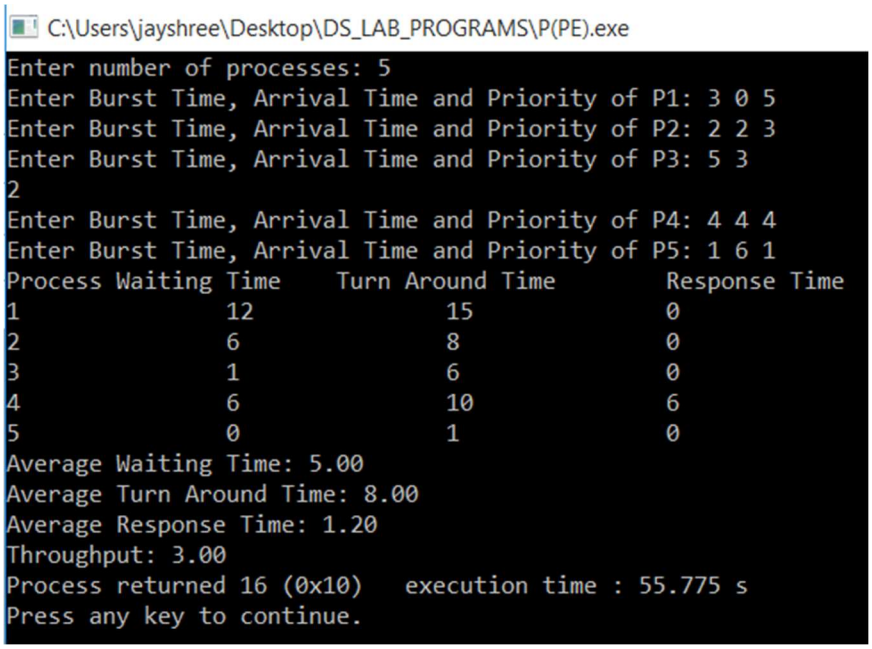
```
#include <stdio.h>

#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);

}
```

Result:



Code: Priority Scheduling (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;
```

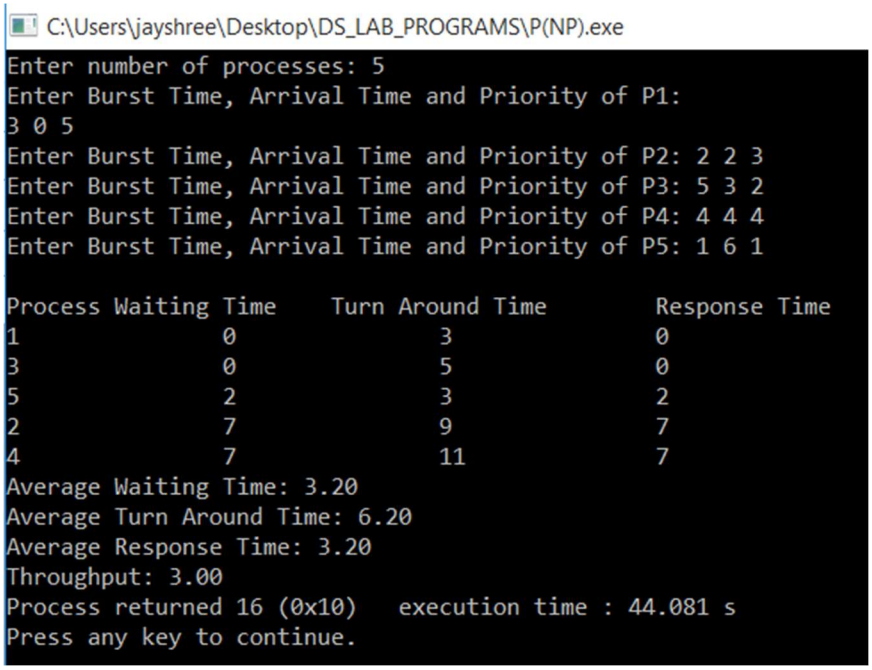
```
    int v;
};

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);
}
```

Result:



Code: Round Robin (Experiment with different quantum sizes for RR algorithm)

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

#define MAX_PROCESSES 10
```

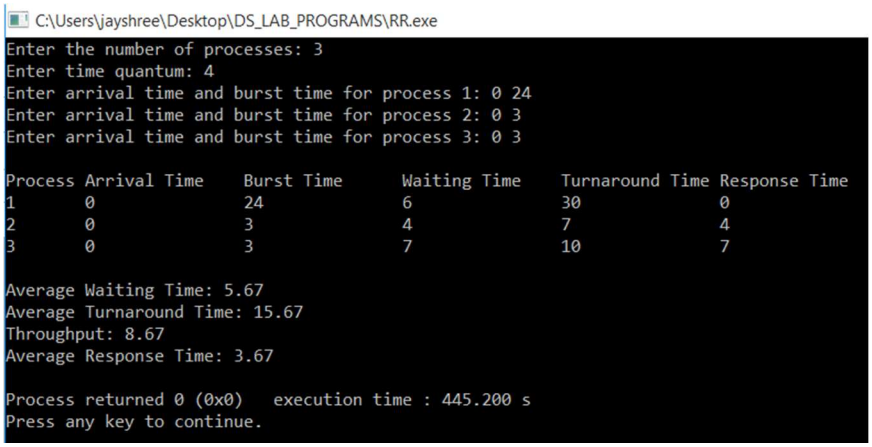
```
struct Process {
    int id;
    int at;
    int bt;
    int wt;
    int tat;
    int st;
    int et;
    int rt;
    int vi;
    int obt;
};

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;
}
```

Result:



Program - 3

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

Code:

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

struct P{
```

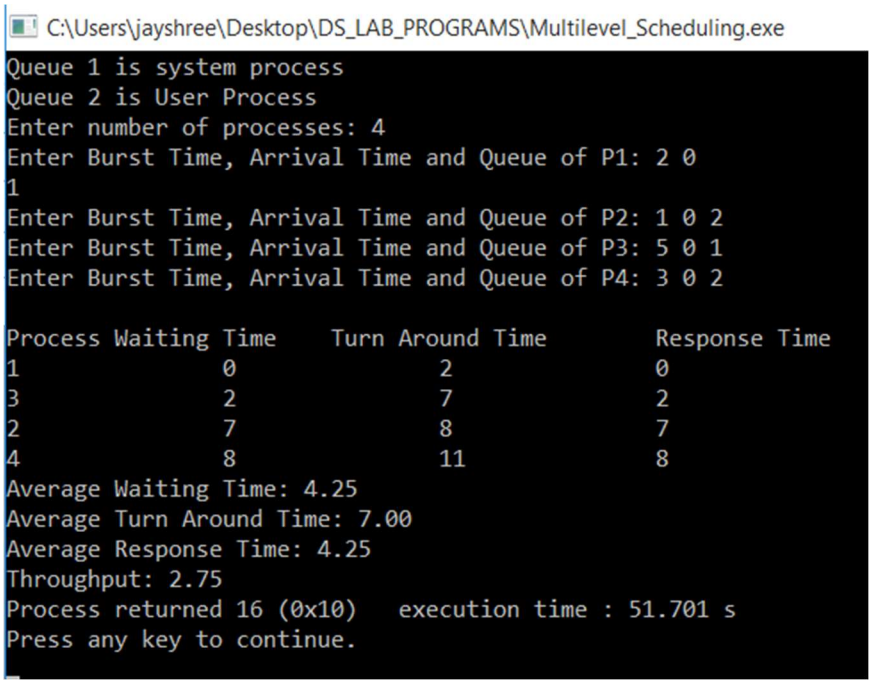
```
    int id;
    int bt;
    int at;
    int q;
    int wt;
    int tat;
    int rt;
    int st;
    int et;
    int v;
};
void main(){
    int n,ct=0;
    float awt=0,atat=0,art=0,tp;
    printf("Queue 1 is system process\nQueue 2 is User Process\n");
    printf("Enter number of processes: ");
    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);
}
```

Result:



Program - 4

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

a) Rate- Monotonic b) Earliest-deadline First c) Proportional scheduling

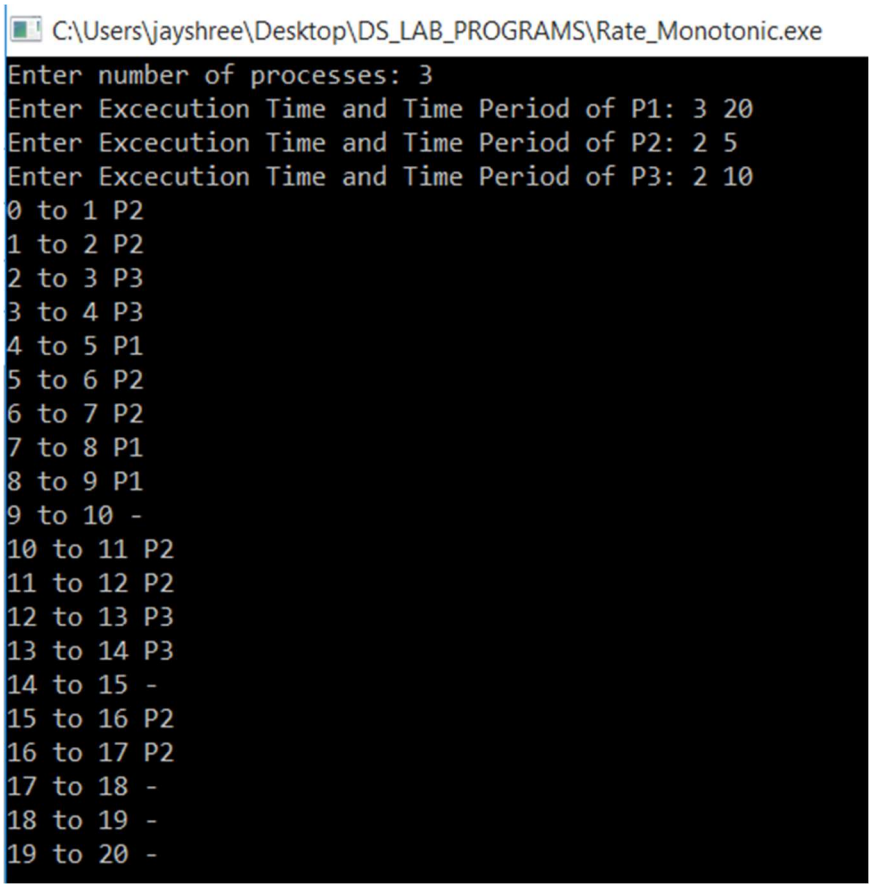
Code: a) Rate- Monotonic

```
#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);
        }
    }
}
```

Result:



Code: b) Earliest-deadline First

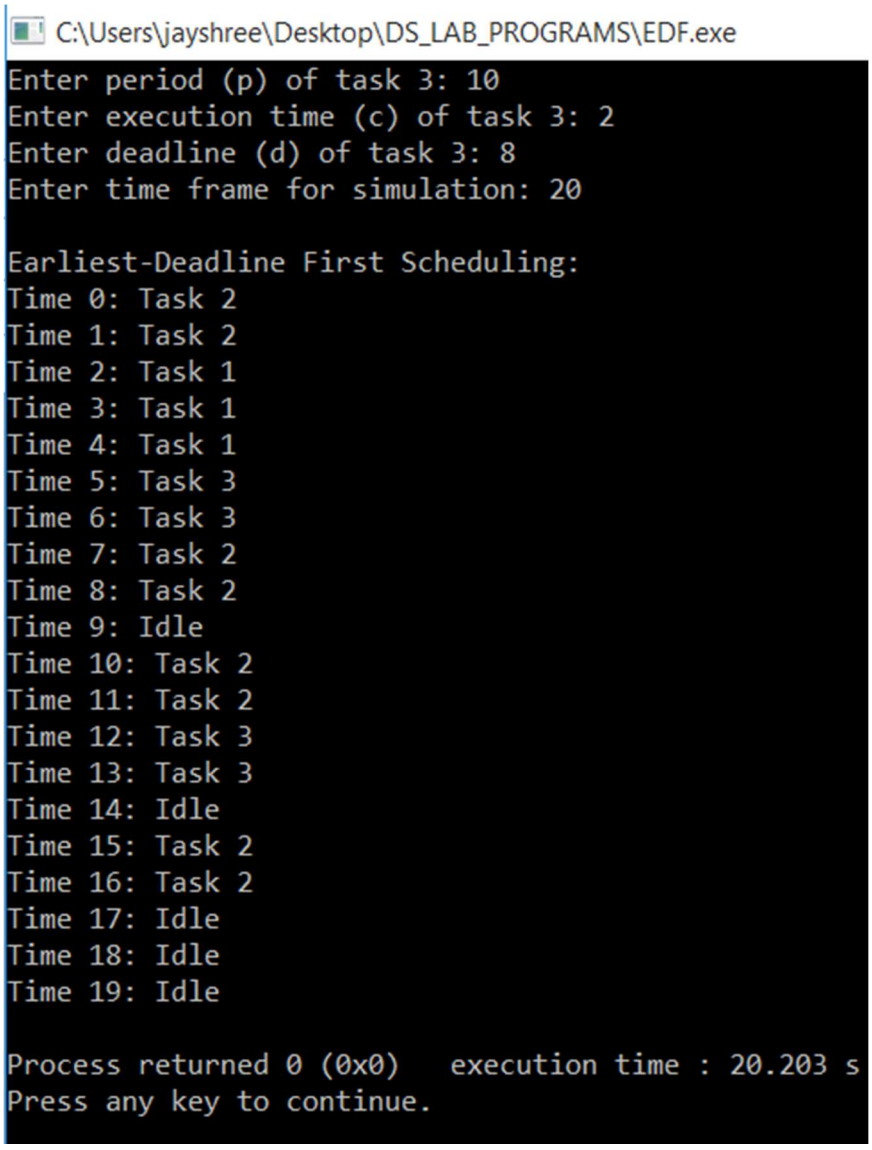
```
#include <stdio.h>

#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;
}
```

Result:

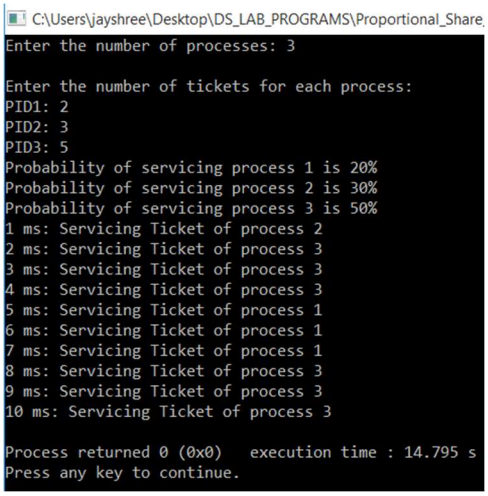


Code: c) Proportional scheduling

```
#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;
}
```

Result:



Program - 5

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

Code:

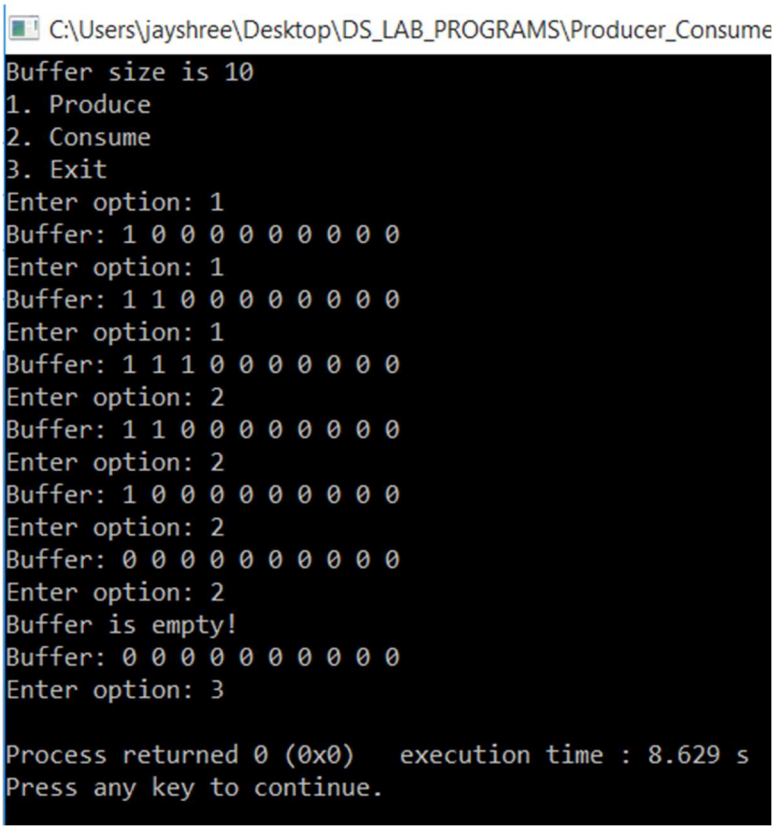
```
#include <stdio.h>


#define MAX 10


int sharedMemory[MAX];

int top = -1;

int mutex = 1;

int empty = MAX;

int full = 0;


void wait(int *s){

   (*s)--;
```

```
}

void signal(int *s){
   (*s)++;
}

void producer(int A[], int *top, int *m, int *e, int *f){
   if(*e != 0){
      wait(m);
      A[++(*top)] = 1; // Produce an item (here represented by 1)
      wait(e);
      signal(f);
      signal(m);
   } else {
      printf("Buffer is full!\n");
   }
}

void consumer(int A[], int *top, int *m, int *e, int *f){
   if(*f != 0){
      wait(m);
      A[(*top)--] = 0; // Consume an item (here represented by setting it to 0)
      signal(e);
      wait(f);
      signal(m);
   } else {
      printf("Buffer is empty!\n");
```

```
    }
}

void printBuffer(int A[], int top) {
    printf("Buffer: ");
    for(int i = 0; i < MAX; i++) {
        if(i <= top) {
            printf("%d ", A[i]);
        } else {
            printf("0 ");
        }
    }
    printf("\n");
}

int main(){
    int optn;
    printf("Buffer size is %d\n", MAX);
    printf("1. Produce\n2. Consume\n3. Exit\n");
    while(1){
        printf("Enter option: ");
        scanf("%d", &optn);
        switch(optn) {
            case 1:
                producer(sharedMemory, &top, &mutex, &empty, &full);
                printBuffer(sharedMemory, top);
                break;
```

```
            case 2:
                consumer(sharedMemory, &top, &mutex, &empty, &full);
                printBuffer(sharedMemory, top);
                break;
            case 3:
                return 0;
            default:
                printf("Invalid option!\n");
                break;
        }
    }
    return 0;
}
```

Result:



Program - 6

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

Code:

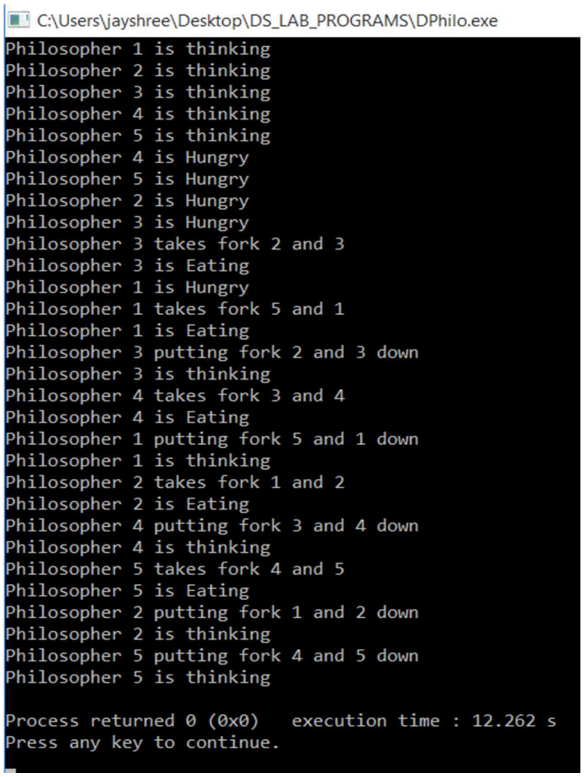
```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

#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 neighbors 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)
{
    int c = 0;
    while (c < 1) {
        int* i = num;
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
        c++;
    }
}
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);
}
```

Result:



Program - 7

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

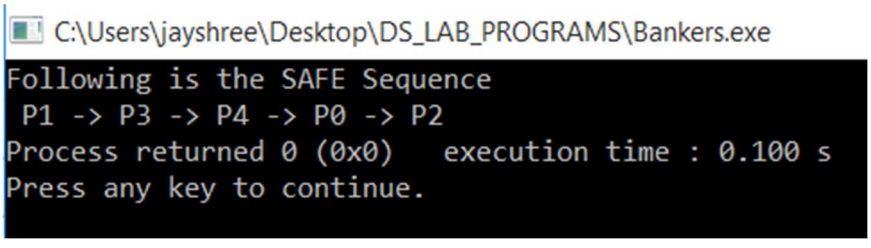
Code:

```
#include <stdio.h>
int main()
{
   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);
}
```

Result:



Program - 8

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

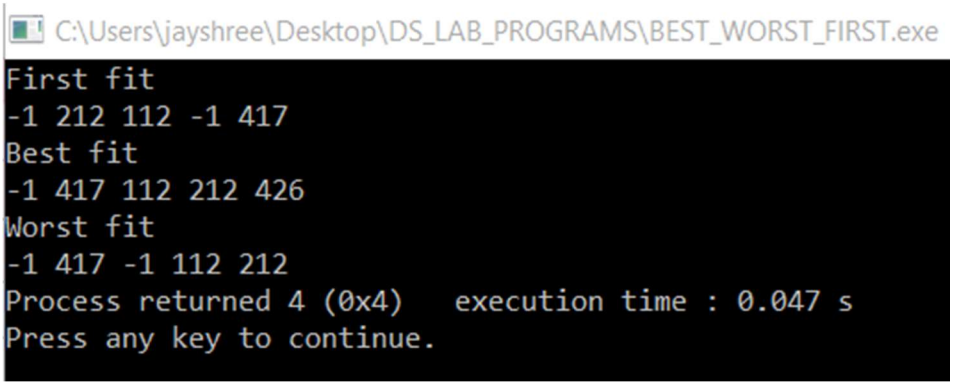
Code:

```
#include<stdio.h>


void main()
{


   int a[5], v[4];
   int m[] = {100, 500, 200, 300, 600};
   int p[] = {212, 417, 112, 426};


   for(int j=0;j<5;j++)
   {
      a[j] = -1;
   }



   for(int i = 0;i<4;i++)
   {
      for(int j = 0;j<5;j++)
      {
         if(p[i] < m[j]  && a[j] == -1)
         {
            a[j] = p[i];
```

```
            break;
        }
    }
}
printf("First fit\n");
for(int i=0;i<5;i++)
{
    printf("%d ",a[i]);

}
for(int j=0;j<5;j++)
{
    a[j] = -1;
}
int index;
for(int i=0;i<4;i++)
{
    index = -1;
    for(int j=0;j<5;j++)
    {
        if(m[j] >= p[i] && a[j] == -1 &&(index == -1 || m[j] < m[index])){
            index = j;
        }
    }

    if(index != -1)
        a[index] = p[i];
```

```
    }
    printf("\nBest fit\n");
    for(int i=0;i<5;i++)
        printf("%d ",a[i]);
    for(int j=0;j<5;j++)
        a[j] = -1;
    for(int i=0;i<4;i++)
    {
        index = -1;
        for(int j=0;j<5;j++)
        {
            if(m[j] >= p[i] && a[j] == -1 &&(index == -1 || m[j] > m[index])){
                index = j;
            }
        }
        if(index != -1)
          a[index] = p[i];
    }
    printf("\nWorst fit\n");
    for(int i=0;i<5;i++)
    {
        printf("%d ",a[i]);
    }
}
```

Result:



Program - 9

Question: Write a C program to simulate deadlock detection

Code:

```
//Write a C program to simulate deadlock detection
#include <stdio.h>
#include <stdbool.h>

#define NUM_PROCESSES 5
#define NUM_RESOURCES 3

int available[NUM_RESOURCES];
int allocation[NUM_PROCESSES][NUM_RESOURCES];
int request[NUM_PROCESSES][NUM_RESOURCES];
int avail_matrix[NUM_PROCESSES + 1][NUM_RESOURCES];

bool deadlockDetection(int *safeSequence) {
   int work[NUM_RESOURCES];
   bool finish[NUM_PROCESSES] = {false};
```

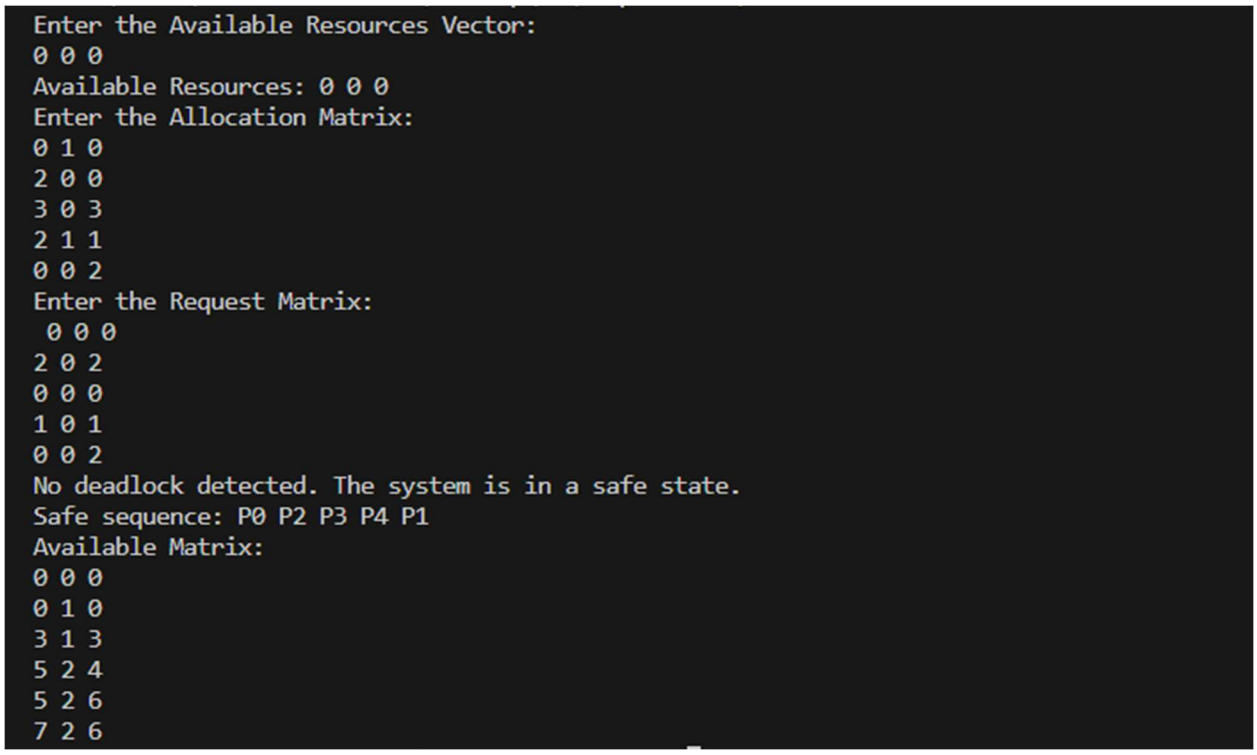
```
for (int i = 0; i < NUM_RESOURCES; i++) {
    work[i] = available[i];
    avail_matrix[0][i] = work[i];
}

int count = 0;
while (count < NUM_PROCESSES) {
    bool found = false;
    for (int i = 0; i < NUM_PROCESSES; i++) {
        if (!finish[i]) {
            bool canProceed = true;
            for (int j = 0; j < NUM_RESOURCES; j++) {
                if (request[i][j] > work[j]) {
                    canProceed = false;
                    break;
                }
            }
            if (canProceed) {
                for (int j = 0; j < NUM_RESOURCES; j++) {
                    work[j] += allocation[i][j];
                }
                safeSequence[count++] = i;
                finish[i] = true;
                found = true;
```

```
                for (int k = 0; k < NUM_RESOURCES; k++) {
                    avail_matrix[count][k] = work[k];
                }
            }
        }
    }

    if (!found) {
        break;
    }
  }

  for (int i = 0; i < NUM_PROCESSES; i++) {
    if (!finish[i]) {
        printf("Deadlock detected. Process P%d is in deadlock.\n", i);
        return false;
    }
  }
  printf("No deadlock detected. The system is in a safe state.\n");
  printf("Safe sequence: ");
  for (int i = 0; i < NUM_PROCESSES; i++) {
    printf("P%d ", safeSequence[i]);
  }
  printf("\n");
  return true;
}
int main() {
```

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

printf("Available Resources: ");
for (i = 0; i < NUM_RESOURCES; i++) {
   printf("%d ", available[i]);
}
printf("\n");
printf("Enter the Allocation Matrix:\n");
for (i = 0; i < NUM_PROCESSES; i++) {
   for (j = 0; j < NUM_RESOURCES; j++) {
      scanf("%d", &allocation[i][j]);
   }
}
printf("Enter the Request Matrix:\n");
for (i = 0; i < NUM_PROCESSES; i++) {
   for (j = 0; j < NUM_RESOURCES; j++) {
      scanf("%d", &request[i][j]);
   }
}
int safeSequence[NUM_PROCESSES];
if (deadlockDetection(safeSequence)) {
   printf("Available Matrix:\n");
   for (i = 0; i <= NUM_PROCESSES; i++) {
```

```
            for (j = 0; j < NUM_RESOURCES; j++) {
                printf("%d ", avail_matrix[i][j]);
            }
            printf("\n");
        }
    }

    return 0;
}
```

Result:



Program - 10

Question: Write a C program to simulate page replacement algorithms

a) FIFO b) LRU c) Optimal

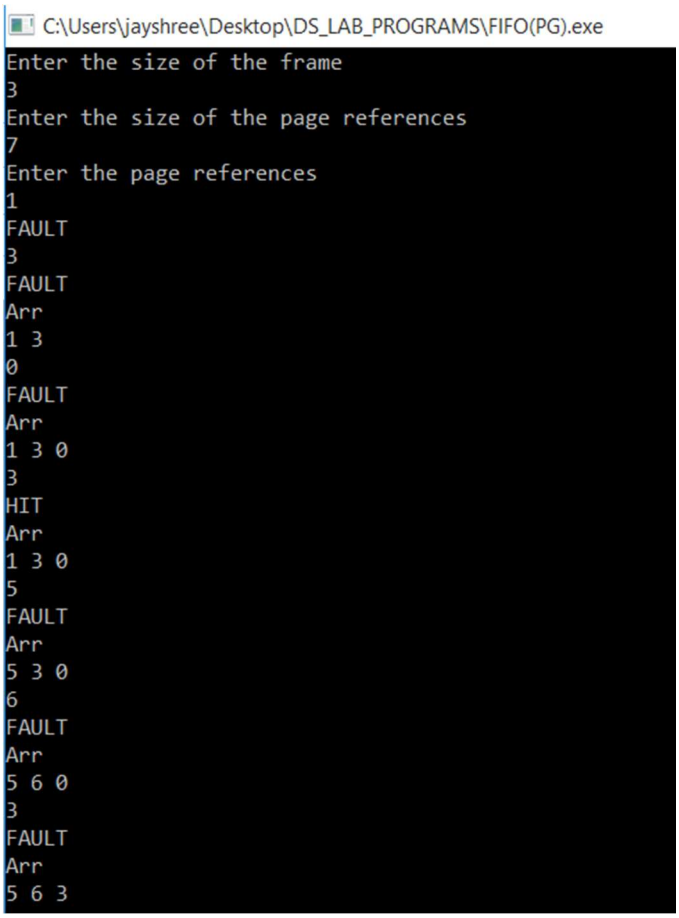
Code: a) FIFO

```
#include<stdio.h>
int front = -1;
int rear = -1;
void push(int a[], int n, int s, int flag)
{
    if(front == -1 && rear ==-1)
    {
        rear = front = 0;
        a[rear] = s;
    }
    else{

        for(int i=0;i<=rear;i++){
            if(a[i] == s)
            {
                flag = 1;
                printf("HIT\n");
                break;
            }
        }
        if(flag == 0)
        {
            printf("FAULT\n");
```

```
            if(rear == n-1)
            {
                a[front] = s;
                //front = (front+1) % n;
                (front +=1 )%n;
            }
            else{
                rear = (rear + 1) % n;
                a[rear] = s;
            }
        }
        printf("Arr\n");
        for(int i=0;i<=rear;i++)
        {
            printf("%d ",a[i]);
        }
        printf("\n");
    }
}
void main(){
    int n;
    printf("Enter the size of the frame \n");
    scanf("%d",&n);
    int a[n];
    int pg;
    printf("Enter the size of the page references\n");
    scanf("%d", &pg);
    int pr[pg];
    printf("Enter the page references\n");
    for(int i=0; i < pg;i++)
```

```
    {
        scanf("%d",&pr[i]);
        push(a, n, pr[i], 0);
    }
}
```

Result:



Code: b) LRU c) Optimal

```
#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 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) {
```

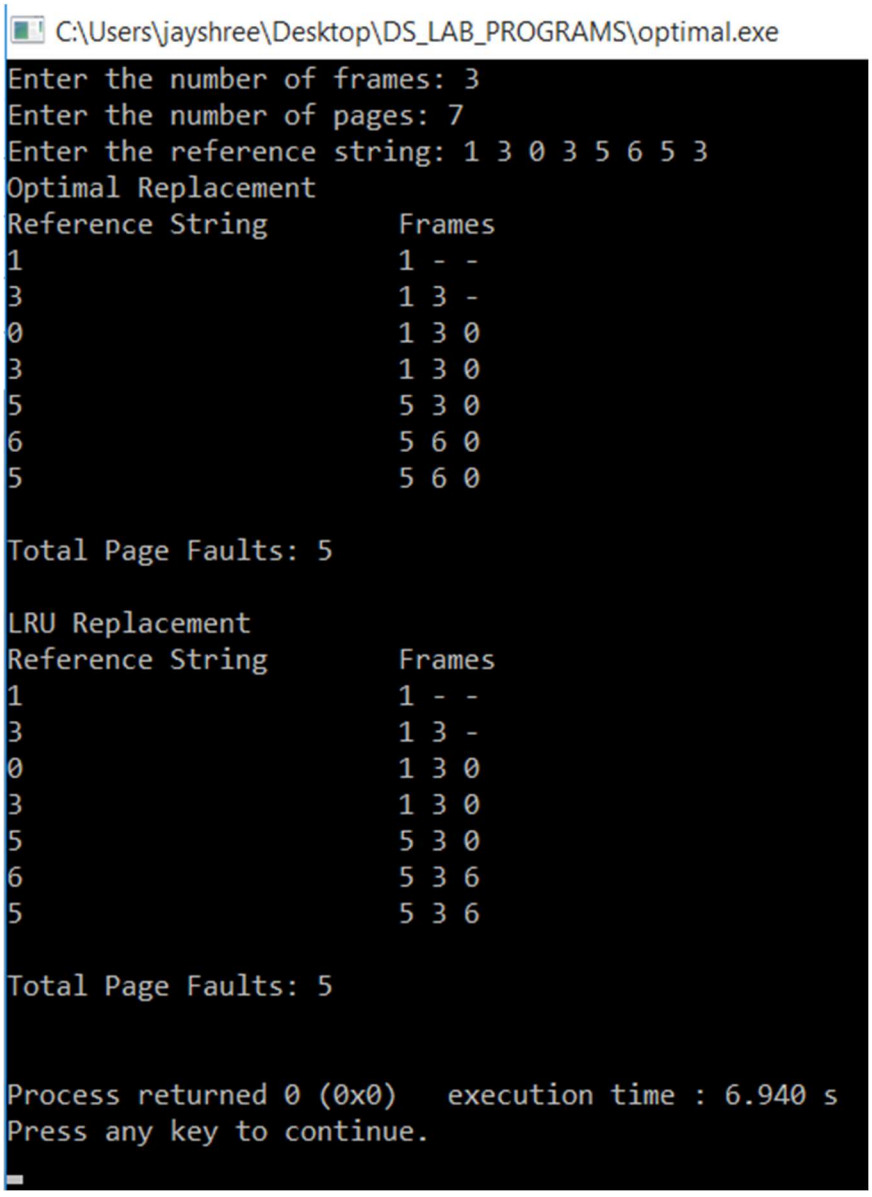
```
                farthest = j;
                index = i;
            }
            break;
        }
    }
    // If the page is not found in future, return this index
    if (j == numPages) {
        return i;
    }
  }

  // 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]);
    }
    optPageReplacement(pages, numPages, numFrames);
    lruPageReplacement(pages, numPages, numFrames);
    return 0;
}
```

Result:


C:\Users\jayshree\Desktop\DS_LAB_PROGRAMS\optimal.exe
Enter the number of frames: 3
Enter the number of pages: 7
Enter the reference string: 1 3 0 3 5 6 5 3
Optimal Replacement
Reference String Frames
1 1 - -
3 1 3 -
0 1 3 0
3 1 3 0
5 5 3 0
6 5 6 0
5 5 6 0
Total Page Faults: 5
LRU Replacement
Reference String Frames
1 1 - -
3 1 3 -
0 1 3 0
3 1 3 0
5 5 3 0
6 5 3 6
5 5 3 6
Total Page Faults: 5
Process returned 0 (0x0) execution time : 6.940 s
Press any key to continue.