Principles of Operating System [CBS1006]

ASSESSMENT - 4[LAB]

SIDDHARTHA PURWAR| 19BBS0072

06-April-2021

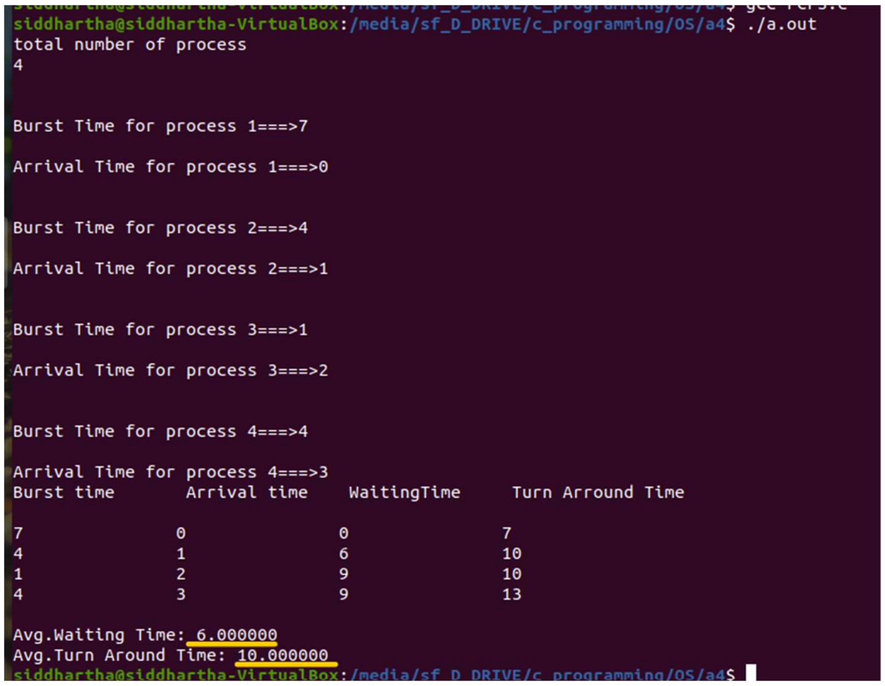
1. Code the following scheduling algorithms in C and print the performance parameters for an arbitrary set of inputs (Process IDs, Burst Times, Arrival Times, Priorities). Use of appropriate data structures will be appreciable.
    a. FCFS
    b. SJF
    c. SRTF
    d. PRIORITY
    e. ROUND ROBIN

a) FCFS CODE:

```
#include <stdio.h>

struct inputs{
    int arrival_time;
    int waiting_time;
    int turn_arround_time;
    int brust_time;
};

int i = 0, j = 0, n = 0, t, total_waiting_time = 0, total_turn_arround_time = 0,
pre_brust = 0;
float average_waiting_time = 0.0, average_turn_around_time = 0.0;

void fcfs(struct inputs *process){
    process[0].waiting_time = 0;
    int j, i;
    for (j = 0; j < n; ){
        for(i = 0; i < n ;i++){
            if((process[i].arrival_time <= total_turn_arround_time) &&
(process[i].turn_arround_time == 0)){
                process[i].waiting_time = pre_brust - process[i].arrival_time;
                process[i].waiting_time = (process[i].waiting_time > 0) ?
process[i].waiting_time : 0;
                process[i].turn_arround_time = process[i].waiting_time +
process[i].brust_time;
                total_turn_arround_time = total_turn_arround_time +
process[i].turn_arround_time;
                total_waiting_time = total_waiting_time +
process[i].waiting_time;
                pre_brust += process[i].brust_time;
                j++;
            }
        }

    }
}
int main()
{
```

```
printf("total number of process\n");
scanf("%d",&n);
struct inputs process[n];

for(i=0;i<n;i++)
{
 printf("\n\nBurst Time for process %d===>", i+ 1);
 scanf("\t %d",&process[i].brust_time);
 printf("\nArrival Time for process %d===>", i + 1);
  scanf("\t%d",&process[i].arrival_time);
 process[i].turn_arround_time = process[i].waiting_time = 0;
}

process[0].waiting_time = 0;
struct inputs t;

fcfs(process);

printf("Pid\tBurst time\t Arrival time\t WaitingTime\t Turn Arround Time\n");
for(i=0;i<n;i++){
    printf("\n%d\t\t%d\t\t%d\t\t%d",process[i].brust_time,
process[i].arrival_time, process[i].waiting_time,process[i].turn_arround_time);
}

average_waiting_time = (float)total_waiting_time / n;
average_turn_around_time = (float)total_turn_arround_time / n;
printf("\n\nAvg.Waiting Time: %f\nAvg.Turn Around Time:
%f\n",average_waiting_time,average_turn_around_time);
return 0;
}
```

FCFS OUTPUT:

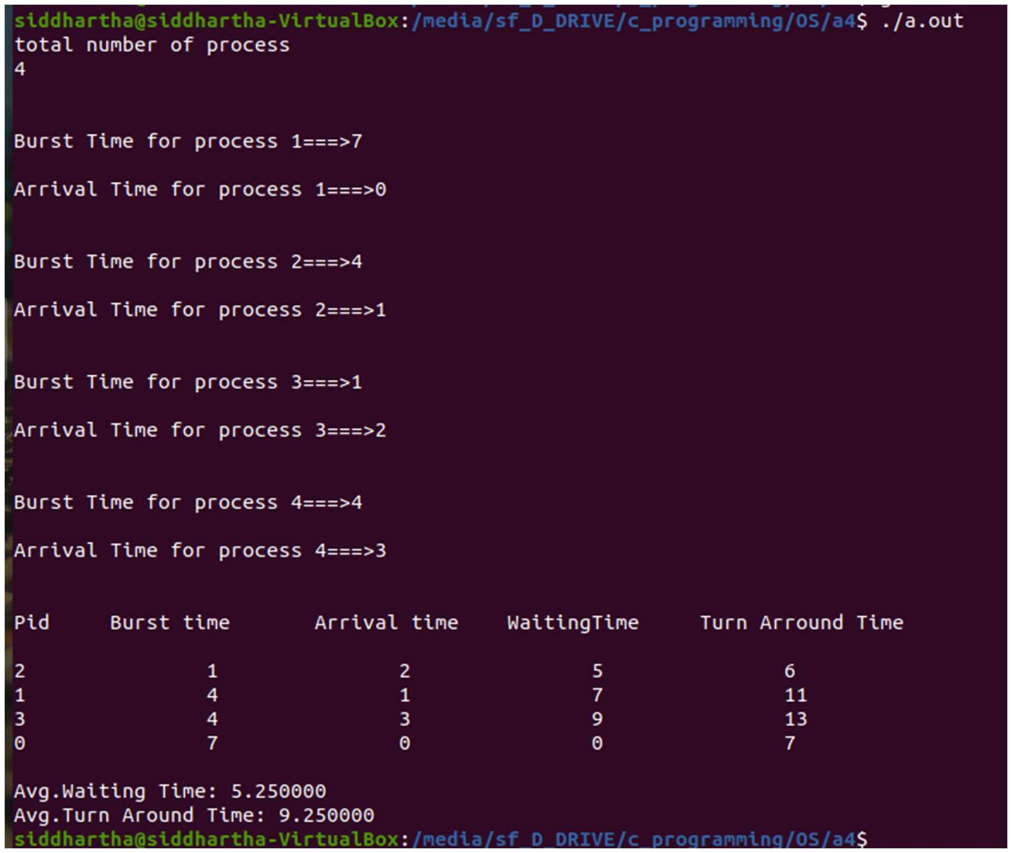


```
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$ gcc fcfs.c
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$ ./a.out
total number of process
4


Burst Time for process 1===>7

Arrival Time for process 1===>0


Burst Time for process 2===>4

Arrival Time for process 2===>1


Burst Time for process 3===>1

Arrival Time for process 3===>2


Burst Time for process 4===>4

Arrival Time for process 4===>3
Burst time       Arrival time    WaitingTime     Turn Arround Time

7                0               0               7
4                1               6               10
1                2               9               10
4                3               9               13

Avg.Waiting Time: 6.000000
Avg.Turn Around Time: 10.000000
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$
```

b) SJF CODE:

```
#include <stdio.h>

struct inputs{
    int arrival_time;
    int waiting_time;
    int turn_arround_time;
    int brust_time;
    int id;
};
int i = 0, j = 0, n = 0, t, total_waiting_time = 0, total_turn_arround_time = 0,
pre_brust = 0;
float average_waiting_time = 0.0, average_turn_around_time = 0.0;

void fcfs(struct inputs *process){
    process[0].waiting_time = 0;
    int j, i;
    for (j = 0; j < n; ){
        for(i = 0; i < n ;i++){
            if((process[i].arrival_time <= total_turn_arround_time) &&
(process[i].turn_arround_time == 0)){
                process[i].waiting_time = pre_brust - process[i].arrival_time;
                process[i].waiting_time = (process[i].waiting_time > 0) ?
process[i].waiting_time : 0;
                process[i].turn_arround_time = process[i].waiting_time +
process[i].brust_time;
                total_turn_arround_time = total_turn_arround_time +
process[i].turn_arround_time;
                total_waiting_time = total_waiting_time +
process[i].waiting_time;
                pre_brust += process[i].brust_time;
                j++;
            }
        }

    }
}
int main()
{
printf("total number of process\n");
scanf("%d",&n);
struct inputs process[n];
for(i=0;i<n;i++)
{
 printf("\n\nBurst Time for process %d===>", i+ 1);
 scanf("\t %d",&process[i].brust_time);
 printf("\nArrival Time for process %d===>", i + 1);
 scanf("\t%d",&process[i].arrival_time);
 process[i].id = i;
 process[i].turn_arround_time = process[i].waiting_time = 0;
}
process[0].waiting_time = 0;
struct inputs t;
//sorting according to Brust_time
for (i=0;i<n;i++){
    for(j=i+1;j<n;j++){
        if (process[i].brust_time > process[j].brust_time){
            t = process[j];
            process[j] = process[i];
            process[i] = t;
        }
    }
}
//after sorting SJF scheduling is like FCFS scheduling
fcfs(process);

printf("\n\nPid\tBurst time\t Arrival time\t WaitingTime\t Turn Arround Time\n");
for(i=0;i<n;i++){
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",process[i].id ,process[i].brust_time,
process[i].arrival_time, process[i].waiting_time,process[i].turn_arround_time);
}
average_waiting_time = (float)total_waiting_time / n;
average_turn_around_time = (float)total_turn_arround_time / n;
printf("\n\nAvg.Waiting Time: %f\nAvg.Turn Around Time:
%f\n",average_waiting_time,average_turn_around_time);
return 0;
}
```

SJF OUTPUT:

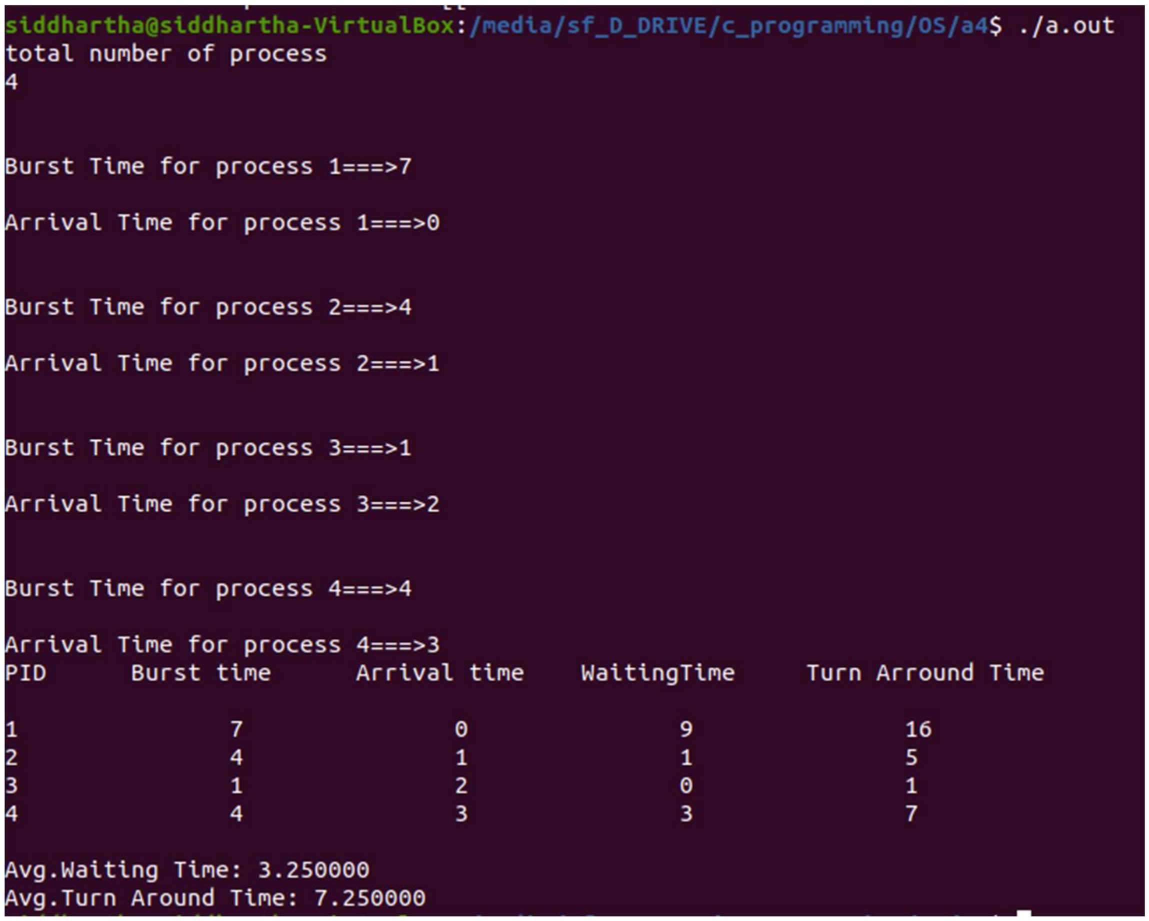


```
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$ ./a.out
total number of process
4


Burst Time for process 1===>7

Arrival Time for process 1===>0


Burst Time for process 2===>4

Arrival Time for process 2===>1


Burst Time for process 3===>1

Arrival Time for process 3===>2


Burst Time for process 4===>4

Arrival Time for process 4===>3


Pid     Burst time       Arrival time    WaitingTime     Turn Arround Time

2               1               2               5               6
1               4               1               7               11
3               4               3               9               13
0               7               0               0               7

Avg.Waiting Time: 5.250000
Avg.Turn Around Time: 9.250000
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$
```

c) SRTF CODE:

```
#include <stdio.h>

struct inputs{
    int arrival_time;
    int waiting_time;
    int turn_arround_time;
    int burst_time;
    int temp_burst_time;
    int id;
};


int i = 0, j = 0, n = 0, t, total_waiting_time = 0, total_turn_arround_time = 0,
pre_brust = 0, time = 0, count = 0, smallest = 0, end = 0;
float average_waiting_time = 0.0, average_turn_around_time = 0.0;


int main()
{
printf("total number of process\n");
scanf("%d",&n);
struct inputs process[n];

for(i=0;i<n;i++)
{
 printf("\n\nBurst Time for process %d===>", i+ 1);
 scanf("\t %d",&process[i].burst_time);
 process[i].temp_burst_time = process[i].burst_time;
 printf("\nArrival Time for process %d===>", i + 1);
 scanf("\t%d",&process[i].arrival_time);
 process[i].id = i;
 process[i].turn_arround_time = process[i].waiting_time = 0;
 process[i].id = i + 1;
}
      process[i].burst_time = 1000;
      for(time = 0; count != n; time++)
      {
            smallest = i;
            for(i = 0; i < n; i++)
            {
                  if(process[i].arrival_time <= time && process[i].burst_time <
process[smallest].burst_time && process[i].burst_time > 0)
                  {
                        smallest = i;
                  }
            }
            process[smallest].burst_time--;
            if(process[smallest].burst_time == 0)
            {
                  count++;
                  end = time + 1;
                  process[smallest].waiting_time = end -
process[smallest].arrival_time- process[smallest].temp_burst_time;
                  total_waiting_time +=  process[smallest].waiting_time;
                  process[smallest].turn_arround_time =  end -
process[smallest].arrival_time;
                  total_turn_arround_time += process[smallest].turn_arround_time;
            }
      }
printf("PID\t Burst time\t Arrival time\t WaitingTime\t Turn Arround Time\n");
for(i=0;i<n;i++){
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",process[i].id,
process[i].temp_burst_time, process[i].arrival_time,
process[i].waiting_time,process[i].turn_arround_time);
}

average_waiting_time = (float)total_waiting_time / n;
average_turn_around_time = (float)total_turn_arround_time / n;
printf("\n\nAvg.Waiting Time: %f\nAvg.Turn Around Time:
%f\n",average_waiting_time,average_turn_around_time);
return 0;
}
```

SRTF OUTPUT:

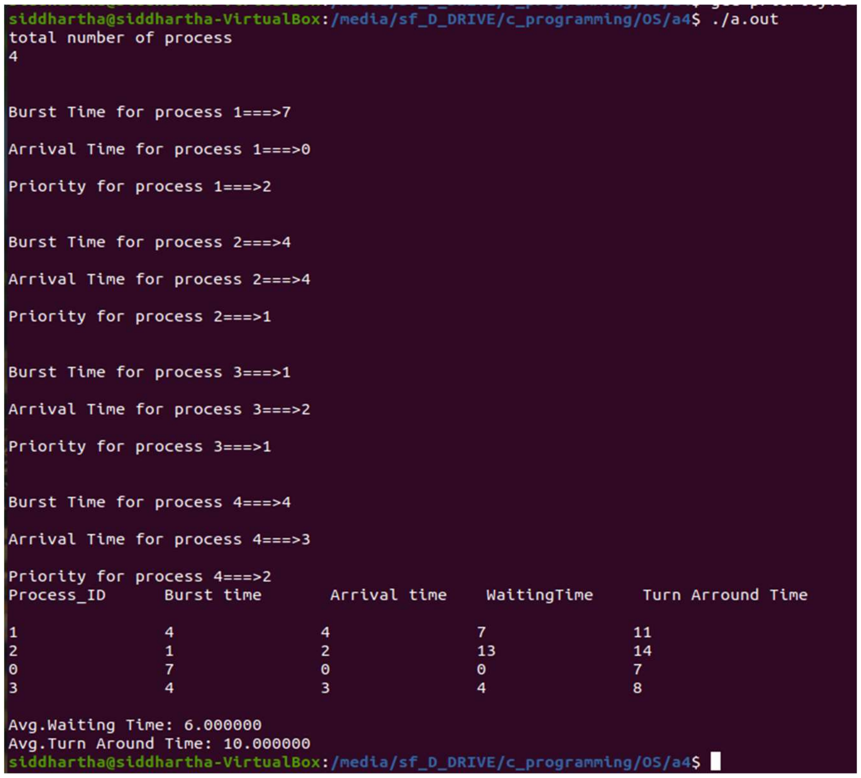


```
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$ ./a.out
total number of process
4


Burst Time for process 1===>7

Arrival Time for process 1===>0


Burst Time for process 2===>4

Arrival Time for process 2===>1


Burst Time for process 3===>1

Arrival Time for process 3===>2


Burst Time for process 4===>4

Arrival Time for process 4===>3
PID      Burst time      Arrival time     WaitingTime      Turn Arround Time

1               7               0               9               16
2               4               1               1               5
3               1               2               0               1
4               4               3               3               7

Avg.Waiting Time: 3.250000
Avg.Turn Around Time: 7.250000
```

d) PRIORITY CODE:

```
#include <stdio.h>

struct inputs{
    int arrival_time;
    int waiting_time;
    int turn_arround_time;
    int brust_time;
    int priority;
    int id;
};
int i = 0, j = 0, n = 0, t, total_waiting_time = 0, total_turn_arround_time = 0,
pre_brust = 0;
float average_waiting_time = 0.0, average_turn_around_time = 0.0;

void fcfs(struct inputs *process){
    process[0].waiting_time = 0;
    int j, i;
    for (j = 0; j < n; ){
        for(i = 0; i < n ;i++){
            if((process[i].arrival_time <= total_turn_arround_time) &&
(process[i].turn_arround_time == 0)){
                process[i].waiting_time = pre_brust - process[i].arrival_time;
                process[i].waiting_time = (process[i].waiting_time > 0) ?
process[i].waiting_time : 0;
                process[i].turn_arround_time = process[i].waiting_time +
process[i].brust_time;
                total_turn_arround_time = total_turn_arround_time +
process[i].turn_arround_time;
                total_waiting_time = total_waiting_time +
process[i].waiting_time;
                pre_brust += process[i].brust_time;
                 j++;
            }
        }
    }
}
int main(){
printf("total number of process\n");
scanf("%d",&n);
struct inputs process[n];
for(i=0;i<n;i++){
 printf("\n\nBurst Time for process %d===>", i+ 1);
 scanf("\t %d",&process[i].brust_time);
 printf("\nArrival Time for process %d===>", i + 1);
 scanf("\t%d",&process[i].arrival_time);
 printf("\nPriority for process %d===>", i + 1);
 scanf("\t%d",&process[i].priority);
 process[i].id = i;
 process[i].turn_arround_time = process[i].waiting_time = 0;
}
process[0].waiting_time = 0;
struct inputs t;
//sorting according to priority
for (i=0;i<n;i++){
    for(j=i+1;j<n;j++){
        if (process[i].priority > process[j].priority ){
            t = process[j];
            process[j] = process[i];
            process[i] = t;
        }
    }
}
//after sorting priority scheduling is like FCFS scheduling
fcfs(process);

printf("Process_ID\tBurst time\t Arrival time\t WaitingTime\t Turn Arround
Time\n");
for(i=0;i<n;i++){
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",process[i].id, process[i].brust_time,
process[i].arrival_time, process[i].waiting_time,process[i].turn_arround_time);
}
average_waiting_time = (float)total_waiting_time / n;
average_turn_around_time = (float)total_turn_arround_time / n;
printf("\n\nAvg.Waiting Time: %f\nAvg.Turn Around Time:
%f\n",average_waiting_time,average_turn_around_time);
}
```

PRIORITY OUTPUT: (*Not getting expected Output)*

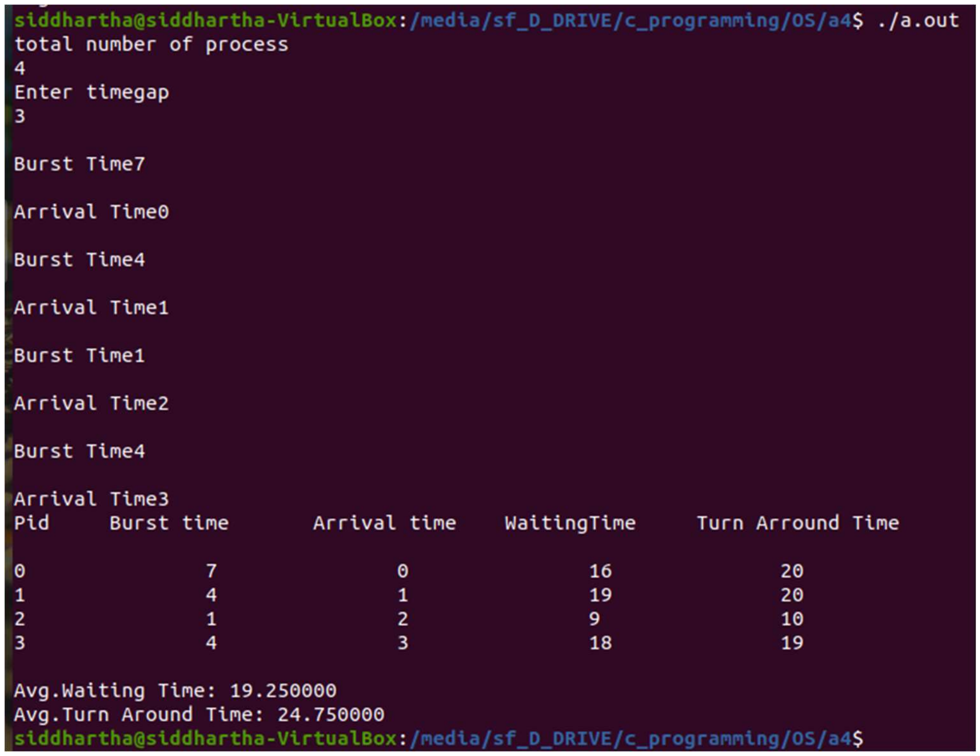


```
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$ ./a.out
total number of process
4


Burst Time for process 1===>7

Arrival Time for process 1===>0

Priority for process 1===>2


Burst Time for process 2===>4

Arrival Time for process 2===>4

Priority for process 2===>1


Burst Time for process 3===>1

Arrival Time for process 3===>2

Priority for process 3===>1


Burst Time for process 4===>4

Arrival Time for process 4===>3

Priority for process 4===>2
Process_ID      Burst time       Arrival time    WaitingTime     Turn Arround Time

1               4                4               7               11
2               1                2               13              14
0               7                0               0               7
3               4                3               4               8

Avg.Waiting Time: 6.000000
Avg.Turn Around Time: 10.000000
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$
```

e) ROUND ROBIN CODE:

```
#include <stdio.h>

struct inputs{
    int arrival_time;
    int waiting_time;
    int turn_arround_time;
    int brust_time;
    int temp_brust_time;
    int id;
};
int time , i = 0, j = 0, n = 0, t, total_waiting_time = 0,
total_turn_arround_time = 0,  pre_brust = 0;
float average_waiting_time = 0.0, average_turn_around_time = 0.0;

void round_robin(struct inputs *process){
    process[0].waiting_time = 0;
    int j, i;
    for (j = 0; j < n; ){
             for(i = 0; i < n ;i++){
                if((process[i].arrival_time <= total_turn_arround_time) &&
(process[i].brust_time != -1)){
                process[i].waiting_time = pre_brust - process[i].arrival_time;
                process[i].waiting_time = (process[i].waiting_time > 0) ?
process[i].waiting_time : 0;
                process[i].turn_arround_time = process[i].waiting_time +
process[i].brust_time;
                total_turn_arround_time = total_turn_arround_time +
process[i].turn_arround_time;
                total_waiting_time = total_waiting_time +
process[i].waiting_time;
                process[i].waiting_time = (process[i].waiting_time > 0) ?
process[i].waiting_time : 0;
                pre_brust += process[i].brust_time;
                process[i].brust_time = process[i].brust_time - time;
                }
                 if(process[i].brust_time <= 0){
                        j++;
                    process[i].brust_time = -1;
                }
        }
    }
}
int main(){
printf("total number of process\n");
scanf("%d",&n);
printf("Enter timegap\n");
scanf("%d", &time);
struct inputs process[n];
for(i=0;i<n;i++)
{
 printf("\nBurst Time");
 scanf("\n %d",&process[i].brust_time);
 printf("\nArrival Time");
 process[i].temp_brust_time = process[i].brust_time;
 scanf("\n %d",&process[i].arrival_time);
 process[i].id = i;
 process[i].turn_arround_time = process[i].waiting_time = 0;
}
process[0].waiting_time = 0;
struct inputs t;

round_robin(process);

printf("Pid\tBurst time\t Arrival time\t WaitingTime\t Turn Arround Time\n");
for(i=0;i<n;i++){
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",process[i].id
,process[i].temp_brust_time, process[i].arrival_time,
process[i].waiting_time,process[i].turn_arround_time);
}

average_waiting_time = (float)total_waiting_time / n;
average_turn_around_time = (float)total_turn_arround_time / n;
printf("\n\nAvg.Waiting Time: %f\nAvg.Turn Around Time:
%f\n",average_waiting_time,average_turn_around_time);
return 0;
}
```

ROUND ROBIN OUTPUT *(Not giving expected output)*



```
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$ ./a.out
total number of process
4
Enter timegap
3

Burst Time7

Arrival Time0

Burst Time4

Arrival Time1

Burst Time1

Arrival Time2

Burst Time4

Arrival Time3
Pid     Burst time      Arrival time    WaitingTime     Turn Arround Time

0               7               0               16              20
1               4               1               19              20
2               1               2               9               10
3               4               3               18              19

Avg.Waiting Time: 19.250000
Avg.Turn Around Time: 24.750000
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$
```

2. Create two POSIX threads. Let the first thread copy the contents of "file1.txt" to "file2.txt" and let the second thread collapse all spaces more than one to one space in an input string. Write the main function to test the working of the two threads.

CODE:

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#define MAXNAME 50
#define MAXLENGTH 5000

char file1[MAXNAME];
char file2[MAXNAME];
char output_string[MAXLENGTH];


void* copy_file(void * arg){ //for thread_1
    FILE *fp1, *fp2;
    int c = 0;
    if(!(fp1 = fopen(file1, "r"))){
        perror("File 1 cannot be opened");
    }
    if(!(fp2 = fopen(file2, "w"))){
        perror("File 2 cannot be opened");
    }
    while((c = fgetc(fp1)) != EOF){
        fputc(c,fp2);
    }
    fclose(fp1);
    fclose(fp2);
    pthread_exit(NULL);
}

void * space_operation(void * arg){ //for thread_2
    FILE *fp0, *fp3;
    char d[1];
    int flag = -1;
    if(!(fp0 = fopen(file2, "r"))){
        perror("File 2 cannot be opened");
    }
    while((d[0] = fgetc(fp0)) != EOF){
         if((d[0] == ' ') && (flag == -1)){
            flag = 1;
            strcat(output_string, d);
          }
        if((d[0] == ' ') && (flag == 1)){
            }
            else{
              strcat(output_string, d);
               flag = -1;
            }
    }
    fclose(fp0);
    pthread_exit(NULL);
}

//DRIVER CODE
int main(){
    pthread_t t1, t2;

    printf("Enter name of file 1\n");
    scanf("%s",file1);
    printf("Enter name of file 2\n");
    scanf("%s",file2);

    if((pthread_create(&t1, NULL, copy_file, NULL))){
        perror("Thread t1 cannot be created");
    }
    pthread_join(t1, NULL);

    if((pthread_create(&t2, NULL, space_operation, NULL))){
        perror("Thread t2 cannot be created");
    }
    pthread_join(t2, NULL);

    printf("\nString is \n%s", output_string);
    return 0;
}
```

OUTPUT:

Before executing program:

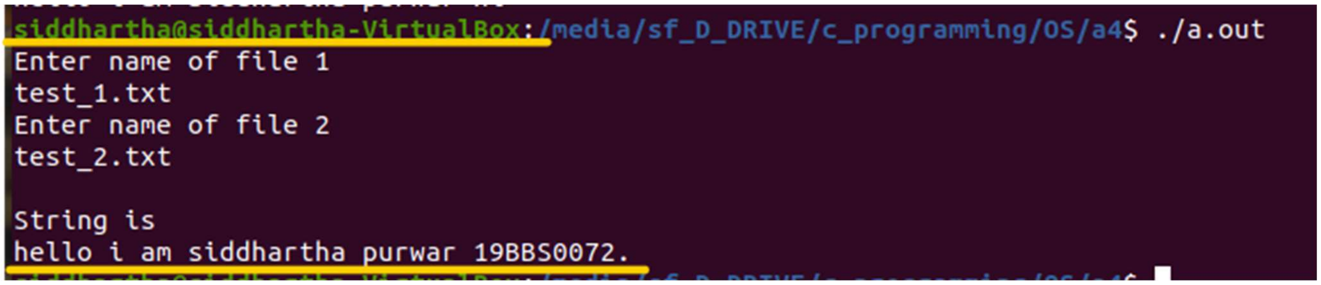
Content of test_1.txt: *"hello i am siddhartha   purwar   19BBS0072."*

Content of test_2.txt: "   "

After executing program:

Content of test_1.txt: *"hello i am siddhartha   purwar   19BBS0072."*

Content of test_2.txt: *"hello i am siddhartha   purwar   19BBS0072."*



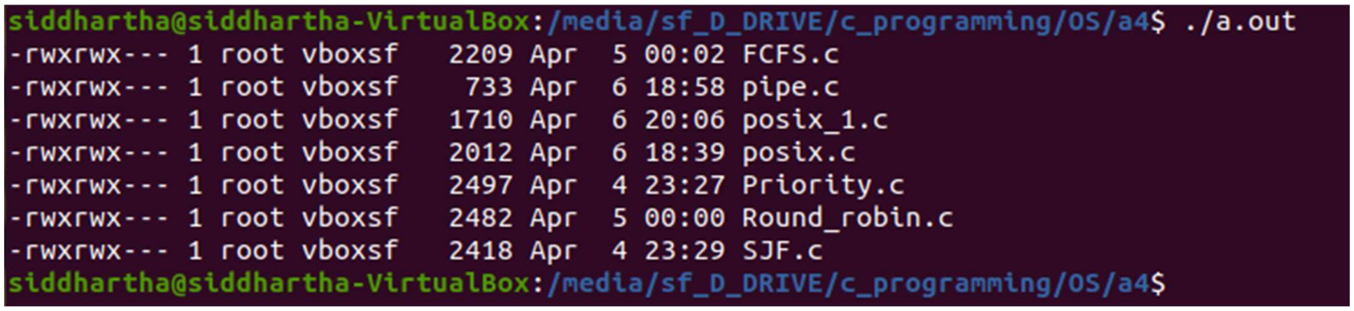
```
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$ ./a.out
Enter name of file 1
test_1.txt
Enter name of file 2
test_2.txt

String is
hello i am siddhartha purwar 19BBS0072.
```

3. Write a C to implement the pipe given below.

    ls -l | grep .c$

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main() {
        int pid;
        int fd[2]; //one for read end and one for write end
        pipe(fd);
        pid=fork();

     if( pid < 0 ) {
          perror( "fork");
          exit( -1 );
     }

        if( pid==0) {

                close(1); //no need to write to stdout
                dup (fd[1]);
                close(fd[0]);
                close(fd[1]);
                execl("/usr/bin/ls","ls", "-l",NULL);

        }
        else {
                close(0); //no need to read from stdin
                dup (fd[0]);
                close(fd[0]);
                close(fd[1]);
                execl("/usr/bin/grep","grep", ".c$" ,NULL);
        }
}
```



```
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$ ./a.out
-rwxrwx--- 1 root vboxsf   2209 Apr  5 00:02 FCFS.c
-rwxrwx--- 1 root vboxsf    733 Apr  6 18:58 pipe.c
-rwxrwx--- 1 root vboxsf   1710 Apr  6 20:06 posix_1.c
-rwxrwx--- 1 root vboxsf   2012 Apr  6 18:39 posix.c
-rwxrwx--- 1 root vboxsf   2497 Apr  4 23:27 Priority.c
-rwxrwx--- 1 root vboxsf   2482 Apr  5 00:00 Round_robin.c
-rwxrwx--- 1 root vboxsf   2418 Apr  4 23:29 SJF.c
siddhartha@siddhartha-VirtualBox:/media/sf_D_DRIVE/c_programming/OS/a4$
```