**CPU SCHEDULING**

1. **FCFS :**

**CODE :**

#include <stdio.h>

#include <stdlib.h>

struct p{

int arrival, id, burst, waiting, tat;

};

int cmpfnc1(const void\* a, const void\* b){

struct p\* A = (struct p\*)a;

struct p\* B = (struct p\*)b;

if (A-> arrival > B->arrival) return 1;

if (B-> arrival > A->arrival) return -1;

if (A->arrival == B->arrival) return A->id - B->id;

}

int cmpfnc2(const void\* a, const void\* b){

struct p\* A = (struct p\*)a;

struct p\* B = (struct p\*)b;

return A->id - B->id;

}

void calc(struct p arr[],int n){

qsort(arr, n, sizeof(struct p), cmpfnc1);

int time = 0;

float avg\_tat = 0.0, avg\_wait = 0.0;

for(int i=0; i < n; i++){

time+= arr[i].burst;

arr[i].tat = time - arr[i].arrival;

arr[i].waiting = arr[i].tat - arr[i].burst;

avg\_tat+= arr[i].tat, avg\_wait+= arr[i].waiting;

}

printProcess(arr, n);

printf("Average turnaround time = %.2f\n",avg\_tat/n );

printf("Average waiting time = %.2f\n",avg\_wait/n );

}

void printProcess(struct p arr[],int n){

qsort(arr, n, sizeof(struct p), cmpfnc2);

printf("Process : \n");

printf("Process\_ID Arrival Burst TAT Waiting\n");

for(int i=0; i < n; i++) printf("\t%d\t%d %d %d %d\n",arr[i].id, arr[i].arrival, arr[i].burst, arr[i].tat, arr[i].waiting);

}

int main(){

int n;

printf("Enter the number of processes : ");

scanf("%d",&n);

struct p arr[n];

for(int i=0; i < n; i++){

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

scanf("%d",&arr[i].arrival);

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

scanf("%d",&arr[i].burst);

arr[i].id = i + 1;

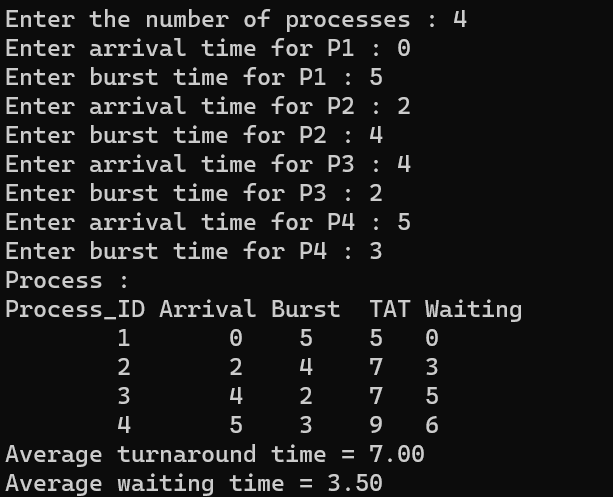
}

calc(arr, n);

return 0;

}

OUTPUT:



1. **SJF**

**CODE:**

// we will be implementing non-preemptive SJF technique

#include <stdio.h>

#include <stdlib.h>

struct p{

int arrival, id, burst, waiting, tat;

};

void swap(struct p\* x, struct p\* y){

struct p temp = \*x;

\*x = \*y;

\*y = temp;

}

int cmpfnc1(const void\* a, const void\* b){

struct p\* A = (struct p\*) a;

struct p\* B = (struct p\*) b;

if(A->arrival < B->arrival) return -1;

if(A->arrival > B->arrival) return 1;

if(A->arrival == B->arrival){

if(A->burst > B->burst) return 1;

if(A->burst < B->burst) return -1;

else if(A->burst == B->burst) return A->id > B->id ? 1 : -1;

}

}

int cmpfnc2(const void\* a, const void\* b){

struct p\* A = (struct p\*)a;

struct p\* B = (struct p\*)b;

return A->id - B->id;

}

void gantt(struct p arr[], int n){

qsort(arr, n, sizeof(struct p), cmpfnc1);

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

for(int j = i; j < n-1; j++) if(arr[j].burst > arr[j+1].burst && arr[j+1].arrival <= arr[i-1].burst) swap(&arr[j], &arr[j+1]);

}

void printProcess(struct p arr[],int n){

qsort(arr, n, sizeof(struct p), cmpfnc2);

printf("Process : \n");

printf("Process\_ID Arrival Burst TAT Waiting\n");

for(int i=0; i < n; i++) printf("\t%d\t%d %d %d %d\n",arr[i].id, arr[i].arrival, arr[i].burst, arr[i].tat, arr[i].waiting);

}

void calc(struct p arr[], int n) {

int time = 0;

float avg\_tat = 0.0, avg\_wait = 0.0;

for(int i=0; i < n; i++){

time+= arr[i].burst;

arr[i].tat = time - arr[i].arrival;

arr[i].waiting = arr[i].tat - arr[i].burst;

avg\_tat+= arr[i].tat, avg\_wait+= arr[i].waiting;

}

printProcess(arr, n);

printf("Average turnaround time = %.2f\n",avg\_tat/n );

printf("Average waiting time = %.2f\n",avg\_wait/n );

}

int main(){

int n;

printf("Enter the number of processes : ");

scanf("%d",&n);

struct p arr[n];

for(int i=0; i < n; i++){

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

scanf("%d",&arr[i].arrival);

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

scanf("%d",&arr[i].burst);

arr[i].id = i + 1;

}

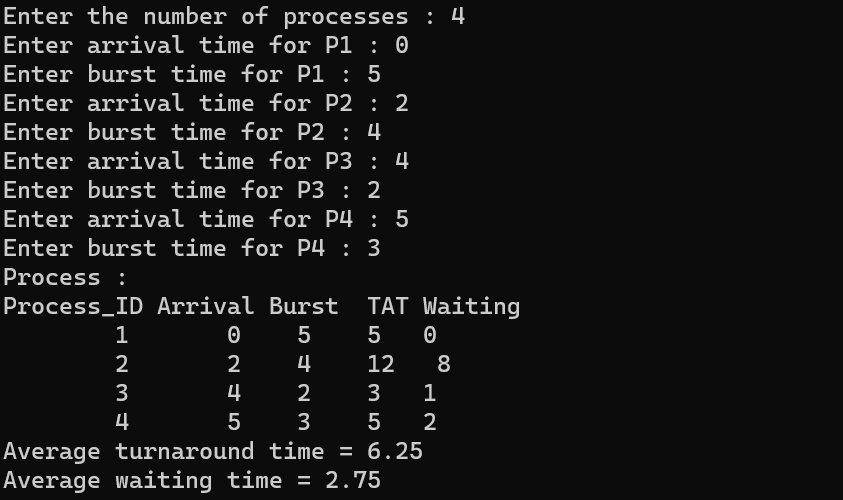
gantt(arr, n);

calc(arr, n);

return 0;

}

OUTPUT:



1. **ROUND ROBIN :**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

struct p{

int arrival, id, burst, waiting, tat, na, nb;

};

int cmpfnc1(const void\* a, const void\* b){

struct p\* A = (struct p\*)a;

struct p\* B = (struct p\*)b;

if (A-> arrival > B->arrival) return 1;

if (A-> arrival < B->arrival) return -1;

if (A->arrival == B->arrival) return A->id > B->id ? 1 : -1;

}

int cmpfnc2(const void\* a, const void\* b){

struct p\* A = (struct p\*)a;

struct p\* B = (struct p\*)b;

return A->id - B->id;

}

int cmpfnc3(const void\* a, const void\* b){

struct p\* A = (struct p\*)a;

struct p\* B = (struct p\*)b;

return A->na > B->na ? 1 : -1;

}

int total\_time(struct p arr[], int n){

int init\_time = 0;

for(int i=0; i < n; i++) init\_time+= arr[i].burst;

return init\_time;

}

int return\_blocks(struct p arr[], int n, int tq){

int blocks = 0;

for(int i=0; i < n; i++) {

if(arr[i].burst > tq )blocks+= (int)ceil((double)arr[i].burst / tq);

else if (arr[i].burst <= tq) blocks+= 1;

} return blocks;

}

void printProcess(struct p arr[],int n){

qsort(arr, n, sizeof(struct p), cmpfnc2);

printf("Process : \n");

printf("Process\_ID Arrival Burst TAT Waiting\n");

for(int i=0; i < n; i++) printf("\t%d\t%d %d %d %d\n",arr[i].id, arr[i].arrival, arr[i].burst, arr[i].tat, arr[i].waiting);

}

void gantt(struct p arr[], int n, int tq){

qsort(arr,n, sizeof(struct p), cmpfnc1);

int total\_burst = total\_time(arr, n);

int init\_time = 0,j=0, block\_size = return\_blocks(arr,n, tq);

struct p\* rq = (struct p\*)malloc(sizeof(struct p) \*block\_size);

for(int i=0; i < n , init\_time < total\_burst && j < block\_size; i++ ){

label : printf("");

if (i > 0) qsort(arr, i,sizeof(struct p), cmpfnc3);

for(int k = 0; k < i; k++){

if(arr[k].na < arr[i].na && arr[k].nb > 0){

if(arr[k].nb <= tq) init\_time += arr[k].nb, arr[k].nb = 0, arr[k].na = arr[k].tat = init\_time;

else if (arr[k].nb > tq) init\_time+= tq, arr[k].nb -= tq, arr[k].na = arr[k].tat = init\_time ;

rq[j++] = arr[k];

}

}

if (arr[i].nb > 0){

if(arr[i].nb <= tq) init\_time += arr[i].nb, arr[i].nb = 0, arr[i].na = arr[i].tat = init\_time;

else if (arr[i].nb > tq) init\_time+= tq, arr[i].nb -= tq, arr[i].na = arr[i].tat = init\_time ;

rq[j++] = arr[i];}

if (i == n-1 && init\_time < total\_burst) goto label;

}

qsort(arr, n, sizeof(struct p),cmpfnc2);

}

void calc(struct p arr[],int n){

float avg\_tat = 0.0, avg\_wait = 0.0;

for(int i=0; i < n; i++){

arr[i].tat-= arr[i].arrival;

arr[i].waiting = arr[i].tat - arr[i].burst;

avg\_tat+= arr[i].tat, avg\_wait+= arr[i].waiting;

}

printProcess(arr, n);

printf("Average turnaround time = %.2f\n",avg\_tat/n );

printf("Average waiting time = %.2f\n",avg\_wait/n );

}

int main(){

int n, tq;

printf("Enter the number of processes : ");

scanf("%d",&n);

printf("Enter time quantum : ");

scanf("%d",&tq);

struct p arr[n];

for(int i=0; i < n; i++){

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

scanf("%d",&arr[i].arrival);

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

scanf("%d",&arr[i].burst);

arr[i].id = i + 1, arr[i].na = arr[i].arrival, arr[i].nb = arr[i].burst,arr[i].tat = 0;

}

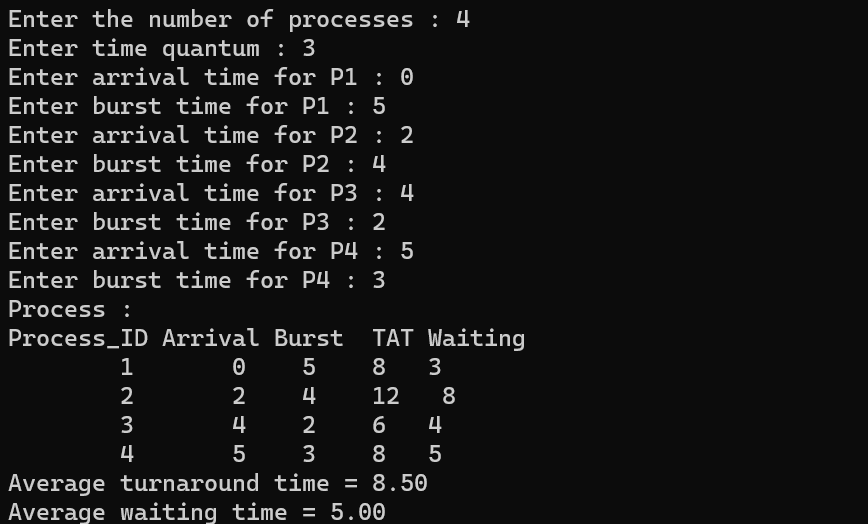
gantt(arr, n, tq);

calc(arr, n);

return 0;

}

OUTPUT:



1. PRIORITY SCHEDULING :

CODE:

// we will be implementing non-preemptive priority scheduling

#include <stdio.h>

#include <stdlib.h>

struct p{

int arrival, id, burst, waiting, tat, priority;

};

void swap(struct p\* x, struct p\* y){

struct p temp = \*x;

\*x = \*y;

\*y = temp;

}

int cmpfnc1(const void\* a, const void\* b){

struct p\* A = (struct p\*)a;

struct p\* B = (struct p\*)b;

if (A->arrival > B->arrival) return 1;

if (A->arrival < B->arrival) return -1;

if (A->arrival == B->arrival){

if (A->priority > B->priority) return 1;

if (A->priority < B->priority) return -1;

if (A->priority == B->priority) return A->id > B->id ? 1 : -1;

}

}

int cmpfnc2(const void\* a, const void\* b){

struct p\* A = (struct p\*)a;

struct p\* B = (struct p\*)b;

return A->id - B->id;

}

void gantt(struct p arr[], int n){

qsort(arr, n, sizeof(struct p), cmpfnc1);

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

for(int j = i; j < n-1; j++) if(arr[j].priority > arr[j+1].priority && arr[j+1].arrival <= arr[i-1].burst) swap(&arr[j], &arr[j+1]);

}

void printProcess(struct p arr[],int n){

qsort(arr, n, sizeof(struct p), cmpfnc2);

printf("Process : \n");

printf("Process\_ID Arrival Burst Priority TAT Waiting\n");

for(int i=0; i < n; i++) printf("\t%d\t%d %d %d %d %d\n",arr[i].id, arr[i].arrival, arr[i].burst,arr[i].priority, arr[i].tat, arr[i].waiting);

}

void calc(struct p arr[], int n) {

int time = 0;

float avg\_tat = 0.0, avg\_wait = 0.0;

for(int i=0; i < n; i++){

time+= arr[i].burst;

arr[i].tat = time - arr[i].arrival;

arr[i].waiting = arr[i].tat - arr[i].burst;

avg\_tat+= arr[i].tat, avg\_wait+= arr[i].waiting;

}

printProcess(arr, n);

printf("Average turnaround time = %.2f\n",avg\_tat/n );

printf("Average waiting time = %.2f\n",avg\_wait/n );

}

int main(){

int n;

printf("Enter the number of processes : ");

scanf("%d",&n);

struct p arr[n];

for(int i=0; i < n; i++){

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

scanf("%d",&arr[i].arrival);

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

scanf("%d",&arr[i].burst);

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

scanf("%d",&arr[i].priority);

arr[i].id = i + 1;

}

gantt(arr, n);

calc(arr, n);

return 0;

}

OUTPUT:

