# DIGITAL ASSIGNMENT – 2

Name: Gagan Chordia
Reg. No: 19BCE0788
Course: Operating Systems
Slot: L9+L10

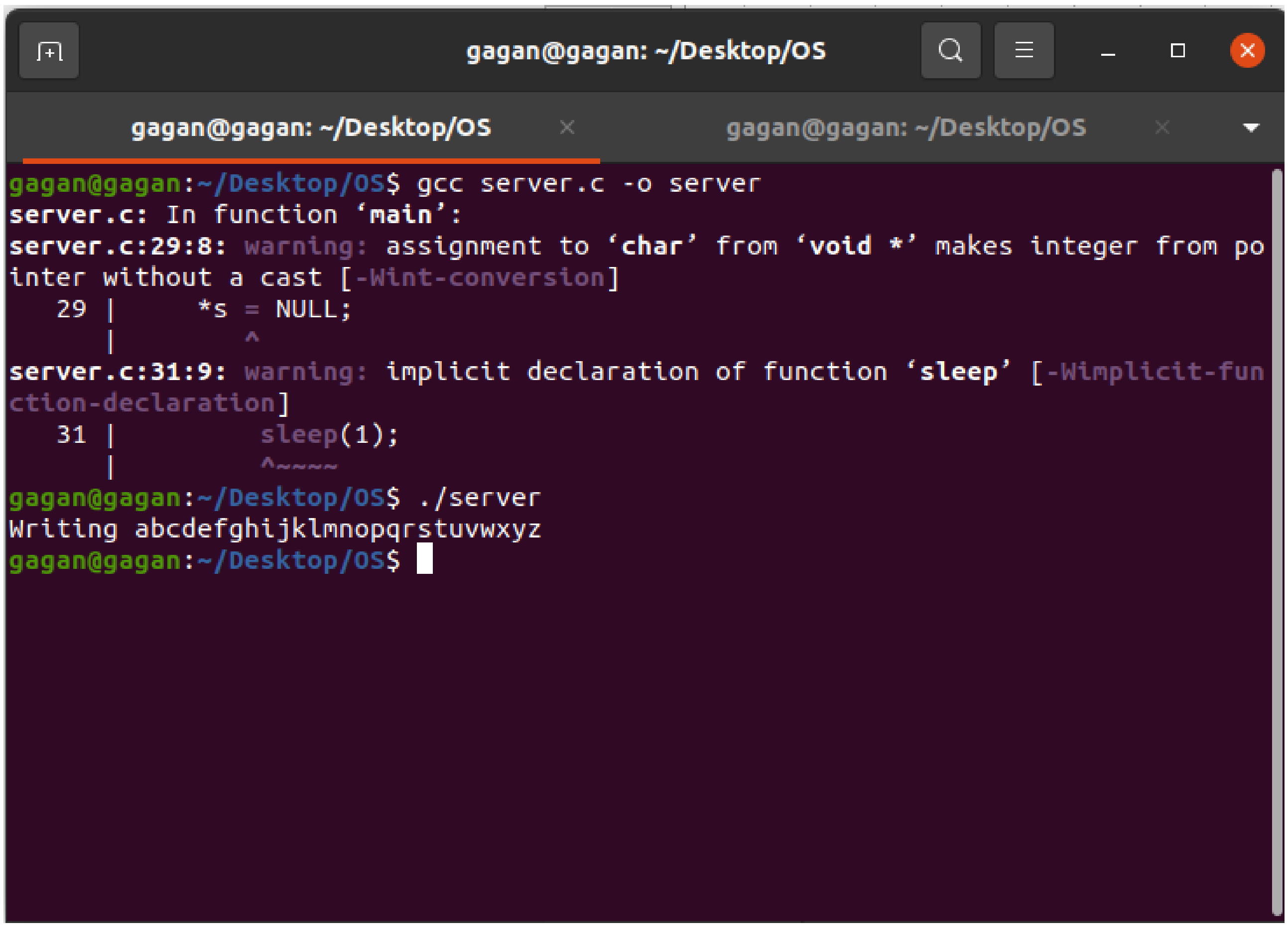
# Questions from Previous DA

1. Assume that two processes named client and server running in the system. It is required that these two processes should communicate with each other using shared memory concept. The server writes alphabets from a..z to the shared memory .the client should read the alphabets from the shared memory and convert it to A...Z. Write a program to demonstrate the above mentioned scenario.

Server C Code:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include<stdlib.h>
#define SHMSZ 27

void main(){
char c;
int shmid;
key_t key;
char *shm, *s;
key = 5678;
if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0) {
perror("shmget");
exit(1);
}
if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
perror("shmat");
exit(1);
}
s = shm;
printf("Writing ");
for (c = 'a'; c <= 'z'; c++){
printf("%c",c);
*s++ = c;
}
printf("\n");
*s = NULL;
while (*shm != '*'){
sleep(1);
}
exit(0);
}
```

Server Output:


gagan@gagan: ~/Desktop/OS
gagan@gagan: ~/Desktop/OS
gagan@gagan: ~/Desktop/OS

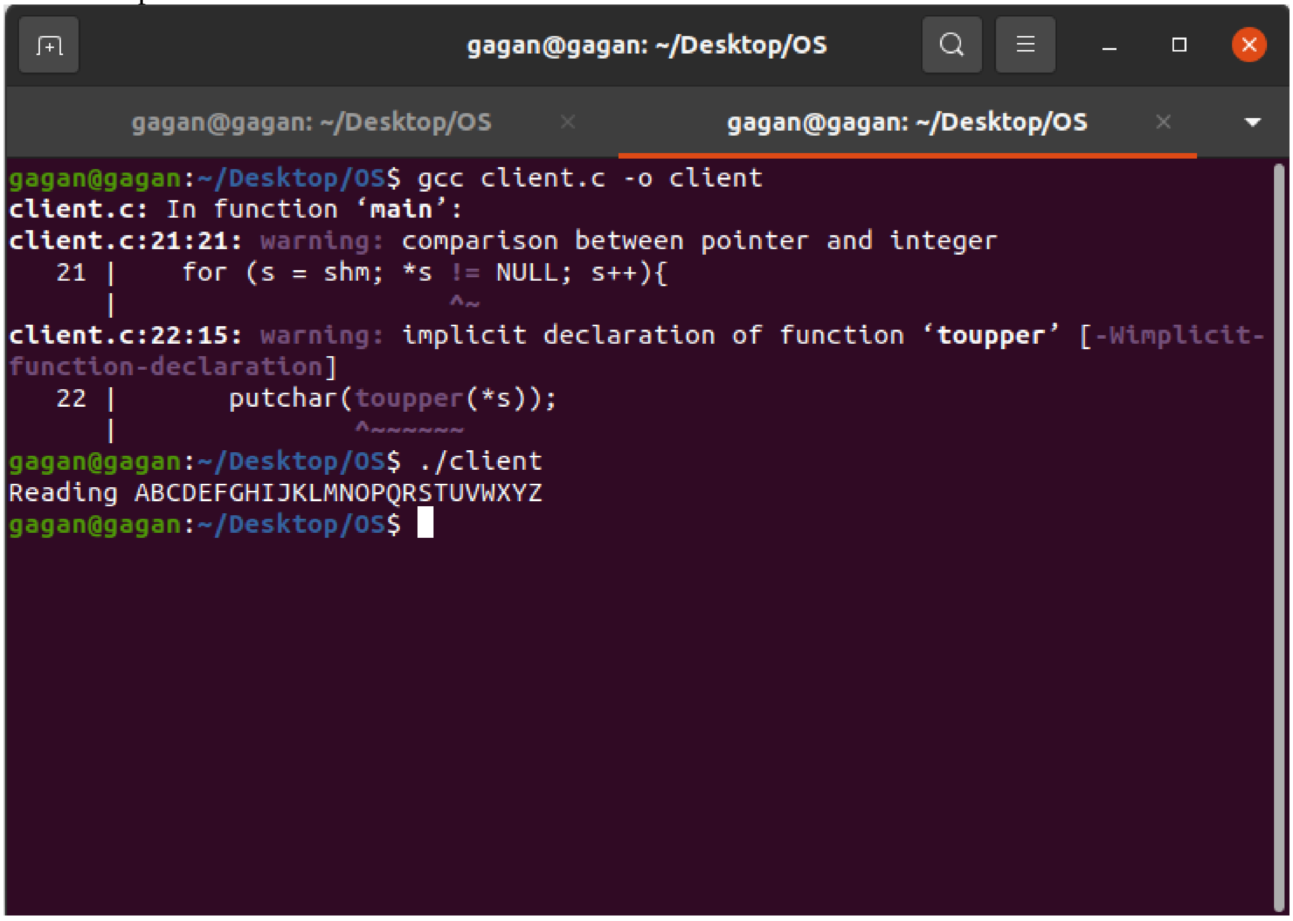
```
gagan@gagan:~/Desktop/OS$ gcc server.c -o server
server.c: In function ‘main’:
server.c:29:8: warning: assignment to ‘char’ from ‘void *’ makes integer from po
inter without a cast [-Wint-conversion]
   29 |     *s = NULL;
      |        ^
server.c:31:9: warning: implicit declaration of function ‘sleep’ [-Wimplicit-fun
ction-declaration]
   31 |         sleep(1);
      |         ^~~~~
gagan@gagan:~/Desktop/OS$ ./server
Writing abcdefghijklmnopqrstuvwxyz
gagan@gagan:~/Desktop/OS$ 
```

Client C Code:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include<stdlib.h>
#define SHMSZ 27
void main(){
int shmid;
key_t key;
char *shm, *s;
key = 5678;
if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
perror("shmget");
exit(1);
}
if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
perror("shmat");
exit(1);
}
printf("Reading ");
for (s = shm; *s != NULL; s++){
putchar(toupper(*s));
}
putchar('\n');
```

```
*shm = '*';
exit(0);
}
```

Client Output:



gagan@gagan: ~/Desktop/OS
gagan@gagan: ~/Desktop/OS
gagan@gagan: ~/Desktop/OS

```
gagan@gagan:~/Desktop/OS$ gcc client.c -o client
client.c: In function ‘main’:
client.c:21:21: warning: comparison between pointer and integer
   21 |     for (s = shm; *s != NULL; s++){
      |                      ^~
client.c:22:15: warning: implicit declaration of function ‘toupper’ [-Wimplicit-
function-declaration]
   22 |        putchar(toupper(*s));
      |                ^~~~~~~
gagan@gagan:~/Desktop/OS$ ./client
Reading ABCDEFGHIJKLMNOPQRSTUVWXYZ
gagan@gagan:~/Desktop/OS$ 
```

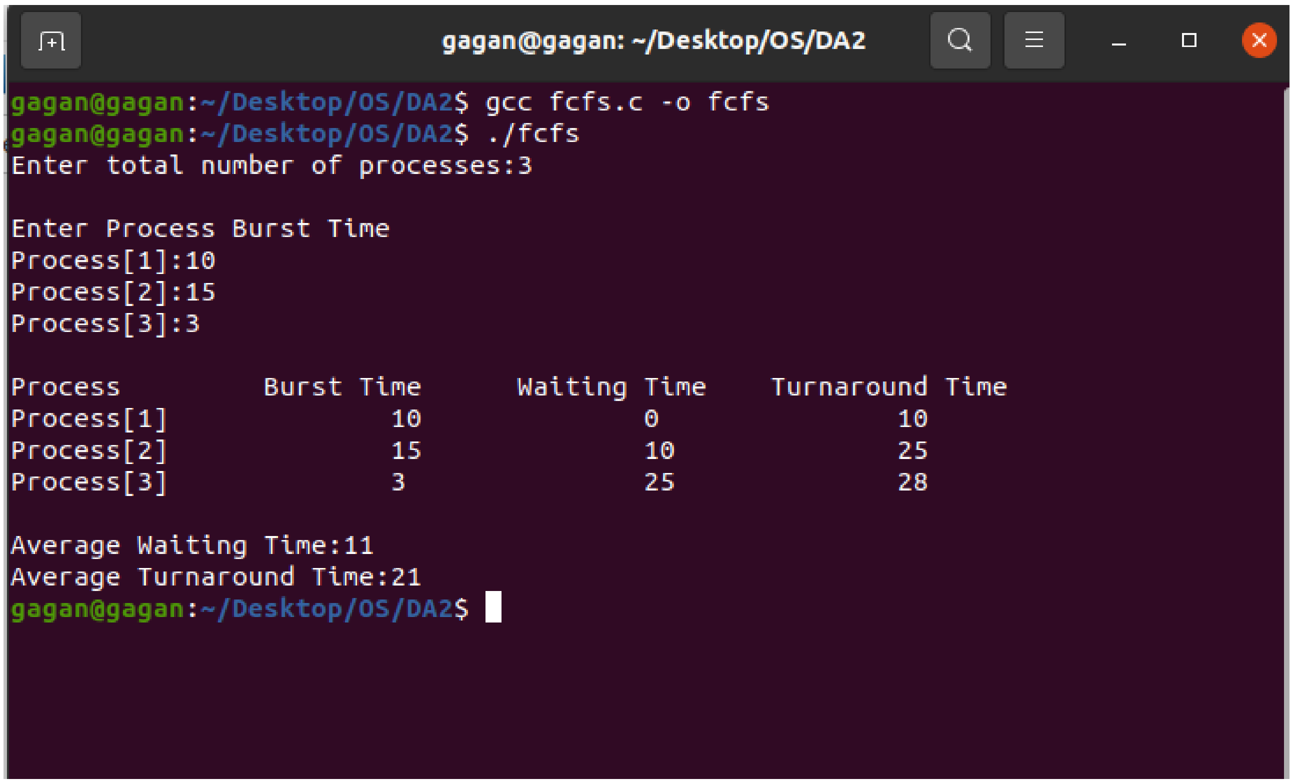
# Current DA

1. FCFS

C Code:

```
#include<stdio.h>
int main(){
int n, burstTime[5], waitingTime[5], turnAroundTime[5], avgWaitingTime = 0, avgTurnAroundTime = 0, i,
j;
printf("Enter total number of processes:");
scanf("%d",&n);
printf("\nEnter Process Burst Time\n");
for(i = 0; i < n; i++){
printf("Process[%d]:",i+1);
scanf("%d",&burstTime[i]);
}
waitingTime[0] = 0;
for(i = 1; i < n; i++){
waitingTime[i] = 0;
for(j = 0; j < i; j++)
waitingTime[i] += burstTime[j];
}
printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++){
turnAroundTime[i] = burstTime[i] + waitingTime[i];
avgWaitingTime += waitingTime[i];
avgTurnAroundTime += turnAroundTime[i];
printf("\nProcess[%d]\t\t%d\t\t%d\t\t%d",i+1, burstTime[i], waitingTime[i], turnAroundTime[i]);
}
avgWaitingTime /= n;
avgTurnAroundTime /= n;

printf("\n\nAverage Waiting Time:%d", avgWaitingTime);
printf("\nAverage Turnaround Time:%d\n", avgTurnAroundTime);
return 0;
}
```

Output:

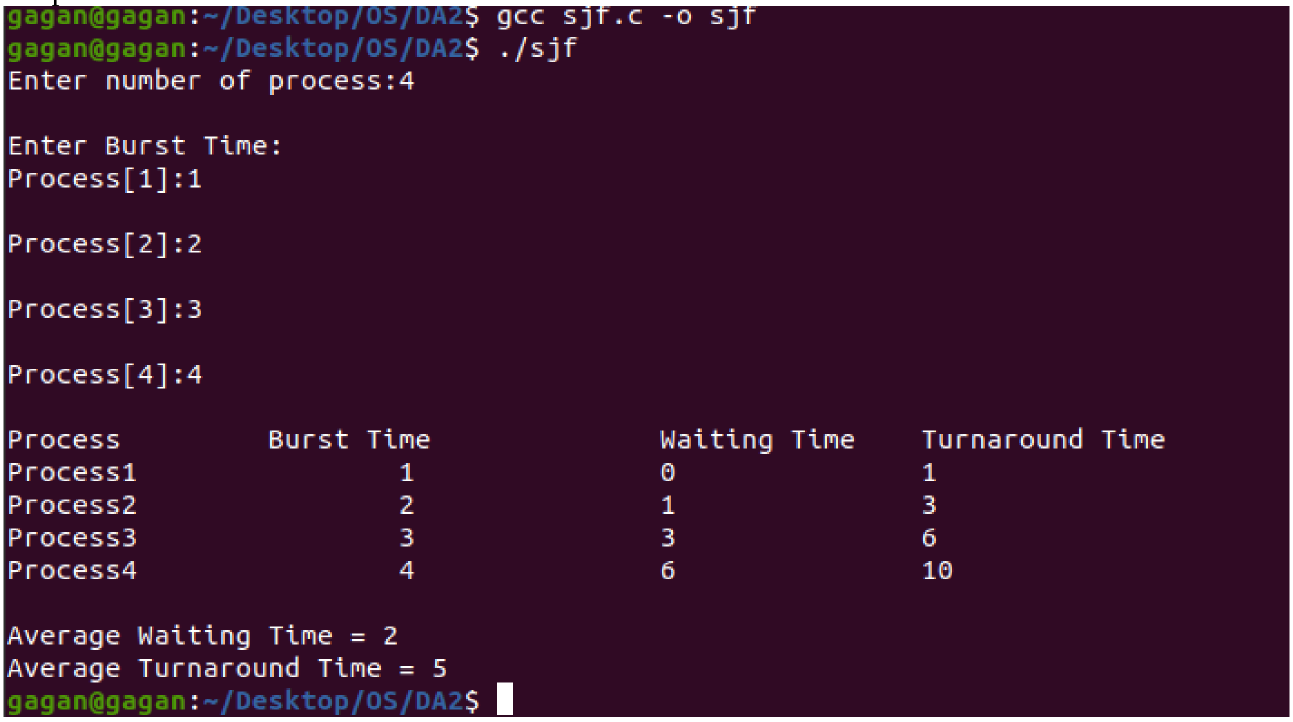


```
gagan@gagan:~/Desktop/OS/DA2$ gcc fcfs.c -o fcfs
gagan@gagan:~/Desktop/OS/DA2$ ./fcfs
Enter total number of processes:3

Enter Process Burst Time
Process[1]:10
Process[2]:15
Process[3]:3

Process         Burst Time      Waiting Time    Turnaround Time
Process[1]              10              0               10
Process[2]              15              10              25
Process[3]              3               25              28

Average Waiting Time:11
Average Turnaround Time:21
gagan@gagan:~/Desktop/OS/DA2$
```

2. SJF
C Code:

```
#include<stdio.h>

int main(){
int burstTime[5],process[5], waitingTime[5], turnAroundTime[5], i, j, n, total = 0, pos, temp,
avgWaitingTime, avgTurnAroundTime;
printf("Enter number of process:");
scanf("%d",&n);
printf("\nEnter Burst Time: ");
for(i = 0;i < n;i++){
printf("\nProcess[%d]:",i+1);
scanf("%d",&burstTime[i]);
process[i] = i + 1;
}
//sorting of burst times
for(i = 0;i < n;i++){
pos = i;
for(j = i + 1;j < n;j++){
if(burstTime[j] < burstTime[pos])
pos = j;
}
temp = burstTime[i];
burstTime[i] = burstTime[pos];
burstTime[pos] = temp;
temp = process[i];
process[i] = process[pos];
process[pos] = temp;
}
waitingTime[0] = 0;
for(i = 1;i < n;i++){
waitingTime[i] = 0;
for(j = 0;j < i;j++)
waitingTime[i] += burstTime[j];
total += waitingTime[i];
}
avgWaitingTime = total / n;
total = 0;
printf("\nProcess\t\tBurst Time\t\tWaiting Time\tTurnaround Time");
for(i = 0;i < n;i++){
turnAroundTime[i] = burstTime[i] + waitingTime[i];
total += turnAroundTime[i];
printf("\nProcess%d\t\t%d\t\t%d\t\t%d", process[i], burstTime[i], waitingTime[i], turnAroundTime[i]);
}
avgTurnAroundTime = total / n;
printf("\n\nAverage Waiting Time = %d",avgWaitingTime);
printf("\nAverage Turnaround Time = %d\n",avgTurnAroundTime);
}
```

Output:



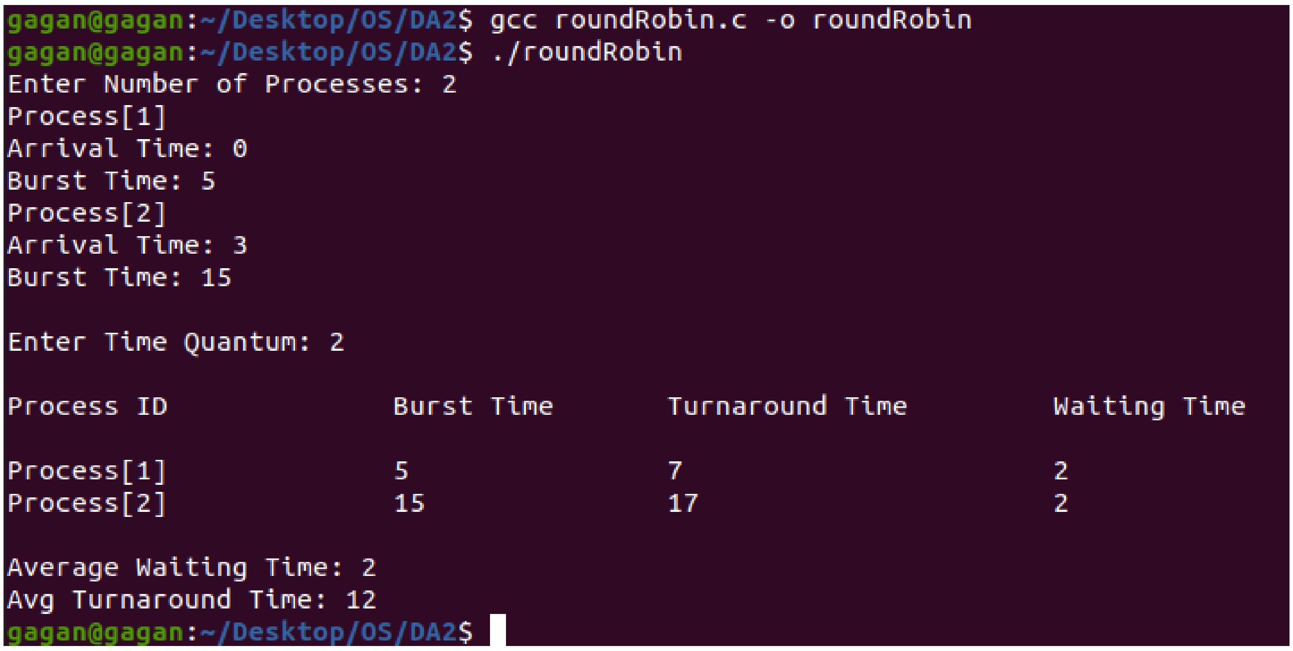
```
gagan@gagan:~/Desktop/OS/DA2$ gcc sjf.c -o sjf
gagan@gagan:~/Desktop/OS/DA2$ ./sjf
Enter number of process:4

Enter Burst Time:
Process[1]:1

Process[2]:2

Process[3]:3

Process[4]:4

Process         Burst Time              Waiting Time    Turnaround Time
Process1                1               0               1
Process2                2               1               3
Process3                3               3               6
Process4                4               6               10

Average Waiting Time = 2
Average Turnaround Time = 5
gagan@gagan:~/Desktop/OS/DA2$
```

3. Round Robin

C Code:

```
#include<stdio.h>
int main(){
int i, n, total = 0, x, counter = 0, timeQuantum;
int waitingTime = 0, turnAroundTime = 0, arrivalTime[5], burstTime[5], temp[5];
int avgWaitingTime, avgTurnAroundTime;
printf("Enter Number of Processes: ");
scanf("%d", &n);
x = n;
for(i = 0; i < n; i++){
printf("Process[%d]\n", i + 1);
printf("Arrival Time: ");
scanf("%d", &arrivalTime[i]);
printf("Burst Time: ");
scanf("%d", &burstTime[i]);
temp[i] = burstTime[i];
}
printf("\nEnter Time Quantum: ");
scanf("%d", &timeQuantum);
printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");
for(total = 0, i = 0; x != 0;){
if(temp[i] <= timeQuantum && temp[i] > 0){
total = total + temp[i];
temp[i] = 0;
counter = 1;
}
else if(temp[i] > 0){
temp[i] = temp[i] - timeQuantum;
total = total + timeQuantum;
}
if(temp[i] == 0 && counter == 1){
x--;
printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burstTime[i], total - arrivalTime[i], total -
arrivalTime[i] - burstTime[i]);
waitingTime = waitingTime + total - arrivalTime[i] - burstTime[i];
turnAroundTime = turnAroundTime + total - arrivalTime[i];
counter = 0;
}
if(i == n - 1){
i = 0;
}
else if(arrivalTime[i + 1] <= total){
i++;
}
else{
i = 0;
}
}
avgWaitingTime = waitingTime / n;
avgTurnAroundTime = turnAroundTime / n;
printf("\n\nAverage Waiting Time: %d", avgWaitingTime);
printf("\nAvg Turnaround Time: %d\n", avgTurnAroundTime);
return 0;
}
```

Output:



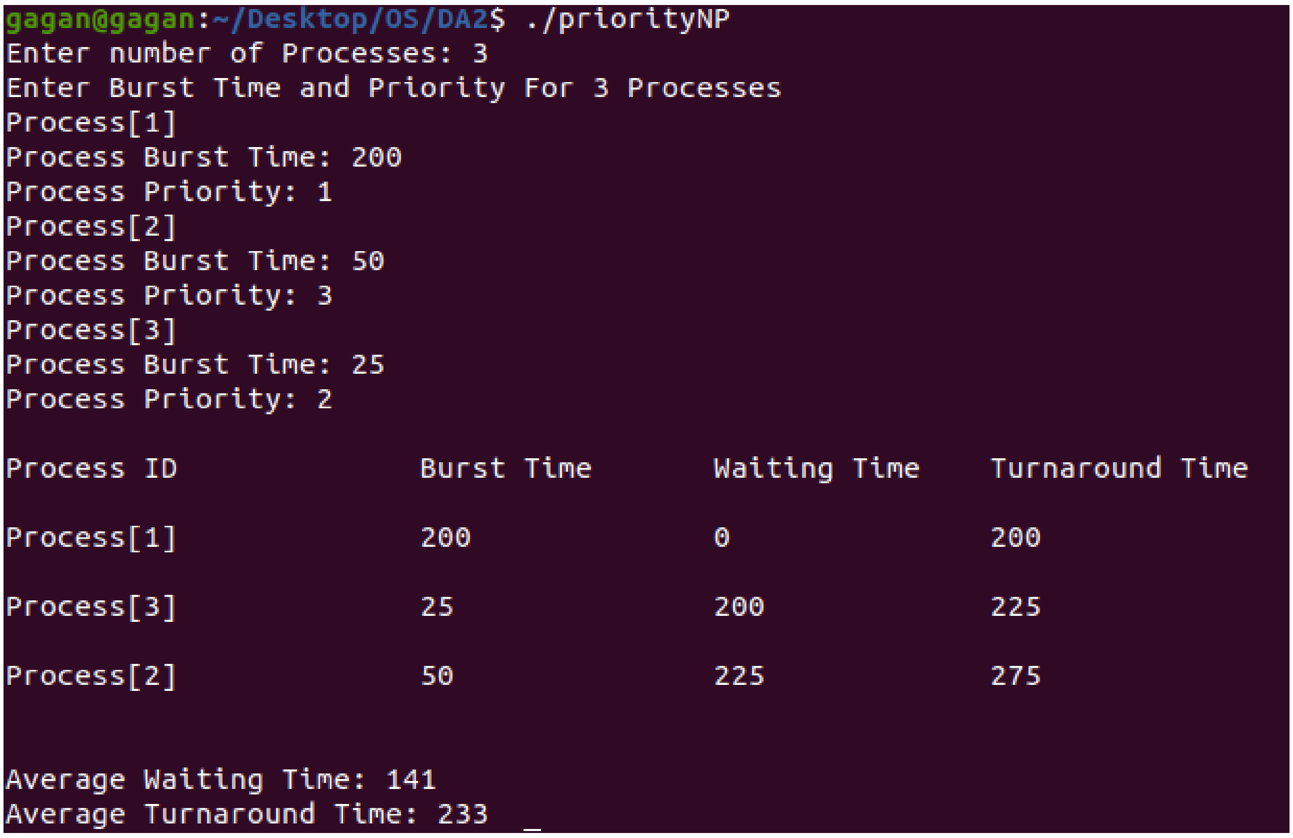
```
gagan@gagan:~/Desktop/OS/DA2$ gcc roundRobin.c -o roundRobin
gagan@gagan:~/Desktop/OS/DA2$ ./roundRobin
Enter Number of Processes: 2
Process[1]
Arrival Time: 0
Burst Time: 5
Process[2]
Arrival Time: 3
Burst Time: 15

Enter Time Quantum: 2

Process ID              Burst Time        Turnaround Time         Waiting Time

Process[1]              5                 7                       2
Process[2]              15                17                      2

Average Waiting Time: 2
Avg Turnaround Time: 12
gagan@gagan:~/Desktop/OS/DA2$ 
```

4. Priority

C Code: Non Preemptive

```
#include<stdio.h>
int main(){
int burstTime[5], process[5], waitingTime[5], turnAroundTime[5], priority[5];
int i, j, n, total = 0, position, temp;
int avgWaitingTime, avgTurnAroundTime;
printf("Enter number of Processes: ");
scanf("%d", &n);
printf("Enter Burst Time and Priority For %d Processes\n", n);
for(i = 0; i < n; i++){
printf("Process[%d]\n", i + 1);
printf("Process Burst Time: ");
scanf("%d", &burstTime[i]);
printf("Process Priority: ");
scanf("%d", &priority[i]);
process[i] = i + 1;
}
for(i = 0; i < n; i++){
position = i;
for(j = i + 1; j < n; j++){
if(priority[j] < priority[position]){
position = j;
}
}
temp = priority[i];
priority[i] = priority[position];
priority[position] = temp;
temp = burstTime[i];
burstTime[i] = burstTime[position];
burstTime[position] = temp;
temp = process[i];
process[i] = process[position];
process[position] = temp;
}
waitingTime[0] = 0;
for(i = 1; i < n; i++)
{
waitingTime[i] = 0;
for(j = 0; j < i; j++)
{
waitingTime[i] = waitingTime[i] + burstTime[j];
}
total += waitingTime[i];
}
avgWaitingTime = total / n;
total = 0;
printf("\nProcess ID\t\tBurst Time\t Waiting Time\t Turnaround Time\n");
for(i = 0; i < n; i++)
{
turnAroundTime[i] = burstTime[i] + waitingTime[i];
total += turnAroundTime[i];
printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\n", process[i], burstTime[i], waitingTime[i],
turnAroundTime[i]);
}
avgTurnAroundTime = total / n;
printf("\n\nAverage Waiting Time: %d", avgWaitingTime);
printf("\nAverage Turnaround Time: %d\n", avgTurnAroundTime);
return 0;
}
```

Output:

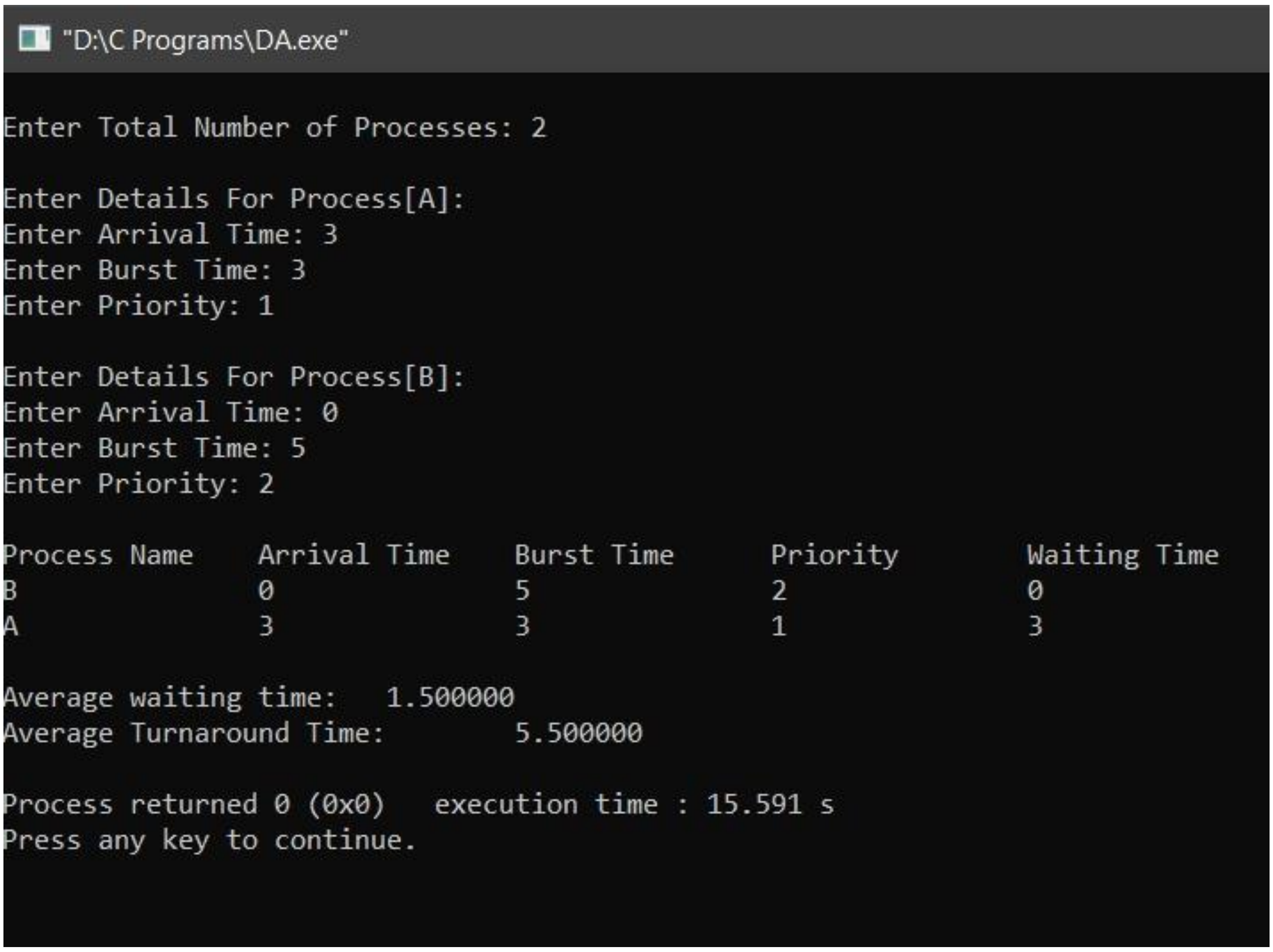

```
gagan@gagan:~/Desktop/OS/DA2$ ./priorityNP
Enter number of Processes: 3
Enter Burst Time and Priority For 3 Processes
Process[1]
Process Burst Time: 200
Process Priority: 1
Process[2]
Process Burst Time: 50
Process Priority: 3
Process[3]
Process Burst Time: 25
Process Priority: 2

Process ID              Burst Time       Waiting Time     Turnaround Time

Process[1]              200              0                200

Process[3]              25               200              225

Process[2]              50               225              275


Average Waiting Time: 141
Average Turnaround Time: 233 _
```

C Code: Preemptive

```
#include<stdio.h>
struct process
{
char processName;
int arrivalTime, burstTime, ct, waitingTime, turnAroundTime, priority;
int status;
}processQueue[5];
int n;
void ArrivalTimeSorting()
{
struct process temp;
int i, j;
for(i = 0; i < n - 1; i++)
{
for(j = i + 1; j < n; j++)
{
if(processQueue[i].arrivalTime > processQueue[j].arrivalTime)
{
temp = processQueue[i];
processQueue[i] = processQueue[j];
processQueue[j] = temp;
}
}
}
}
int main()
{
```

```
int i, time = 0, burstTime = 0, largest;
char c;
float waitingTime = 0, turnAroundTime = 0, avgWaitingTime, avgTurnAroundTime;
printf("\nEnter Total Number of Processes:\t");
scanf("%d", &n);
for(i = 0, c = 'A'; i < n; i++, c++)
{
processQueue[i].processName = c;
printf("\nEnter Details For Process[%C]:\n", processQueue[i].processName);
printf("Enter Arrival Time:\t");
scanf("%d", &processQueue[i].arrivalTime );
printf("Enter Burst Time:\t");
scanf("%d", &processQueue[i].burstTime);
printf("Enter Priority:\t");
scanf("%d", &processQueue[i].priority);
processQueue[i].status = 0;
burstTime += processQueue[i].burstTime;
}
ArrivalTimeSorting();
processQueue[4].priority = -9999;
printf("\nProcess Name\tArrival Time\tBurst Time\tPriority\tWaiting Time");
for(time = processQueue[0].arrivalTime; time < burstTime; time++)
{
largest = 4;
for(i = 0; i < n; i++)
{
if(processQueue[i].arrivalTime <= time && processQueue[i].status != 1 && processQueue[i].priority >
processQueue[largest].priority)
{
largest = i;
}
}
time += processQueue[largest].burstTime;
processQueue[largest].ct = time;
processQueue[largest].waitingTime = processQueue[largest].ct - processQueue[largest].arrivalTime -
processQueue[largest].burstTime;
processQueue[largest].turnAroundTime = processQueue[largest].ct - processQueue[largest].arrivalTime;
processQueue[largest].status = 1;
waitingTime = waitingTime + processQueue[largest].waitingTime;
turnAroundTime = turnAroundTime + processQueue[largest].turnAroundTime;
printf("\n%c\t\t%d\t\t%d\t\t%d\t\t%d", processQueue[largest].processName,
processQueue[largest].arrivalTime, processQueue[largest].burstTime, processQueue[largest].priority,
processQueue[largest].waitingTime);
}
avgWaitingTime = waitingTime / n;
avgTurnAroundTime = turnAroundTime / n;
printf("\n\nAverage waiting time:\t%f\n", avgWaitingTime);
printf("Average Turnaround Time:\t%f\n", avgTurnAroundTime);
return 0;
}
```

Output: (Ran in Windows due to unknown errors in Ubuntu)



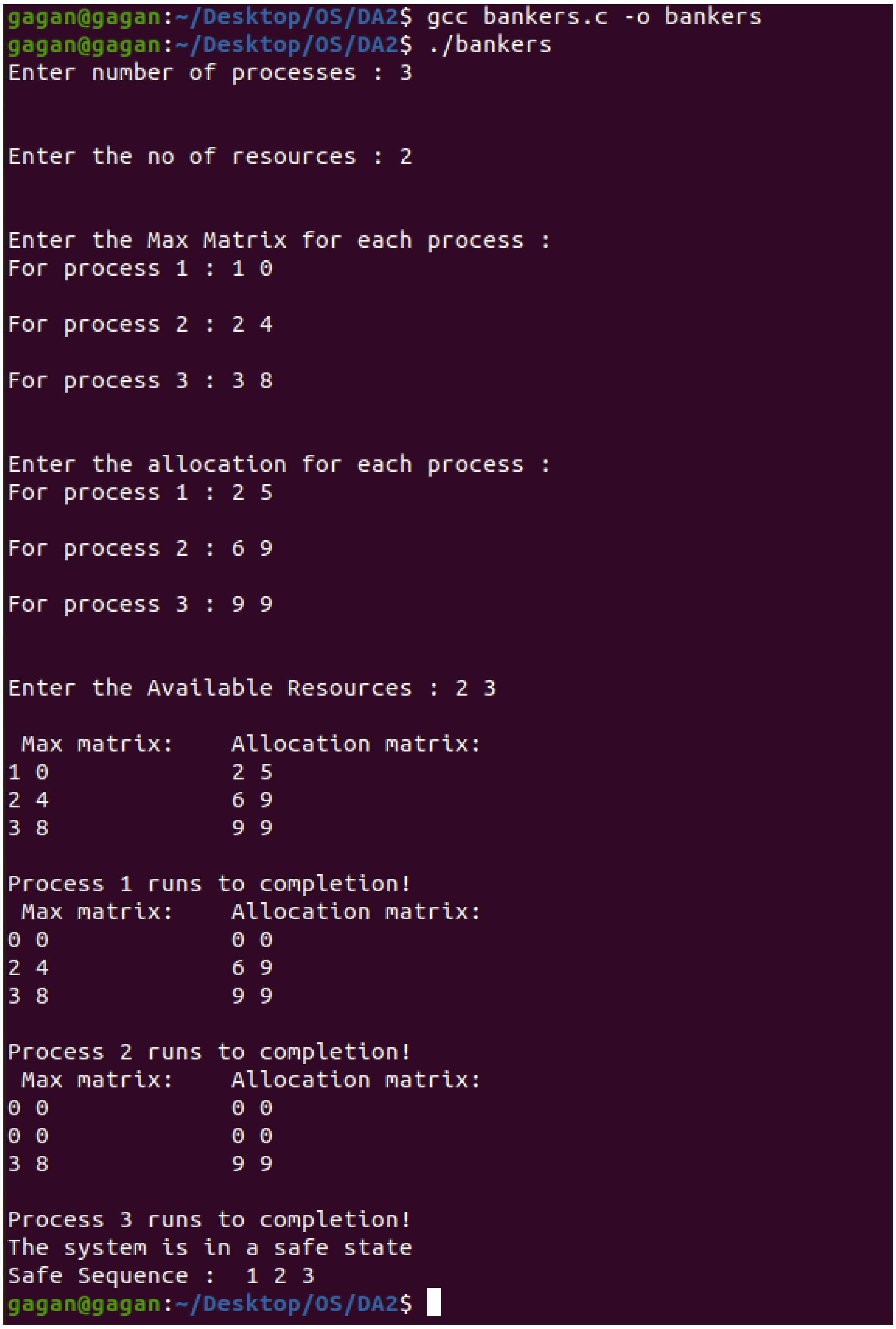
```
Enter Total Number of Processes: 2

Enter Details For Process[A]:
Enter Arrival Time: 3
Enter Burst Time: 3
Enter Priority: 1

Enter Details For Process[B]:
Enter Arrival Time: 0
Enter Burst Time: 5
Enter Priority: 2

Process Name    Arrival Time    Burst Time      Priority        Waiting Time
B               0               5               2               0
A               3               3               1               3

Average waiting time:   1.500000
Average Turnaround Time:        5.500000

Process returned 0 (0x0)   execution time : 15.591 s
Press any key to continue.
```

Bankers Algorithm
C Code:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];
int p, r, i, j, process, count;
count = 0;

printf("Enter number of processes : ");
scanf("%d", &p);

for(i = 0; i< p; i++)
completed[i] = 0;

printf("\n\nEnter the no of resources : ");
scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");
for(i = 0; i < p; i++)
{
printf("\nFor process %d : ", i + 1);
for(j = 0; j < r; j++)
scanf("%d", &max[i][j]);
```

```
}

printf("\n\nEnter the allocation for each process : ");
for(i = 0; i < p; i++)
{
printf("\nFor process %d : ",i + 1);
for(j = 0; j < r; j++)
scanf("%d", &alloc[i][j]);
}

printf("\n\nEnter the Available Resources : ");
for(i = 0; i < r; i++)
scanf("%d", &avail[i]);

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

for(j = 0; j < r; j++)
need[i][j] = max[i][j] - alloc[i][j];

do
{
printf("\n Max matrix:\tAllocation matrix:\n");

for(i = 0; i < p; i++)
{
for( j = 0; j < r; j++)
printf("%d ", max[i][j]);
printf("\t\t");
for( j = 0; j < r; j++)
printf("%d ", alloc[i][j]);
printf("\n");
}

process = -1;

for(i = 0; i < p; i++)
{
if(completed[i] == 0)
{
process = i ;
for(j = 0; j < r; j++)
{
if(avail[j] < need[i][j])
{
process = -1;
break;
}
}
}
if(process != -1)
break;
}

if(process != -1)
{
printf("\nProcess %d runs to completion!", process + 1);
safeSequence[count] = process + 1;
count++;
for(j = 0; j < r; j++)
{
avail[j] += alloc[process][j];
alloc[process][j] = 0;
max[process][j] = 0;
completed[process] = 1;
}
}
}
while(count != p && process != -1);

if(count == p)
{
printf("\nThe system is in a safe state\n");
printf("Safe Sequence : ");
for( i = 0; i < p; i++)
printf("%d ", safeSequence[i]);
printf("\n");
}
else
printf("\nThe system is in an unsafe state");

}
```

Output:



```
gagan@gagan:~/Desktop/OS/DA2$ gcc bankers.c -o bankers
gagan@gagan:~/Desktop/OS/DA2$ ./bankers
Enter number of processes : 3


Enter the no of resources : 2


Enter the Max Matrix for each process :
For process 1 : 1 0

For process 2 : 2 4

For process 3 : 3 8


Enter the allocation for each process :
For process 1 : 2 5

For process 2 : 6 9

For process 3 : 9 9


Enter the Available Resources : 2 3

 Max matrix:    Allocation matrix:
1 0             2 5
2 4             6 9
3 8             9 9

Process 1 runs to completion!
 Max matrix:    Allocation matrix:
0 0             0 0
2 4             6 9
3 8             9 9

Process 2 runs to completion!
 Max matrix:    Allocation matrix:
0 0             0 0
0 0             0 0
3 8             9 9

Process 3 runs to completion!
The system is in a safe state
Safe Sequence :  1 2 3
gagan@gagan:~/Desktop/OS/DA2$ 
```