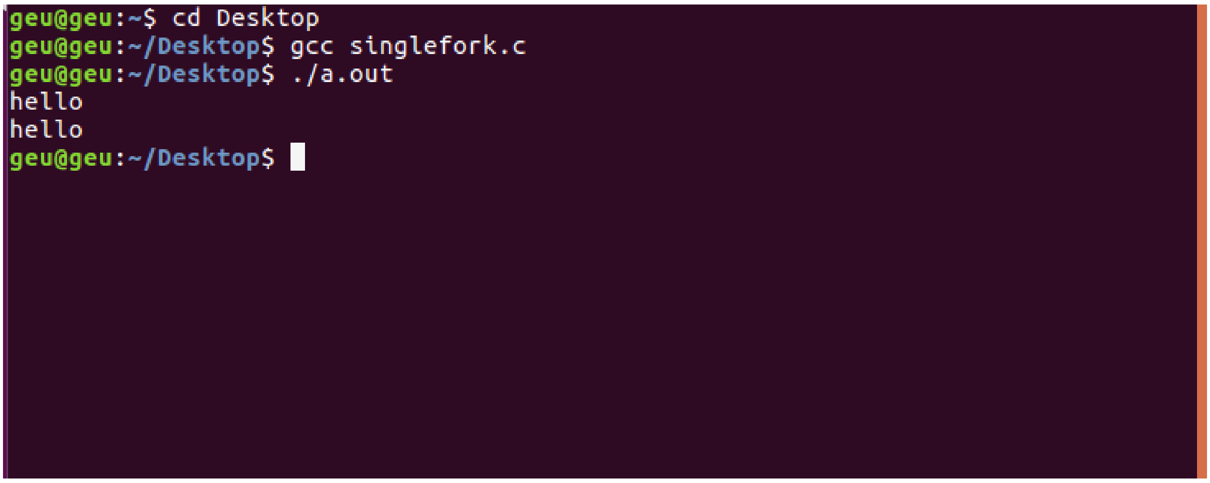
WEEK - 1

Program 1.a) Implementation of single fork( )

```
#include<stdio.h>
#include<unistd.h>
int main()
 {
   fork();
   printf("hello");
   return 0;
 }
```

OUTPUT

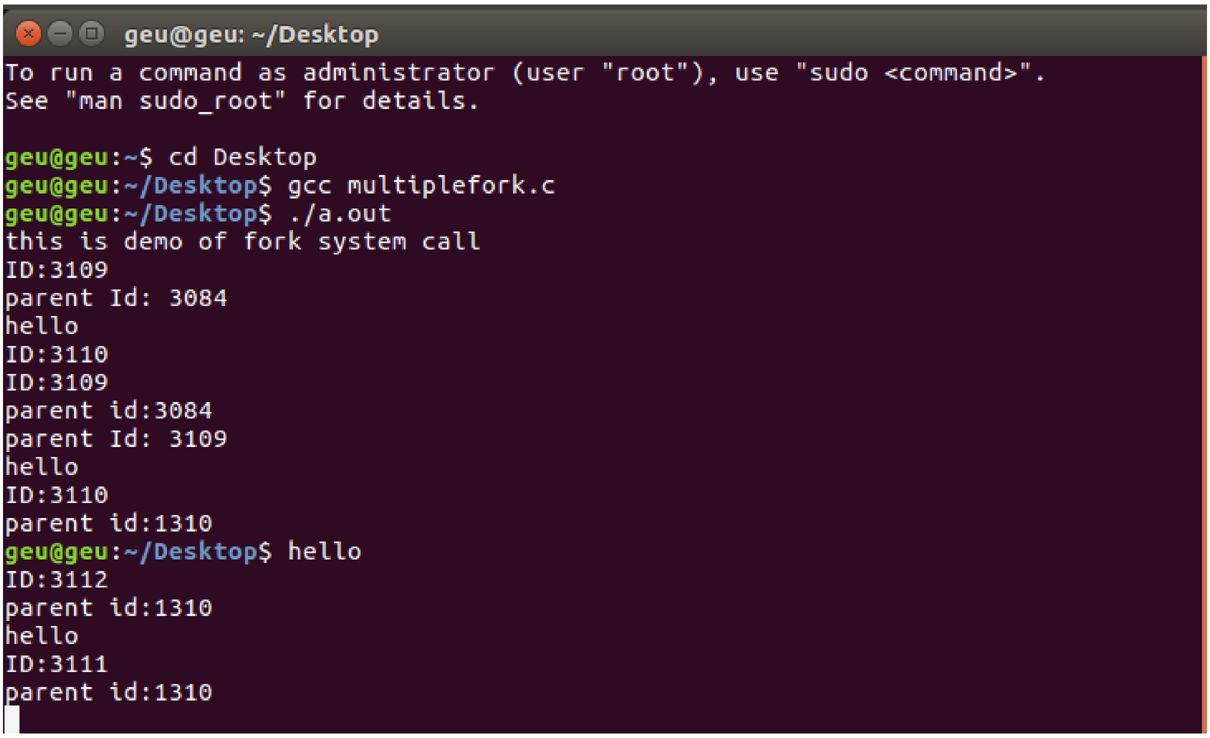


```
geu@geu:~$ cd Desktop
geu@geu:~/Desktop$ gcc singlefork.c
geu@geu:~/Desktop$ ./a.out
hello
hello
geu@geu:~/Desktop$ 
```

Program 1.b) Implementation of multiple fork( )

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    printf("this is demo of fork system call\n");
    fork();
    printf("ID:%d\n",getpid());
    printf("parent Id: %d\n",getppid());
    fork();
    printf("hello\n");
    printf("ID:%d\n",getpid());
    printf("parent id:%d\n",getppid());
    return 0;
}
```

OUTPUT

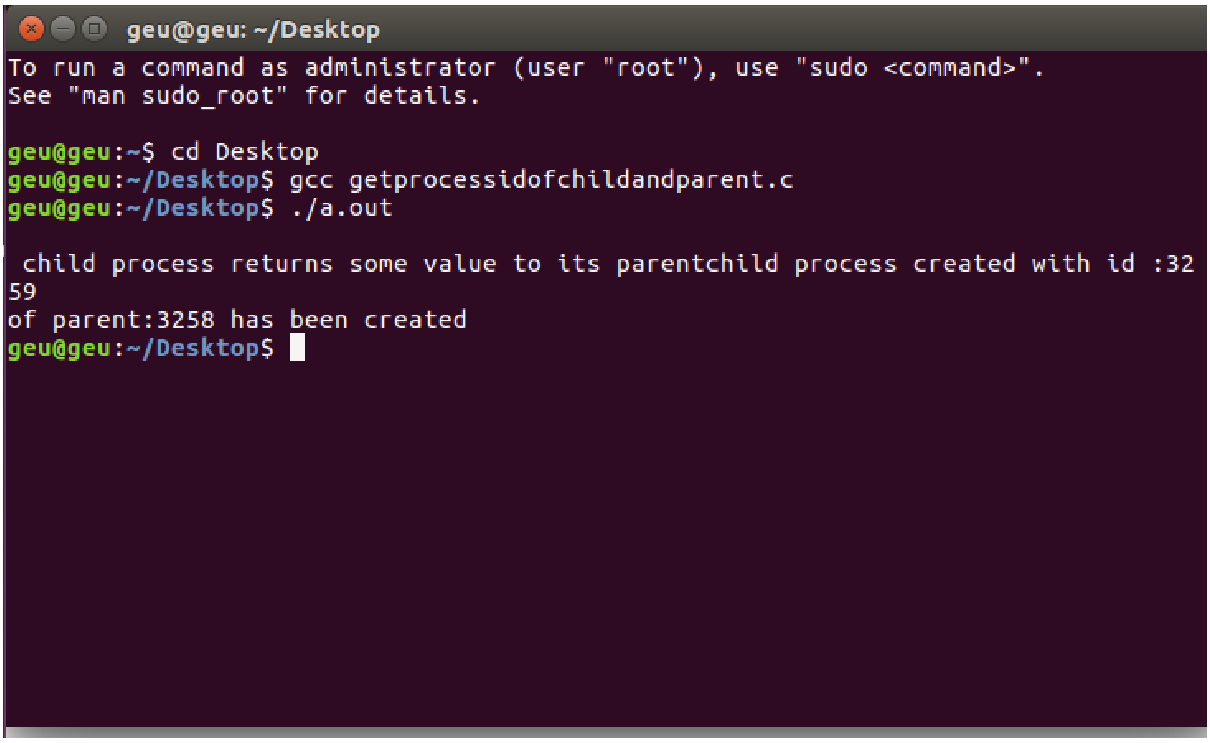


```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

geu@geu:~$ cd Desktop
geu@geu:~/Desktop$ gcc multiplefork.c
geu@geu:~/Desktop$ ./a.out
this is demo of fork system call
ID:3109
parent Id: 3084
hello
ID:3110
ID:3109
parent id:3084
parent Id: 3109
hello
ID:3110
parent id:1310
geu@geu:~/Desktop$ hello
ID:3112
parent id:1310
hello
ID:3111
parent id:1310
```

Program 1.c) Get process ID's of Child and Parent process

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    int pid;
    pid=fork();
    if(pid==0)
    {
        printf("child process created with id :%d\n",getpid());
        printf("of parent:%d has been created\n",getppid());
    }
    if(pid>0)
    {
        printf("\n child process returns some value to its parent");
    }
    return 0;
}
```

OUTPUT

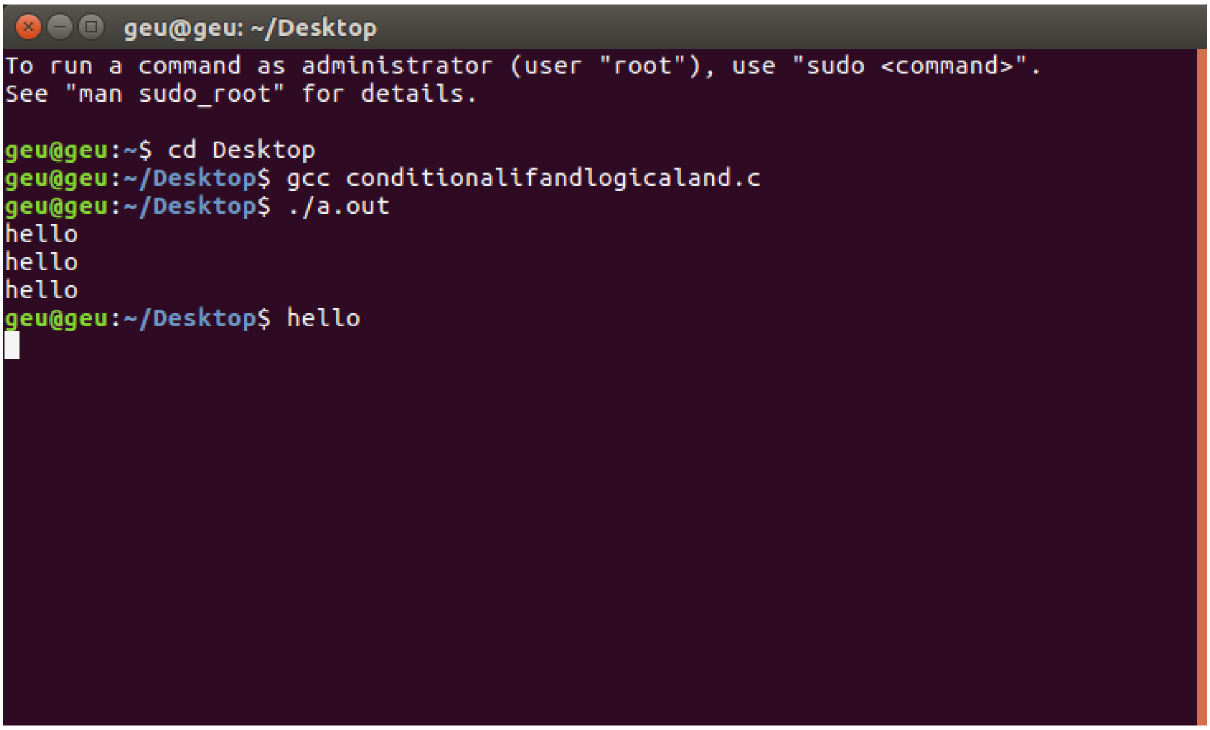


```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

geu@geu:~$ cd Desktop
geu@geu:~/Desktop$ gcc getprocessidofchildandparent.c
geu@geu:~/Desktop$ ./a.out

 child process returns some value to its parentchild process created with id :32
59
of parent:3258 has been created
geu@geu:~/Desktop$ 
```

Program 1.d) Fork with conditional if and logical AND operator (&&)

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    if(fork() && fork())
    {
        fork();
    }
    printf("hello \n");
    return 0;
}
```

OUTPUT

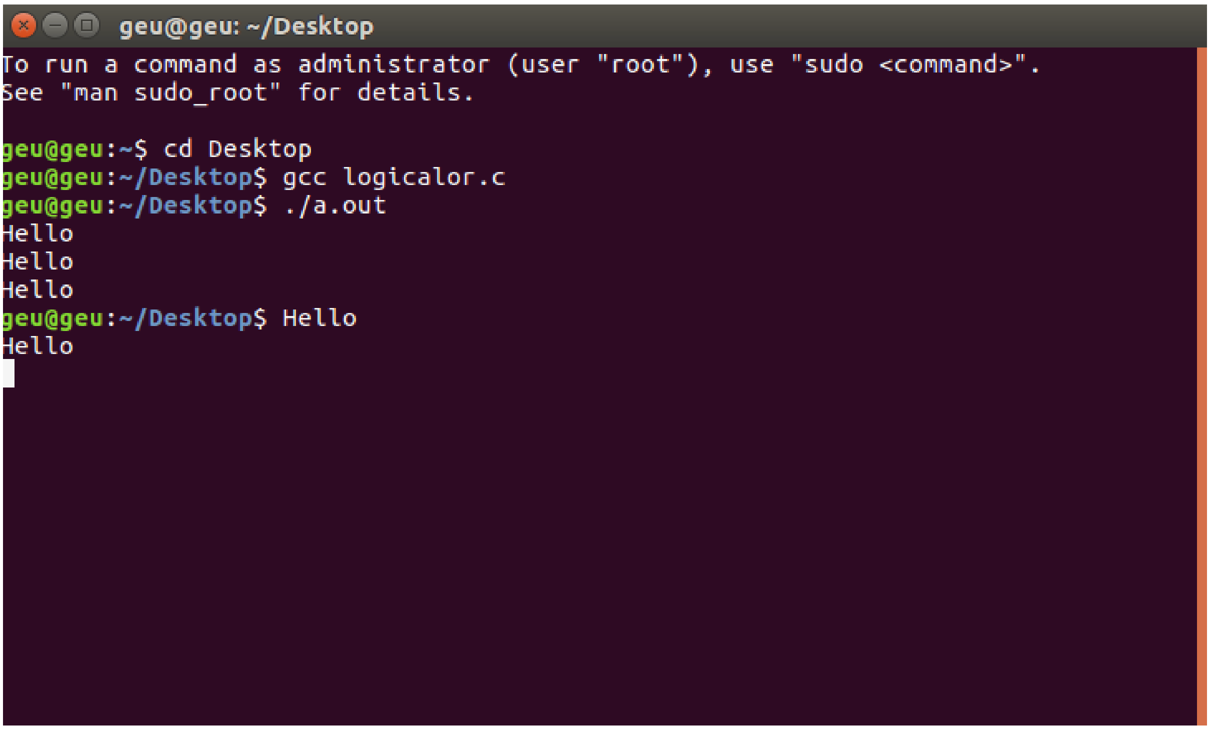


```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

geu@geu:~$ cd Desktop
geu@geu:~/Desktop$ gcc conditionalifandlogicaland.c
geu@geu:~/Desktop$ ./a.out
hello
hello
hello
geu@geu:~/Desktop$ hello
```

Program 1.e) Fork with conditional if and logical OR operator (||)

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    if(fork() || fork())
    {
        fork();
    }
        printf("Hello \n");
        return 0;
}
```

OUTPUT

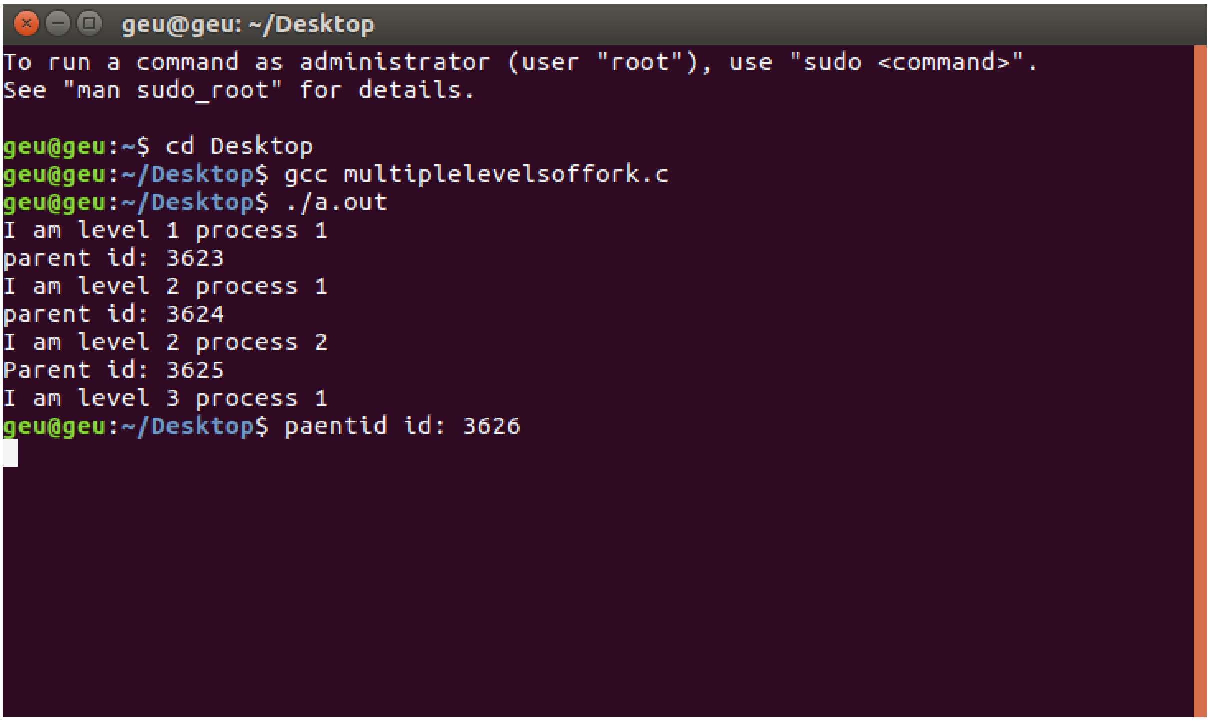


```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

geu@geu:~$ cd Desktop
geu@geu:~/Desktop$ gcc logicalor.c
geu@geu:~/Desktop$ ./a.out
Hello
Hello
Hello
geu@geu:~/Desktop$ Hello
Hello
```

Program 1.f) Demonstrate multiple levels of fork system call with their process id's

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    int id1=fork(), id2=fork();
    if(id1>0 && id2>0)
    {
        printf("I am level 1 process 1\n");
        printf("parent id: %d\n",getpid());
    }
    else if(id1==0 && id2>0)
    {
        printf("I am level 2 process 1\n");
        printf("parent id: %d\n",getpid());
    }
    else if(id1>0 && id2==0)
    {
        printf("I am level 2 process 2\n");
        printf("Parent id: %d\n",getpid());
    }
    else if(id1==0 && id2==0)
    {
        printf("I am level 3 process 1\n");
        printf("paentid id: %d\n",getpid());
    }
    else
    {
        printf("unsuccessful creation of process\n");
    }
}
```

OUTPUT



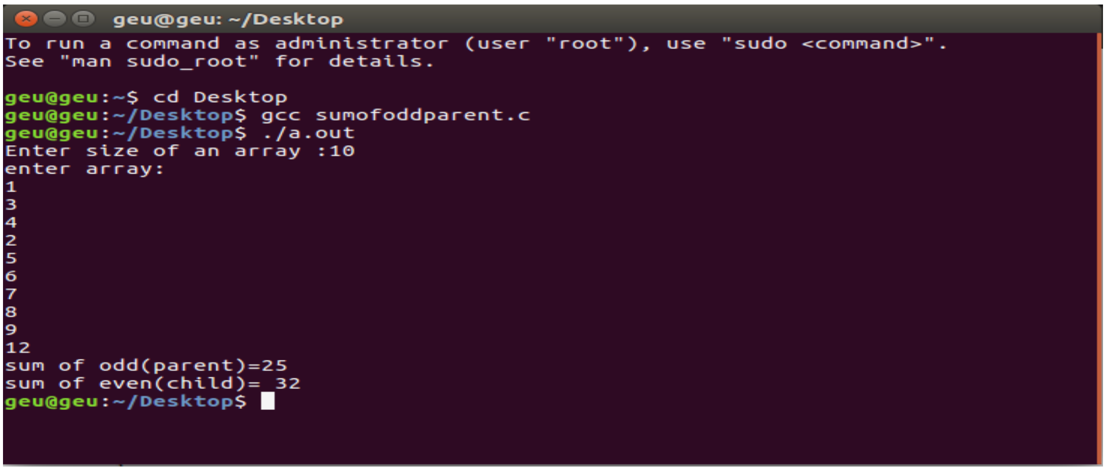
```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

geu@geu:~$ cd Desktop
geu@geu:~/Desktop$ gcc multiplelevelsoffork.c
geu@geu:~/Desktop$ ./a.out
I am level 1 process 1
parent id: 3623
I am level 2 process 1
parent id: 3624
I am level 2 process 2
Parent id: 3625
I am level 3 process 1
geu@geu:~/Desktop$ paentid id: 3626
```

WEEK - 2

Program 2) Implement a program that computers sum of odd numbers in parent process and sum of even numbers in child process using fork system call.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
    int arr[10];
    int a,n,sume=0,sumo=0;
    printf("Enter size of an array :");
    scanf("%d",&n);
    printf("enter array:\n");
    for(int i=0;i<n;i++) {
        scanf("%d",&arr[i]);
    }
    a=fork();
    if(a>0){
        for(int i=0;i<n;i++){
            if(arr[i]%2!=0){
                sumo=sumo+arr[i];
            }
        }
        printf("sum of odd(parent)=%d\n",sumo);
        exit(0);
    }
    else if(a==0){
        for(int i=0;i<n;i++) {
            if(arr[i]%2==0){
                sume=sume+arr[i];
            }
        }
        printf("sum of even(child)= %d\n",sume);
        exit(0);
    }
    else
    printf("unsuccessful ceation of process\n");
}
```

OUTPUT



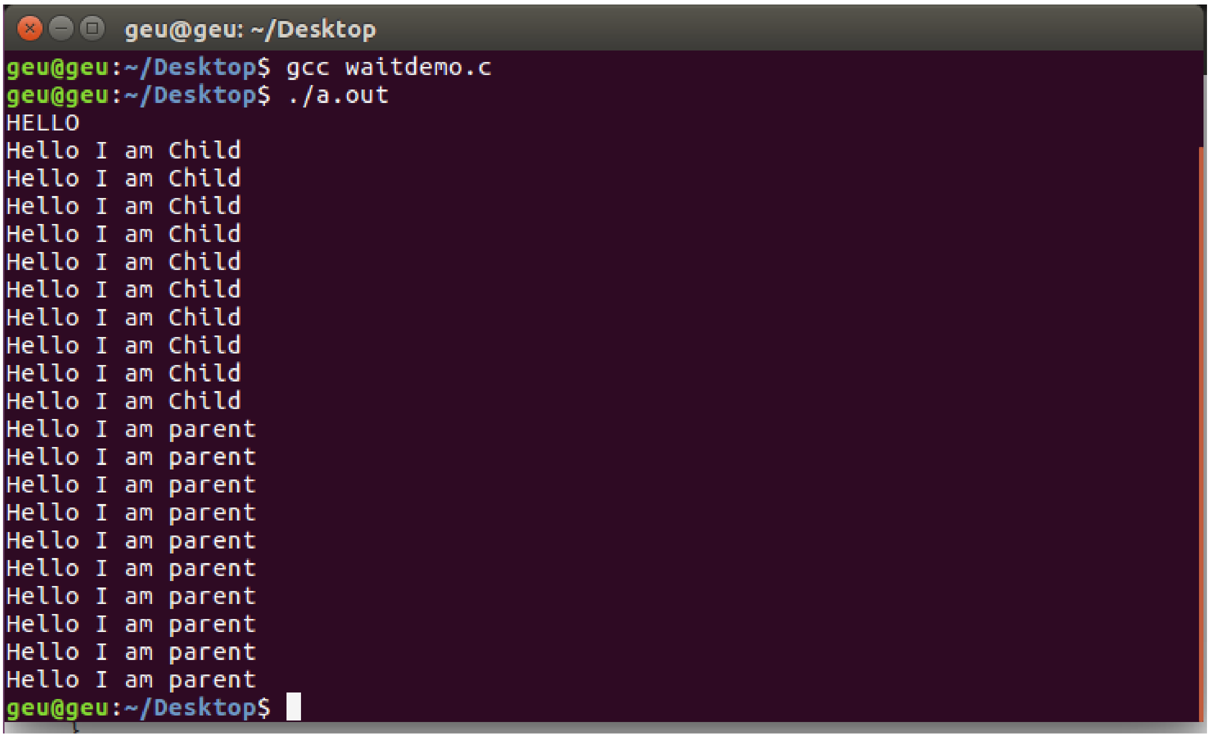
```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

geu@geu:~$ cd Desktop
geu@geu:~/Desktop$ gcc sumofoddparent.c
geu@geu:~/Desktop$ ./a.out
Enter size of an array :10
enter array:
1
3
4
2
5
6
7
8
9
12
sum of odd(parent)=25
sum of even(child)= 32
geu@geu:~/Desktop$
```

WEEK - 3

Program 3) Demonstrate the working of wait() system call

```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
int main()
{
    pid_t id;
    int i;
    id=fork();
    if(id==0)
    {
        for(i=0;i<10;i++)
        {
            printf("Hello I am Child\n");
        }
    }
    else if(id>0)
    {
        printf("HELLO\n");
        wait(NULL);
        for(i=0;i<10;i++)
        {
            printf("Hello I am parent\n");
        }
    }
}
```

OUTPUT



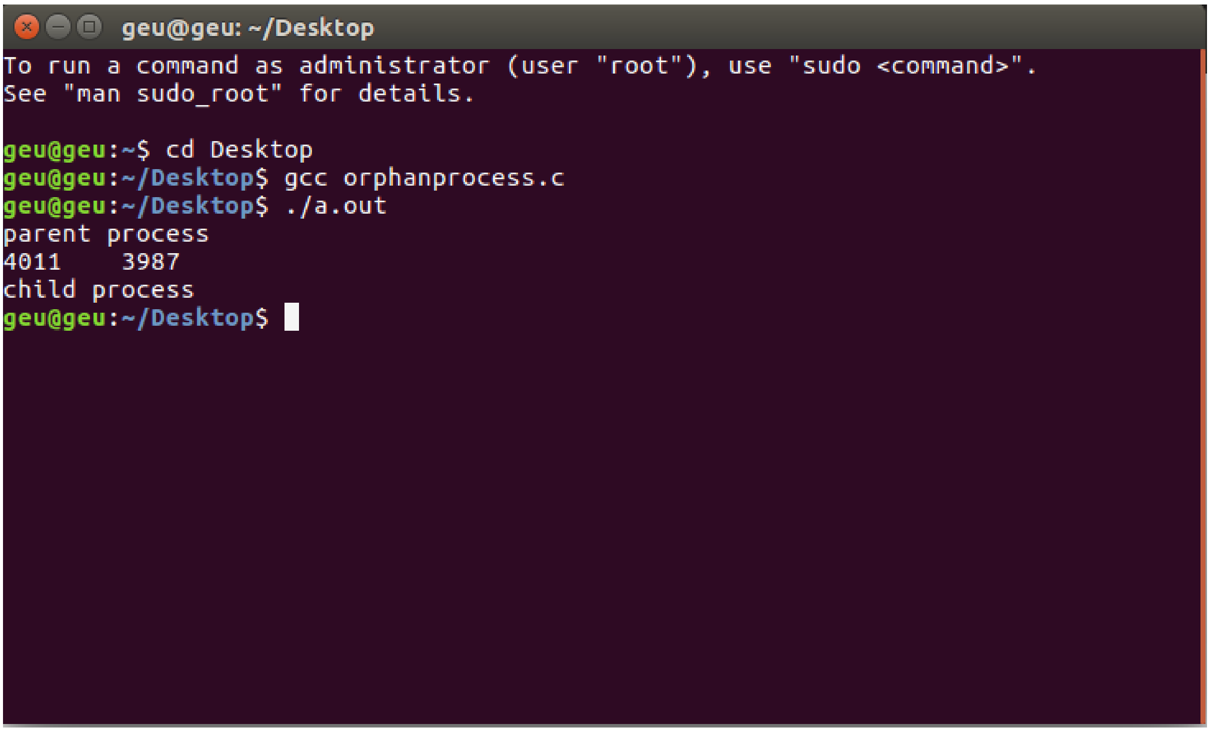
```
geu@geu:~/Desktop$ gcc waitdemo.c
geu@geu:~/Desktop$ ./a.out
HELLO
Hello I am Child
Hello I am Child
Hello I am Child
Hello I am Child
Hello I am Child
Hello I am Child
Hello I am Child
Hello I am Child
Hello I am Child
Hello I am Child
Hello I am parent
Hello I am parent
Hello I am parent
Hello I am parent
Hello I am parent
Hello I am parent
Hello I am parent
Hello I am parent
Hello I am parent
Hello I am parent
geu@geu:~/Desktop$
```

WEEK - 4

Program 4.a) Demonstrate Orphan process

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>
int main()
{
    pid_t id;
    id=fork();
    if(id>0)
    {
        printf("parent process\n");
        printf("%d\t%d\n",getpid(),getppid());
        exit(0);
        printf("ABC");
    }
    else if(id==0)
    {
        printf("child process\n");
        sleep(50);
        printf("%d\t%d\n",getpid(),getppid());
    }
}
```

OUTPUT

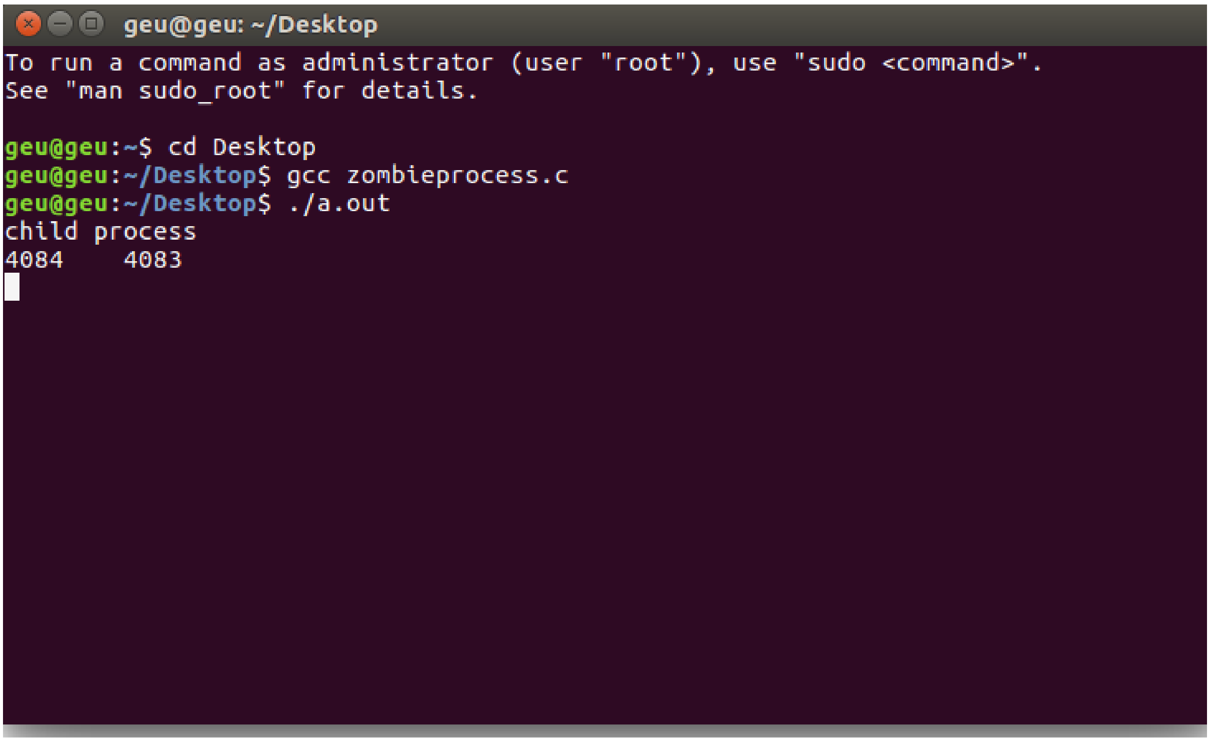


```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

geu@geu:~$ cd Desktop
geu@geu:~/Desktop$ gcc orphanprocess.c
geu@geu:~/Desktop$ ./a.out
parent process
4011    3987
child process
geu@geu:~/Desktop$ 
```

Program 4.b) Demonstrate Zombie process

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>
int main()
{
    pid_t id;
    id=fork();
    if(id>0)
    {
        sleep(50);
        printf("parent process\n");
        printf("%d\t%d\n",getpid(),getppid());
    }
    else if(id==0)
    {
        printf("child process\n");

        printf("%d\t%d\n",getpid(),getppid());
    }
}
```

OUTPUT



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

geu@geu:~$ cd Desktop
geu@geu:~/Desktop$ gcc zombieprocess.c
geu@geu:~/Desktop$ ./a.out
child process
4084    4083
```

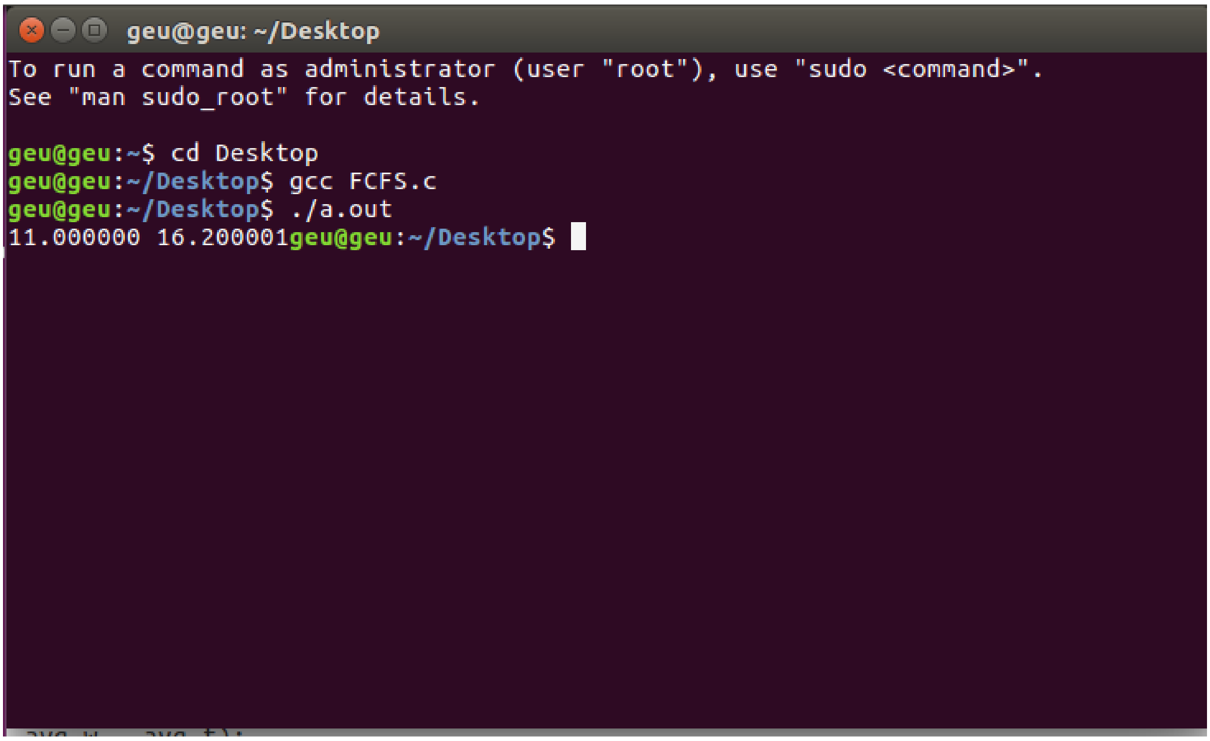
WEEK - 5

Program 5.a) Implement FCFS (First Come First Served) Scheduling Algorithm

Approach 1

```
#include <stdio.h>
#include <unistd.h>
int main()
{
int arrival[5] = {0 , 0 , 0 , 0 , 0};
int burst[5] = {7 , 2 , 8 , 5 , 4};
int wait[5] = {0 , 0 , 0 , 0 , 0};
int tat[5] = {0 , 0 , 0 , 0 , 0};
int i , sum_wait=0 , sum_tat=0;
float avg_w , avg_t;
for(i=1 ; i<5 ; i++)
{
wait[i] = wait[i-1]+burst[i-1];
}
for(i=0 ; i<5 ; i++)
{
tat[i] = wait[i] + burst[i];
sum_wait+=wait[i];
sum_tat+=tat[i];
}
avg_w = sum_wait/5.0;
avg_t = sum_tat/5.0;
printf("%f %f",avg_w , avg_t);
return 0;
}
```

OUTPUT

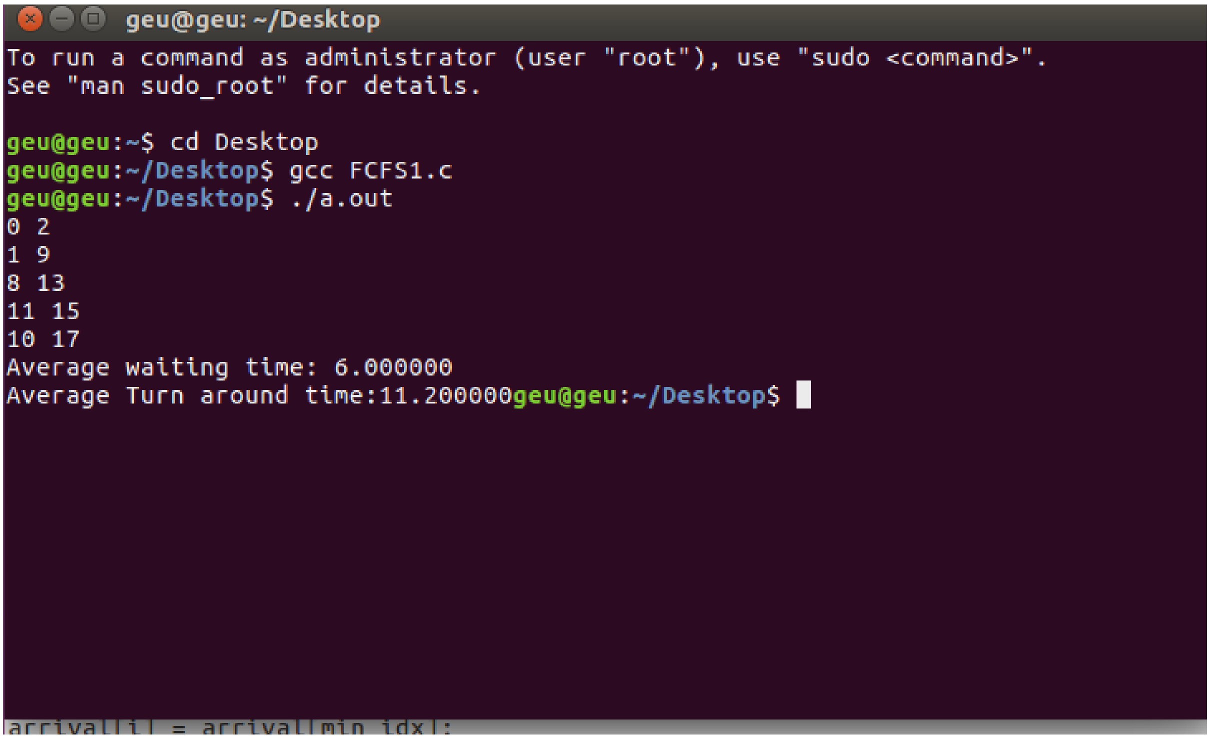


```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

geu@geu:~$ cd Desktop
geu@geu:~/Desktop$ gcc FCFS.c
geu@geu:~/Desktop$ ./a.out
11.000000 16.200001geu@geu:~/Desktop$ 
```

Approach 2

```
#include <stdio.h>
//FCFS
int main()
{
int arrival[5] = {5,0,1,1,2};
int burst[5] = {7,2,8,5,4};
int wait[5] = {0,0,0,0,0};
int tat[5] = {0,0,0,0,0};
int sum_wait=0 , sum_tat=0;
//selection sort
for(int i=0 ; i<5-1 ; i++)
{
int min_idx = i;
for(int j=i+1 ; j<5 ; j++)
{
if(arrival[j]<arrival[min_idx])
{
min_idx = j;
}
}
if(min_idx!=i)
{
int temp = arrival[i];
arrival[i] = arrival[min_idx];
arrival[min_idx] = temp;
temp = burst[i];
burst[i] = burst[min_idx];
burst[min_idx] = temp;
}
}
wait[0] = 0;
tat[0] = burst[0];
sum_wait+=wait[0];
sum_tat+=tat[0];
printf("%d %d\n",wait[0] , tat[0]);
for(int i=1 ; i<5 ; i++)
{
wait[i] = wait[i-1] + burst[i-1] - arrival[i];
tat[i] = wait[i] + burst[i];
printf("%d %d\n",wait[i] , tat[i]);
sum_wait+=wait[i];
sum_tat+=tat[i];
}
printf("Average waiting time: %f\nAverage Turn around
time:%f",sum_wait/5.0 , sum_tat/5.0);
return 0;
}
```

OUTPUT



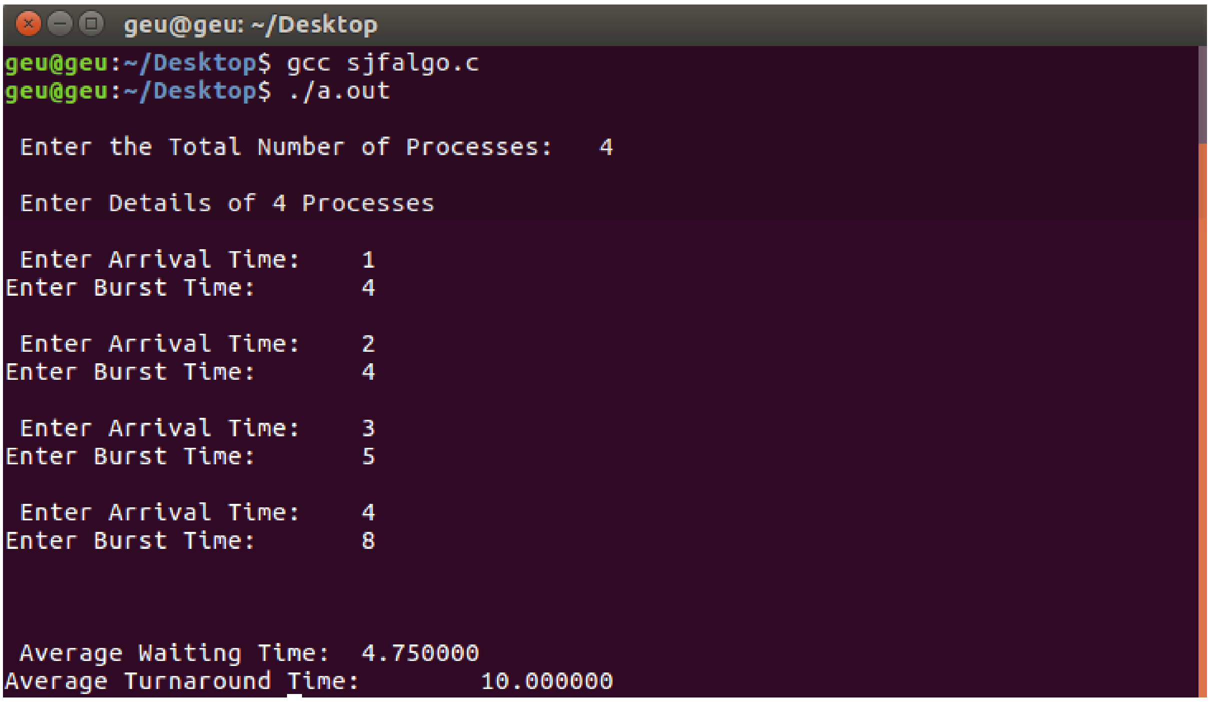
```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

geu@geu:~$ cd Desktop
geu@geu:~/Desktop$ gcc FCFS1.c
geu@geu:~/Desktop$ ./a.out
0 2
1 9
8 13
11 15
10 17
Average waiting time: 6.000000
Average Turn around time:11.200000geu@geu:~/Desktop$ 
```

```
arrival[i] = arrival[min_idx];
```

Program 5.b) Implement SJF (Shortest Job First) Scheduling Algorithm

```
#include <stdio.h>
int main()
{
      int arrival_time[10], burst_time[10], temp[10];
      int i, smallest, count = 0, time, limit;
      double wait_time = 0, turnaround_time = 0, end;
      float average_waiting_time, average_turnaround_time;
      printf("\n Enter the Total Number of Processes:\t");
      scanf("%d", &limit);
      printf("\n Enter Details of %d Processes \n", limit);
      for(i = 0; i < limit; i++)
      {
            printf("\n Enter Arrival Time:\t");
            scanf("%d", &arrival_time[i]);
            printf("Enter Burst Time:\t");
            scanf("%d", &burst_time[i]);
            temp[i] = burst_time[i];
      }
      burst_time[9] = 9999;
      for(time = 0; count != limit; time++)
      {
            smallest = 9;
            for(i = 0; i < limit; i++)
            {
                  if(arrival_time[i] <= time && burst_time[i] <
burst_time[smallest] && burst_time[i] > 0)
                  {
                        smallest = i;
                  }
            }
            burst_time[smallest]--;
            if(burst_time[smallest] == 0)
            {
                  count++;
                  end = time + 1;
                  wait_time = wait_time + end - arrival_time[smallest]
- temp[smallest];
                  turnaround_time = turnaround_time + end -
arrival_time[smallest];
            }
      }
      average_waiting_time = wait_time / limit;
      average_turnaround_time = turnaround_time / limit;
      printf("\n\n\n Average Waiting Time:\t%lf\n",
average_waiting_time);
      printf("Average Turnaround Time:\t%lf\n",
average_turnaround_time);
      return 0;
}
```

OUTPUT



```
geu@geu:~/Desktop$ gcc sjfalgo.c
geu@geu:~/Desktop$ ./a.out

 Enter the Total Number of Processes:   4

 Enter Details of 4 Processes

 Enter Arrival Time:    1
Enter Burst Time:       4

 Enter Arrival Time:    2
Enter Burst Time:       4

 Enter Arrival Time:    3
Enter Burst Time:       5

 Enter Arrival Time:    4
Enter Burst Time:       8



 Average Waiting Time:  4.750000
Average Turnaround Time:        10.000000
```

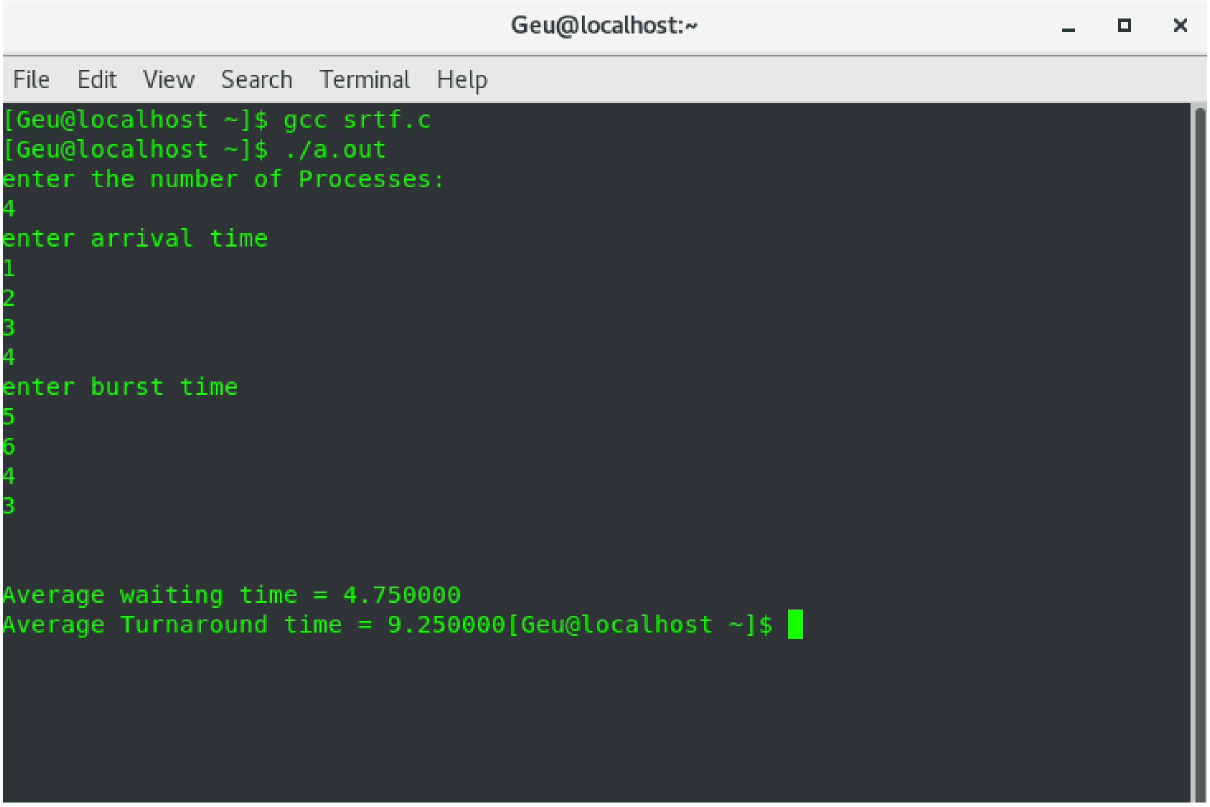
WEEK – 6

Program 6) Implement SRTF (Shortest Remaining Time First) Scheduling Algorithm

```
#include <stdio.h>
int main()
{
 int a[10],b[10],x[10],i,j,smallest,count=0,time,n;
 double avg=0,tt=0,end;
  printf("enter the number of Processes:\n");
  scanf("%d",&n);
 printf("enter arrival time\n");
 for(i=0;i<n;i++)
 scanf("%d",&a[i]);
 printf("enter burst time\n");
 for(i=0;i<n;i++)
 scanf("%d",&b[i]);
 for(i=0;i<n;i++)
 x[i]=b[i];
  b[9]=9999;
 for(time=0;count!=n;time++)
 {
   smallest=9;
  for(i=0;i<n;i++)
  {
   if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )
   smallest=i;
  }
  b[smallest]--;
  if(b[smallest]==0)
  {
   count++;
   end=time+1;
   avg=avg+end-a[smallest]-x[smallest];
```

```
     tt= tt+end-a[smallest];
    }
  }
 printf("\n\nAverage waiting time = %lf\n",avg/n);
    printf("Average Turnaround time = %lf",tt/n);
    return 0;
}
```

OUTPUT



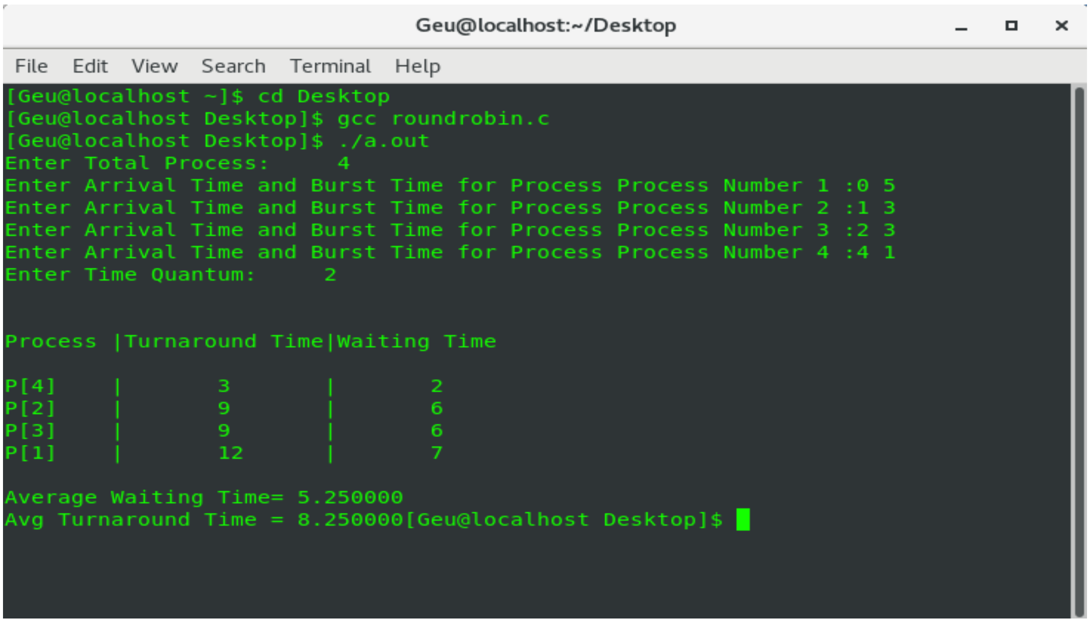
```
[Geu@localhost ~]$ gcc srtf.c
[Geu@localhost ~]$ ./a.out
enter the number of Processes:
4
enter arrival time
1
2
3
4
enter burst time
5
6
4
3


Average waiting time = 4.750000
Average Turnaround time = 9.250000[Geu@localhost ~]$ 
```

WEEK – 7

Program 7) Implement Round Robin Scheduling Algorithm

```
#include<stdio.h>

int main()

{

  int cnt,j,n,t,remain,flag=0,tq;

  int wt=0,tat=0,at[10],bt[10],rt[10];

  printf("Enter Total Process:\t ");

  scanf("%d",&n);

  remain=n;

  for(cnt=0;cnt<n;cnt++)

  {

    printf("Enter Arrival Time and Burst Time for Process Process
Number %d :",cnt+1);

    scanf("%d",&at[cnt]);

    scanf("%d",&bt[cnt]);

    rt[cnt]=bt[cnt];

  }

  printf("Enter Time Quantum:\t");

  scanf("%d",&tq);

  printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");

  for(t=0,cnt=0;remain!=0;)

  {

    if(rt[cnt]<=tq && rt[cnt]>0)

    {

      t+=rt[cnt];

      rt[cnt]=0;

      flag=1;

    }

    else if(rt[cnt]>0)

    {

      rt[cnt]-=tq;

      t+=tq;
    }
    if(rt[cnt]==0 && flag==1)
    {
      remain--;
      printf("P[%d]\t|\t%d\t|\t%d\n",cnt+1,t-at[cnt],t-at[cnt]-
bt[cnt]);
      wt+=t-at[cnt]-bt[cnt];
      tat+=t-at[cnt];
      flag=0;
    }
    if(cnt==n-1)
      cnt=0;
    else if(at[cnt+1]<=t)
      cnt++;
    else
      cnt=0;
  }
  printf("\nAverage Waiting Time= %f\n",wt*1.0/n);
  printf("Avg Turnaround Time = %f",tat*1.0/n);
  return 0;
}
```

OUTPUT



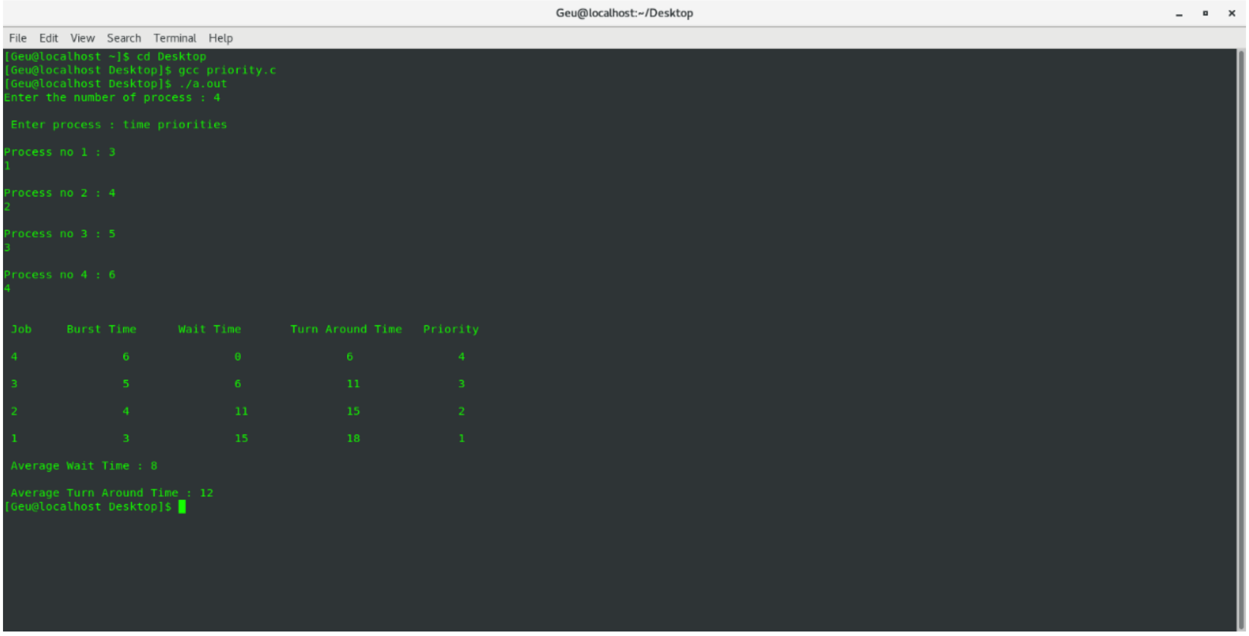
```
[Geu@localhost ~]$ cd Desktop
[Geu@localhost Desktop]$ gcc roundrobin.c
[Geu@localhost Desktop]$ ./a.out
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Process Number 1 :0 5
Enter Arrival Time and Burst Time for Process Process Number 2 :1 3
Enter Arrival Time and Burst Time for Process Process Number 3 :2 3
Enter Arrival Time and Burst Time for Process Process Number 4 :4 1
Enter Time Quantum:      2


Process |Turnaround Time|Waiting Time

P[4]    |       3       |       2
P[2]    |       9       |       6
P[3]    |       9       |       6
P[1]    |       12      |       7

Average Waiting Time= 5.250000
Avg Turnaround Time = 8.250000[Geu@localhost Desktop]$ 
```

WEEK – 8

Program 8) Implement Priority Scheduling Algorithm.

```
#include<stdio.h>
int main() {
   int x,n,p[10],pp[10],pt[10],w[10],t[10],awt,atat,i,j;
   printf("Enter the number of process : ");
   scanf("%d",&n);
   printf("\n Enter process : time priorities \n");
   for(i=0;i<n;i++)
    {
      printf("\nProcess no %d : ",i+1);
      scanf("%d  %d",&pt[i],&pp[i]);
      p[i]=i+1;
    }
  for(i=0;i<n-1;i++){
     for(j=i+1;j<n;j++)
     {
       if(pp[i]<pp[j])
       {
         x=pp[i];
         pp[i]=pp[j];
         pp[j]=x;
         x=pt[i];
         pt[i]=pt[j];
         pt[j]=x;
         x=p[i];
         p[i]=p[j];
         p[j]=x;
      }
   }
}
w[0]=0;
awt=0;
t[0]=pt[0];
atat=t[0];
for(i=1;i<n;i++)
 {
   w[i]=t[i-1];
   awt+=w[i];
   t[i]=w[i]+pt[i];
   atat+=t[i];
 }
printf("\n\n Job \t Burst Time \t Wait Time \t Turn Around Time
Priority \n");
for(i=0;i<n;i++)
  printf("\n %d \t\t %d  \t\t %d \t\t %d \t\t %d
\n",p[i],pt[i],w[i],t[i],pp[i]);
awt/=n;
atat/=n;
printf("\n Average Wait Time : %d \n",awt);
printf("\n Average Turn Around Time : %d \n",atat);
return 0;
}
```

OUTPUT



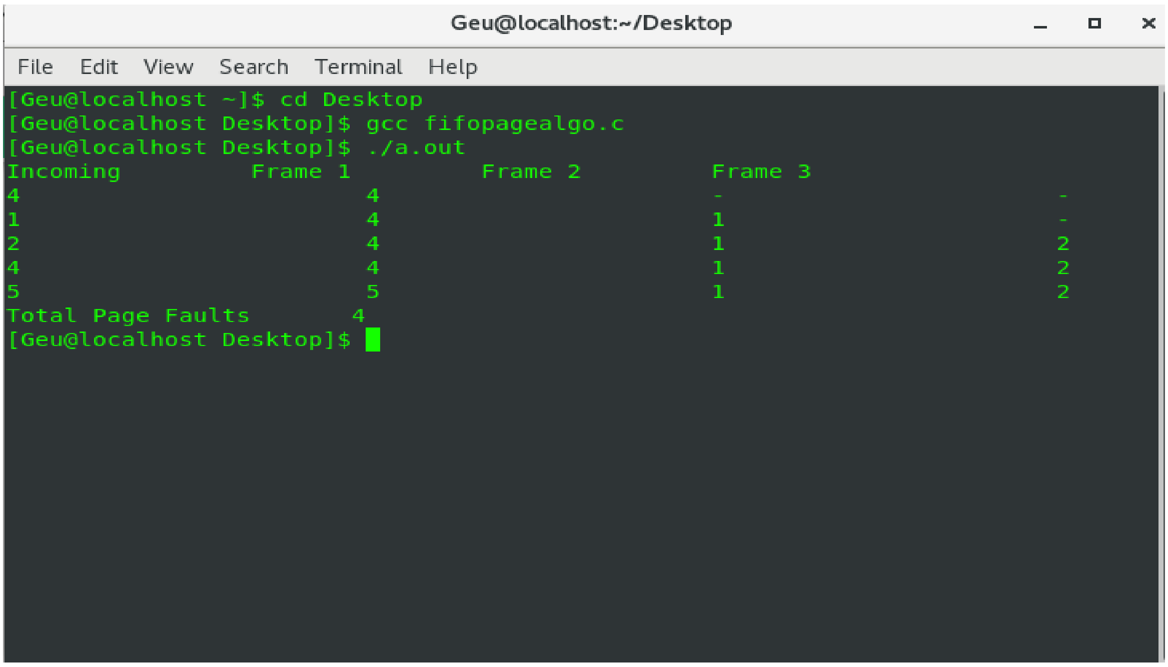
```
[Geu@localhost ~]$ cd Desktop
[Geu@localhost Desktop]$ gcc priority.c
[Geu@localhost Desktop]$ ./a.out
Enter the number of process : 4

 Enter process : time priorities

Process no 1 : 3
1

Process no 2 : 4
2

Process no 3 : 5
3

Process no 4 : 6
4


 Job     Burst Time      Wait Time       Turn Around Time   Priority

 4              6              0              6              4

 3              5              6              11             3

 2              4              11             15             2

 1              3              15             18             1

 Average Wait Time : 8

 Average Turn Around Time : 12
[Geu@localhost Desktop]$ 
```

WEEK – 9

Program 9) Implement FIFO (First in First Out) Page Replacement Algorithm.

```
#include <stdio.h>
int main()
{
    int incomingStream[] = {4, 1, 2, 4, 5};
    int pageFaults = 0;
    int frames = 3;
    int m, n, s, pages;
    pages = sizeof(incomingStream)/sizeof(incomingStream[0]);
    printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3");
    int temp[frames];
    for(m = 0; m < frames; m++)
    {
        temp[m] = -1;
    }
    for(m = 0; m < pages; m++)
    {
        s = 0;
        for(n = 0; n < frames; n++)
        {
            if(incomingStream[m] == temp[n])
            {
                s++;
                pageFaults--;
            }
        }
        pageFaults++;
        if((pageFaults <= frames) && (s == 0))
        {
            temp[m] = incomingStream[m];
        }
        else if(s == 0)
        {
            temp[(pageFaults - 1) % frames] = incomingStream[m];
        }
        printf("\n");
        printf("%d\t\t\t",incomingStream[m]);
        for(n = 0; n < frames; n++)
        {
            if(temp[n] != -1)
                printf(" %d\t\t\t", temp[n]);
            else
                printf(" - \t\t\t");
        }
    }
    printf("\nTotal Page Faults \t%d\n", pageFaults);
    return 0;
}
```

OUTPUT



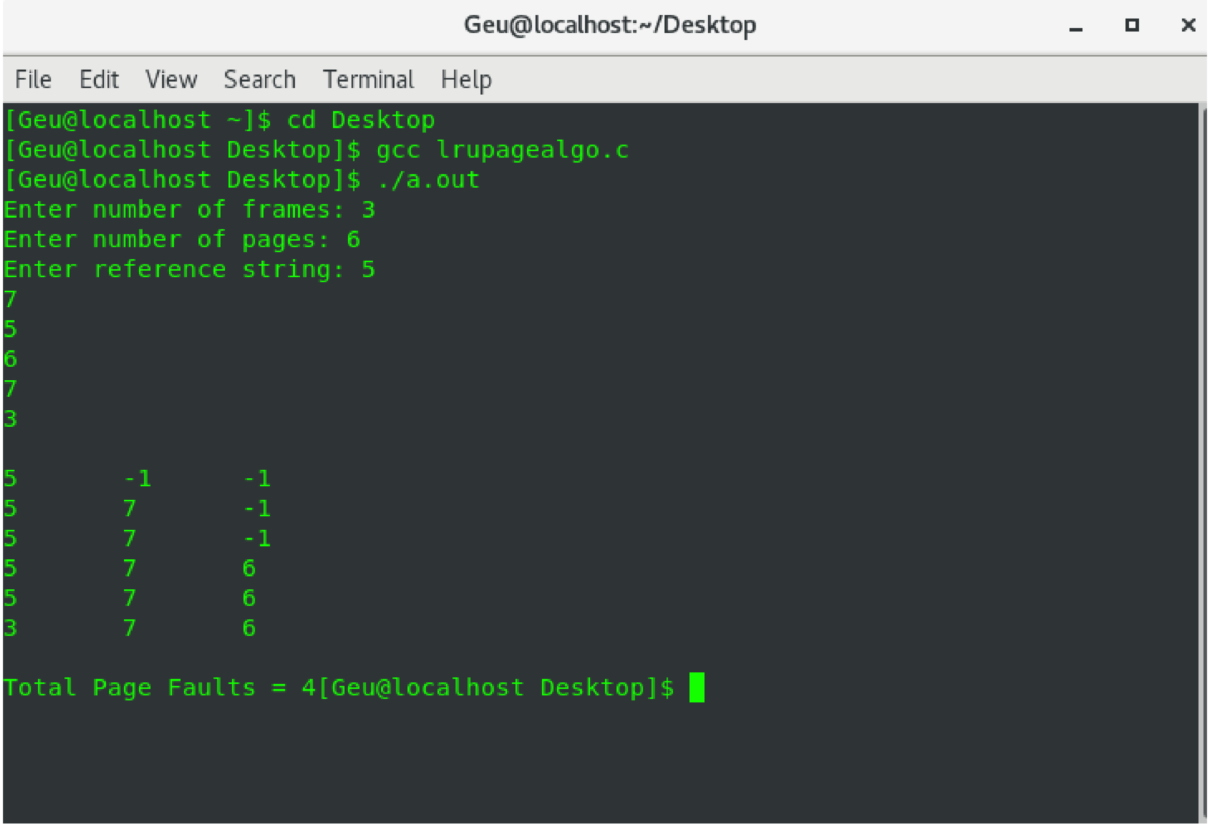
```
[Geu@localhost ~]$ cd Desktop
[Geu@localhost Desktop]$ gcc fifopagealgo.c
[Geu@localhost Desktop]$ ./a.out
Incoming         Frame 1         Frame 2         Frame 3
4                        4                       -                       -
1                        4                       1                       -
2                        4                       1                       2
4                        4                       1                       2
5                        5                       1                       2
Total Page Faults       4
[Geu@localhost Desktop]$ 
```

WEEK – 10

Program 10) Implement LRU (Least Recently Used) Page Replacement Algorithm.

```
#include<stdio.h>

int findLRU(int time[], int n){

int i, minimum = time[0], pos = 0;

for(i = 1; i < n; ++i){

if(time[i] < minimum){

minimum = time[i];

pos = i;

}

}

return pos;

}

int main()

{

    int no_of_frames, no_of_pages, frames[10], pages[30], counter = 
0, time[10], flag1, flag2, i, j, pos, faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no_of_frames);

printf("Enter number of pages: ");

scanf("%d", &no_of_pages);

printf("Enter reference string: ");

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

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

    }

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

     frames[i] = -1;

    }

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

     flag1 = flag2 = 0;

     for(j = 0; j < no_of_frames; ++j){

     if(frames[j] == pages[i]){

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

OUTPUT



```
[Geu@localhost ~]$ cd Desktop
[Geu@localhost Desktop]$ gcc lrupagealgo.c
[Geu@localhost Desktop]$ ./a.out
Enter number of frames: 3
Enter number of pages: 6
Enter reference string: 5
7
5
6
7
3

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

Total Page Faults = 4[Geu@localhost Desktop]$ 
```

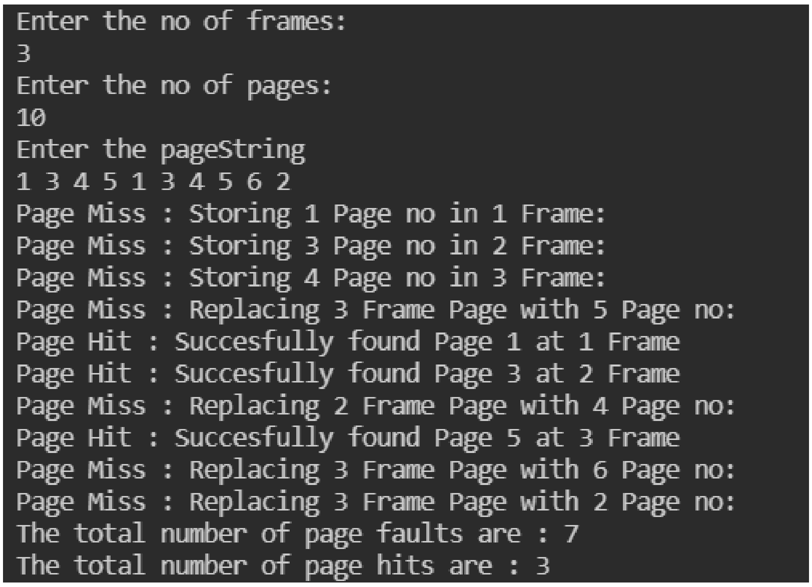
WEEK – 11

Program 11) Implement MRU (Most Recently Used) Page Replacement Algorithm.

```
#include <stdio.h>

#include <stdlib.h>

int find(int no_of_frames, int *frames, int tofound)

{

    int index = -1;

    int i=0;

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

    {

        if (frames[i] == tofound)

        {

            index = i;

        }

    }

    return index;

}

int traverse(int no_of_pages, int *pageString, int i, int tofound)

{

    int index = -1;

    int j=0;

    for (j = i - 1; j >= 0; j--)

    {

        if (pageString[j] == tofound)

        {

            return j;

        }

    }

    return index;

}


int find_MRU(int *pageString, int i, int no_of_pages, int
no_of_frames, int *frames)
```

```
{
    int *flag;
    flag = (int *)calloc(no_of_frames, sizeof(int));
    int j=0;
    for (j = 0; j < no_of_frames; j++)
    {
        flag[j] = 0;
    }
    int index = -1;
    int idx = -1;
    int max = -1; //just some high value later to be replaced
    for (j = 0; j < no_of_frames; j++)
    {
        idx = traverse(no_of_pages, pageString, i, frames[j]);
        if (idx != -1)
        {
            if (idx > max)
            {
                max = idx;
                index = j;
            }
            flag[j] = 1;
        }
    }
    free(flag);
    return index;
}

int main()
{
    int no_of_frames, no_of_pages;
    printf("Enter the no of frames:\n");
    scanf("%d", &no_of_frames);
    printf("Enter the no of pages:\n");
    scanf("%d", &no_of_pages);
    printf("Enter the pageString\n");
    int *pageString;
    pageString = (int *)calloc(no_of_pages, sizeof(int));
    int i;
    for (i = 0; i < no_of_pages; i++)
    {
        scanf("%d", &pageString[i]);
    }
    int *frames;
    frames = (int *)calloc(no_of_frames, sizeof(int));
    for (int i = 0; i < no_of_frames; i++)
    {
        frames[i] = -1;
    }
    int index = 0;
    int no_of_page_faults = 0;
    int no_of_page_hits = 0;
    int idx;
    int count = 0;
    for (i = 0; i < no_of_pages; i++)
    {
        if (count < no_of_frames)
        {
            idx = find(no_of_frames, frames, pageString[i]);
            if (idx != -1)
            {
                no_of_page_hits++;
                printf("Page Hit : Succesfully found Page %d at %d
Frame\n", pageString[i], idx + 1);
            }
            else
            {
                frames[count] = pageString[i];
                printf("Page Miss : Storing %d Page no in %d
Frame:\n", pageString[i], count + 1);
                count++;
                no_of_page_faults++;
            }
        }
        else
        {
            idx = find(no_of_frames, frames, pageString[i]);
            if (idx != -1)
            {
                no_of_page_hits++;
                printf("Page Hit : Succesfully found Page %d at %d
Frame\n", pageString[i], idx + 1);
            }
            else
            {
                index = find_MRU(pageString, i, no_of_pages,
no_of_frames, frames);
                printf("Page Miss : Replacing %d Frame Page with %d
Page no:\n", index + 1, pageString[i]);
                no_of_page_faults++;
                frames[index] = pageString[i];
            }
        }
    }
    printf("The total number of page faults are : %d\n",
no_of_page_faults);
    printf("The total number of page hits are : %d\n",
no_of_page_hits);
    return 0;
}
```

OUTPUT



```
Enter the no of frames:
3
Enter the no of pages:
10
Enter the pageString
1 3 4 5 1 3 4 5 6 2
Page Miss : Storing 1 Page no in 1 Frame:
Page Miss : Storing 3 Page no in 2 Frame:
Page Miss : Storing 4 Page no in 3 Frame:
Page Miss : Replacing 3 Frame Page with 5 Page no:
Page Hit : Succesfully found Page 1 at 1 Frame
Page Hit : Succesfully found Page 3 at 2 Frame
Page Miss : Replacing 2 Frame Page with 4 Page no:
Page Hit : Succesfully found Page 5 at 3 Frame
Page Miss : Replacing 3 Frame Page with 6 Page no:
Page Miss : Replacing 3 Frame Page with 2 Page no:
The total number of page faults are : 7
The total number of page hits are : 3
```

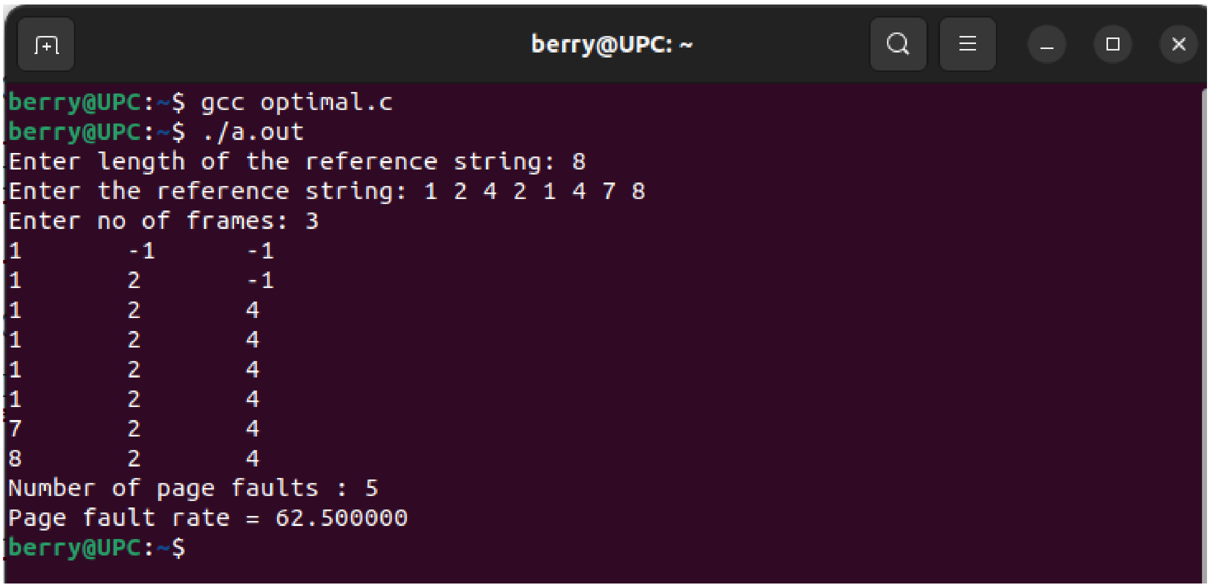
WEEK – 12

Program 12) Implement Optimal Page Replacement Algorithm.

```
#include<stdio.h>
int fr[3], n, m;
void display();
int main()
{
int i,j,page[20],fs[10];
int max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0;
float pr;
printf("Enter length of the reference string: ");
scanf("%d",&n);
printf("Enter the reference string: ");
for(i=0;i<n;i++)
scanf("%d",&page[i]);
printf("Enter no of frames: ");
scanf("%d",&m);
for(i=0;i<m;i++)
fr[i]=-1;
pf=m;
for(j=0;j<n;j++)
{
flag1=0;
flag2=0;
for(i=0;i<m;i++)
{
if(fr[i]==page[j])
{
flag1=1;
flag2=1;
break;
}
```

```
}
if(flag1==0)
{
for(i=0;i<m;i++)
{
if(fr[i]==-1)
{
fr[i]=page[j];
flag2=1;
break;
}
}
}
if(flag2==0)
{
for(i=0;i<m;i++)
lg[i]=0;
for(i=0;i<m;i++)
{
for(k=j+1;k<=n;k++)
{
if(fr[i]==page[k])
{
lg[i]=k-j;
break;
}
}
}
found=0;
for(i=0;i<m;i++)
{
if(lg[i]==0)
{
index=i;
found = 1;
break;
}
}
if(found==0)
{
max=lg[0]; index=0;
for(i=0;i<m;i++)
{
if(max<lg[i])
{
max=lg[i];
index=i;
}
}
}
fr[index]=page[j];
pf++;
}
display();
}
printf("Number of page faults : %d\n", pf);
pr=(float)pf/n*100;
printf("Page fault rate = %f \n", pr);
return 0;
}
void display()
{
int i; for(i=0;i<m;i++)
printf("%d\t",fr[i]);
printf("\n");

}
```

OUTPUT



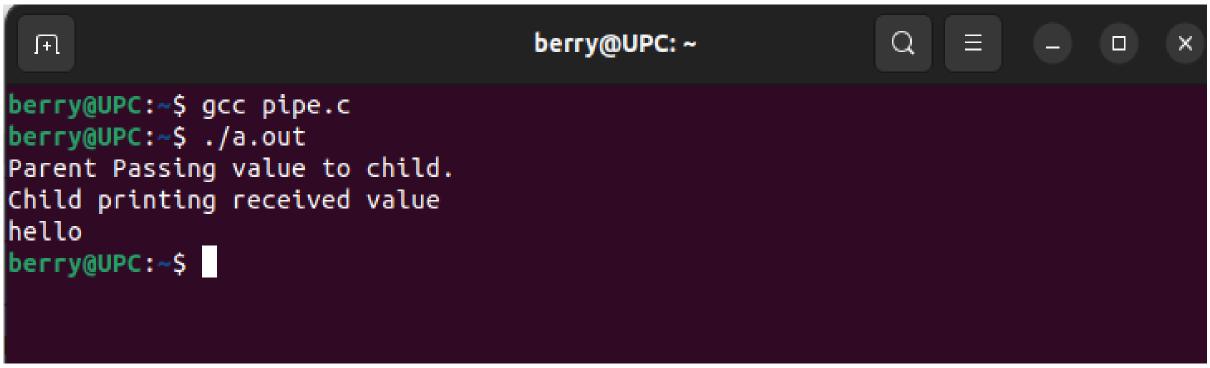
```
berry@UPC:~$ gcc optimal.c
berry@UPC:~$ ./a.out
Enter length of the reference string: 8
Enter the reference string: 1 2 4 2 1 4 7 8
Enter no of frames: 3
1       -1      -1
1       2       -1
1       2       4
1       2       4
1       2       4
1       2       4
7       2       4
8       2       4
Number of page faults : 5
Page fault rate = 62.500000
berry@UPC:~$
```

WEEK – 13

Program 13) Implementation of PIPE.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main()   {
    int fd[2], n;
    char buffer[100];
    pid_t p;
    pipe(fd);
    p = fork();
    if (p > 0)
    {
        printf("Parent Passing value to child.\n");
        write(fd[1], "hello\n", 6);
        wait(NULL);
    }
    else
    {
        printf("Child printing received value\n");
        n = read(fd[0], buffer, 100);
        write(1, buffer, n);
    }
}
```

OUTPUT



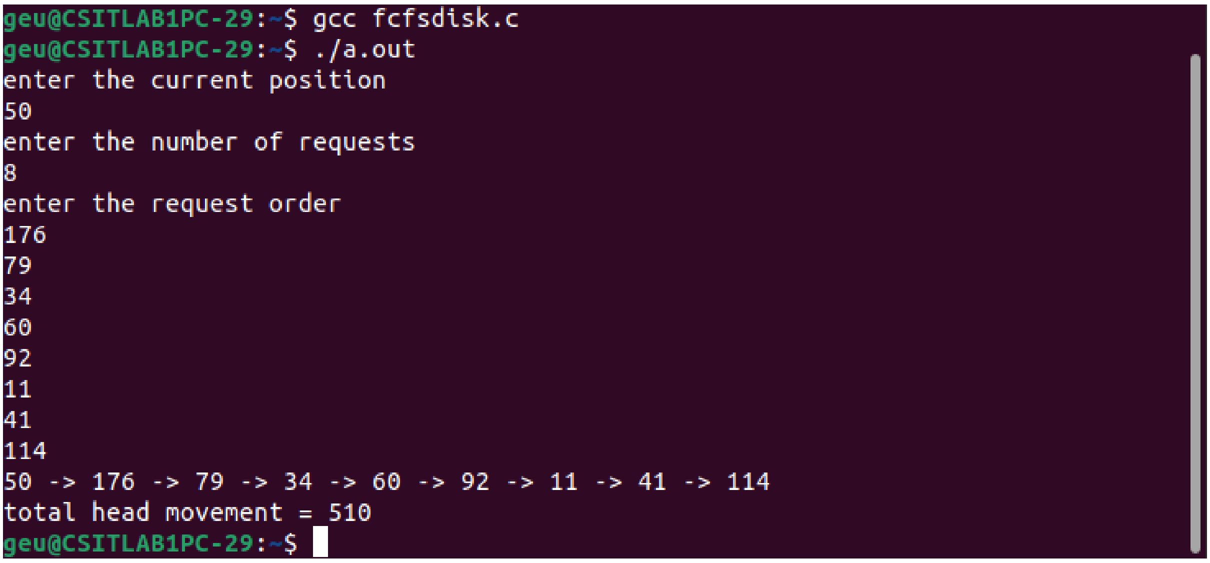
```
berry@UPC:~$ gcc pipe.c
berry@UPC:~$ ./a.out
Parent Passing value to child.
Child printing received value
hello
berry@UPC:~$ 
```

WEEK – 14

Program 14)a) Implement FCFS (First Come First Serve) Disk Scheduling Algorithm.

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,n,req[50],mov=0,cp;
    printf("enter the current position\n");
    scanf("%d",&cp);
    printf("enter the number of requests\n");
    scanf("%d",&n);
    printf("enter the request order\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&req[i]);
    }
    mov=mov+abs(cp-req[0]); // abs is used to calculate the absolute
value
    printf("%d -> %d",cp,req[0]);
    for(i=1;i<n;i++)
    {
        mov=mov+abs(req[i]-req[i-1]);
        printf(" -> %d",req[i]);
    }
    printf("\n");
    printf("total head movement = %d\n",mov);
}
```

OUTPUT

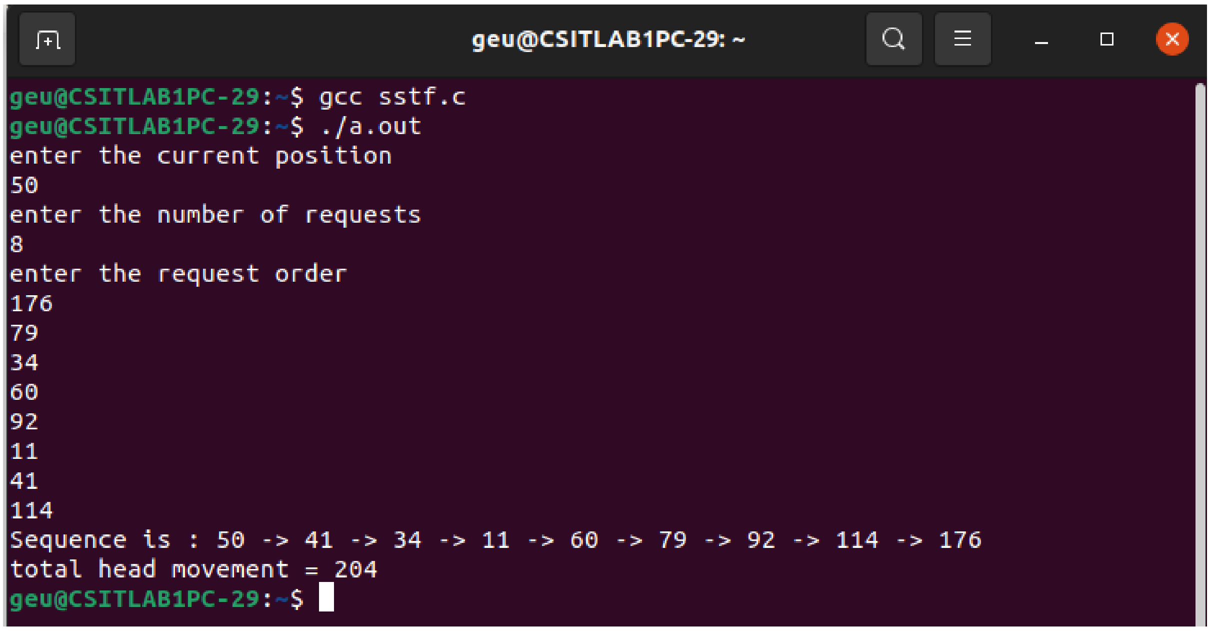


```
geu@CSITLAB1PC-29:~$ gcc fcfsdisk.c
geu@CSITLAB1PC-29:~$ ./a.out
enter the current position
50
enter the number of requests
8
enter the request order
176
79
34
60
92
11
41
114
50 -> 176 -> 79 -> 34 -> 60 -> 92 -> 11 -> 41 -> 114
total head movement = 510
geu@CSITLAB1PC-29:~$ 
```

Program 14)b) Implement SSTF (Shortest Seek Time First) Disk Scheduling Algorithm.

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,n,k,req[50],mov=0,cp,index[50],min,a[50],j=0,mini,cp1;
    printf("enter the current position\n");
    scanf("%d",&cp);
    printf("enter the number of requests\n");
    scanf("%d",&n);
    cp1=cp;
    printf("enter the request order\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&req[i]);
    }
    for(k=0;k<n;k++)
    {
    for(i=0;i<n;i++)
    {
        index[i]=abs(cp-req[i]); // calculate distance of each
request from current position
    }
    // to find the nearest request
    min=index[0];
    mini=0;
    for(i=1;i<n;i++)
    {
        if(min>index[i])
        {
            min=index[i];
            mini=i;
        }
    }
    a[j]=req[mini];
    j++;
    cp=req[mini]; // change the current position value to next
request
    req[mini]=999;
    } // the request that is processed its value is changed so that
it is not processed again
    printf("Sequence is : ");
    printf("%d",cp1);
    mov=mov+abs(cp1-a[0]);     // head movement
    printf(" -> %d",a[0]);
    for(i=1;i<n;i++)
    {
        mov=mov+abs(a[i]-a[i-1]); ///head movement
        printf(" -> %d",a[i]);
    }
    printf("\n");
    printf("total head movement = %d\n",mov);
}
```

OUTPUT



```
geu@CSITLAB1PC-29:~$ gcc sstf.c
geu@CSITLAB1PC-29:~$ ./a.out
enter the current position
50
enter the number of requests
8
enter the request order
176
79
34
60
92
11
41
114
Sequence is : 50 -> 41 -> 34 -> 11 -> 60 -> 79 -> 92 -> 114 -> 176
total head movement = 204
geu@CSITLAB1PC-29:~$ 
```