**OS - Practical - 6**

**Name: Rishabh Jain**

**Branch & Sem: CSE - IV**

**Sec: A**

**Roll no.:54**

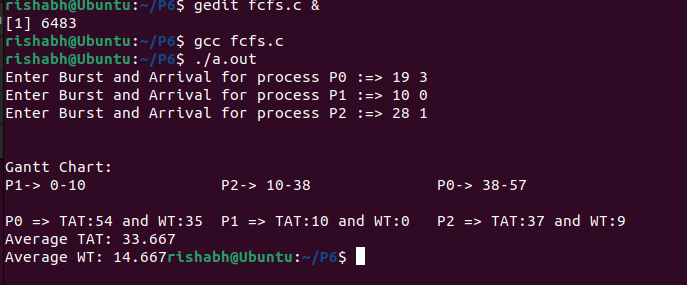
**Aim:** Given the list of processes, their CPU burst time, arrival time, priority and time quantum. Display/print the Gantt chart; compute the average

waiting time and average turnaround time for given scheduling policy.

**FCFS:**

#include<stdio.h>  
#include<stdlib.h>  
  
typedef struct Process{  
    int burst;  
    int arrival;  
    int tat;  
    int wt;  
}P;  
  
void initialize(P\* p,int b,int a){  
    p->burst = b;  
    p->arrival = a;  
    p->tat = 0;  
    p->wt = 0;  
}  
  
int main(){  
    P parr[3];  
    int done[3];  
  
    for(int i=0;i<sizeof(parr)/sizeof(parr[0]);i++){  
        done[i] = 0;  
    }  
  
    //setting Up Processes data :  
    for(int i=0;i<sizeof(parr)/sizeof(parr[0]);i++){  
        printf("Enter Burst and Arrival for process P%d :=> ",i);  
        int b,a;  
        scanf("%d %d",&b,&a);  
        initialize(&parr[i],b,a);  
    }  
  
   int total=0;  
   int n = sizeof(parr)/sizeof(parr[0]);  
   P pdash = parr[0];  
   //finding first arrived process  
  
   for(int i=1;i<n;i++){  
        if(pdash.arrival>parr[i].arrival){  
            pdash = parr[i];  
        }  
   }  
  
   //processing on first arrived process:  
   if(pdash.arrival!=0){  
    total+=pdash.arrival;  
   }  
     
   //finding smallest  
   printf("\n\nGantt Chart:\n");  
   int pcount=0;  
   while(pcount!=n){  
    initialize(&pdash,0,999);  
    int idx = -1;  
        for(int i=0;i<n;i++){  
            if(pdash.arrival>parr[i].arrival && done[i]!=1){  
                pdash = parr[i];  
                idx = i;  
            }  
        }  
        //got arrived process  
        if(idx!=-1){  
            int old = total;  
            total+=pdash.burst;  
  
            parr[idx].tat = total-parr[idx].arrival;  
            parr[idx].wt = parr[idx].tat-parr[idx].burst;  
             
            done[idx]=1;  
            printf("P%d-> %d-%d\t\t",idx,old,total);  
        }  
        pcount++;  
  
   }  
  
    printf("\n\n");  
    int totTAT=0,totWT=0;  
    for(int i=0;i<n;i++){  
        printf("P%d => TAT:%d and WT:%d \t",i,parr[i].tat,parr[i].wt);  
        totTAT+=parr[i].tat; totWT+=parr[i].wt;  
    }  
  
    printf("\nAverage TAT: %.3f\n",(float)totTAT/n);  
    printf("Average WT: %.3f",(float)totWT/n);  
  
}

OUTPUT :



**SJF(Non-Preemptive):**

#include<stdio.h>

#include<stdlib.h>

typedef struct Process{

int burst;

int arrival;

int tat;

int wt;

}P;

void initialize(P\* p,int b,int a){

p->burst = b;

p->arrival = a;

p->tat = 0;

p->wt = 0;

}

int main(){

P parr[5];

int done[5];

for(int i=0;i<sizeof(parr)/sizeof(parr[0]);i++){

done[i] = 0;

}

//setting Up Processes data :

for(int i=0;i<sizeof(parr)/sizeof(parr[0]);i++){

printf("Enter Burst and Arrival for process P%d :=> ",i);

int b,a;

scanf("%d %d",&b,&a);

initialize(&parr[i],b,a);

}

int total=0;

int n = sizeof(parr)/sizeof(parr[0]);

P arr[n];

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

arr[i] = parr[i];

}

for(int i=0;i<n-1;i++){

for(int j=0;j<n-i-1;j++){

if(arr[j].arrival>arr[j+1].arrival){

P temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

printf("\n");

int x;

total = arr[0].arrival+arr[0].burst;

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

if(parr[i].arrival == arr[0].arrival && parr[i].burst == arr[0].burst){

parr[i].tat = total-parr[i].arrival;

parr[i].wt = parr[i].tat-parr[i].burst;

x = i;

}

}

//Actual Logic:

printf("Gantt Chart: \n");

printf("P%d: 0-%d\t",x,total);

int pcount=0;

while(pcount!=n-1){

int hold=999;

int idx;

int prevProcess;

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

if(done[i]!=1 && arr[i].arrival<=total){

if(hold>arr[i].burst){

hold = arr[i].burst;

idx = i; prevProcess = i;

}

}

}

//Found Process

if(hold!=999){

int dash = total;

total+=hold;

done[idx] = 1;

pcount++;

int y;

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

if(parr[i].arrival == arr[idx].arrival && parr[i].burst == arr[idx].burst){

parr[i].tat = total-parr[i].arrival;

parr[i].wt = parr[i].tat-parr[i].burst;

y = i;

}

}

printf("P%d: %d-%d\t",y,dash,total);

}else if(hold==999){

//For Excess Time gap in Between

int least=9999;

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

if(i!=prevProcess && done[i]!=1 && arr[i].arrival<least){

least = arr[i].arrival;

}

}

printf("%d\n",least);

least = least-total;

total += least;

}

}

printf("\n\n%d\n",total);

float avgTAT=0,avgWT=0;

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

printf("for Process (%d) => TAT: %d\t, WT: %d\n",i,parr[i].tat,parr[i].wt);

avgTAT+=parr[i].tat;

avgWT+=parr[i].wt;

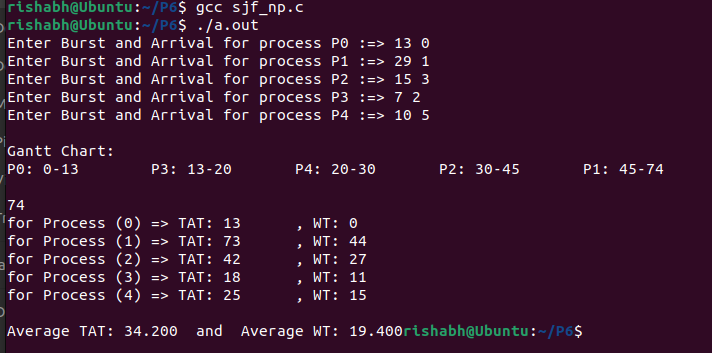
}

printf("\nAverage TAT: %.3f and Average WT: %.3f",avgTAT/n,avgWT/n);

return 0;

}

OUTPUT:



**Priority(Non-Preemptive):**

#include<stdio.h>

#include<stdlib.h>

typedef struct Process{

int burst;

int arrival;

int priority;

int tat;

int wt;

}P;

void initialize(P\* p,int b,int a,int pb){

p->burst = b;

p->arrival = a;

p->priority = pb;

p->tat = 0;

p->wt = 0;

}

int main(){

P parr[5];

int done[5];

for(int i=0;i<sizeof(parr)/sizeof(parr[0]);i++){

done[i] = 0;

}

//setting Up Processes data :

for(int i=0;i<sizeof(parr)/sizeof(parr[0]);i++){

printf("Enter Burst , Arrival and Priority for process P%d :=> ",i);

int b,a,pb;

scanf("%d %d %d",&b,&a,&pb);

initialize(&parr[i],b,a,pb);

}

int total=0;

int n = sizeof(parr)/sizeof(parr[0]);

P arr[n];

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

arr[i] = parr[i];

}

for(int i=0;i<n-1;i++){

for(int j=0;j<n-i-1;j++){

if(arr[j].arrival>arr[j+1].arrival){

P temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

printf("\n");

int x;

total = arr[0].arrival+arr[0].burst;

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

if(parr[i].arrival == arr[0].arrival && parr[i].priority == arr[0].priority){

parr[0].tat = total-parr[0].arrival;

parr[0].wt = parr[0].tat-parr[0].burst;

x =i;

}

}

//Actual Logic:

printf("Gantt Chart: \n");

printf("P%d: 0-%d\t\t",x,total);

int pcount=0;

while(pcount!=n-1){

int hold=999;

int idx;

int prevProcess;

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

if(done[i]!=1 && arr[i].arrival<=total){

if(hold>arr[i].priority){

hold = arr[i].priority;

idx = i; prevProcess = i;

}

}

}

//Found Process

if(hold!=999){

int dash = total;

total+=arr[idx].burst;

done[idx] = 1;

pcount++;

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

if(parr[i].arrival == arr[idx].arrival && parr[i].priority == arr[idx].priority){

parr[i].tat = total-parr[i].arrival;

parr[i].wt = parr[i].tat-parr[i].burst;

}

}

printf("P%d: %d-%d\t",idx,dash,total);

}else if(hold==999){

//For Excess Time gap in Between

int least=9999;

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

if(i!=prevProcess && done[i]!=1 && arr[i].arrival<least){

least = arr[i].arrival;

}

}

printf("%d\n",least);

least = least-total;

total += least;

}

}

printf("\n\n%d\n",total);

float avgTAT=0,avgWT=0;

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

printf("for Process (%d) => TAT: %d\t, WT: %d\n",i,parr[i].tat,parr[i].wt);

avgTAT+=parr[i].tat;

avgWT+=parr[i].wt;

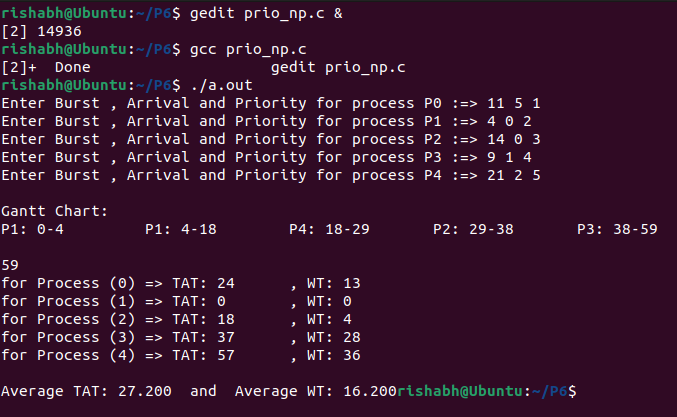
}

printf("\nAverage TAT: %.3f and Average WT: %.3f",avgTAT/n,avgWT/n);

return 0;

}

OUTPUT:



**Priority(Preemptive):**

#include<stdio.h>

#include<stdlib.h>

typedef struct Process{

int burst;

int arrival;

int priority;

int holdBurst;

int tat;

int wt;

}P;

void initialize(P\* p,int b,int a,int pd){

p->burst = b;

p->arrival = a;

p->holdBurst = b;

p->priority = pd;

p->tat = 0;

p->wt = 0;

}

int main(){

P parr[4];

int done[4];

for(int i=0;i<sizeof(parr)/sizeof(parr[0]);i++){

done[i] = 0;

}

//setting Up Processes data :

for(int i=0;i<sizeof(parr)/sizeof(parr[0]);i++){

printf("Enter Burst , Arrival and Priority for process P%d :=> ",i);

int b,a,pd;

scanf("%d %d %d",&b,&a,&pd);

initialize(&parr[i],b,a,pd);

}

int total=0;

int n = sizeof(parr)/sizeof(parr[0]);

P arr[n];

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

arr[i] = parr[i];

}

for(int i=0;i<n-1;i++){

for(int j=0;j<n-i-1;j++){

if(arr[j].arrival>arr[j+1].arrival){

P temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

printf("\n");

// total = arr[0].arrival+arr[0].burst;

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

printf("%d\t",arr[i].burst);

// if(parr[i].arrival == arr[0].arrival && parr[i].burst == arr[0].burst){

// parr[0].tat = total-parr[0].arrival;

// parr[0].wt = parr[0].tat-parr[0].burst;

// }

}

//Actual Logic:

printf("Gantt Chart: \n");

// printf("P0: 0-%d\t",total);

int lastArrival = 0;

// int pcount=0;

P\* moving = &arr[0];

int movingIDX=0;

int toReach = arr[n-1].arrival;

if(arr[0].arrival!=0){

lastArrival+=arr[0].arrival;

total+=arr[0].arrival;

}

while(lastArrival != toReach){

if(lastArrival==0){

moving->burst--;

lastArrival++;

total++;

}else{

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

if(done[i]!=1)

if(lastArrival>=arr[i].arrival && moving->priority>arr[i].priority){

moving = &arr[i];

movingIDX = i;

}

}

if(moving->burst!=0){

moving->burst--;

int oldTotal = total;

lastArrival++; total++;

if(arr[movingIDX].burst==0){

done[movingIDX] = 1;

arr[movingIDX].tat = total-arr[movingIDX].arrival;

arr[movingIDX].wt = arr[movingIDX].tat-arr[movingIDX].holdBurst;

//set new moving

P p;

initialize(&p,999,0,999);

moving = &p;

movingIDX=-1;

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

if(done[i]!=1 && lastArrival>=arr[i].arrival){

if(moving->priority>arr[i].priority){

moving = &arr[i];

movingIDX = i;

}

}

}

}

}

}

}

//Remaining bursts:

int allDone = 1;

while(allDone){

//Setting up allDone:

int finding=0;

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

if(done[i]==1){

finding++;

}

}

if(finding==n){

allDone=0;

break;

}

P p;

initialize(&p,999,0,999);

moving = &p;

movingIDX=-1;

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

if(done[i]!=1 && lastArrival>=arr[i].arrival){

if(moving->priority>arr[i].priority){

moving = &arr[i];

movingIDX = i;

}

}

}

int oldTotal = total;

total+=moving->burst;

moving->burst=0;

done[movingIDX]=1;

arr[movingIDX].tat = total-arr[movingIDX].arrival;

arr[movingIDX].wt = arr[movingIDX].tat - arr[movingIDX].holdBurst;

}

printf("\n\n%d\n\n",total);

float avgTAT=0,avgWT=0;

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

avgTAT+=(float)arr[i].tat;

printf("TAT: %d\t",arr[i].tat);

avgWT+=(float)arr[i].wt;

printf("WT: %d\n",arr[i].wt);

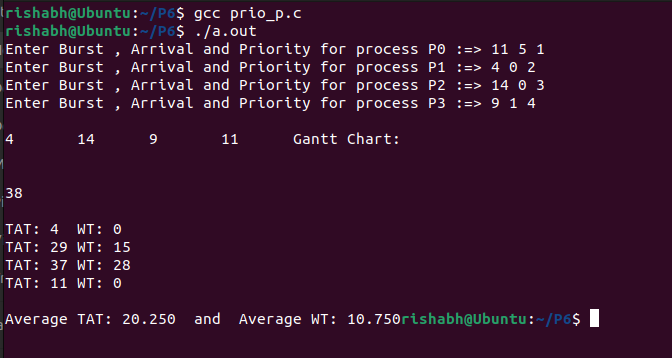
}

printf("\nAverage TAT: %.3f and Average WT: %.3f",avgTAT/n,avgWT/n);

return 0;

}

OUTPUT:



**SRTF:**

#include<stdio.h>

#include<stdlib.h>

typedef struct Process{

int burst;

int holdBurst;

int arrival;

int tat;

int wt;

}P;

void initialize(P\* p,int b,int a){

p->burst = b;

p->arrival = a;

p->holdBurst = b;

p->tat = 0;

p->wt = 0;

}

int main(){

P parr[5];

int done[5];

for(int i=0;i<sizeof(parr)/sizeof(parr[0]);i++){

done[i] = 0;

}

//setting Up Processes data :

for(int i=0;i<sizeof(parr)/sizeof(parr[0]);i++){

printf("Enter Burst and Arrival for process P%d :=> ",i);

int b,a;

scanf("%d %d",&b,&a);

initialize(&parr[i],b,a);

}

int total=0;

int n = sizeof(parr)/sizeof(parr[0]);

P arr[n];

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

arr[i] = parr[i];

}

for(int i=0;i<n-1;i++){

for(int j=0;j<n-i-1;j++){

if(arr[j].arrival>arr[j+1].arrival){

P temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

printf("\n");

//Actual Logic:

int lastArrival = 0;

// int pcount=0;

P\* moving = &arr[0];

int movingIDX=0;

int toReach = arr[n-1].arrival;

if(arr[0].arrival!=0){

lastArrival+=arr[0].arrival;

total+=arr[0].arrival;

}

while(lastArrival != toReach){

if(lastArrival==0){

moving->burst--;

lastArrival++;

total++;

}else{

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

if(done[i]!=1)

if(lastArrival>=arr[i].arrival && moving->burst>arr[i].burst){

moving = &arr[i];

movingIDX = i;

}

}

if(moving->burst!=0){

moving->burst--;

lastArrival++; total++;

if(arr[movingIDX].burst==0){

done[movingIDX] = 1;

arr[movingIDX].tat = total-arr[movingIDX].arrival;

arr[movingIDX].wt = arr[movingIDX].tat-arr[movingIDX].holdBurst;

//set new moving

P p;

initialize(&p,999,0);

moving = &p;

movingIDX=-1;

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

if(done[i]!=1 && lastArrival>=arr[i].arrival){

if(moving->burst>arr[i].burst){

moving = &arr[i];

movingIDX = i;

}

}

}

}

}

}

}

//Remaining bursts:

int allDone = 1;

while(allDone){

//Setting up allDone:

int finding=0;

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

if(done[i]==1){

finding++;

}

}

if(finding==n){

allDone=0;

break;

}

P p;

initialize(&p,999,0);

moving = &p;

movingIDX=-1;

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

if(done[i]!=1 && lastArrival>=arr[i].arrival){

if(moving->burst>arr[i].burst){

moving = &arr[i];

movingIDX = i;

}

}

}

total+=moving->burst;

moving->burst=0;

done[movingIDX]=1;

arr[movingIDX].tat = total-arr[movingIDX].arrival;

arr[movingIDX].wt = arr[movingIDX].tat-arr[movingIDX].holdBurst;

}

printf("\n\n%d\n\n",total);

float avgTAT=0,avgWT=0;

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

avgTAT+=(float)arr[i].tat;

printf("TAT: %d\t",arr[i].tat);

avgWT+=(float)arr[i].wt;

printf("WT: %d\n",arr[i].wt);

}

printf("\nAverage TAT: %.3f and Average WT: %.3f",avgTAT/n,avgWT/n);

return 0;

}

OUTPUT:



**Round Robin:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Process {

int arrival;

int burst, holdBurst;

int tat, wt;

} P;

typedef struct Node {

P p;

struct Node\* next;

} Node;

void initializeProcess(P\* p, int b, int a) {

p->arrival = a;

p->burst = b;

p->holdBurst = b;

}

void initializeNode(Node\* newNode, P\* p) {

newNode->p = \*p;

newNode->next = NULL;

}

void addNode(Node\*\* head, P\* p) {

Node\* new = (Node\*)malloc(sizeof(Node));

if (new == NULL) {

printf("Memory allocation failed.\n");

return;

}

initializeNode(new, p);

if (\*head == NULL) {

\*head = new;

}

else {

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = new;

}

}

Node\* removeNode(Node\*\* head) {

if (\*head == NULL) {

printf("Queue is empty.\n");

return NULL;

}

Node\* temp = \*head;

\*head = temp->next;

temp->next = NULL;

return temp;

}

void display(Node\*\* head) {

Node\* temp = \*head;

if (temp == NULL) {

return;

}

while (temp != NULL) {

printf("%d %d\n", temp->p.burst, temp->p.arrival);

temp = temp->next;

}

}

int main() {

Node\* head = NULL;

P parr[5];

int done[5];

int quantum = 0;

int n = sizeof(parr) / sizeof(parr[0]);

printf("Enter Time Quantum for Round Robin: ");

scanf("%d", &quantum);

printf("\n");

for (int i = 0; i < sizeof(parr) / sizeof(parr[0]); i++) {

done[i] = 0;

}

// Setting up Processes data:

for (int i = 0; i < sizeof(parr) / sizeof(parr[0]); i++) {

printf("Enter Burst and Arrival for process P%d :=> ", i);

int b, a;

scanf("%d %d", &b, &a);

initializeProcess(&parr[i], b, a);

}

P arr[n];

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

arr[i] = parr[i];

}

//finding First Arrived and adding it to Ready Queue:

for(int i=0;i<n-1;i++){

for(int j=0;j<n-i-1;j++){

if(arr[j].arrival>arr[j+1].arrival){

P temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

int total = 0,pcount=0;

total+=arr[0].arrival;

addNode(&head,&arr[0]);

printf("\n\n");

while(pcount!=n){

Node\* moving = removeNode(&head);

int Idx;

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

if(moving->p.arrival==arr[i].arrival && moving->p.holdBurst==arr[i].holdBurst){

Idx = i;

}

}

if(quantum>moving->p.burst){

int old = total;

total+=moving->p.burst;

moving->p.burst=0;

done[Idx] = 1;

//Finding In actual array

int duplicateIdx;

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

if(parr[i].arrival==arr[Idx].arrival && parr[i].holdBurst==arr[Idx].holdBurst){

duplicateIdx = i;

}

}

printf("P%d = %d-%d\t\t",duplicateIdx,old,total);

arr[Idx].tat = total-arr[Idx].arrival;

arr[Idx].wt = arr[Idx].tat - arr[Idx].holdBurst;

pcount++;

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

if(arr[i].arrival<=total && arr[i].arrival>=old && done[i]!=1 && i!=Idx){

addNode(&head,&arr[i]);

}

}

}else{

int old = total;

total+=quantum;

moving->p.burst-=quantum;

int duplicateIdx;

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

if(parr[i].arrival==arr[Idx].arrival && parr[i].holdBurst==arr[Idx].holdBurst){

duplicateIdx = i;

}

}

printf("P%d = %d-%d\t\t",duplicateIdx,old,total);

// if(){

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

if(arr[i].arrival<=total && arr[i].arrival>=old && done[i]!=1 && i!=Idx){

addNode(&head,&arr[i]);

}

}

// }

if(moving->p.burst!=0){

addNode(&head,&moving->p);

}else if(moving->p.burst==0){

pcount++;

done[Idx] = 1;

arr[Idx].tat = total-arr[Idx].arrival;

arr[Idx].wt = arr[Idx].tat - arr[Idx].holdBurst;

}

}

}

printf("\n\nANSWER: %d\n\n",total);

int avgTAT=0,avgWT=0;

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

printf("%d %d\t\t",arr[i].tat,arr[i].wt);

avgTAT+=arr[i].tat;

avgWT+=arr[i].wt;

}

printf("\n\nTAT average:%.3f\t WT average: %.3f",(float)avgTAT/n,(float)avgWT/n);

return 0;

}

OUTPUT:

