CSE2005 OS – LAB 01

Rupin
20BCE1837

Shell, Unix, and fork()

a) Shell Programming

1. Types of Shells available to the system

```
rupin@rupin:~/Desktop/CSE2005$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/bin/dash
/usr/bin/dash
```

2. Show where the bash file is located

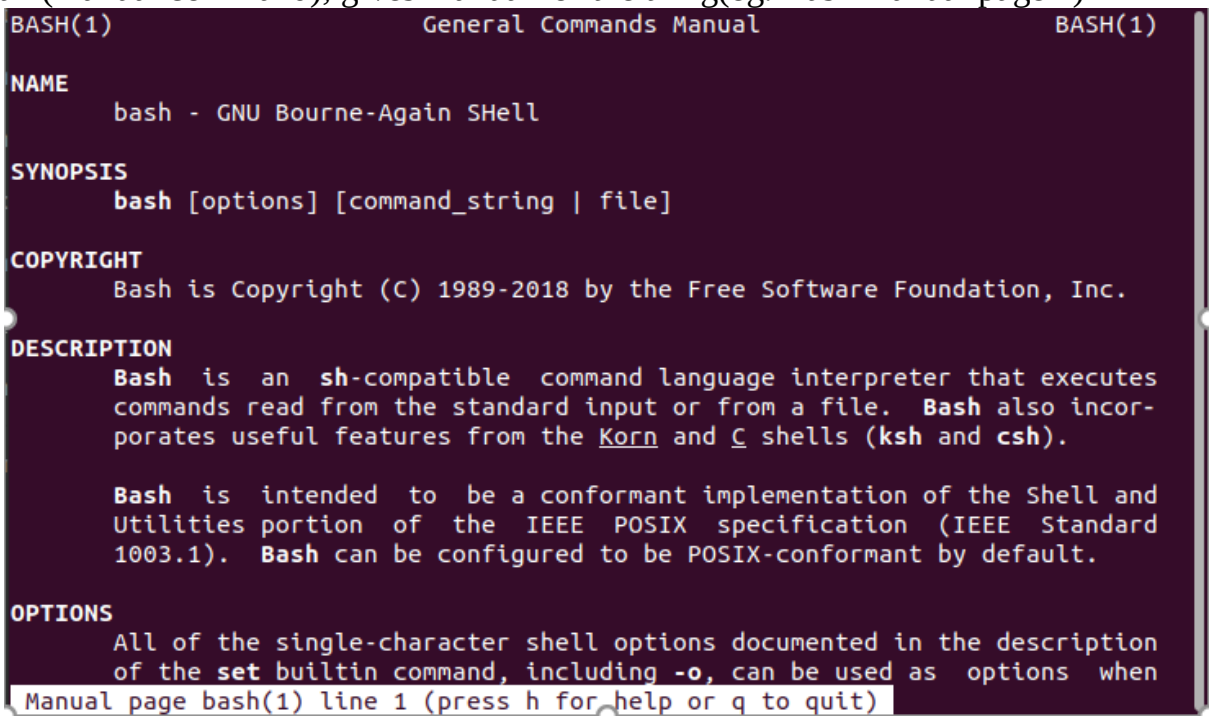
```
rupin@rupin:~/Desktop/CSE2005$ which bash
/usr/bin/bash
```

3. Contrasting non-shell programs with shell programs

```
rupin@rupin:~/Desktop/CSE2005$ which cd
rupin@rupin:~/Desktop/CSE2005$ which ls
/usr/bin/ls
```

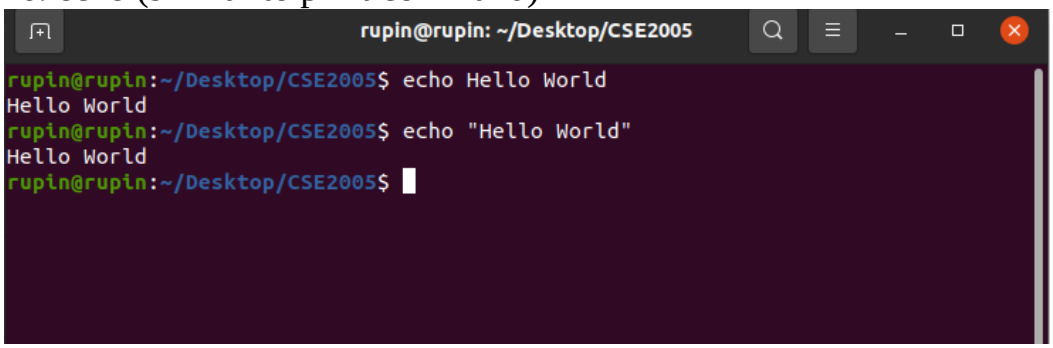
(cd is not shell, while ls is a shell program)

4. man (manual command), gives manual for the thing(eg. Bash manual page 1)

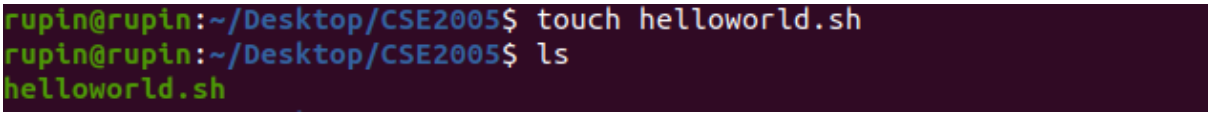


5. showcasing cd (change directory) command

6. echo (similar to print command)



7. touch command to create shell executable



Running shell file

```
rupin@rupin:~/Desktop/CSE2005$ ./helloworld.sh
bash: ./helloworld.sh: Permission denied
rupin@rupin:~/Desktop/CSE2005$
```

(permission denied as creating with touch)

8. chmod to change file perms

```
rupin@rupin:~/Desktop/CSE2005$ ls -al
total 8
drwxrwxr-x 2 rupin rupin 4096 Mar 15 22:53 .
drwxr-xr-x 3 rupin rupin 4096 Mar 15 22:48 ..
-rw-rw-r-- 1 rupin rupin    0 Mar 15 22:53 helloworld.sh
rupin@rupin:~/Desktop/CSE2005$ chmod +x helloworld.sh
rupin@rupin:~/Desktop/CSE2005$ ls -al
total 8
drwxrwxr-x 2 rupin rupin 4096 Mar 15 22:53 .
drwxr-xr-x 3 rupin rupin 4096 Mar 15 22:48 ..
-rwxrwxr-x 1 rupin rupin    0 Mar 15 22:53 helloworld.sh
rupin@rupin:~/Desktop/CSE2005$
```

(notice the change in helloWorld.sh in first and second execution of ls -al)

```
rupin@rupin:~/Desktop/CSE2005$ ./helloworld.sh
Hello World
rupin@rupin:~/Desktop/CSE2005$
```

9. set $variable

```
1 echo Hello World
2
3 echo $PwD
4 echo $HOME
5
```

```
rupin@rupin:~/Desktop/CSE2005$ ./helloworld.sh
Hello World
Todays date is 09-01-2022
/home/rupin
```

10. # (pound/hash) comments the text preceded by only # is not taken into account

11.read

```
1 echo Hello World
2
3 date=09-01-2022
4 echo "Todays date is $date"
5
6 #echo $PwD
7 #echo $HOME
8 read name
9
10 echo 'My name is : ' $name
```

```
rupin@rupin:~$ cd Desktop
rupin@rupin:~/Desktop$ cd CSE2005
rupin@rupin:~/Desktop/CSE2005$ ./helloworld
bash: ./helloworld: No such file or directory
rupin@rupin:~/Desktop/CSE2005$ ./helloworld.sh
Hello World
Todays date is 09-01-2022
Rupin
My name is :  Rupin
```

12. read -sp (read password silently)

```
read -sp 'Enter your name secretly : ' name

echo 'My name is : ' $name
```

```
rupin@rupin:~/Desktop/CSE2005$ ./helloworld.sh
Hello World
Todays date is 09-01-2022
Enter your name secretly : My name is :  Myname is secret
rupin@rupin:~/Desktop/CSE2005$
```

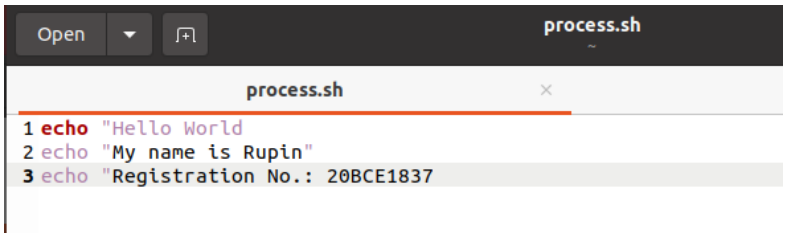
Rupin
20BCE1837

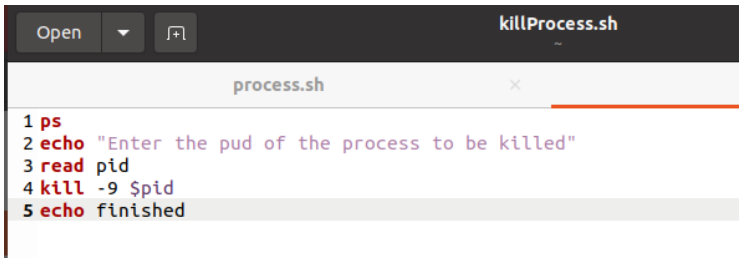
LAB 2

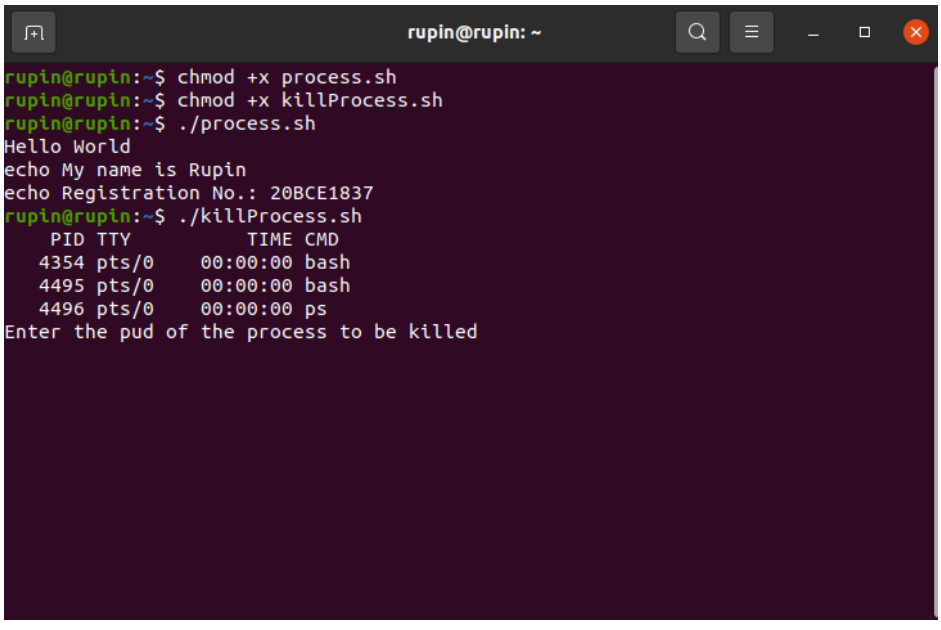
Aim: 1. To create and kill a process,
2. To perform addition, subtraction, multiplication and division,
3. To write a program for system input-output calls, and
4. To perform a bootloader example program.

Output:

Create and Kill a Process

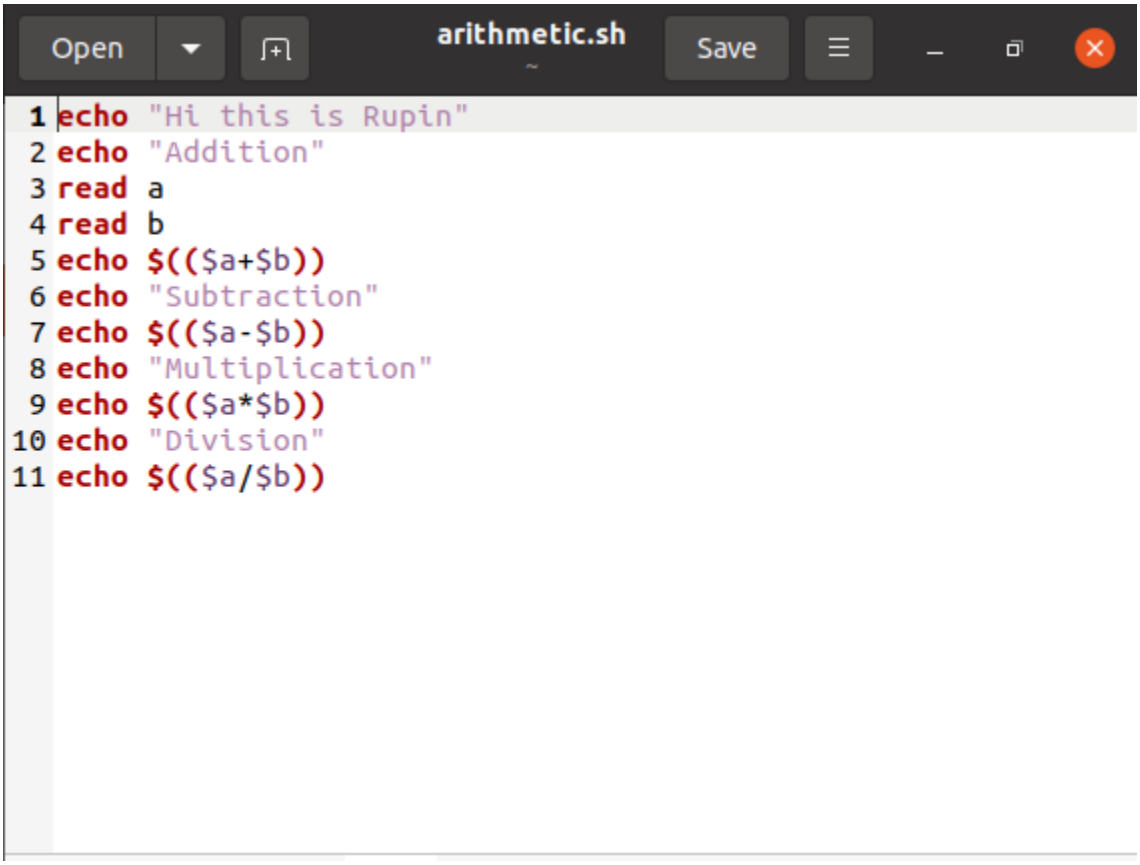


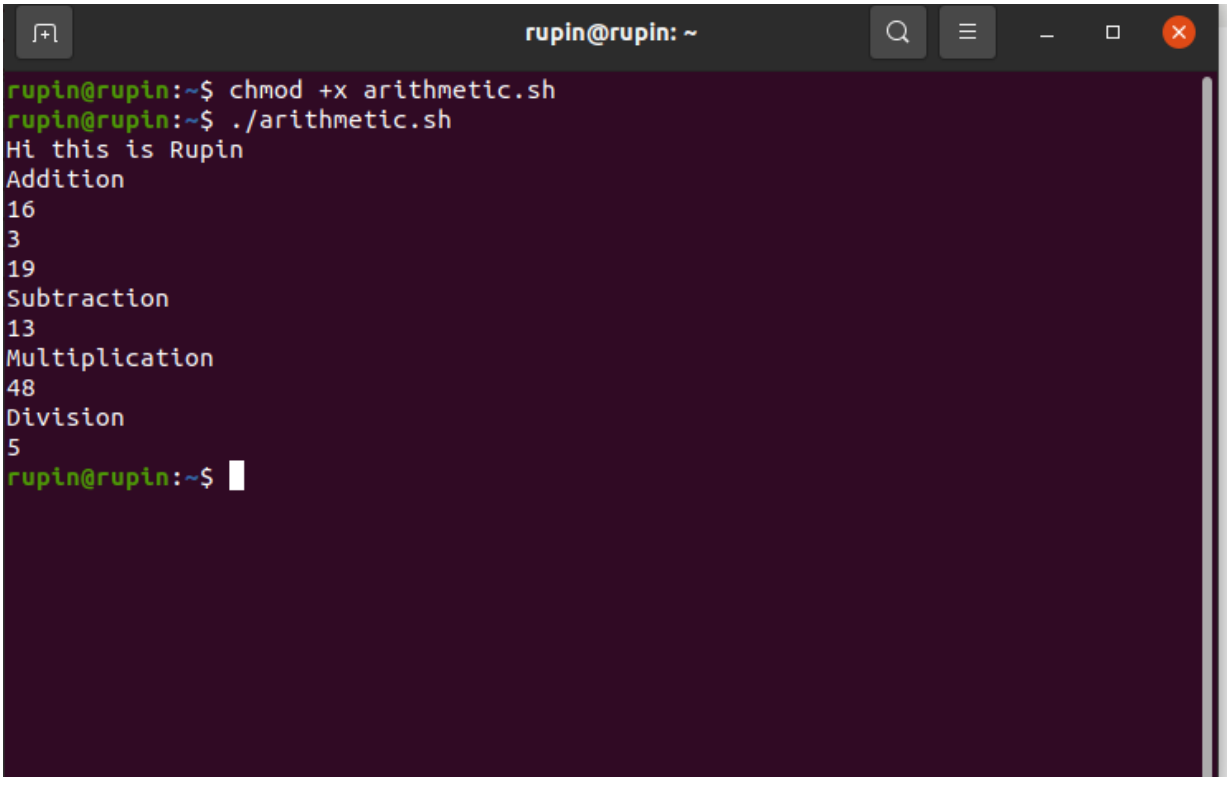




On clicking enter the Shell terminates

Calculator:





Rupin
20BCE1837

Lab No-3 Date: 20.01.2021

Aim:

1. To implement FCFS without arrival time
2. To implement FCFS with arrival time
3. To implement SJF without arrival time
4. To implement SJF without arrival time

Algorithm:

➢ FCFS

Step1: Create the number of process.
Siep2: Get the ID and Service time for each process.
Step3: Initially, Waiting time of first process is zero and Total time for the first process is the starting time of that process.
Step4: Calculate the Total time and Processing time for the remaining processes.
StepS: Waiting time of one process is the Total time of the previous process.
Step6: Total time of process is calculated by adding Waiting time and Service time.
Step7: Total waiting time is calculated by adding the waiting time for lack process.
Step&: Total turn around time is calculated by adding all total time of each process.
Step9: Calculate Average waiting time by dividing the total waiting time by total number of process.
Step 10: Calculate Average turn around time by dividing the total time by the number of process.

➢ SJF

Step1 1: Display the result.
Step1:Get the number of process.
Step2:Get the id and service time for each process.
Step3:Initially the waiting time of first short process as 0 and total time of first short is process the service time/gf that process.
Step4:Calculate the total time and waiting time of remaining process.
StepS: Waiting time of one process is the total time of the previous process.
Step6:Total time of process is calculated by adding the waiting time and service time of each process.
Step7:Total waiting time calculated by adding the waiting time of each process.

Step8:Total turn around time calculated by adding all total time of each process.
Step9:calculate average waiting time by dividing the total waiting time by total number of process.
Step10:Calculate average turn around time by dividing the total waiting time by total no of process
Step11: Display the result

Code:

```c
#include<stdio.h>
int TIME=0;
int TOTAL_TIME;
struct Process
{
char process[5];
int brust_time;
int exit_time;
};
int main()
{
int n;
printf("Enter the nunmber of the process: ");
scanf("%d",&n);
struct Process P[n];
for(int i=0;i<n;i++)
{
printf("Enter name of the process: ");
scanf("%s",P[i].process);
printf("Enter Brust time for process %d :",i+1);
scanf("%d",&P[i].brust_time);
}
for(int i=0;i<n;i++)
{
printf("Executing process %s. . .\n",P[i].process);
TIME+=P[i].brust_time;
P[i].exit_time=TIME;
}
printf("Total time elasped %d\n",TIME);
printf("Summary\n");
for(int i=0;i<n;i++)
{
printf("Process %s. . .\n",P[i].process);
printf("Brust time:%d \t Exit Time:%d \n",P[i].brust_time,P[i].exit_time);
}
}

```

Output:

```
rupin@rupin:~/Desktop/CSE2005$ gcc fcfs_without_arrival.c
rupin@rupin:~/Desktop/CSE2005$ ./a.out
Enter the nunmber of the process: 5
Enter name of the process: p1
Enter Brust time for process 1 :5
Enter name of the process: p2
Enter Brust time for process 2 :10
Enter name of the process: p3
Enter Brust time for process 3 :15
Enter name of the process: p4
Enter Brust time for process 4 :20
Enter name of the process: p5
Enter Brust time for process 5 :30
Executing process p1. . .
Executing process p2. . .
Executing process p3. . .
Executing process p4. . .
Executing process p5. . .
Total time elasped 80
Summary
Process p1. . .
Brust time:5     Exit Time:5
Process p2. . .
Brust time:10    Exit Time:15
Process p3. . .
Brust time:15    Exit Time:30
Process p4. . .
Brust time:20    Exit Time:50
Process p5. . .
Brust time:30    Exit Time:80
rupin@rupin:~/Desktop/CSE2005$
```

2.

```c
#include<stdio.h>
#include<string.h>
int main()
{
char pn[10][10],t[10];
int arr[10],bur[10],star[10],finish[10],tat[10],wt[10],i,j,n,temp;
int totwt=0,tottat=0;
printf("Enter the number of processes: ");
scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("Enter the ProcessName, Arrival Time & Burst Time: ");
scanf("%s%d%d",&pn[i],&arr[i],&bur[i]);
}
for(i=0; i<n; i++)
{
for(j=0; j<n; j++)
{
if(arr[i]<arr[j])
{
temp=arr[i];
arr[i]=arr[j];
arr[j]=temp;
temp=bur[i];
bur[i]=bur[j];
bur[j]=temp;
strcpy(t, pn[i]);
```

```
strcpy(t, pn[i]);
strcpy(pn[i],pn[j]);
strcpy(pn[j],t);
}

}
}
for(i=0; i<n; i++)
{
if(i==0)
star[i]=arr[i];
else
star[i]=finish[i-1];
wt[i]=star[i]-arr[i];
finish[i]=star[i]+bur[i];
tat[i]=finish[i]-arr[i];
}
printf("\nPName Arrtime Burtime WaitTime Start TAT Finish");
for(i=0; i<n; i++)
{
printf("\n%s\t%3d\t%3d\t%3d\t%3d\t%6d\t%6d",pn[i],arr[i],bur[i],wt[i],star[i],ta
totwt+=wt[i];
tottat+=tat[i];
}
printf("\nAverage Waiting time:%f",(float)totwt/n);
printf("\nAverage Turn Around Time:%f",(float)tottat/n);
return 0;
}
```

Output:

```
rupin@rupin:~/Desktop/CSE2005$ gcc fcfs_with_arr.c
fcfs_with_arr.c: In function ‘main’:
fcfs_with_arr.c:13:9: warning: format ‘%s’ expects argument of type ‘char *’, but argument 2 has type ‘char (*)[10]’ [-Wformat=]
   13 | scanf("%s%d%d",&pn[i],&arr[i],&bur[i]);
      |        ~^      ~~~~~~
      |         |      |
      |         char * char (*)[10]
rupin@rupin:~/Desktop/CSE2005$ ./a.out
Enter the number of processes: 3
Enter the ProcessName, Arrival Time & Burst Time: p1 3 5
Enter the ProcessName, Arrival Time & Burst Time: p2 5 7
Enter the ProcessName, Arrival Time & Burst Time: p3 2 9

PName Arrtime Burtime WaitTime Start TAT Finish
p3        2       9       0       2          9      11
p1        3       5       8      11         13      16
p2        5       7      11      16         18      23
Average Waiting time:6.333333
rupin@rupin:~/Desktop/CSE2005$
```

3.

```c
#include<stdio.h>
int main()
{
int bt[20],p[20],wt[20], tat[20], i, j,n, total=0, pos, temp;
float avg_wt,avg_tat;
printf("Enter number of process:");
scanf("%d",&n);

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

for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n; j++)
{
if(bt[j]<bt[pos])
pos=j;
}

temp=bt[i];
bt[i]=bt [pos];
bt [pos] =temp;

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

wt [0]=0;


for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0; j<i; j++)
wt[i]+=bt[j];

total+=wt [i];
}

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

printf("\nProcess\t  Burst Time  \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i];
total+=tat[i];
printf("\np%d\t\t  %d\t\t  %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}

avg_tat=(float)total/n;
printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Output:

```
rupin@rupin:~/Desktop/CSE2005$ gcc sjf_without_arr.c
rupin@rupin:~/Desktop/CSE2005$ ./a.out
Enter number of process:5

Enter Burst Time:
p1:4
p2:3
p3:8
p4:1
p5:2

Process    Burst Time     Waiting Time     Turnaround Time
p4                 1                0                       1
p5                 2                1                       3
p2                 3                3                       6
p1                 4                6                       10
p3                 8                10                      18

Average Waiting Time=4.000000
Average Turnaround Time=7.600000
rupin@rupin:~/Desktop/CSE2005$
```

4.

```c
#include<stdio.h>
#include<string.h>
int main()
{
int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
int totwt=0,totta=0;
float awt,ata;
char pn[10][10],t[10];
printf("Enter the number of process: ");
scanf("%d",&n);
for(i=0; i<n; i++)
{
printf("Enter process name, arrival time& execution time;");
scanf("%s%d%d",pn[i],&at[i],&et[i]);
}
for(i=0; i<n; i++)
for(j=0; j<n; j++)
{
if(et[i]<et[j])
{
temp=at[i];
at[i]=at[j];
at[j]=temp;
temp=et[i];
et[i]=et[j];
et[j]=temp;

strcpy(t,pn[i]);
strcpy(pn[i],pn[j]);
strcpy(pn[j],t);
}
}
for(i=0; i<n; i++)
{
if(i==0)
st[i]=at[i];
else
st[i]=ft[i-1];
wt[i]=st[i]-at[i];
ft[i]=st[i]+et[i];
ta[i]=ft[i]-at[i];
totwt+=wt[i];
totta+=ta[i];
}
awt=(float)totwt/n;
ata=(float)totta/n;
printf("\nPname\tarrivaltime\texecutiontime\twaitingtime\ttatime");
for(i=0; i<n; i++)
printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],wt[i],ta[i]);
printf("\nAverage waiting time is:%f",awt);
printf("\nAverage turnaroundtime is:%f",ata);
return 0;
}

```

Output:

Result:

All the scheduling programmes are implemented sucessfully.

Lab-4

Rupin
20BCE1837

Priority Scheduling

Code:

```c
#include <stdio.h>
#include<string.h>
int time = 0;


struct process
{
   char process[5];
   int burst_time, exit_time, arrival_time, turnaround,wait_time;
   char t[5];
   int priority;
};

int main()
{
   int n, avg;
   int total = 0;
   int tot=0;
   printf("Number of processes to be created:");
   scanf("%d", &n);

   struct process P[n];
   for (int i = 0; i < n; i++)
   {
      printf("ID of the process %d: ", i + 1);
      scanf("%s", P[i].process);
      printf("Arrival time of the process %s: ", P[i].process);
      scanf("%d", &P[i].arrival_time);
      printf("Burst time of the process %s: ", P[i].process);
      scanf("%d", &P[i].burst_time);
      printf("priority of the process %s: ", P[i].process);
      scanf("%d", &P[i].priority);
```

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


            temp = P[i].arrival_time;
            P[i].arrival_time=P[pos].arrival_time;
            P[pos].arrival_time = temp;

            strcpy(P[i].t,P[i].process);
            strcpy(P[i].process,P[pos].process);
            strcpy(P[pos].process,P[i].t);

        }
    }
    for (int i = 0; i < n; i++)
    {
        printf("Executing the process %s...\n", P[i].process);
        time += P[i].burst_time;
        P[i].exit_time = time;
        P[i].turnaround = P[i].exit_time - P[i].arrival_time;
    }
    printf("Total time elapsed %d\n", time);
    for (int i = 0; i < n; i++)
    {
        printf("Process %s...\n", P[i].process);
        printf("Burst time:%d \t TurnAround time:%d \t Exit time:%d\n
Priority:%d\n", P[i].burst_time, P[i].turnaround,
P[i].exit_time,P[i].priority);
    }
```

```
    for (int i = 0; i < n; i++)
    {
        total += P[i].turnaround;
    }
    for (int i = 0; i < n; i++)
    {
        tot += P[i].wait_time;
    }
    avg = (float)(total / n);
    printf("Average Turn Around time:%d\n", avg);
    float avg1 = (float)(total / n);
    printf("Average Waiting Time:%f\n", (float)avg1);
    return 0;
}
```

Output:

```
$ ./a.exe
Enter the number of the process
4
Enter the arrival time , burst time and priority of the process
AT BT PT
0  3  3
1  2  2
2  5  4
3  4  1
ID WT TAT
1 6     9
2 0     2
3 7     12
4 0     4
Avg waiting time of the process  is 3.250000
Avg turn around time of the process is 6.750000
```

```
Number of processes to be created:4
ID of the process 1: p1
Arrival time of the process p1: 0
Burst time of the process p1: 3
priority of the process p1: 3
ID of the process 2: p2
Arrival time of the process p2: 1
Burst time of the process p2: 5
priority of the process p2: 1
ID of the process 3: p3
Arrival time of the process p3: 2
Burst time of the process p3: 4
priority of the process p3: 4
ID of the process 4: p4
Arrival time of the process p4: 3
Burst time of the process p4: 2
priority of the process p4: 4
Executing the process p2...
Executing the process p1...
Executing the process p3...
Executing the process p4...
Total time elapsed 14
Process p2...
Burst time:5     TurnAround time:4       Exit time:5
 Priority:3
Process p1...
Burst time:3     TurnAround time:8       Exit time:8
 Priority:1
Process p3...
Burst time:4     TurnAround time:10      Exit time:12
 Priority:4
Process p4...
Burst time:2     TurnAround time:11      Exit time:14
 Priority:4
Average Turn Around time:8
Average Waiting Time:8.000000
```

Preemptive Priority scheduling

Code:

```
#include<stdio.h>

struct process
```

```
{
    int WT,AT,BT,TAT,PT;
};

struct process a[10];

int main()
{
    int n,temp[10],t,count=0,short_p;
    float total_WT=0,total_TAT=0,Avg_WT,Avg_TAT;
    printf("Enter the number of the process\n");
    scanf("%d",&n);
    printf("Enter the arrival time , burst time and priority of the
process\n");
    printf("AT BT PT\n");
    for(int i=0;i<n;i++)
    {
        scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);

        temp[i]=a[i].BT;
    }

    a[9].PT=10000;

    for(t=0;count!=n;t++)
    {
        short_p=9;
        for(int i=0;i<n;i++)
        {
            if(a[short_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)
            {
                short_p=i;
            }
        }

        a[short_p].BT=a[short_p].BT-1;

        if(a[short_p].BT==0)
        {
            count++;
```

```
            a[short_p].WT=t+1-a[short_p].AT-temp[short_p];
            a[short_p].TAT=t+1-a[short_p].AT;

            total_WT=total_WT+a[short_p].WT;
            total_TAT=total_TAT+a[short_p].TAT;

        }
    }

    Avg_WT=total_WT/n;
    Avg_TAT=total_TAT/n;

    printf("ID WT TAT\n");
    for(int i=0;i<n;i++)
    {
        printf("%d %d\t%d\n",i+1,a[i].WT,a[i].TAT);
    }

    printf("Avg waiting time of the process  is %f\n",Avg_WT);
    printf("Avg turn around time of the process is %f\n",Avg_TAT);

    return 0;
}
```

Output:

```
$ ./a.exe
Enter the number of the process
4
Enter the arrival time , burst time and priority of the process
AT BT PT
0  3  3
1  2  2
2  5  4
3  4  1
ID WT TAT
1 6     9
2 0     2
3 7     12
4 0     4
Avg waiting time of the process  is 3.250000
Avg turn around time of the process is 6.750000
```

Non-Preemptive Priority Scheduling-

Code:

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

```c
int time = 0;

struct process
{
    char process[5];
    int burst_time, exit_time, arrival_time, turnaround,wait_time;
    char t[5];
    int priority;
};

int main()
{
    int n, avg;
    int total = 0;
    int tot=0;
    printf("Number of processes to be created:");
    scanf("%d", &n);

    struct process P[n];
    for (int i = 0; i < n; i++)
    {
        printf("ID of the process %d: ", i + 1);
        scanf("%s", P[i].process);
        printf("Arrival time of the process %s: ", P[i].process);
        scanf("%d", &P[i].arrival_time);
        printf("Burst time of the process %s: ", P[i].process);
        scanf("%d", &P[i].burst_time);
        printf("priority of the process %s: ", P[i].process);
        scanf("%d", &P[i].priority);
    }
    for (int i=0;i<n;i++)
    {
        int pos = i;
        for(int j=i+1;j<n;j++)
        {
```

```
            if (P[j].priority < P[i].priority)
            {
               pos=j;
            }
            int temp = P[i].burst_time;
            P[i].burst_time=P[pos].burst_time;
            P[pos].burst_time = temp;


            temp = P[i].arrival_time;
            P[i].arrival_time=P[pos].arrival_time;
            P[pos].arrival_time = temp;

            strcpy(P[i].t,P[i].process);
            strcpy(P[i].process,P[pos].process);
            strcpy(P[pos].process,P[i].t);

        }
    }
    for (int i = 0; i < n; i++)
    {
        printf("Executing the process %s...\n", P[i].process);
        time += P[i].burst_time;
        P[i].exit_time = time;
        P[i].turnaround = P[i].exit_time - P[i].arrival_time;
    }
    printf("Total time elapsed %d\n", time);
    for (int i = 0; i < n; i++)
    {
        printf("Process %s...\n", P[i].process);
        printf("Burst time:%d \t TurnAround time:%d \t Exit time:%d\n
Priority:%d\n", P[i].burst_time, P[i].turnaround,
P[i].exit_time,P[i].priority);
    }
    for (int i = 0; i < n; i++)
    {
        total += P[i].turnaround;
    }
    for (int i = 0; i < n; i++)
    {
```

```
        tot += P[i].wait_time;
    }
    avg = (float)(total / n);
    printf("Average Turn Around time:%d\n", avg);
    float avgWaiting = (float)(total / n);
    printf("Average Waiting Time:%f\n", (float)avgWaiting);
    return 0;
}
```

Output:

```
Numberofprocessestobecreated:4 ID ofthe process 1: p1
Arrival time of the process p1: 0
Burst time of the process p1: 3
priority of the process p1: 3
ID of the process 2: p2
Arrival time of the process p2: 1
Burst time of the process p2: 2
priority of the process p2: 2
ID of the process 3: p3
Arrival time of the process p3: 2
Burst time of the process p3: 5
priority of the process p3: 4
ID of the process 4: p4
Arrival time of the process p4: 3
Burst time of the process p4: 4
priority of the process p4: 1
Executing the process p4...
Executing the process p1...
Executing the process p2...
Executing the process p3...
Total time elapsed 14
Process p4...
Burst time:4     TurnAround time:1
 Priority:3                                  Exit time:4
Process p1. . .
Burst  t1me:3          TurnAround t1me:7
 Pr1ority : 2
                                             Exit time:7
Process p2. . .
Bursttime:2            TurnAroundtime:8 Priority:4
Process p3...
                                             Exit time:9
Burst time:5     TurnAround time:12
 Priority:1
Average Turn Around time:7
                                             Exit time:14
Average Waiting Time:7.000000
```

Round Robin scheduling w/o Arrival time

Code:

```
#include<iostream>
using namespace std;

void findWaitingTime(int processes[], int n,
             int bt[], int wt[], int quantum)
{
    int rem_bt[n];
    for (int i = 0 ; i < n ; i++)
        rem_bt[i] = bt[i];

    int t = 0;

    while (1)
    {
        bool done = true;

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

            if (rem_bt[i] > 0)
            {
                done = false;

                if (rem_bt[i] > quantum)
                {

                    t += quantum;

                    rem_bt[i] -= quantum;
                }
```

```
                else
                {

                    t = t + rem_bt[i];

                    wt[i] = t - bt[i];

                    rem_bt[i] = 0;
                }
            }
        }

        if (done == true)
        break;
    }
}

void findTurnAroundTime(int processes[], int n,
                        int bt[], int wt[], int tat[])
{


    for (int i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}


void findavgTime(int processes[], int n, int bt[],
                                    int quantum)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;


    findWaitingTime(processes, n, bt, wt, quantum);


    findTurnAroundTime(processes, n, bt, wt, tat);
```

```
    cout << "Processes "<< " Burst time "
         << " Waiting time " << " Turn around time\n";


    for (int i=0; i<n; i++)
    {
        total_wt = total_wt +
        wt[i]; total_tat =
        total_tat + tat[i];
        cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
            << wt[i] <<"\t\t " << tat[i] <<endl;
    }

  cout << "Average waiting time = "
      << (float)total_wt / (float)n;
  cout << "\nAverage turn around time = "
      << (float)total_tat / (float)n;
}


int main()
{
    int processes[] = { 1, 2, 3};
    int n = sizeof processes / sizeof

    processes[0]; int burst_time[] = {5,3,4};

    int quantum = 2;
    findavgTime(processes, n, burst_time,
    quantum); return 0;
```

Output:

```
$ ./a.exe
Processes  Burst time  Waiting time  Turn around time
 1             5         7              12
 2             3         6              9
 3             4         7              11
Average waiting time = 6.66667
Average turn around time = 10.6667
```

Rupin
20BCE1837
LAB-5

Round Robin with arrival time

```c
#include<stdio.h>

int main()
{

int count,j,n,time,remain,flag=0
int wait_time=0,turnaround_time=
printf("Enter Total Process:\t "
scanf("%d",&n);
remain=n;
for(count=0;count<n;count++)
{
printf("Enter Arrival Time and B
scanf("%d",&at[count]);
scanf("%d",&bt[count]);
rt[count]=bt[count];
}
printf("Enter Time Quantum:\t");
scanf("%d",&time_quantum);
printf("\n\nProcess\t|Turnaround
for(time=0,count=0;remain!=0;)
{
if(rt[count]<=time_quantum && rt
{
time+=rt[count]; rt[count]=0; fl
}
else if(rt[count]>0)
{
rt[count]-=time_quantum; time+=t
}
if(rt[count]==0 && flag==1)
{

remain--;
printf("P[%d]\t|\t%d\t|\t%d\n",c
wait_time+=time-at[count]-bt[cou
}
}
if(count==n-1) count=0;
else if(at[count+1]<=time) co
else
count=0;
}
printf("\nAverage Waiting Tim
printf("Avg Turnaround Time =

return 0;
}
```

```
rupin@rupin:~/Desktop/CSE2005$ gcc roundrobin_with_arrtime.c
rupin@rupin:~/Desktop/CSE2005$ ./a.out
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Process Number 1: 1
5
Enter Arrival Time and Burst Time for Process Process Number 2: 2
6
Enter Arrival Time and Burst Time for Process Process Number 3: 3
7
Enter Arrival Time and Burst Time for Process Process Number 4: 4
8
Enter Time Quantum:     20


Process |Turnaround Time|Waiting Time

P[1]    |       4       |       -1
P[2]    |       9       |       3
P[3]    |       15      |       8
P[4]    |       22      |       14

Average Waiting Time= 6.000000
Avg Turnaround Time = 12.500000rupin@rupin:~/Desktop/CSE2005$
```

Round robin with very long time-quantum-

```
rupin@rupin:~/Desktop/CSE2005$ gcc roundrobin_with_arrtime.c
rupin@rupin:~/Desktop/CSE2005$ ./a.out
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Process Number 1: 1
2
Enter Arrival Time and Burst Time for Process Process Number 2: 2
2
Enter Arrival Time and Burst Time for Process Process Number 3: 3
4
Enter Arrival Time and Burst Time for Process Process Number 4: 4
3
Enter Time Quantum:     2


Process |Turnaround Time|Waiting Time

P[1]    |       1       |       -1
P[2]    |       2       |       0
P[3]    |       7       |       3
P[4]    |       7       |       4

Average Waiting Time= 1.500000
Avg Turnaround Time = 4.250000rupin@rupin:~/Desktop/CSE2005$
```

Bankers Algorithm

```c
#include <stdio.h>
#include <stdlib.h>
void final_output(int k[][10], int n, int p)
{
int i, j;
for (i = 0; i < n; i++)
{
printf("\n");
for (j = 0; j < p; j++)
{
printf("%d\t", k[i][j]);
}
}
}
//Banker's Algorithm
void Banker(int A[][10], int N[][10],
int M[10][10], int W[1][10], int *n, int *m)
{
int i, j;
printf("\n Enter total number of processes : ");
scanf("%d", n);
printf("\n Enter total number of resources : ");
scanf("%d", m);
for (i = 0; i < *n; i++)
{
printf("\n Process %d\n", i + 1);
for (j = 0; j < *m; j++)
{
printf(" Allocation for resource %d : ", j + 1);
scanf("%d", &A[i][j]);
printf(" Maximum for resource %d : ", j + 1);
scanf("%d", &M[i][j]);
}
}
printf("\n Available resources : \n");
for (i = 0; i < *m; i++)
{
printf(" Resource %d : ", i + 1);
scanf("%d", &W[0][i]);
}
```

```
41
42 for (i = 0; i < *n; i++)
43 for (j = 0; j < *m; j++)
44 N[i][j] = M[i][j] - A[i][j];
45
46 printf("\n ********Allocation Matrix**********");
47 final_output(A, *n, *m);
48 printf("\n ********Maximum Requirement Matrix" "*********");
49 final_output(M, *n, *m);
50 printf("\n ***********Need Matrix*********");
51 final_output(N, *n, *m);
52 }
53 int safety(int A[][10], int N[][10],int B[1][10], int n, int m, int a[])
54 {
55
56 int i, j, k, x = 0, f1 = 0, f2 = 0;
57 int F[10], W[1][10];
58 for (i = 0; i < n; i++)
59 F[i] = 0;
60 for (i = 0; i < m; i++)
61 W[0][i] = B[0][i];
62
63 for (k = 0; k < n; k++)
64 {
65 for (i = 0; i < n; i++)
66 {
67 if (F[i] == 0)
68 {
69 f2 = 0;
70 for (j = 0; j < m; j++)
71 {
72 if (N[i][j] > W[0][j]) f2 = 1;
73 }
74 if (f2 == 0 && F[i] == 0)
75 {
76 for (j = 0; j < m; j++)
77 W[0][j] += A[i][j]; F[i] = 1;
78 f1++;
79 a[x++] = i;
80 }
81 }
82 }
```

```
}
if (f1 == n)

return 1;
}
return 0;
}

void request(int A[10][10], int N[10][10], int B[10][10], int pid, int K)
{
int rmat[1][10];
int i;
printf("\n Enter additional request : \n");
for (i = 0; i < K; i++)
{
printf(" Request for resource %d : ", i + 1);
scanf("%d", &rmat[0][i]);
}

for (i = 0; i < K; i++)
if (rmat[0][i] > N[pid][i])
{
printf("\n ******Error encountered******\n");
exit(0);
}

for (i = 0; i < K; i++)  if (rmat[0][i] > B[0][i])
{
printf("\n ******Resources unavailable*****\n");
exit(0);
}

for (i = 0; i < K; i++)
{
B[0][i] -= rmat[0][i];
A[pid][i] += rmat[0][i];
N[pid][i] -= rmat[0][i];
}
}
int banker(int A[][10], int N[][10], int W[1][10], int n, int m)
{
int j, i, a[10];
j = safety(A, N, W, n, m, a);
```

```
j = safety(A, N, W, n, m, a);

if (j != 0)
{
printf("\n\n");
printf("\n A safety sequence has been " "detected.\n");
for (i = 0; i < n; i++) printf(" P%d ", a[i]); printf("\n");
return 1;
}
else
{
printf("\n Deadlock has occured.\n"); return 0;
}
}
int main()
{
int All[10][10], Max[10][10], Need[10][10]
, W[1][10];
int n, m, pid, c, r;
printf("\n *********DEADLOCK AVOIDANCE USING" "BANKER'S ALGORITHM**********\n");
Banker(All, Need, Max, W, &n, &m); r = banker(All, Need, W, n, m);
if (r != 0)
{
printf("\n Do you want make an additional" "request for any of the process ? (1=Yes|0=No)");
   scanf("%d", &c);
if (c == 1)
{
printf("\n Enter process number : ");
 scanf("%d", &pid);
request(All, Need, W, pid - 1, m);
r = banker(All, Need, W, n, m);
if (r == 0)
{
exit(0);
}
}
}
else

exit(0);
return 0;
}

```

Output

```
rupin@rupin:~/Desktop/CSE2005$ gcc banker.c
rupin@rupin:~/Desktop/CSE2005$ ./a/out
bash: ./a/out: No such file or directory
rupin@rupin:~/Desktop/CSE2005$ ./a.out

 *********DEADLOCK AVOIDANCE USINGBANKER'S ALGORITHM***********

 Enter total number of processes : 2

 Enter total number of resources : 10

 Process 1
 Allocation for resource 1 : 1
 Maximum for resource 1 : 2
 Allocation for resource 2 : 1
 Maximum for resource 2 : 3
 Allocation for resource 3 : 1
 Maximum for resource 3 : 2
 Allocation for resource 4 : 3
 Maximum for resource 4 : 1
 Allocation for resource 5 : 5
 Maximum for resource 5 : 3
 Allocation for resource 6 : 2
 Maximum for resource 6 : 5
 Allocation for resource 7 : 3
 Maximum for resource 7 : 5
 Allocation for resource 8 : 2
 Maximum for resource 8 : 4
 Allocation for resource 9 : 2
 Maximum for resource 9 : 4
 Allocation for resource 10 : 3
 Maximum for resource 10 : 2

 Process 2
 Allocation for resource 1 : 4
 Maximum for resource 1 : 2
 Allocation for resource 2 : 4
 Maximum for resource 2 : 2
 Allocation for resource 3 : 5
 Maximum for resource 3 : 3
 Allocation for resource 4 : 5
 Maximum for resource 4 : 3
 Allocation for resource 5 : 6
 Maximum for resource 5 : 3
 Allocation for resource 6 : 6
 Maximum for resource 6 : 3
 Allocation for resource 7 : 4
 Maximum for resource 7 : 2
 Allocation for resource 8 : 3
 Maximum for resource 8 : 5
 Allocation for resource 9 : 3
 Maximum for resource 9 : 5
 Allocation for resource 10 : 2
 Maximum for resource 10 : 6

 Available resources :
 Resource 1 : 1
 Resource 2 : 6
 Resource 3 : 2
 Resource 4 : 6
 Resource 5 : 2
 Resource 6 : 7
 Resource 7 : 3
 Resource 8 : 7
 Resource 9 : 4
 Resource 10 : 8

 ********Allocation Matrix**********
1       1       1       3       5       2       3       2       2       3
4       4       5       5       6       6       4       3       3       2
 ********Maximum Requirement Matrix*********
2       3       2       1       3       5       5       4       4       2
2       2       3       3       3       3       2       5       5       6
 ***********Need Matrix*********
1       2       1       -2      -2      3       2       2       2       -1
-2      -2      -2      -2      -3      -3      -2      2       2       4
```

```
 ********Allocation Matrix**********
1       1       1       3       5       2       3       2       2       3
4       4       5       5       6       6       4       3       3       2
 ********Maximum Requirement Matrix*********
2       3       2       1       3       5       5       4       4       2
2       2       3       3       3       3       2       5       5       6
 ***********Need Matrix*********
1       2       1       -2      -2      3       2       2       2       -1
-2      -2      -2      -2      -3      -3      -2      2       2       4


 A safety sequence has been detected.
 P0  P1

 Do you want make an additionalrequest for any of the process ? (1=Yes|0=No)^[[A
```

LAB 6

20BCE1837

Aim: Banker Algorithm

Code:

```
#include  <stdio.h>
#include <stdlib.h>
int maxm = 100;

void display(int k[][maxm], int n, int p)
{
        int i, j;
        for (i = 0; i < n; i++)
        {
        for (j = 0; j < p; j++)
        {
        printf("%d\t", k[i][j]);
        }
        printf("\n");
        }
}

void Banker(int allocation[][maxm], int need[][maxm], int max[maxm][maxm], int
resource[1][maxm], int *n, int *m)
{
        int i, j;
        printf("Enter total number of processes: ");
        scanf("%d", n);
        printf("Enter total number of resources: ");
        scanf("%d", m);
        for (i = 0; i < *n; i++)
```

```
        {
        printf("\nProcess %d\n", (i+1));
        for (j = 0; j < *m; j++)
        {
        printf("Allocation for resource %d: ", (j+1));
        scanf("%d", &allocation[i][j]);
        printf("Maximum for resource %d: ", (j+1));
        scanf("%d", &max[i][j]);
        }
        }
        printf("\nAvailable resources:\n");
        for (i = 0; i < *m; i++)
        {
        printf("Resource %d: ", i + 1);
        scanf("%d", &resource[0][i]);
        }

        for (i = 0; i < *n; i++)
        for (j = 0; j < *m; j++)
        need[i][j] = max[i][j] - allocation[i][j];

        printf("\nAllocation Matrix:\n");
        display(allocation, *n, *m);
        printf("\nMaximum Requirement Matrix:\n");
        display(max, *n, *m);
        printf("\nNeed Matrix:\n");
        display(need, *n, *m);
}

int safety(int allocation[][maxm], int need[][maxm], int B[1][maxm], int n, int m,
int a[])
{

        int i, j, k, x = 0, f1 = 0, f2 = 0;
        int F[maxm], resource[1][maxm];
```

```
        for (i=0; i<n; i++)
        F[i] = 0;
        for (i=0; i<m; i++)
        resource[0][i] = B[0][i];

        for (k = 0; k < n; k++)
        {
        for (i = 0; i < n; i++)
        {
        if (F[i] == 0)
        {
                f2 = 0;
                for (j = 0; j < m; j++)
                {
                if (need[i][j] > resource[0][j])
                f2 = 1;
                }
                if (f2 == 0 && F[i] == 0)
                {
                for (j = 0; j < m; j++)
                resource[0][j] += allocation[i][j];
                F[i] = 1;
                f1++;
                a[x++] = i;
                }
        }
        }
        if (f1 == n)
        return 1;
        }
        return 0;
}

void request(int allocation[maxm][maxm], int need[maxm][maxm], int
B[maxm][maxm], int indx, int K)
```

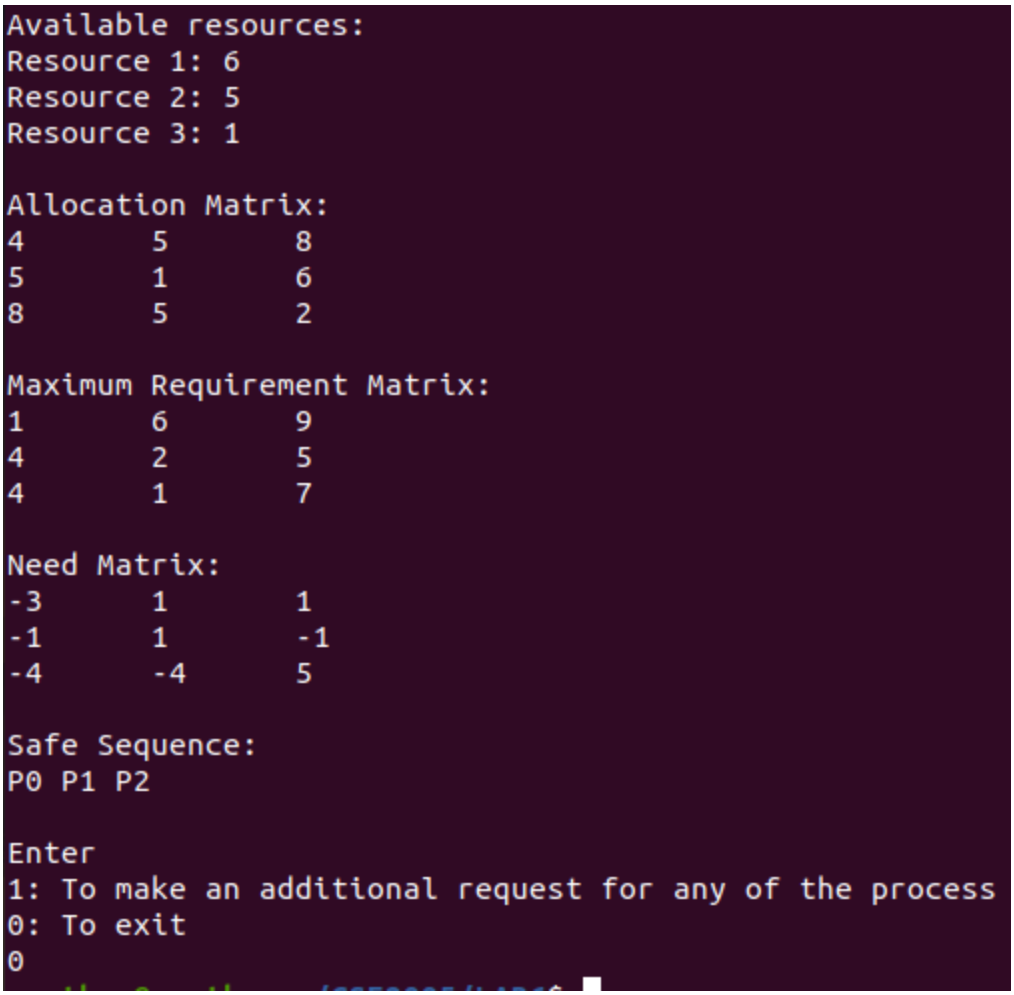
```
{
        int rr[1][maxm];
        int i;
        printf("\nEnter additional request\n");
        for (i = 0; i < K; i++)
        {
        printf("Request for resource %d: ", (i+1));
        scanf("%d", &rr[0][i]);
        }

        for (i = 0; i < K; i++)
        if (rr[0][i] > need[indx][i])
        {
        printf("\nError encountered\n");
        exit(0);
        }

        for (i = 0; i < K; i++)
        if (rr[0][i] > B[0][i])
        {
        printf("\nResources unavailable\n");
        exit(0);
        }

        for (i = 0; i < K; i++)
        {
        B[0][i] -= rr[0][i];
        allocation[indx][i] += rr[0][i];
        need[indx][i] -= rr[0][i];
        }
}

int banker(int allocation[][maxm], int need[][maxm], int resource[1][maxm], int n,
int m)
{
```

```
        int j, i, a[maxm];
        j = safety(allocation, need, resource, n, m, a);
        if (j != 0)
        {
        printf("\nSafe Sequence:\n");
        for (i = 0; i < n; i++)
        printf("P%d ", a[i]);
        printf("\n");
        return 1;
        }
        else
        {
        printf("\n Deadlock has occured.\n");
        return 0;
        }
}

int main()
{
        int All[maxm][maxm], Max[maxm][maxm], Need[maxm][maxm],
resource[1][maxm];
        int n, m, indx, c, r;
        Banker(All, Need, Max, resource, &n, &m);
        r = banker(All, Need, resource, n, m);
        if (r!=0)
        {
        printf("\nEnter\n1: To make an additional request for any of the process\n0:
To exit\n");
        scanf("%d", &c);
        if (c==1)
        {
        printf("\nEnter process number: ");
        scanf("%d", &indx);
        request(All, Need, resource, indx - 1, m);
        r = banker(All, Need, resource, n, m);
```

```
        if (r == 0)
        {
                exit(0);
        }
        }
        }
        return 0;
}
```

Output:

```
Enter total number of processes: 3
Enter total number of resources: 3

Process 1
Allocation for resource 1: 4
Maximum for resource 1: 1
Allocation for resource 2: 5
Maximum for resource 2: 6
Allocation for resource 3: 8
Maximum for resource 3: 9

Process 2
Allocation for resource 1: 5
Maximum for resource 1: 4
Allocation for resource 2: 1
Maximum for resource 2: 2
Allocation for resource 3: 6
Maximum for resource 3:
5

Process 3
Allocation for resource 1: 8
Maximum for resource 1: 4
Allocation for resource 2: 5
Maximum for resource 2: 1
Allocation for resource 3: 2
Maximum for resource 3: 7
```


Available resources:
Resource 1: 6
Resource 2: 5
Resource 3: 1
Allocation Matrix:
4 5 8
5 1 6
8 5 2
Maximum Requirement Matrix:
1 6 9
4 2 5
4 1 7
Need Matrix:
-3 1 1
-1 1 -1
-4 -4 5
Safe Sequence:
P0 P1 P2
Enter
1: To make an additional request for any of the process
0: To exit
0

LAB 7

Aim: 1. Peterson Solution
2. Producer consumer problem using Semaphore

Peterson Solution

Code:

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

int flag[2];
int turn;

void lock(int i)
{
   flag[i] = 0;
   turn = 1-i;
   while((flag[1-i]) && (turn==(1-i)));
}

void unlock(int i)
{
   flag[i] = 0;
}

void* fn0(void *test){
   lock(0);
   printf("Process 0");
   unlock(0);
}
```

```
void* fn1(void *test){
   lock(1);
   printf("Process 1");
   unlock(1);
}

int main()
{
   pthread_t p1, p2;
   pthread_create(&p1, NULL, fn0, (void*)0);
   pthread_create(&p2, NULL, fn1, (void*)1);
   pthread_join(p1, NULL);
   pthread_join(p2, NULL);
   return 0;
}
```

OUTPUT:

Producer consumer problem using semaphore

Algorithm:

1.

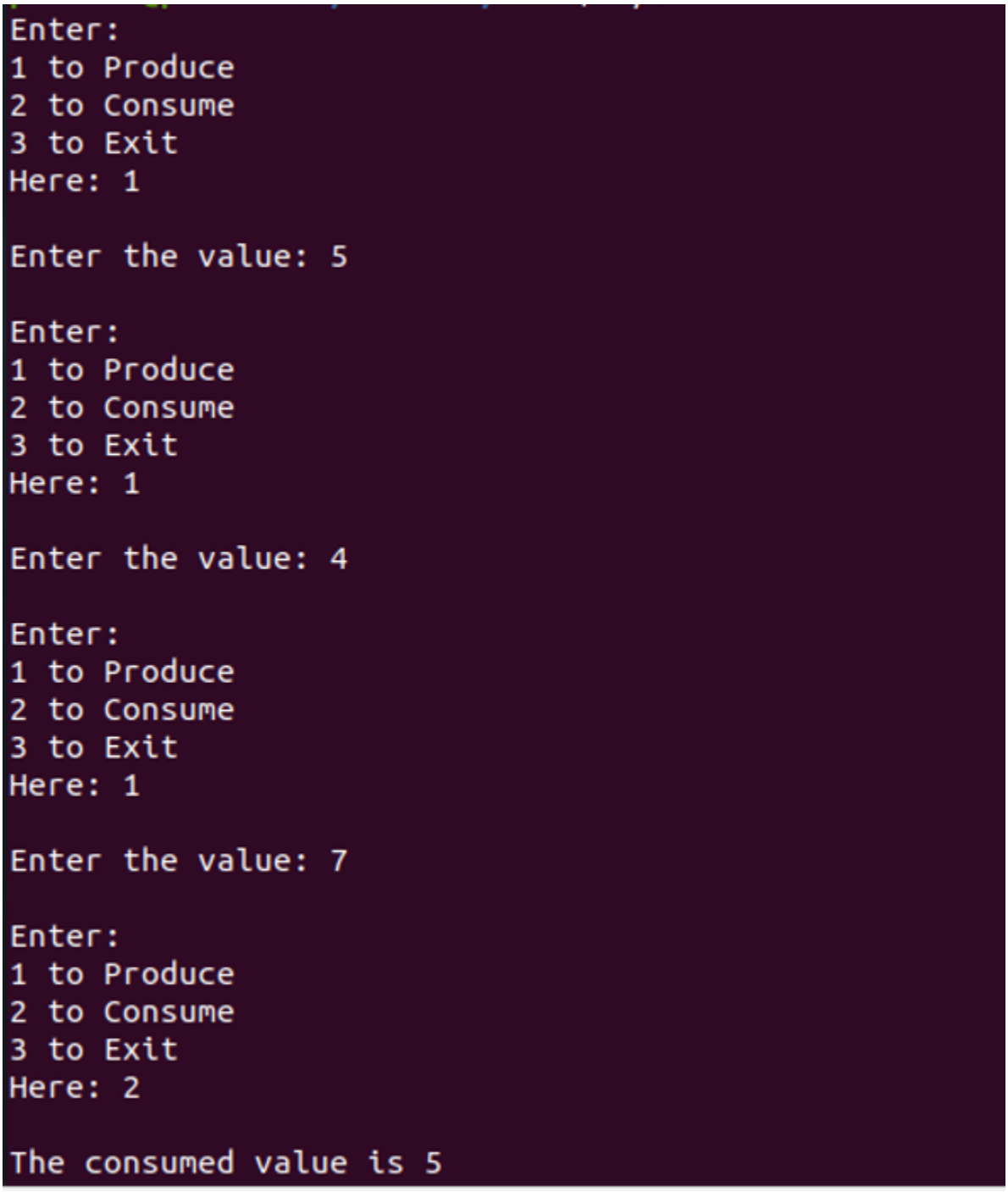
Code:

```
#include <stdio.h>
int main()
{
        int bsize = 10, i = 0, o = 0, pr, cn, ch = 0;
        int bufr[bsize];
        while (ch!=3)
        {
        printf("Enter:\n1 to Produce\n2 to Consume\n3 to Exit\nHere: ");
        scanf("%d", &ch);
        switch (ch)
```

```
{
case 1:
        if ((i + 1) % bsize == o)
        printf("\nBuffer is Full");
        else
        {
        printf("\nEnter the value: ");
        scanf("%d", &pr);
        printf("\n");
        bufr[i] = pr;
        i = (i + 1) % bsize;
        }
        break;
case 2:
        if (i == o)
        printf("\nBuffer is Empty");
        else
        {
        cn = bufr[o];
        printf("\nThe consumed value is %d\n\n", cn);
```

```
            o = (o + 1) % bsize;
            }
            break;
        }
        }
        return 0;
}
```

OUTPUT:



LAB 8

LAB 8

Aim: 1. Peterson Solution

2. Producer Consumer problem with Semaphore
3. Producer Consumer problem without Semaphore

Peterson Solution

Code:

#include <stdio.h> int flag[2] = {0}; int turn;

void entryChecker(int pid){ flag[pid] = 1;

turn = pid;

int k = flag[1-pid];

while((k==1) && (turn == pid));

}

void exitSetter(int pid){ flag[pid] = 0;

}

void processDoer(int i){ entryChecker(i);

{

```
printf("Inside the critical section of the code\n");
}
exitSetter(i);
}

int main(){ processDoer(1); processDoer(3);
 return 0;
}
```

OUTPUT:

```
Inside the critical section of the code
Inside the critical section of the code
```

Producer Consumer problem with Semaphore Code:

```
#include <stdio.h>
int mutex = 0, empty, full = 0, in=0, out=0, n; int buffer[10];

void wait(int k){ while(k<0){
printf("\nCannot Add Item\n");
}
k--;
}

void signal(int k){ k++;
}
```

```
void producer(){ do{
wait(empty); wait(mutex); printf("Enter an item: "); scanf("%d", &buffer[in]); in++;
signal(mutex); signal(full);
}while(in<n);
}

void consumer(){ do{
wait(full); wait(mutex);
printf("\nConsumed item: %d", buffer[out]); out++;

signal(mutex); signal(empty);
}while(out<n);
}

int main(){
printf("Enter the size: "); scanf("%d", &n);
empty = n; while(in<n){
producer();
}
while(in!=out){ consumer();
```

}

printf("\n"); return 0;

}

OUTPUT:

Producer Consumer problem without Semaphore

Code:

#include <stdio.h>

int count = 0, in=0, out=0, n; const int bsize = 5;

int buffer[5];

void producer(){ while(1){

while(count == bsize); printf("Enter value in buffer: "); scanf("%d", &buffer[in]);

in++;

in = in%bsize; count++;

}

```
}

void consumer(){ while(1){
while(count == 0);
printf("Consumed value %d", buffer[out]); out++;
out = out%bsize; count--;
}
}

int main(){
printf("Enter the size: "); scanf("%d", &n); while(in<n){
producer();
}

while(in!=out){ consumer();
}
printf("\n"); return 0;
}
```

OUTPUT:

LAB 9

LAB 9

Aim: 1. Dining Philosophers Problem
2. Reader Writer Problem

Dining Philosophers Problem

Code:

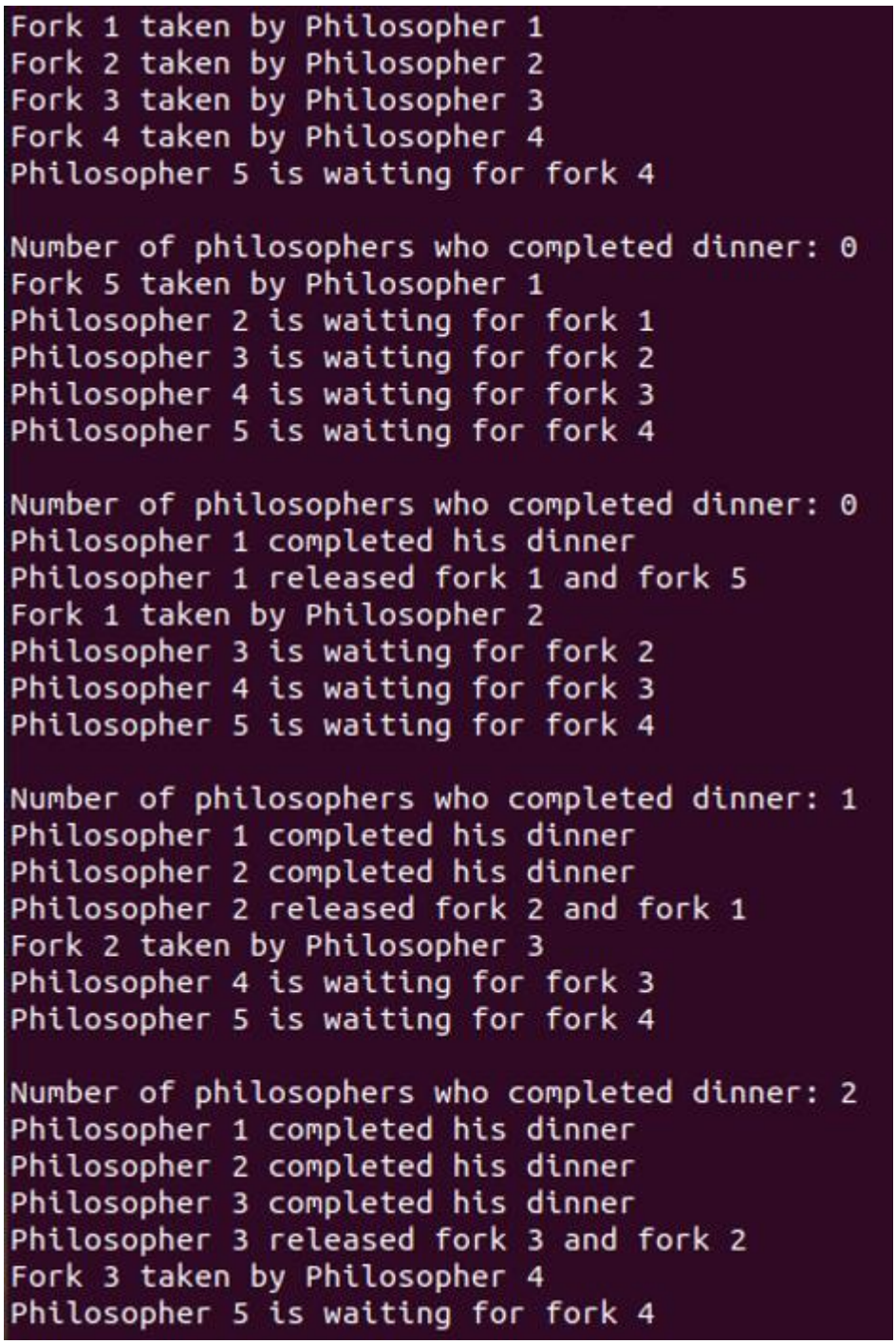
```
#include <stdio.h>
#define n 5
int cp = 0, i;

struct fork
{
        int taken;
} forkTaken[n];

struct philosp
{
        int l;
        int r;
} pstatus[n];

void goForDinner(int pID)
{
        if (pstatus[pID].l == 10 && pstatus[pID].r == 10){
        printf("Philosopher %d completed his dinner\n", pID + 1);
        }

        else if (pstatus[pID].l == 1 && pstatus[pID].r == 1){
        printf("Philosopher %d completed his dinner\n", pID + 1);
        pstatus[pID].l = pstatus[pID].r = 10;
```

```
        int otherFork = pID - 1;
        if (otherFork == -1){
        otherFork = (n - 1);
        }

        forkTaken[pID].taken = forkTaken[otherFork].taken = 0;
        printf("Philosopher %d released fork %d and fork %d\n", pID + 1, pID + 1,
otherFork + 1);
        cp++;
        }
        else if (pstatus[pID].l == 1 && pstatus[pID].r == 0){
        if (pID == (n - 1)){
        if (forkTaken[pID].taken == 0){
                forkTaken[pID].taken = pstatus[pID].r = 1;
                printf("Fork %d taken by philosopher %d\n", pID + 1, pID + 1);
        }
        else{
                printf("Philosopher %d is waiting for fork %d\n", pID + 1, pID + 1);
        }
        }
        else{

        int dpID = pID;
        pID -= 1;

        if (pID == -1)
                pID = (n - 1);

        if (forkTaken[pID].taken == 0)
        {
                forkTaken[pID].taken = pstatus[dpID].r = 1;
                printf("Fork %d taken by Philosopher %d\n", pID + 1, dpID + 1);
        }
        else
        {
                printf("Philosopher %d is waiting for fork %d\n", dpID + 1, pID + 1);
```

```
        }
        }
        }
        else if (pstatus[pID].l == 0){
        if (pID == (n - 1)){
        if (forkTaken[pID - 1].taken == 0){
                forkTaken[pID - 1].taken = pstatus[pID].l = 1;
                printf("Fork %d taken by Philosopher %d\n", pID, pID + 1);
        }
        else{
                printf("Philosopher %d is waiting for fork %d\n", pID + 1, pID);
        }
        }
        else{

        if (forkTaken[pID].taken == 0){
                forkTaken[pID].taken = pstatus[pID].l = 1;
                printf("Fork %d taken by Philosopher %d\n", pID + 1, pID + 1);
        }
        else{
                printf("Philosopher %d is waiting for fork %d\n", pID + 1, pID + 1);
        }
        }
        }
}


int main()
{
        for (i = 0; i < n; i++){
        forkTaken[i].taken = pstatus[i].l = pstatus[i].r = 0;
        }

        while (cp < n){
        for (i = 0; i < n; i++){
        goForDinner(i);
        }
```

```
        printf("\nNumber of philosophers who completed dinner: %d\n", cp);
        }
        return 0;
}
```

OUTPUT:



Lab-10

Rupin
20BCE1837

Fixed Allocation:

.

```
#include
<iostream> using
namespace std; int
main()
{
   int n,t;
   cout<<"Enter the number of
   processes: "; cin>>n;
   int
   proc[n],vis[n];
   for(int
   i=0;i<n;i++){
      cout<<"Enter the size of process"<<i+1<<": ";
      cin>>proc[i];
      vis[i]=0;
   }
   cout<<"Enter the number of
   blocks: "; cin>>t;
   int
   wastage=0;
   int
   block[t],vb[t];
   for(int i=0;i<t;i++){
      cout<<"Enter the capacity of block"<<i+1<<": ";
      cin>>block[i];
      wastage+=block[i
      ]; vb[i]=0;
   }
   cout <<"Press 1 for FirstFit, press 2 for BestFit, press 3 for
   WorstFit."<<endl; cout << "Enter the choice of fit: ";
   int o;
   cin >>
   o;
   if(o==1){
   for(int i=0;i<n;i++){
      for(int
      j=0;j<t;j++){
         if(proc[i]<=block[j] && vb[j]==0){
            vis[i]=1;
            vb[j]=1;
            block[j]-
            =proc[i];
```

```
            wastage-
            =proc[i];
            cout<<"Process"<<i+1<<" allocated to
            block"<<j+1<<endl; break;
        }
    }
}
```

```
}
else
if(o==2){ int
m,flag=0;
for(int i=0;i<n;i++){
   int min=1000;
   for(int
   j=0;j<t;j++){
      if(proc[i]<=block[j] &&
         vb[j]==0){ if(block[j]<min)
         {
            min=block[
            j]; m=j;
            vis[i]=1;
         }
      }
   }
   if(vis[i]==1){
   vb[m]=1;
   block[m]-
   =proc[i];
   wastage-
   =proc[i];
   cout<<"Process"<<i+1<<" allocated to block"<<m+1<<endl;
   }
}
}
else
if(o==3){ int
m,flag=0;
for(int
   i=0;i<n;i++){
   int max=0;
   for(int j=0;j<t;j++){
      if(proc[i]<=block[j] &&
      vb[j]==0){
         if(block[j]>max)
         {
            max=block[j
            ]; m=j;
            vis[i]=1;
         }
      }
   }
   if(vis[i]==1){
   vb[m]=1;
   block[m]-
   =proc[i];
   wastage-
   =proc[i];
   cout<<"Process"<<i+1<<" allocated to block"<<m+1<<endl;
   }
}}
for(int
   i=0;i<n;i++){
   if(vis[i]==0)
   cout<<"Process"<<i+1<<" not allocated to any block."<<endl;
```

```
    }
    cout<<"Total wastage of memory is:
    "<<wastage<<"KB"; return 0;
}
```

Output for First Fit-

```
Enter the number of processes: 4
Enter the size of process1: 20
Enter the size of process2: 200
Enter the size of process3: 500
Enter the size of process4: 50
Enter the number of blocks: 4
Enter the capacity of block1: 30
Enter the capacity of block2: 50
Enter the capacity of block3: 200
Enter the capacity of block4: 700
Press 1 for FirstFit, press 2 for BestFit, press 3 for WorstFit.
Enter the choice of fit: 1
Process1 allocated to block1
Process2 allocated to block3
Process3 allocated to block4
Process4 allocated to block2
Total wastage of memory is: 210KB
```

Output for Best Fit-

```
Enter the number of processes: 4
Enter the size of process1: 20
Enter the size of process2: 300
Enter the size of process3: 500
Enter the size of process4: 50
Enter the number of blocks: 4
Enter the capacity of block1: 30
Enter the capacity of block2: 50
Enter the capacity of block3: 200
Enter the capacity of block4: 700
Press 1 for FirstFit, press 2 for BestFit, press 3 for WorstFit.
Enter the choice of fit: 2
Process1 allocated to block1
Process2 allocated to block4
Process4 allocated to block2
Process3 not allocated to any block.
Total wastage of memory is: 610KB
```

Output for Worst Fit-

```
Enter the number of processes: 4
Enter the size of process1: 20
Enter the size of process2: 200
Enter the size of process3: 500
Enter the size of process4: 50
Enter the number of blocks: 4
Enter the capacity of block1: 30
Enter the capacity of block2: 50
Enter the capacity of block3: 200
Enter the capacity of block4: 700
Press 1 for FirstFit, press 2 for BestFit, press 3 for WorstFit.
Enter the choice of fit: 3
Process1 allocated to block4
Process2 allocated to block3
Process4 allocated to block2
Process3 not allocated to any block.
Total wastage of memory is: 710KB
```

Variable Allocation:

Code:

```
#include
<iostream> using
namespace std;

int
main
()
{
int n, t;
cout << "Enter the number of
processes: "; cin >> n;
int proc[n], vis[n];
for (int i = 0; i < n; i++)
   {
cout << "Enter the size of process" << i + 1 << ":
"; cin >> proc[i];
vis[i] = 0;
}
cout << "Enter the number of
blocks: "; cin >> t;
int wastage =
0; int block[t];
for (int i = 0; i < t; i++)
   {
cout << "Enter the capacity of block" << i + 1 <<
": "; cin >> block[i];
wastage += block[i];
}
cout <<
"Press 1 for FirstFit, press 2 for BestFit, press 3 for WorstFit."
<< endl;
```

```
cout << "Enter the choice of
fit: "; int o;
cin >> o;
if (o ==
1)

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

         {


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

             {
    if (proc[i] <=
       block[j])

                  {
vis[i] = 1;
block[j] -=
proc[i]; wastage
-= proc[i];
cout << "Process" << i +
                     1 << " allocated to block" << j +
1 <<
endl;
break;
}}}}
else if (o == 2)
   {
int m, flag = 0;
for (int i = 0; i < n; i++)
         {
   int min =
      1000;
 for (int j = 0; j < t;
           j++)
             {
    if (proc[i] <=
       block[j])
                 {
    if (block[j] <
               min)
                    {
min =
block[j]; m =
j;

vis[i] = 1;
}
}
}
}
if (vis[i] == 1)
            {
block[m] -=
proc[i]; wastage
-= proc[i];
cout << "Process" << i + 1 << " allocated to block" <<
m + 1 << endl;
}}}
else if(o==3){
int m, flag =
0;
for (int i = 0; i < n; i++)
```

```
{
}
int max = 0;
for (int j = 0; j < t; j++)
{
if (proc[i] <= block[j])
{
```

```
if (block[j] > max)
{
max =
block[j]; m =
j;
vis[i] = 1;
}}}
if (vis[i] == 1)
{
block[m] -=
proc[i]; wastage
-= proc[i];
cout << "Process" << i + 1 << " allocated to block" <<
m + 1 << endl;
}}}
for (int i = 0; i < n; i++)
{
if (vis[i] == 0)
cout << "Process" << i +
1 << " not allocated to any block." << endl;
}
cout << "Total wastage of memory is: " << wastage <<
"KB"; cout<<endl<<"Indraneel-20BCE1154";
return 0;
}
```

Output for First Fit-

```
Enter the number of processes: 4
Enter the size of process1: 20
Enter the size of process2: 200
Enter the size of process3: 500
Enter the size of process4: 50
Enter the number of blocks: 4
Enter the capacity of block1: 30
Enter the capacity of block2: 50
Enter the capacity of block3: 200
Enter the capacity of block4: 700
Press 1 for FirstFit, press 2 for BestFit, press 3 for WorstFit.
Enter the choice of fit: 1
Process1 allocated to block1
Process2 allocated to block3
Process3 allocated to block4
Process4 allocated to block2
Total wastage of memory is: 210KB
Indraneel-20BCE1154
```

Output for Best Fit-

```
Enter the number of processes: 4
Enter the size of process1: 20
Enter the size of process2: 200
Enter the size of process3: 500
Enter the size of process4: 50
Enter the number of blocks: 4
Enter the capacity of block1: 30
Enter the capacity of block2: 150
Enter the capacity of block3: 200
Enter the capacity of block4: 600
Press 1 for FirstFit, press 2 for BestFit, press 3 for WorstFit.
Enter the choice of fit: 2
Process1 allocated to block1
Process2 allocated to block3
Process3 allocated to block4
Process4 allocated to block4
Total wastage of memory is: 210KB
Indraneel-20BCE1154
```

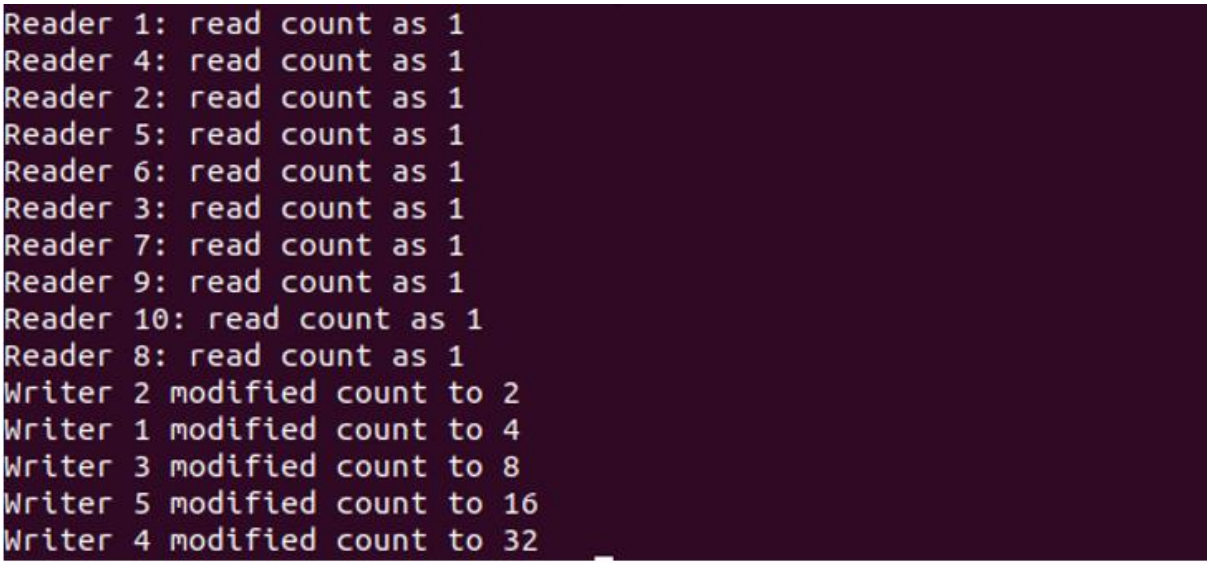
Output for Worst Fit-

```
Enter the number of processes: 4
Enter the size of process1: 20
Enter the size of process2: 200
Enter the size of process3: 500
Enter the size of process4: 50
Enter the number of blocks: 4
Enter the capacity of block1: 30
Enter the capacity of block2: 50
Enter the capacity of block3: 200
Enter the capacity of block4: 700
Press 1 for FirstFit, press 2 for BestFit, press 3 for WorstFit.
Enter the choice of fit: 3
Process1 allocated to block4
Process2 allocated to block4
Process4 allocated to block4
Process3 not allocated to any block.
Total wastage of memory is: 710KB
Indraneel-20BCE1154
```

Reader Writer Problem

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

sem_t w;
pthread_mutex_t mutex;
int count = 1;
int nr = 0;

void *writer(void *wn)
{
        sem_wait(&w);
        count = count*2;
        printf("Writer %d modified count to %d\n",(*((int *)wn)),count);
        sem_post(&w);

}
void *reader(void *rn)
{
        pthread_mutex_lock(&mutex);
        nr++;
        if(nr == 1) {
        sem_wait(&w);
        }
        pthread_mutex_unlock(&mutex);
        printf("Reader %d: read count as %d\n",*((int *)rn),count);
        pthread_mutex_lock(&mutex);
        nr--;
        if(nr == 0) {
        sem_post(&w);
        }
        pthread_mutex_unlock(&mutex);
}
```

```
int main()
{

        pthread_t read[10],write[5];
        pthread_mutex_init(&mutex,
        NULL);sem_init(&w,0,1);
        int i;
        int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        for(i = 0; i < 10; i++) {
        pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);
        }
        for(i = 0; i < 5; i++) {
        pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);
        }

        for(i = 0; i < 10;
        i++) {
        pthread_join(read[i
        ], NULL);
        }
        for(i = 0; i < 5;
        i++) {
        pthread_join(write[
        i], NULL);
        }

        pthread_mutex_destroy(&mu
        tex);sem_destroy(&w);

        return 0;

}
```



LAB 11

Aim: 1. FIFO Page Replacement

2. Optimal Page Replacement

FIFO Page Replacement

Algorithm:

1. Start traversing the pages.
2. Declare the size as the length of the Page.
3. Check the need of the replacement from the page to memory.

4. Check the need of the replacement from the old page to the new page in memory.
5. Form the queue to hold all pages.
6. Insert the required page memory into the queue.
7. Check the bad replacements and page faults.
8. Get the number of processes to be inserted.
9. Show the values.

Code:

```
#include <stdio.h> int main()
{
int i, j = 0, nf, np, pg[50], frame[10], k, av, c = 0;
printf("Enter number of frames: ");
scanf("%d", &nf);
printf("Enter number of pages: "); scanf("%d", &np);
printf("Enter page reference string: "); for (i = 1; i <= np;
i++){
scanf("%d", &pg[i]);
}

for (i = 0; i < nf; i++){ frame[i] = -1;
}

printf("\n");
for (i = 1; i <= np; i++)
```

```
{
printf("%d\t\t", pg[i]); av = 0;
for (k = 0; k < nf; k++) if (frame[k] == pg[i])
av = 1; if (av == 0)
{
frame[j] = pg[i]; j = (j + 1) % nf; c++;
for (k = 0; k < nf; k++) printf("%d\t", frame[k]);
}
printf("\n");
}
printf("Page Fault: %d\n", c); return 0;
}
```

LAB 11

Aim: 1. FIFO Page Replacement

2. Optimal Page Replacement

FIFO Page Replacement

Algorithm:

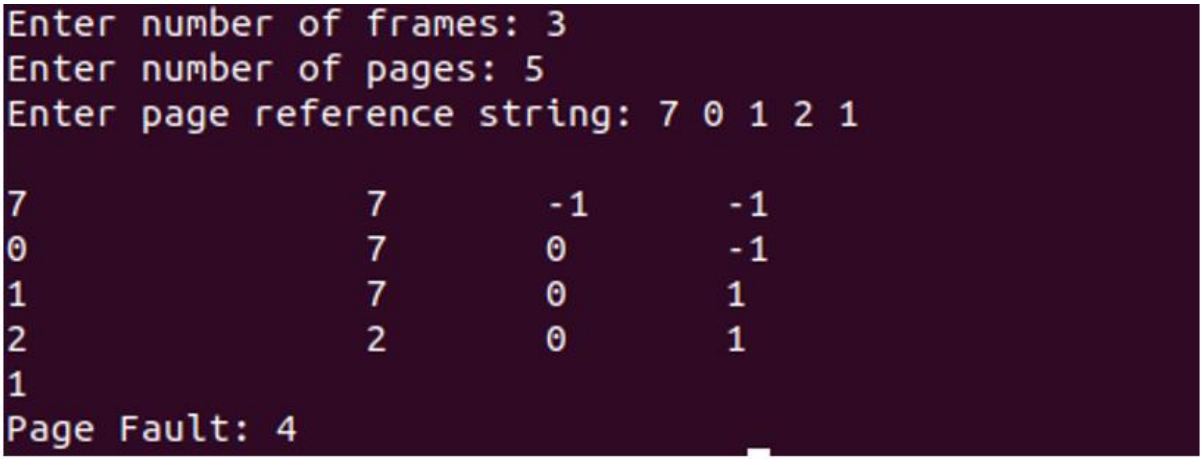
1. Start traversing the pages.
2. Declare the size as the length of the Page.
3. Check the need of the replacement from the page to memory.
4. Check the need of the replacement from the old page to the new page in memory.
5. Form the queue to hold all pages.
6. Insert the required page memory into the queue.
7. Check the bad replacements and page faults.
8. Get the number of processes to be inserted.
9. Show the values.

Code:

```
#include <stdio.h> int main()
{
int i, j = 0, nf, np, pg[50], frame[10], k, av, c = 0;
printf("Enter number of frames: ");
scanf("%d", &nf);
```

```
printf("Enter number of pages: "); scanf("%d", &np);
printf("Enter page reference string: "); for (i = 1; i <= np;
i++){
scanf("%d", &pg[i]);
}

for (i = 0; i < nf; i++){ frame[i] = -1;
}

printf("\n");
for (i = 1; i <= np; i++)
{
printf("%d\t\t", pg[i]); av = 0;
for (k = 0; k < nf; k++) if (frame[k] == pg[i])
av = 1; if (av == 0)
{
frame[j] = pg[i]; j = (j + 1) % nf; c++;
for (k = 0; k < nf; k++) printf("%d\t", frame[k]);
}
printf("\n");
}
printf("Page Fault: %d\n", c); return 0;
}
```

OUTPUT:



Optimal Page Replacement

Algorithm:

1. If the referred page is already present, increment the hit count.

2. If not present, find if a page is never referenced in future. If such a page exists, replace this page with a new page. If no such page exists, find a page that is referenced farthest in future. Replace this page with a new page.

Code:

```
#include<stdio.h>


int main()

{

int nf, np, fr[10], pg[30], tmp[10], f1, f2, f3, i, j, k, pos, max, fl = 0; printf("Enter number of frames: ");

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

```
printf("Enter number of pages: "); scanf("%d", &np);
printf("Enter page reference string: "); for(i = 0; i < np;
++i){
scanf("%d", &pg[i]);
}

for(i = 0; i < nf; ++i){ fr[i] = -1;
}

for(i = 0; i < np; ++i){ f1 = f2 = 0;
for(j = 0; j < nf; ++j){ if(fr[j] == pg[i]){
f1 = f2 = 1;
break;
}
}

if(f1 == 0){ for(j=0; j<nf; ++j){
if(fr[j]==-1){ fl++;
fr[j] = pg[i]; f2 = 1;
break;
}
}
} if(f2==0){ f3 = 0;
for(j=0; j<nf; j++){
tmp[j] = -1;
```

```
for(k=i + 1; k<np; k++){ if(fr[j] == pg[k]){
tmp[j] = k; break;
}
}
}
for(j = 0; j < nf; j++){ if(tmp[j] == -1){ pos = j;
f3 = 1;
break;
}
} if(f3==0){
max = tmp[0]; pos = 0;

for(j=1; j<nf; j++){ if(tmp[j] > max){

max = tmp[j]; pos = j;
}
}
}
fr[pos] = pg[i]; fl++;
}

printf("\n");

for(j=0; j<nf; j++){ printf("%d\t", fr[j]);
```

```
}
}
printf("\n\nTotal Page Faults = %d\n", fl);
return 0;
}
```

```
Enter number of frames: 3
Enter number of pages: 6
Enter page reference string: 2 3 4 2 1 1

2       -1      -1
2       3       -1
2       3       4
2       3       4
1       3       4
1       3       4

Total Page Faults = 4
```

LAB 12

Aim: 1. LRU Page Replacement

2. Comparing FIFO, Optimal & LRU Page Replacement Algorithms

LRU Page Replacement

Algorithm:

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by the counter value
7. Stack them according to the selection.
8. Display the values
9. Stop the process

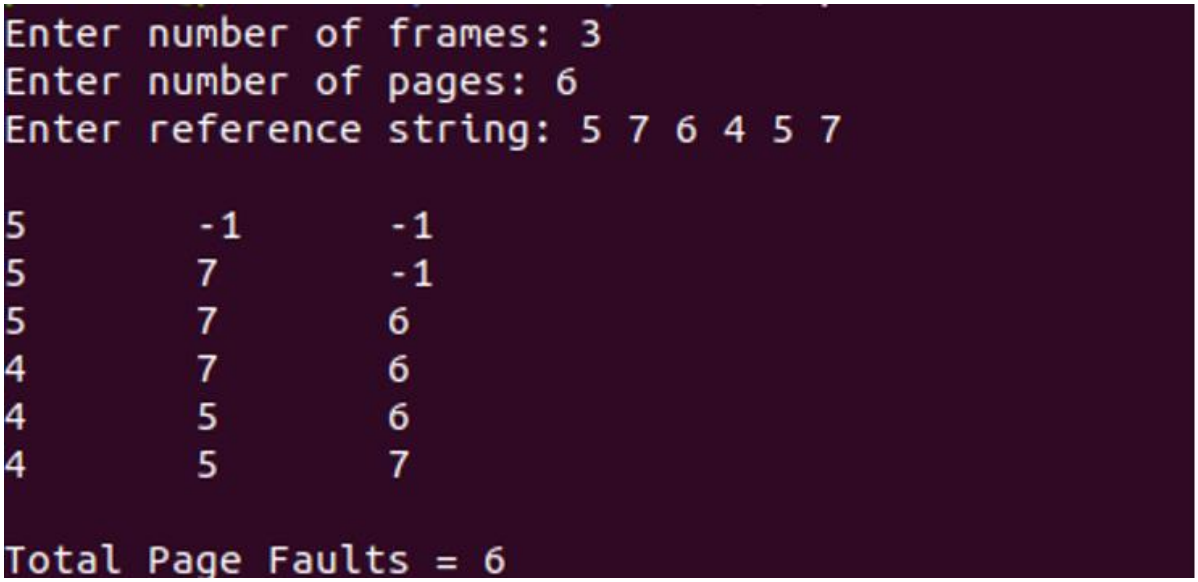
Code:

```
#include <stdio.h>

int LRU(int time[], int n)
{
int i, mn = time[0], p = 0;
```

```
for (i = 1; i < n; ++i)
{
if (time[i] < mn)
{
mn = time[i]; p = i;
}

}
return p;
}

int main()
{
int nf, np, fr[10], pg[30], cn = 0, time[10], f1, f2, i, j, p, fl
= 0; printf("Enter number of frames: ");
scanf("%d", &nf);
printf("Enter number of pages: "); scanf("%d", &np);
printf("Enter reference string: "); for (i = 0; i < np; ++i)
{
scanf("%d", &pg[i]);
}

for (i = 0; i < nf; ++i)
{
```

```
fr[i] = -1;
}

for (i = 0; i < np; ++i)
{
f1 = f2 = 0;

for (j = 0; j < nf; ++j)
{
if (fr[j] == pg[i])
{
cn++; time[j] = cn; f1 = f2 = 1;
break;
}

}

if (f1 == 0)
{
for (j = 0; j < nf; ++j)
{
if (fr[j] == -1)
{
cn++; fl++;
fr[j] = pg[i]; time[j] = cn; f2 = 1;
```

```
break;
}
}
}

if (f2 == 0)
{
p = LRU(time, nf); cn++;
fl++;
fr[p] = pg[i]; time[p] = cn;
}

printf("\n");

for (j = 0; j < nf; ++j)
{
printf("%d\t", fr[j]);
}
}
printf("\n\nTotal Page Faults = %d\n", fl);


return 0;
}
```
OUTPUT:



Comparing FIFO, Optimal & LRU Page Replacement Algorithms Algorithm:

1. Write code for FIFO Page Replacement in a function
2. Write code for Optimal Page Replacement in a function
3. Write code for LRU Page Replacement in a function
4. Execute all 3 in the main function.

Code:

```
#include <stdio.h>


void FIFO(int nf, int np, int pg[], int fr[]){ int i, j = 0, k, av, fl = 0;
for (i = 1; i <= np; i++)
{
printf("%d\t\t", pg[i]); av = 0;
for (k = 0; k < nf; k++) if (fr[k] == pg[i])
```

```
av = 1; if (av == 0)
{
fr[j] = pg[i];
j = (j + 1) % nf; fl++;
for (k = 0; k < nf; k++) printf("%d\t", fr[k]);
}
printf("\n");
}
printf("\n\nTotal Page Faults = %d\n", fl);
}

void optimal(int nf, int np, int pg[], int fr[]){
int tmp[10], f1, f2, f3, i, j, k, pos, max, fl = 0; for(i = 0; i <
np; ++i){

f1 = f2 = 0;
for(j = 0; j < nf; ++j){ if(fr[j] == pg[i]){
f1 = f2 = 1;
break;
}
}
if(f1 == 0){ for(j=0; j<nf; ++j){
if(fr[j]==-1){ fl++;
fr[j] = pg[i]; f2 = 1;
break;
```

```
}
}
} if(f2==0){ f3 = 0;
for(j=0; j<nf; j++){
tmp[j] = -1;

for(k=i + 1; k<np; k++){ if(fr[j] == pg[k]){
tmp[j] = k; break;
}
}
}
for(j = 0; j < nf; j++){ if(tmp[j] == -1){ pos = j;
f3 = 1;
break;
}

} if(f3==0){
max = tmp[0]; pos = 0;

for(j=1; j<nf; j++){ if(tmp[j] > max){ max = tmp[j];
pos = j;
}
}
}
fr[pos] = pg[i]; fl++;
```

```
}

printf("\n");

for(j=0; j<nf; j++){ printf("%d\t", fr[j]);
}
}
printf("\n\nTotal Page Faults = %d\n", fl);
}

int LRU(int time[], int n)
{
int i, mn = time[0], p = 0;

for (i = 1; i < n; ++i)
{
if (time[i] < mn)
{
mn = time[i]; p = i;

}
}
return p;
}
```

```
void LRUpr(int nf, int np, int pg[], int fr[]){ int i, j, cn = 0,
time[10], f1, f2, p, fl = 0; for (i = 0; i < np; ++i)
{
f1 = f2 = 0;

for (j = 0; j < nf; j++)
{
if (fr[j] == pg[i])
{
cn++; time[j] = cn; f1 = f2 = 1;
break;
}
}

if (f1 == 0)
{
for (j = 0; j < nf; j++)
{
if (fr[j] == -1)
{
cn++; fl++;
fr[j] = pg[i]; time[j] = cn; f2 = 1;
break;
}
```

```
}
}

if (f2 == 0)
{
p = LRU(time, nf); cn++;
fl++;
fr[p] = pg[i]; time[p] = cn;
}

printf("\n");

for (j = 0; j < nf; j++)
{
printf("%d\t", fr[j]);
}
}
printf("\n\nTotal Page Faults = %d\n", fl);
}

int main()
{
int nf, np, fr[10], pg[30], i; printf("Enter number of
frames: "); scanf("%d", &nf);
printf("Enter number of pages: "); scanf("%d", &np);
```

```
printf("Enter reference string: "); for (i = 0; i < np; ++i)
{
scanf("%d", &pg[i]);
}

for (i = 0; i < nf; ++i)
{
fr[i] = -1;
}
printf("\n\nFIFO PR\n"); FIFO(nf, np, pg, fr);
printf("\n\nOptimal PR"); optimal(nf, np, pg, fr);
printf("\n\nLRU PR"); LRUpr(nf, np, pg, fr); return 0;
}
```

OUTPUT:

```
E      number of frames: 3
E      number of pages: 6
Enter reference string: 5 4 1 5 2 4


FIFO PR
4               4       -1      -1
1               4       1       -1
5               4       1       5
2               2       1       5
4               2       4       5
0               2       4       0


Total Page Faults = 6


Optimal PR
2       4       5
2       4       5
2       1       5
2       1       5
2       1       5
4       1       5

Total Page Faults = 3


LRU PR
4       1       5
4       1       5
4       1       5
4       1       5
2       1       5
2       4       5
```

LAB 13

Aim: 1. FCFS Disk Scheduling

2. SSTF Disk Scheduling

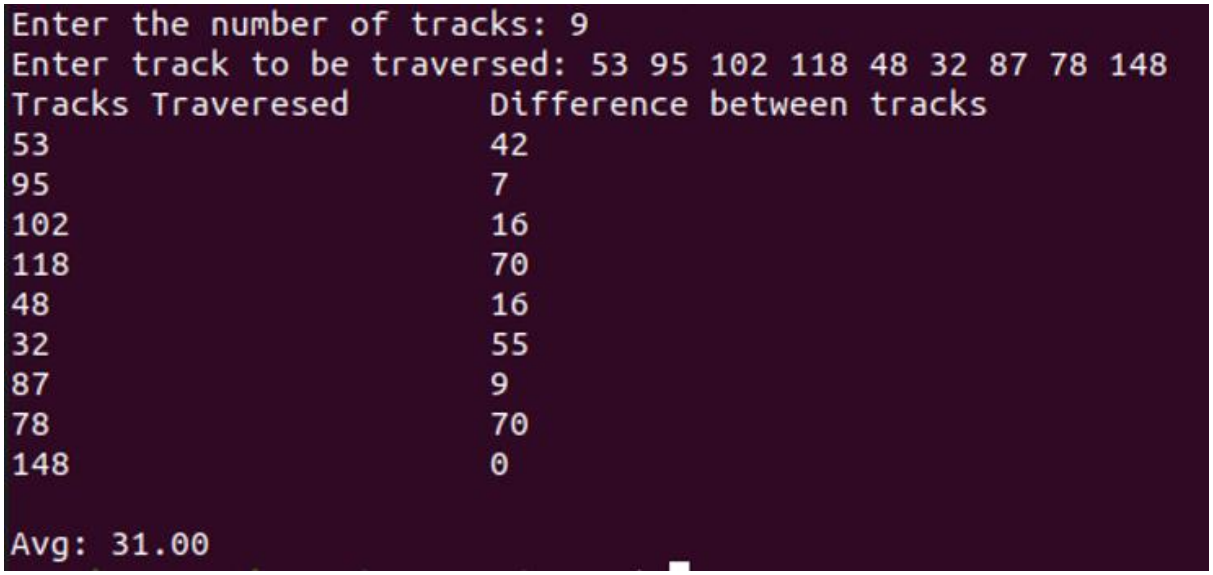
FCFS Disk Scheduling

Code:

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

int main(){
int n, i, tot = 0; double av;
int t[n], tr[n];
printf("Enter the number of tracks: "); scanf("%d", &n);
printf("Enter track to be traversed: "); for(i=0; i<n; i++){
scanf("%d", &t[i]);
}
for(i=0; i<n-1; i++){
tr[i] = abs(t[i+1] - t[i]); tot += tr[i];
}
av = tot/n;
printf("Tracks Traveresed\tDifference between tracks\n");
for(i=0; i<n; i++){
printf("%d\t\t\t%d\n", t[i], tr[i]);
}
printf("\nAvg: %.2f\n", av);
```

}

OUTPUT:



SSTF Disk Scheduling

Code:

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

int main()

{

int i, n, tot=0, h, cn=0, d, j, m; printf("Enter the number of requests: "); scanf("%d",&n);

int r[n];

printf("Enter the requests sequence: "); for(i=0; i<n; i++){

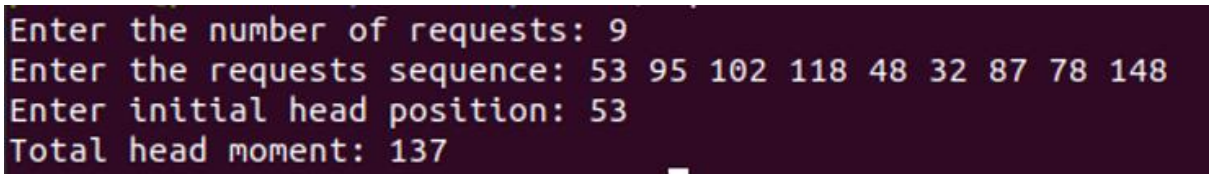
scanf("%d",&r[i]);

}

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

```
while(cn!=n)
{
m = INT_MAX;
for(i=0; i<n; i++)
{
d = abs(r[i] - h); if(d<m)
{
m = d; j = i;
}
}
tot += m; h = r[j];
r[j] = INT_MAX;
cn++;
}

printf("Total head moment: %d\n",tot); return 0;

}
```

OUTPUT:



LAB 14

Aim: 1. Scan Disk Scheduling

2. C-Scan Disk Scheduling

Scan Disk Scheduling

Code:

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

int main()
{
int i , j, n, tot=0, h, s, dir, tmp, pos; printf("Enter the number of requests: "); scanf("%d",&n);
int rq[n];
printf("Enter the requests sequence: "); for(i=0;i<n;i++){
scanf("%d",&rq[i]);
}
printf("Enter initial head position: "); scanf("%d",&h);
printf("Enter total disk size: "); scanf("%d",&s);
printf("Enter the head movement direction for high 1 and for low 0: "); scanf("%d",&dir);

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

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

if(rq[j]>rq[j+1])
{
tmp=rq[j]; rq[j]=rq[j+1]; rq[j+1]=tmp;
}

}
}

for(i=0; i<n; i++)
{
if(h<rq[i])
{
pos=i; break;
}
}

if(dir == 1)
{
for(i=pos; i<n; i++)
{
tot += abs(rq[i]-h); h=rq[i];
```

```
}
tot += abs(s-rq[i-1]-1); h = s-1;
for(i=pos-1; i>=0; i--)
{
tot += abs(rq[i]-h); h=rq[i];
}
}
else if(dir == 0)

{
for(i=pos-1; i>=0; i--)
{
tot += abs(rq[i]-h); h=rq[i];
}
tot += abs(rq[i+1]-0); h =0;
for(i=pos; i<n; i++)
{




}
}
else{

tot += abs(rq[i]-h); h=rq[i];
```

```
printf("Invalid Input!!");
}

printf("Total head movement: %d\n",tot);
return 0;
}
```

OUTPUT:

```
Enter the number of requests: 8
Enter the requests sequence: 95 180 34 119 11 123 62 64
Enter initial head position: 50
Enter total disk size: 200
Enter the head movement direction for high 1 and for low 0: 1
Total head movement: 337
```

C-Scan Disk Scheduling

Code: #include<stdio.h> #include<stdlib.h> int main()

{

int i, j, n, tot=0, h, s, dir, tmp, pos; printf("Enter the number of requests: "); scanf("%d",&n);

int rq[n];

printf("Enter the requests sequence: "); for(i=0;i<n;i++){

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

}

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

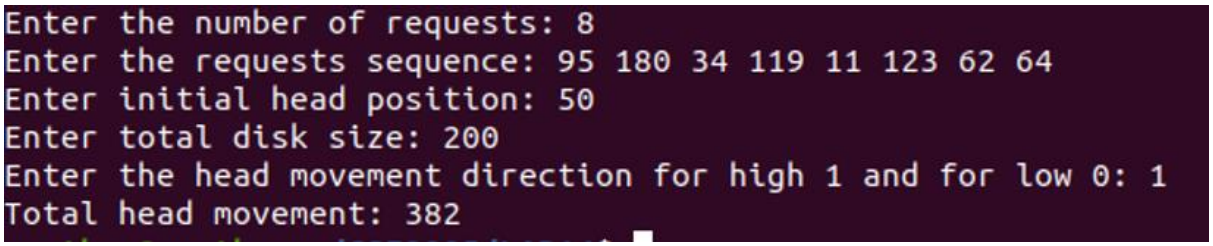
printf("Enter total disk size: "); scanf("%d",&s);

printf("Enter the head movement direction for high 1 and for low 0: "); scanf("%d",&dir);

```
for(i=0; i<n; i++)
{
for(j=0; j<n-i-1; j++)
{
if(rq[j]>rq[j+1])
{
tmp=rq[j]; rq[j]=rq[j+1]; rq[j+1]=tmp;
}

}
}

for(i=0; i<n; i++)
{
if(h<rq[i])
{
pos = i; break;
}
}

if(dir == 1)
{
for(i=pos; i<n; i++)
{
tot += abs(rq[i]-h); h=rq[i];
```

```
}
tot += abs(s-rq[i-1]-1) + abs(s-1); h=0;
for( i=0;i<pos;i++)
{
tot += abs(rq[i]-h); h=rq[i];

}
}
else if(dir == 0)
{
for(i=pos-1; i>=0; i--)
{
tot += abs(rq[i]-h); h=rq[i];
}
tot += abs(rq[i+1]-0) + abs(s-1); h = s-1;
for(i=n-1; i>=pos; i--)

{
tot += abs(rq[i]-h); h=rq[i];
}
}

printf("Total head movement: %d\n",tot);
return 0;
}
```

OUTPUT:


Enter the number of requests: 8
Enter the requests sequence: 95 180 34 119 11 123 62 64
Enter initial head position: 50
Enter total disk size: 200
Enter the head movement direction for high 1 and for low 0: 1
Total head movement: 382