Operating System - CS480

CPU Scheduling Report

Simple Non-Pre-emptive

Table of contents:

Problem Description:

This is a CPU scheduling algorithm that has the algorithm types (FCFS, SJF, SRTF, Priority).
The program will ask the user to enter the number of processes, then CPU times and the priorities for each, thereafter the program will calculate the waiting individually and average waiting time to find out which algorithm is most efficient.

Code Steps:

FCFS:

1. Take number of processes(n) and CPU time(bt[]) for each process
2. Arrival time =0 for all processes and waiting time for first process=0
3. Calculate waiting time for each process (waiting time = sum of CPU time for the previous processes)
4. Calculate total waiting time(sum of waiting time for all processes)
5. Display results
6. Average waiting time total/number of processes
7. Display average waiting time

SJF:

1. Take number of processes(n) and CPU time(bt[]) for each process
2. Sort burst time using selection sort
    - Step 1: set min to the first location
    - Step 2: search the minimum element in bt[]
    - Step 3: swap the first location with the minimum value in bt[]
    - Step 4: swap process number p[] and repeat the steps
3. Arrival time =0 for all processes and waiting time for first process=0
4. Calculate waiting time for each process (waiting time = sum of CPU time for the previous processes)
5. Calculate total waiting time(sum of waiting time for all processes)
6. Sort burst time, process number and waiting time using selection sort in ascending order for process number
7. Display results

8. Average waiting time total/number of processes
9. Display average waiting time

SRTF:

Since the arrival time is equal to 0, SRTF is equivalents to SJF, but if the arrival time for each process is different, then it will follow the same steps as the SJF except that it will check each time for which has less remaining time and if it arrived or not. If not arrived then it will complete the current process, if it's arrived then it will preempt the current process and start the new one, and the procedure will be repeated each time a new process will arrive.

Priority:

1. Take number of processes(n) and CPU time(bt[]) for each process
2. Take priority number for each process
3. Sort priority using selection sort
    - Step 1: set min to the first location
    - Step 2: search the minimum element in pr1[]
    - Step 3: swap the first location with the minimum value in bt[]
    - Step 4: swap process number p[] and bt[] and repeat the steps
4. Arrival time =0 for all processes and waiting time for first process=0
5. Calculate waiting time for each process (waiting time = sum of CPU time for the previous processes)
6. Calculate total waiting time(sum of waiting time for all processes)
7. Sort priority, burst time, process number and waiting time using selection sort in ascending order for process number
8. Display results
9. Average waiting time total/number of processes
10. Display average waiting time

Program Codes:

```
#include <stdio.h>

void FCFS(int n, int bt[], int wt[],int arr);

void SJF(int n, int bt[],int p[], int wt[]);

void PRIORITY(int n, int bt1[],int p1[], int wt1[]);

int main()

{

printf("Enter How many processes: ");

int n;

scanf("%d",&n);

int bt[n]; //burst time

int wt[n]; //waiting time

int p[n]; //process number

int bt1[n]; // copy of the arrays

int wt1[n];

int p1[n];

for (int i = 0; i < n ; i++ ) // take burst time from user

{

printf("Enter CPU time for P%d: ",i+1);

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

p[i]=i+1;

}

for (int i = 0; i < n ; i++ ) // copy contents in another array
```

```
{
bt1[i]=bt[i];
wt1[i]=wt[i];
p1[i]=p[i];
}
int arr=0; //arrival time for all processes is 0
// apply functions for each algorithm
FCFS(n,bt,wt,arr);
SJF(n,bt,p,wt);
PRIORITY(n,bt1,p1,wt1);
return 0;
}
void FCFS(int n, int bt[], int wt[],int arr){
wt[0]=0; //waiting time for first process is always 0
for (int i = 1; i < n ; i++ ) //calculating waiting time for each process
{
wt[i]=bt[i-1]+arr;
arr=wt[i];
}
printf( "\n\t First Come First Served Scheduling\n");
double totalw=0.0;
for (int i = 1; i < n ; i++ ) //total waiting time
totalw+=wt[i];
printf("\nProcesses \tCPU Time\tWaiting Time \n"); //output
```

```
for(int i=0;i<n;i++)
{
printf("\nP %d\t\t %d\t\t %d",i+1,bt[i],wt[i]);
}
double avg=totalw/n; // average waiting time total/number of processes
printf("\n\nAverage Waiting Time %.2f \n",avg);
}
void SJF(int n, int bt[],int p[], int wt[]){
int i,j,k=0,min,temp ;
double total=0.0;
//sorting of burst times
for(i=0;i<n;i++)
{
min=i; // step 1) set min to the first location
for(j=i+1;j<n;j++)
{
if(bt[j]<bt[min]) // step 2) search the minimum element in bt[]
min=j;
}
temp=bt[i]; // step 3) swap the first location with the minimum value in bt[]
bt[i]=bt[min];
bt[min]=temp;
temp=p[i]; // step 4) swap process number p[] and repeat the steps
p[i]=p[min];
```

Since the arrival time is equal to 0
SRTF is equivalents to SJF

```
p[min]=temp;
}
wt[0]=0;
int arr=0;
for (int i = 1; i < n ; i++ ) //calculating waiting time for each process
{
wt[i]=bt[i-1]+arr;
arr=wt[i];
}
for (int i = 1; i < n ; i++ ) //total waiting time
total+=wt[i];
for(i=0;i<n;i++)
{
min=i;
for(j=i+1;j<n;j++)
{
if(p[j]<p[min])
min=j;
}
temp=bt[i];
bt[i]=bt[min];
bt[min]=temp;
temp=p[i];
p[i]=p[min];
```

```
p[min]=temp;
temp=wt[i];
wt[i]=wt[min];
wt[min]=temp;
}
//display results
printf( "\n\t Shortest Job First Scheduling \n");
printf("\nProcesses \tCPU Time\tWaiting Time \n"); //output
for(i=0;i<n;i++)
{
printf("\nP %d\t\t %d\t\t %d",p[i],bt[i],wt[i]);
}
double avg=total/n; // calculate average
printf("\n\nAverage Waiting Time %.2f \n",avg);
}
void PRIORITY(int n, int bt1[],int p1[], int wt1[]){
int pr1[n],i,j,k=0,min,temp;
double avg, total=0.0;
for(i=0;i<n;i++)
{
printf("\nEnter priority for P[%d]: ",i+1);
scanf("%d",&pr1[i]);
p1[i]=i+1; //contains process number
}
```

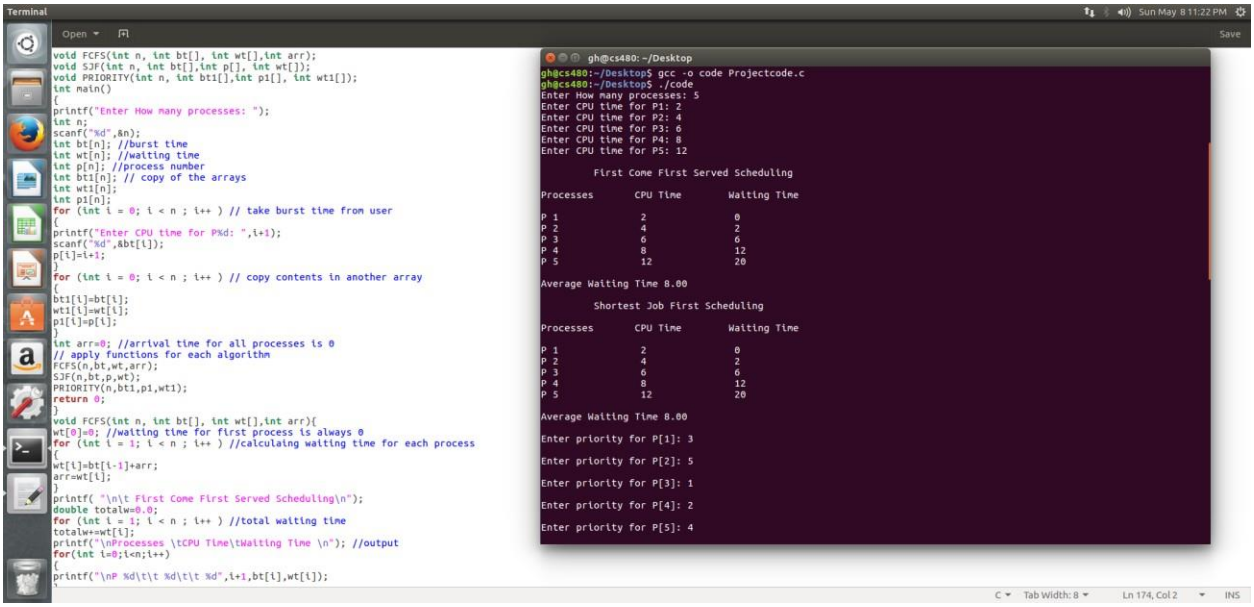
```
//sorting burst time, priority and process number in ascending order using selection sort
for(i=0;i<n;i++)
{
min=i;
for(j=i+1;j<n;j++)
{
if(pr1[j]<pr1[min])
min=j;
}
temp=pr1[i];
pr1[i]=pr1[min];
pr1[min]=temp;
temp=bt1[i];
bt1[i]=bt1[min];
bt1[min]=temp;
temp=p1[i];
p1[i]=p1[min];
p1[min]=temp;
}
wt1[0]=0; //waiting time for first process is zero
int arr=0;
for (int i = 1; i < n ; i++ ) //calculate waiting time
{ wt1[i]=bt1[i-1]+arr;
arr=wt1[i];}
```

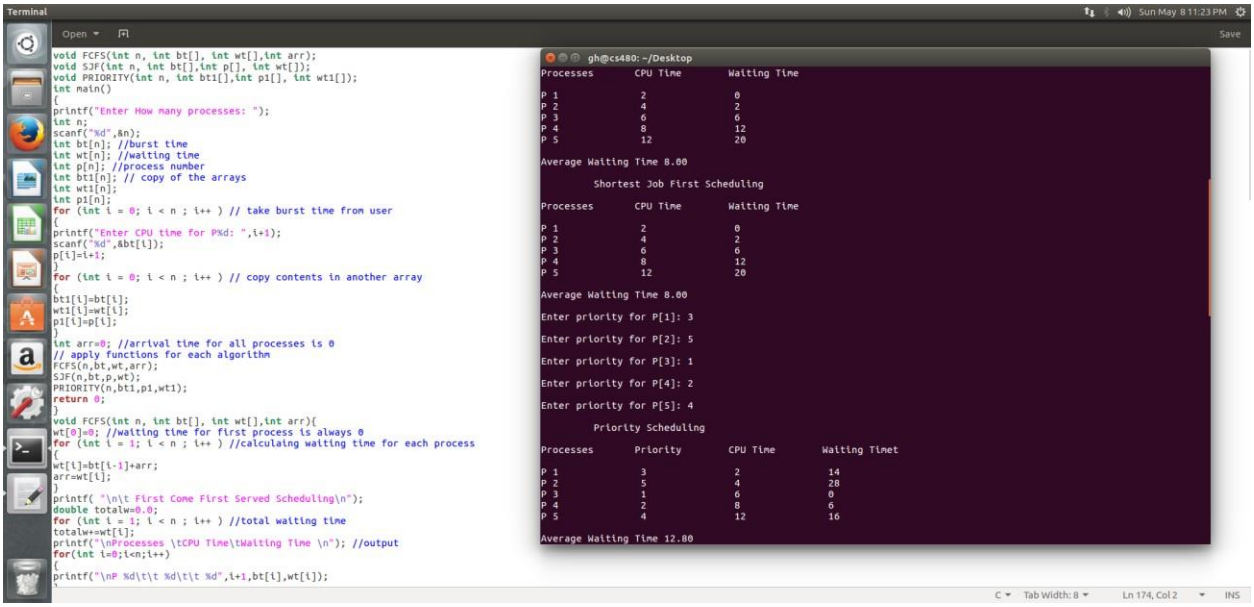
```
for (int i = 1; i < n ; i++ )
total=wt1[i]+total;
for(i=0;i<n;i++)
{
min=i;
for(j=i+1;j<n;j++)
{
if(p1[j]<p1[min])
min=j;
}
temp=pr1[i];
pr1[i]=pr1[min];
pr1[min]=temp;
temp=bt1[i];
bt1[i]=bt1[min];
bt1[min]=temp;
temp=p1[i];
p1[i]=p1[min];
p1[min]=temp;
temp=wt1[i];
wt1[i]=wt1[min];
wt1[min]=temp;
}
printf( "\n\t Priority Scheduling \n");
```

```
printf("\nProcesses \tPriority\tCPU Time\tWaiting Time \n"); //output
for(i=0;i<n;i++)
{
printf("\nP %d\t\t %d\t\t %d\t\t %d",p1[i],pr1[i],bt1[i],wt1[i]);
}
avg=total/n;
printf("\n\nAverage Waiting Time %.2f \n",avg);
}
```

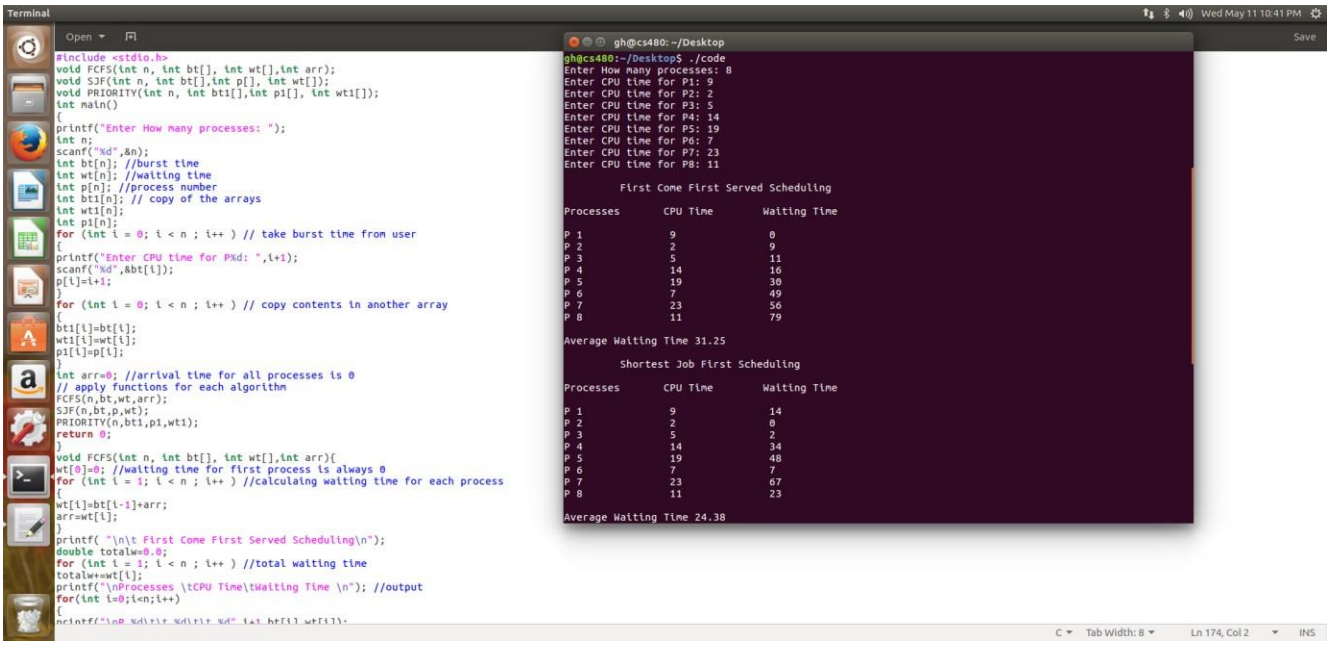
Sample Output:

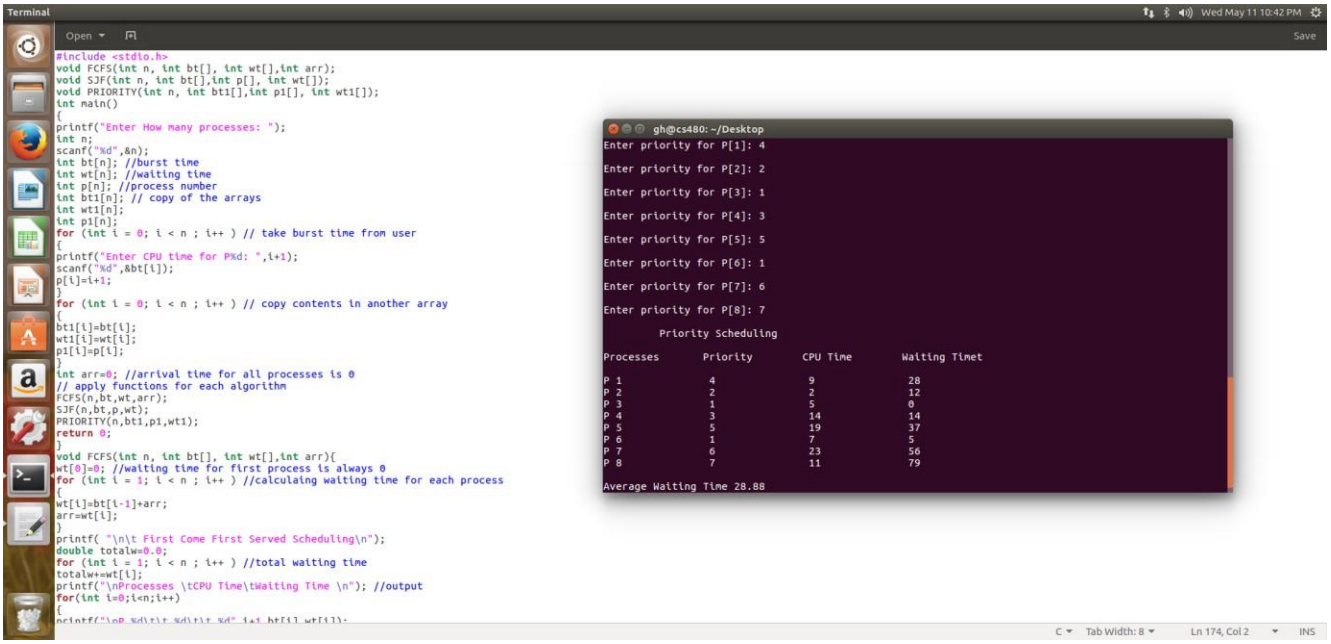
1st:



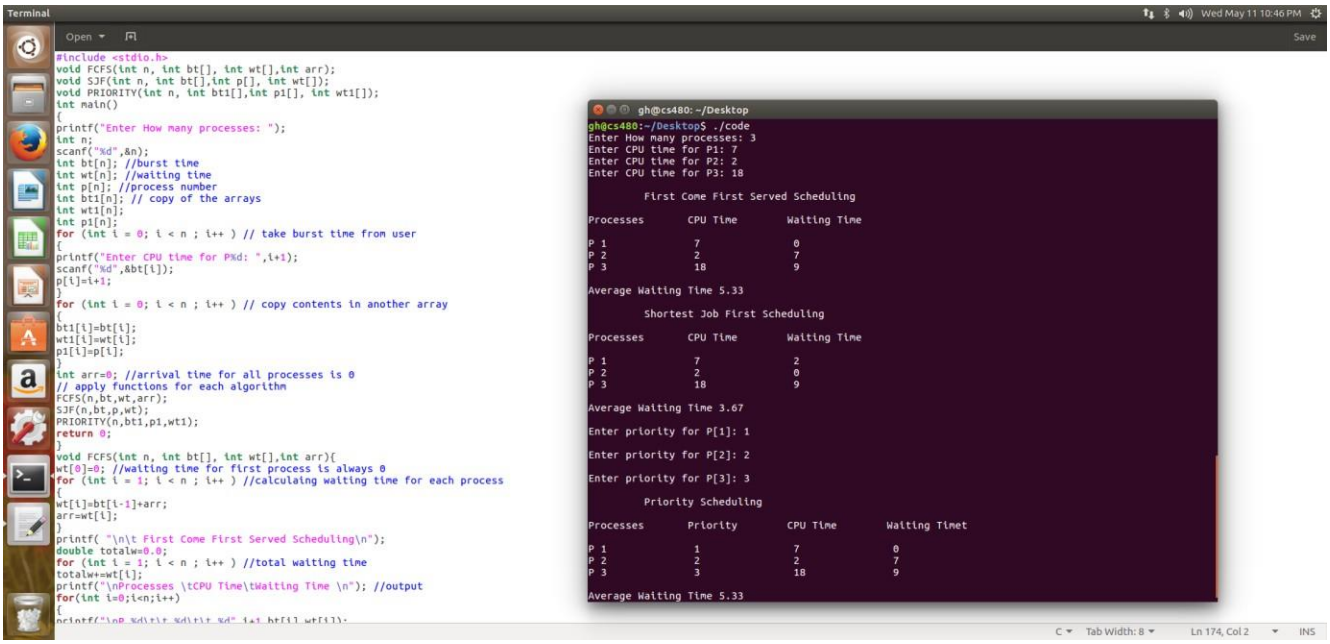


2nd.


Terminal
Wed May 11 10:41 PM
Open
Save
#include <stdio.h>
void FCFS(int n, int bt[], int wt[],int arr);
void SJF(int n, int bt[],int p[], int wt[]);
void PRIORITY(int n, int bt1[],int p1[], int wt1[]);
int main()
{
printf("Enter How many processes: ");
int n;
scanf("%d",&n);
int bt[n]; //burst time
int wt[n]; //waiting time
int p[n]; //process number
int bt1[n]; // copy of the arrays
int wt1[n];
int p1[n];
for (int i = 0; i < n ; i++) // take burst time from user
{
printf("Enter CPU time for P%d: ",i+1);
scanf("%d",&bt[i]);
p[i]=i+1;
}
for (int i = 0; i < n ; i++) // copy contents in another array
{
bt1[i]=bt[i];
wt1[i]=wt[i];
p1[i]=p[i];
}
int arr=0; //arrival time for all processes is 0
// apply functions for each algorithm
FCFS(n,bt,wt,arr);
SJF(n,bt,p,wt);
PRIORITY(n,bt1,p1,wt1);
return 0;
}
void FCFS(int n, int bt[], int wt[],int arr){
wt[0]=0; //waiting time for first process is always 0
for (int i = 1; i < n ; i++) //calculaing waiting time for each process
{
wt[i]=bt[i-1]+arr;
arr=wt[i];
}
printf("\n\t First Come First Served Scheduling\n");
double totalw=0.0;
for (int i = 1; i < n ; i++) //total waiting time
totalw+=wt[i];
printf("\nProcesses \tCPU Time\tWaiting Time \n"); //output
for(int i=0;i<n;i++)
{
gh@cs480: ~/Desktop
gh@cs480:~/Desktop$ ./code
Enter How many processes: 8
Enter CPU time for P1: 9
Enter CPU time for P2: 2
Enter CPU time for P3: 5
Enter CPU time for P4: 14
Enter CPU time for P5: 19
Enter CPU time for P6: 7
Enter CPU time for P7: 23
Enter CPU time for P8: 11
First Come First Served Scheduling
Processes CPU Time Waiting Time
P 1 9 0
P 2 2 9
P 3 5 11
P 4 14 16
P 5 19 30
P 6 7 49
P 7 23 56
P 8 11 79
Average Waiting Time 31.25
Shortest Job First Scheduling
Processes CPU Time Waiting Time
P 1 9 14
P 2 2 0
P 3 5 2
P 4 14 34
P 5 19 48
P 6 7 7
P 7 23 67
P 8 11 23
Average Waiting Time 24.38
C Tab Width: 8 Ln 174, Col 2 INS


Terminal
Wed May 11 10:42 PM
Open
Save
#include <stdio.h>
void FCFS(int n, int bt[], int wt[],int arr);
void SJF(int n, int bt[],int p[], int wt[]);
void PRIORITY(int n, int bt1[],int p1[], int wt1[]);
int main()
{
printf("Enter How many processes: ");
int n;
scanf("%d",&n);
int bt[n]; //burst time
int wt[n]; //waiting time
int p[n]; //process number
int bt1[n]; // copy of the arrays
int wt1[n];
int p1[n];
for (int i = 0; i < n ; i++) // take burst time from user
{
printf("Enter CPU time for P%d: ",i+1);
scanf("%d",&bt[i]);
p[i]=i+1;
}
for (int i = 0; i < n ; i++) // copy contents in another array
{
bt1[i]=bt[i];
wt1[i]=wt[i];
p1[i]=p[i];
}
int arr=0; //arrival time for all processes is 0
// apply functions for each algorithm
FCFS(n,bt,wt,arr);
SJF(n,bt,p,wt);
PRIORITY(n,bt1,p1,wt1);
return 0;
}
void FCFS(int n, int bt[], int wt[],int arr){
wt[0]=0; //waiting time for first process is always 0
for (int i = 1; i < n ; i++) //calculaing waiting time for each process
{
wt[i]=bt[i-1]+arr;
arr=wt[i];
}
printf("\n\t First Come First Served Scheduling\n");
double totalw=0.0;
for (int i = 1; i < n ; i++) //total waiting time
totalw+=wt[i];
printf("\nProcesses \tCPU Time\tWaiting Time \n"); //output
for(int i=0;i<n;i++)
{
gh@cs480: ~/Desktop
Enter priority for P[1]: 4
Enter priority for P[2]: 2
Enter priority for P[3]: 1
Enter priority for P[4]: 3
Enter priority for P[5]: 5
Enter priority for P[6]: 1
Enter priority for P[7]: 6
Enter priority for P[8]: 7
Priority Scheduling
Processes Priority CPU Time Waiting Timet
P 1 4 9 28
P 2 2 2 12
P 3 1 5 0
P 4 3 14 14
P 5 5 19 37
P 6 1 7 5
P 7 6 23 56
P 8 7 11 79
Average Waiting Time 28.88
C Tab Width: 8 Ln 174, Col 2 INS

3rd:



Conclusion:

Shortest Job First is the most efficient algorithm. It minimizes the average wait time because it gives service to less burst time processes before it gives service to processes with larger burst time.