Name: Fiza Mansoor Patel

Roll no: 191 **Div.: C**

PRNo: 0120190425 **Batch: C3**

Assignment 4

Write a menu driven program for implementing CPU Scheduling Algorithms- FCFS(With arrival time), SJF(Preemptive, Non-Preemptive), Priority(Non Preemptive) & Round Robin

```
#include <iostream>
#include <iomanip>
#include <string.h>
#include <stdlib.h>
using namespace std;
//Class for First Come First Serve scheduling
class fcfs
{
 int process[50][4]; //For storing Process information
 int processCount; //Total Processes
public:
 //Constructor
 fcfs()
 {
 cout<<"\n 1] FCFS with Arrival Time";
 cout<<"\n 2] FCFS without Arrival Time";
 cout<<"\n Select One Option: ";
```

```
int option1;
cin>>option1;
reenter:
cout<<"\n--> How many processes want to schedule: ";
cin>>processCount;
if(processCount > 50)
{
cout<<"\n--- Please enter value less than or equal to 50 ---\n";
goto reenter; //If processCount Exceed predeffined value limit
}
memset(process, 0, sizeof(process)); //Filling array with 0
for(int i=0; i<processCount; i++)
{
cout<<"\n--> Process P"<<i;
cout<<"\n\t Burst Time: ";
cin>>process[i][0];
if(option1 == 2)
{
process[i][1] = 0;
}
else
{
cout<<"\t Arrival Time: ";
cin>>process[i][1];
}
}
}
```

```
 //Function for scheduling
 void scheduling()
 {
 cout<<"\n\n--- Gantt Chart ---\n\n";
 cout<<"0";
 for(int i=0, time=0, countIter=0, processesCompleted=0;
i<processCount; i++)
 {
 countIter++; //To keep track of process checked at a given time
 if(countIter > processCount)
 {
 time++;
 cout<<"-"<<time;
 countIter = 0;
 }
 if(process[i][1] <= time) //Checking arrival time
 {
 int diff = process[i][0] - (process[i][3] - process[i][2]);
 if(diff > 0) //Checking whether process is not executed
 {
 countIter = 0;
 process[i][2] = time; //Waiting time
 time += process[i][0];
 process[i][3] = time; //Turnaround time
 cout<<"| P"<<i<<" |"<<time;
 processesCompleted++;
 if(processesCompleted == processCount) //Checking whether
```

```
all processes are completed
 break;
 }
 }
 if(i == processCount-1) //Resetting loop
 i = -1;
 }
 //Printing Output
 cout<<"\n\n"<<setw(5)<<"Process"<<setw(5)<<"WT"<<setw(5)<<"TAT\n";
 for(int i=0; i<processCount; i++)
 {

cout<<setw(5)<<"P"<<i<<setw(5)<<process[i][2]<<setw(5)<<process[i][3]<<"\n";
 }
 }
};
//Class for Shortest Job First scheduling
class sjf
{
int process[50][4]; //For storing Process information
 int processCount; //Total Processes
public:
 //Constructor
 sjf()
 {
 reenter:
 cout<<"\n--> How many processes want to schedule: ";
```

```
 cin>>processCount;
 if(processCount > 50)
 {
 cout<<"\n--- Please enter value less than or equal to 50 ---\n";
 goto reenter; //If processCount Exceed predeffined value limit
 }
 memset(process, 0, sizeof(process)); //Filling array with 0
 for(int i=0; i<processCount; i++)
 {
 cout<<"\n--> Process P"<<i;
 cout<<"\n\t Burst Time: ";
 cin>>process[i][0];
 cout<<"\t Arrival Time: ";
 cin>>process[i][1];
 }
 }
 //Function for non-preemptive scheduling
 void nonPreemptiveScheduling()
 {
 cout<<"\n\n--- Gantt Chart ---\n\n";
 cout<<"0";
 for(int time=0, processesCompleted=0;;)
 {
 int currentProcess = ShortestJob(time); //Finding Shortest job at
a given time
 if(currentProcess == -1) //If no current process available
 {
```

```
 time++;
 cout<<"-"<<time;
 }
 else
 {
 process[currentProcess][2] = time; //Waiting time
 time += process[currentProcess][0];
 process[currentProcess][3] = time; //Turnaround time
 cout<<"| P"<<currentProcess<<" |"<<time;
 processesCompleted++;
 if(processesCompleted == processCount) //Checking whether all
processes are completed
 break;
 }
 }
 //Printing Output
 cout<<"\n\n"<<setw(5)<<"Process"<<setw(5)<<"WT"<<setw(5)<<"TAT\n";
 for(int i=0; i<processCount; i++)
 {

cout<<setw(5)<<"P"<<i<<setw(5)<<process[i][2]<<setw(5)<<process[i][3]<<"\n";
 }
 }
 //Function for non-preemptive scheduling
 void PreemptiveScheduling()
 {
 cout<<"\n\n--- Gantt Chart ---\n\n";
```

```
 for(int time=0, previousProcess=-1, processesCompleted=0;;)
 {
 int currentProcess = ShortestJob(time); //Finding Shortest job at
a given time
 if(currentProcess == -1) //If no current process available
 {
 cout<<time<<"-";
 time++;
 continue;
 }
 if(previousProcess != currentProcess) //If previous process is
completed
 {
 previousProcess = currentProcess;
 cout<<time<<"| P"<<currentProcess<<" |";
 }
 process[currentProcess][3]++; //Total executed time
 time++;
 if(process[currentProcess][0] - (process[currentProcess][3] -
process[currentProcess][2]) <= 0) //If current process is completed
 {
 process[currentProcess][3] = time; //Turnaround time
 process[currentProcess][2] = time -
process[currentProcess][0]; //Waiting time
 processesCompleted++;
 if(processesCompleted == processCount) //Checking whether all
processes are completed
```

```
 {
 cout<<time;
 break;
 }
 }
 }
 //Printing Output
 cout<<"\n\n"<<setw(5)<<"Process"<<setw(5)<<"WT"<<setw(5)<<"TAT\n";
 for(int i=0; i<processCount; i++)
 {

cout<<setw(5)<<"P"<<i<<setw(5)<<process[i][2]<<setw(5)<<process[i][3]<<"\n";
 }
 }
 //Function for finding shortest job at given time
 int ShortestJob(int time)
 {
 int ShortestJobIndex = -1;
 int ShortestBurstTime;
 int i = 0;
 for(i; i<processCount; i++)
 {
 int diff = process[i][0] - (process[i][3] - process[i][2]);
 if(process[i][1] <= time && diff>0) //Process is within arrival
time and not completed
 {
 ShortestJobIndex = i;
```

```
 ShortestBurstTime = process[i][0];
 break;
 }
 }
 for(i; i<processCount; i++)
 {
 int diff = process[i][0] - (process[i][3] - process[i][2]);
 if(process[i][1] <= time && diff>0 && process[i][0] <
ShortestBurstTime) //If process is within arrival time & not completed & have
shortest burst time
 {
 ShortestJobIndex = i;
 ShortestBurstTime = process[i][0];
 }
 }
 return ShortestJobIndex;
 }
};
//Class for Priority Scheduling
class priority
{
 int process[50][5]; //For storing Process information
 int processCount; //Total Processes
public:
 //Constructor
 priority()
 {
```

```
 reenter:
 cout<<"\n--> How many processes want to schedule: ";
 cin>>processCount;
 if(processCount > 50)
 {
 cout<<"\n--- Please enter value less than or equal to 50 ---\n";
 goto reenter; //If processCount Exceed predeffined value limit
 }
 memset(process, 0, sizeof(process)); //Filling array with 0
 for(int i=0; i<processCount; i++)
 {
 cout<<"\n--> Process P"<<i;
 cout<<"\n\t Burst Time: ";
 cin>>process[i][0];
 cout<<"\t Arrival Time: ";
 cin>>process[i][1];
 cout<<"\t Priority: ";
 cin>>process[i][4];
 }
 }
 //Function for non-preemptive scheduling
void nonPreemptiveScheduling()
 {
 cout<<"\n\n--- Gantt Chart ---\n\n";
 cout<<"0";
 for(int time=0, processesCompleted=0;;)
 {
```

```
 int currentProcess = PriorityJob(time); //Finding Priority job at
a given time
 if(currentProcess == -1) //If no current process available
 {
 time++;
 cout<<"-"<<time;
 }
 else
 {
 process[currentProcess][2] = time; //Waiting time
 time += process[currentProcess][0];
 process[currentProcess][3] = time; //Turnaround time
 cout<<"| P"<<currentProcess<<" |"<<time;
 processesCompleted++;
 if(processesCompleted == processCount) //Checking whether all
processes are completed
 break;
 }
 }
 //Printing Output

cout<<"\n\n"<<setw(10)<<"Process"<<setw(10)<<"Priority"<<setw(8)<<"WT"<<setw(1
0)<<"TAT\n";
 for(int i=0; i<processCount; i++)
 {

cout<<setw(6)<<"P"<<i<<setw(10)<<process[i][4]<<setw(10)<<process[i][2]<<setw(
```

```
10)<<process[i][3]<<"\n";
 }
 }
 //Function for non-preemptive scheduling
 void PreemptiveScheduling()
 {
 cout<<"\n\n--- Gantt Chart ---\n\n";
 for(int time=0, previousProcess=-1, processesCompleted=0;;)
 {
 int currentProcess = PriorityJob(time); //Finding Priority job at
a given time
 if(currentProcess == -1) //If no current process available
 {
 cout<<time<<"-";
 time++;
 continue;
 }
 if(previousProcess != currentProcess) //If previous process is
completed
 {
 previousProcess = currentProcess;
 cout<<time<<"| P"<<currentProcess<<" |";
 }
 process[currentProcess][3]++; //Total executed time
 time++;
 if(process[currentProcess][0] - (process[currentProcess][3] -
process[currentProcess][2]) <= 0) //If current process is completed
```

```
 {
 process[currentProcess][3] = time; //Turnaround time
 process[currentProcess][2] = time -
process[currentProcess][0]; //Waiting time
 processesCompleted++;
 if(processesCompleted == processCount) //Checking whether all
processes are completed
 {
 cout<<time;
 break;
 }
 }
 }
 //Printing Output

cout<<"\n\n"<<setw(10)<<"Process"<<setw(10)<<"Priority"<<setw(8)<<"WT"<<setw(1
0)<<"TAT\n";
 for(int i=0; i<processCount; i++)
 {

cout<<setw(6)<<"P"<<i<<setw(10)<<process[i][4]<<setw(10)<<process[i][2]<<setw(
10)<<process[i][3]<<"\n";
 }
 }
 //Function for finding shortest job at given time
 int PriorityJob(int time)
 {
```

```
 int PriorityJobIndex = -1;
 int HighestPriority;
 int i = 0;
 for(i; i<processCount; i++)
 {
 int diff = process[i][0] - (process[i][3] - process[i][2]);
 if(process[i][1] <= time && diff>0) //Process is within arrival
time and not completed
 {
 PriorityJobIndex = i;
 HighestPriority = process[i][4];
 break;
 }
 }
 for(i; i<processCount; i++)
 {
 int diff = process[i][0] - (process[i][3] - process[i][2]);
 if(process[i][1] <= time && diff>0 && process[i][4] <
HighestPriority) //If process is within arrival time & not completed & have
highest priority
 {
 PriorityJobIndex = i;
 HighestPriority = process[i][4];
 }
 }
 return PriorityJobIndex;
 }
```

```
};
//Class for Round Robin Scheduling
class rr
{
 int process[50][4]; //For storing Process information
 int processCount; //Total Processes
 int quantum;
public:
 //Constructor
 rr()
 {
 reenter:
 cout<<"\n--> How many processes want to schedule: ";
 cin>>processCount;
 if(processCount > 50)
 goto reenter; //If processCount Exceed predeffined value limit
 cout<<"\n--> Quantum/Slice Time: ";
 cin>>quantum;
 memset(process, 0, sizeof(process)); //Filling array with 0
 for(int i=0; i<processCount; i++)
 {
 cout<<"\n--> Process P"<<i;
 cout<<"\n\t Burst Time: ";
 cin>>process[i][0];
 cout<<"\t Arrival Time: ";
 cin>>process[i][1];
 }
```

```
 }
 //Function for scheduling
 void scheduling()
 {
 int processesCompleted = 0;
 cout<<"\n\n--- Gantt Chart ---\n\n";
 cout<<"0";
 for(int i=0, time=0, countIter=0; i<processCount; i++)
 {
 countIter++; //To keep track of process checked at a given time
 if(countIter > processCount)
 {
 time++;
 cout<<"-"<<time;
 countIter = 0;
 }
 if(process[i][1] <= time) //Checking arrival time
 {
 int diff = process[i][0] - (process[i][3] - process[i][2]);
 if(diff > 0) //Checking whether process is not executed
 {
 countIter = 0;
 cout<<"| P"<<i<<" |";
 if(diff > quantum) //If remaining execution time is
greater than quantum time
 {
 process[i][3] += quantum;
```

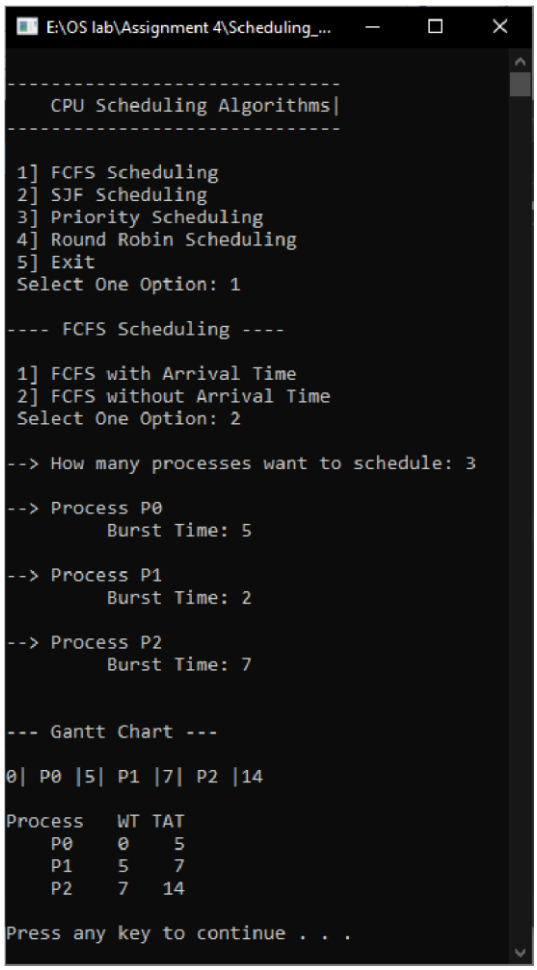
```
 time += quantum;
 }
 else
 {
 time += diff;
 process[i][3] = time - process[i][1]; //Turn Around
Time
 process[i][2] = process[i][3] - process[i][0];
//Waiting Time
 processesCompleted++;
 if(processesCompleted == processCount) //Checking
whether all processes are completed
 {
 cout<<time;
 break;
 }
 }
 cout<<time;
 }
 }
 if(i == processCount-1) //Resetting loop
 i = -1;
 }
 //Printing Output
 cout<<"\n\n"<<setw(5)<<"Process"<<setw(5)<<"WT"<<setw(5)<<"TAT\n";
 for(int i=0; i<processCount; i++)
 {
```

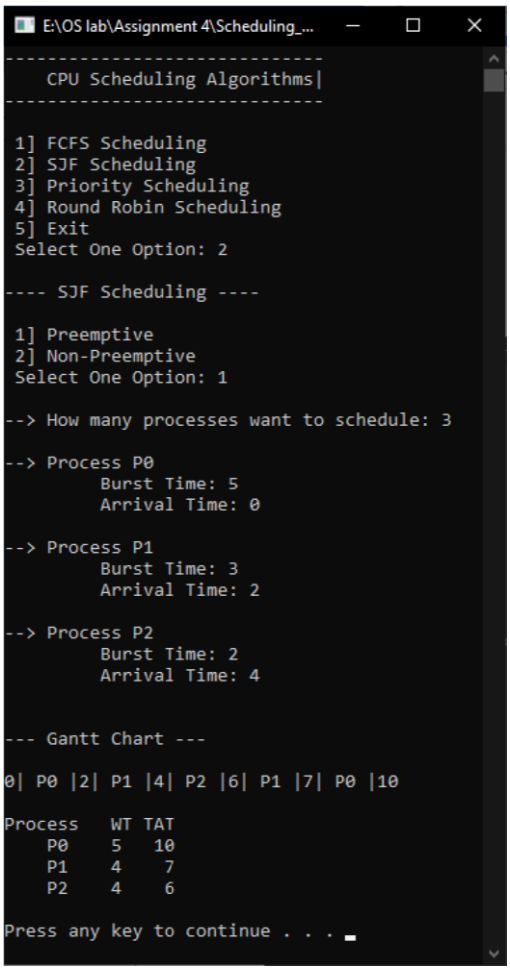
```
cout<<setw(5)<<"P"<<i<<setw(5)<<process[i][2]<<setw(5)<<process[i][3]<<"\n";
 }
 }
};
//Menu driven code
int main()
{
restart:
 system("cls");
 cout<<"\n-------------------------------";
 cout<<"\n CPU Scheduling Algorithms|";
 cout<<"\n-------------------------------\n";
 cout<<"\n 1] FCFS Scheduling";
 cout<<"\n 2] SJF Scheduling";
 cout<<"\n 3] Priority Scheduling";
 cout<<"\n 4] Round Robin Scheduling";
 cout<<"\n 5] Exit";
 cout<<"\n Select One Option: ";
 int option1;
 cin>>option1;
 switch(option1)
 {
 case 1:{
 cout<<"\n---- FCFS Scheduling ----\n";
 fcfs f1;
 f1.scheduling();
```

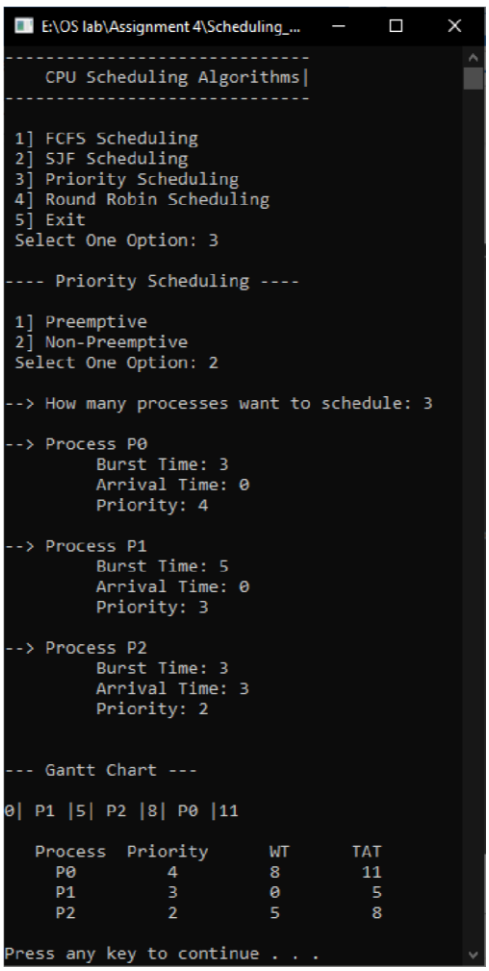
```
break;
}
case 2:{
cout<<"\n---- SJF Scheduling ----\n";
cout<<"\n 1] Preemptive";
cout<<"\n 2] Non-Preemptive";
cout<<"\n Select One Option: ";
int option2;
cin>>option2;
sjf s1;
if(option2 == 1)
s1.PreemptiveScheduling();
else
s1.nonPreemptiveScheduling();
break;
}
case 3:{
cout<<"\n---- Priority Scheduling ----\n";
cout<<"\n 1] Preemptive";
cout<<"\n 2] Non-Preemptive";
cout<<"\n Select One Option: ";
int option2;
cin>>option2;
priority p1;
if(option2 == 1)
p1.PreemptiveScheduling();
else
```

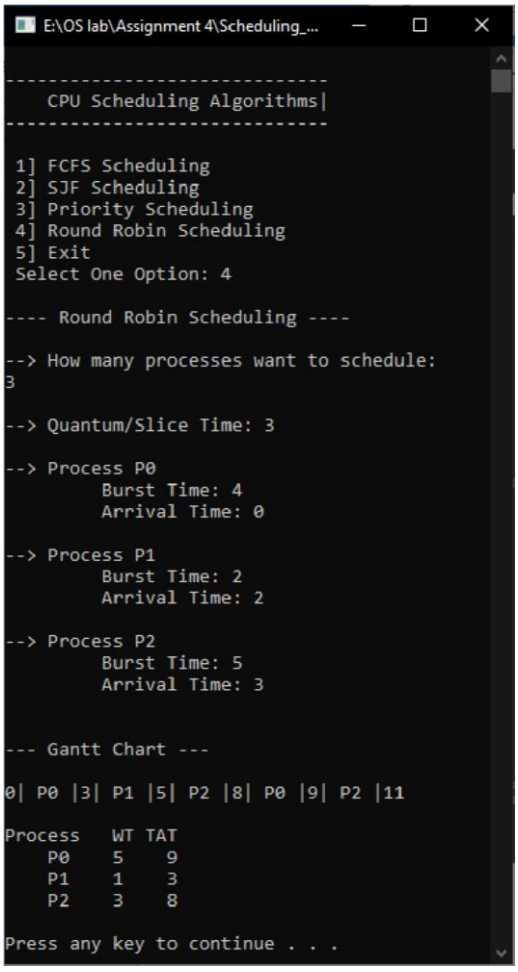
```
 p1.nonPreemptiveScheduling();
 break;
 }
 case 4:{
 cout<<"\n---- Round Robin Scheduling ----\n";
 rr r1;
 r1.scheduling();
 break;
 }
 case 5:
 cout<<"\n\n---- Thanks for Being with Us ----\n\n";
 break;
 default:
 cout<<"\n\n---- Enter a valid option ----\n\n";
 }
 cout<<endl;
 system("pause");
 if(option1 != 5)
 goto restart;
 return 0;
}
```

OUTPUT:


E:\OS lab\Assignment 4\Scheduling_...
CPU Scheduling Algorithms|
1] FCFS Scheduling
2] SJF Scheduling
3] Priority Scheduling
4] Round Robin Scheduling
5] Exit
Select One Option: 1
---- FCFS Scheduling ----
1] FCFS with Arrival Time
2] FCFS without Arrival Time
Select One Option: 2
--> How many processes want to schedule: 3
--> Process P0
Burst Time: 5
--> Process P1
Burst Time: 2
--> Process P2
Burst Time: 7
--- Gantt Chart ---
0| P0 |5| P1 |7| P2 |14
Process WT TAT
P0 0 5
P1 5 7
P2 7 14
Press any key to continue . . .


E:\OS lab\Assignment 4\Scheduling_...
CPU Scheduling Algorithms|
1] FCFS Scheduling
2] SJF Scheduling
3] Priority Scheduling
4] Round Robin Scheduling
5] Exit
Select One Option: 2
---- SJF Scheduling ----
1] Preemptive
2] Non-Preemptive
Select One Option: 1
--> How many processes want to schedule: 3
--> Process P0
Burst Time: 5
Arrival Time: 0
--> Process P1
Burst Time: 3
Arrival Time: 2
--> Process P2
Burst Time: 2
Arrival Time: 4
--- Gantt Chart ---
0| P0 |2| P1 |4| P2 |6| P1 |7| P0 |10
Process WT TAT
P0 5 10
P1 4 7
P2 4 6
Press any key to continue . . .


E:\OS lab\Assignment 4\Scheduling_...
CPU Scheduling Algorithms|
1] FCFS Scheduling
2] SJF Scheduling
3] Priority Scheduling
4] Round Robin Scheduling
5] Exit
Select One Option: 3
---- Priority Scheduling ----
1] Preemptive
2] Non-Preemptive
Select One Option: 2
--> How many processes want to schedule: 3
--> Process P0
Burst Time: 3
Arrival Time: 0
Priority: 4
--> Process P1
Burst Time: 5
Arrival Time: 0
Priority: 3
--> Process P2
Burst Time: 3
Arrival Time: 3
Priority: 2
--- Gantt Chart ---
0| P1 |5| P2 |8| P0 |11
Process Priority WT TAT
P0 4 8 11
P1 3 0 5
P2 2 5 8
Press any key to continue . . .


E:\OS lab\Assignment 4\Scheduling_...
CPU Scheduling Algorithms|
1] FCFS Scheduling
2] SJF Scheduling
3] Priority Scheduling
4] Round Robin Scheduling
5] Exit
Select One Option: 4
---- Round Robin Scheduling ----
--> How many processes want to schedule:
3
--> Quantum/Slice Time: 3
--> Process P0
Burst Time: 4
Arrival Time: 0
--> Process P1
Burst Time: 2
Arrival Time: 2
--> Process P2
Burst Time: 5
Arrival Time: 3
--- Gantt Chart ---
0| P0 |3| P1 |5| P2 |8| P0 |9| P2 |11
Process WT TAT
P0 5 9
P1 1 3
P2 3 8
Press any key to continue . . .