**ARYAMAN MISHRA**

**19BCE1027**

**Aim:**Write an algorithm and implement C program for multi-queue round robin technique for process scheduling. Display the execution sequence (Gantt Chart) and average waiting time (AWT).

**Algorithm:**

0-Start.

1-Import necessary packages stdio,limits and stdlib header files.

2-Define struct to contain all times for processes.

3-Define struct Queue with capacity.

4-Define functions to indicate if Queue is having underflow or overflow.

5-Define functions to push and pop elements in queue from front and rear.

6-Define function to sort elements in Queue using Bubble Sort.

7-Define function to sort arrival time of procces.

8-Input elements-waiting and execution time.

9-Enter Quantum Number as 3.

10-Check range of execution time and put in required queue(less than 10,more than 30 in range between 10 and 30) adnsort using sjf algorithm

11-Apply round robin algorithm.

12-Print gantt chart

13-Print Average Waiting and turnaround time.

14-Stop.

**Code:**

#include<stdio.h>

#include <limits.h>

#include <stdlib.h>

struct times

{

int p,art,but,wtt,tat,rnt;

};

struct Queue {

int front, rear, size;

unsigned capacity;

int\* array;

};

struct Queue\* createQueue(unsigned capacity)

{

struct Queue\* queue = (struct Queue\*)malloc(

sizeof(struct Queue));

queue->capacity = capacity;

queue->front = queue->size = 0;

queue->rear = capacity - 1;

queue->array = (int\*)malloc(

queue->capacity \* sizeof(int));

return queue;

}

int isFull(struct Queue\* queue)

{

return (queue->size == queue->capacity);

}

int isEmpty(struct Queue\* queue)

{

return (queue->size == 0);

}

void enqueue(struct Queue\* queue, int item)

{

if (isFull(queue))

return;

queue->rear = (queue->rear + 1)

% queue->capacity;

queue->array[queue->rear] = item;

queue->size = queue->size + 1;

}

int dequeue(struct Queue\* queue)

{

if (isEmpty(queue))

return INT\_MIN;

int item = queue->array[queue->front];

queue->front = (queue->front + 1)

% queue->capacity;

queue->size = queue->size - 1;

return item;

}

int front(struct Queue\* queue)

{

if (isEmpty(queue))

return INT\_MIN;

return queue->array[queue->front];

}

int rear(struct Queue\* queue)

{

if (isEmpty(queue))

return INT\_MIN;

return queue->array[queue->rear];

}

int sortQueue(struct Queue\* queue,int size){

int i,j;

for (i = 0; i < size; i++) {

for (j = i + 1; j < size; j++) {

if (queue->array[j]<queue->array[i]){

int temp = queue->array[i];

queue->array[i] = queue->array[j];

queue->array[j] = temp;

}

}

}

}

void sortart(struct times a[],int pro)

{

int i,j;

struct times temp;

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

{

for(j=i+1;j<pro;j++)

{

if(a[i].art > a[j].art)

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

return;

}

int main()

{

int i,j,pro,time,remain,flag=0,ts;

struct times a[100];

float avgwt=0,avgtt=0;

struct Queue\* lessThan10Queue = createQueue(100);

struct Queue\* lessThan30Queue = createQueue(100);

struct Queue\* greaterThan30Queue = createQueue(100);

printf("Enter Number Of Processes : ");

scanf("%d",&pro);

remain=pro;

for(i=0;i<pro;i++){

printf("Enter arrival time and Execution time for Process P%d : ",i);

scanf("%d%d",&a[i].art,&a[i].but);

a[i].p = i;

a[i].rnt = a[i].but;

if(a[i].rnt<10){

enqueue(lessThan10Queue,a[i].rnt);

}

else if(a[i].rnt>10 && a[i].rnt<=30){

enqueue(lessThan30Queue,a[i].rnt);

}

else{

enqueue(greaterThan30Queue,a[i].rnt);

}

}

sortQueue(lessThan10Queue,lessThan10Queue->size);

sortQueue(lessThan30Queue,lessThan30Queue->size);

sortQueue(greaterThan30Queue,greaterThan30Queue->size);

printf("Enter Quantum Number : ");

scanf("%d",&ts);

printf("Gantt Chart\n");

printf("0");

for(time=0,i=0;remain!=0;)

{

if(a[i].rnt<=ts && a[i].rnt>0)

{

time = time + a[i].rnt;

printf(" -> [P%d] <- %d",a[i].p,time);

a[i].rnt=0;

flag=1;

}

else if(a[i].rnt > 0)

{

a[i].rnt = a[i].rnt - ts;

time = time + ts;

printf(" -> [P%d] <- %d",a[i].p,time);

}

if(a[i].rnt==0 && flag==1)

{

remain--;

a[i].tat = time-a[i].art;

a[i].wtt = time-a[i].art-a[i].but;

avgwt = avgwt + time-a[i].art-a[i].but;

avgtt = avgtt + time-a[i].art;

flag=0;

}

if(i==pro-1)

i=0;

else if(a[i+1].art <= time)

i++;

else

i=0;

}

printf("Process ID \tArrival Time\tExecution Time\tTurn Around Time\tWaiting Time\n");

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

{

printf("P%d\t\t\t\t%d\t\t\t\t\t%d\t\t\t\t\t%d\t\t\t\t\t%d\n",a[i].p,a[i].art,a[i].but,a[i].tat,a[i].wtt);

}

avgwt = avgwt/pro;

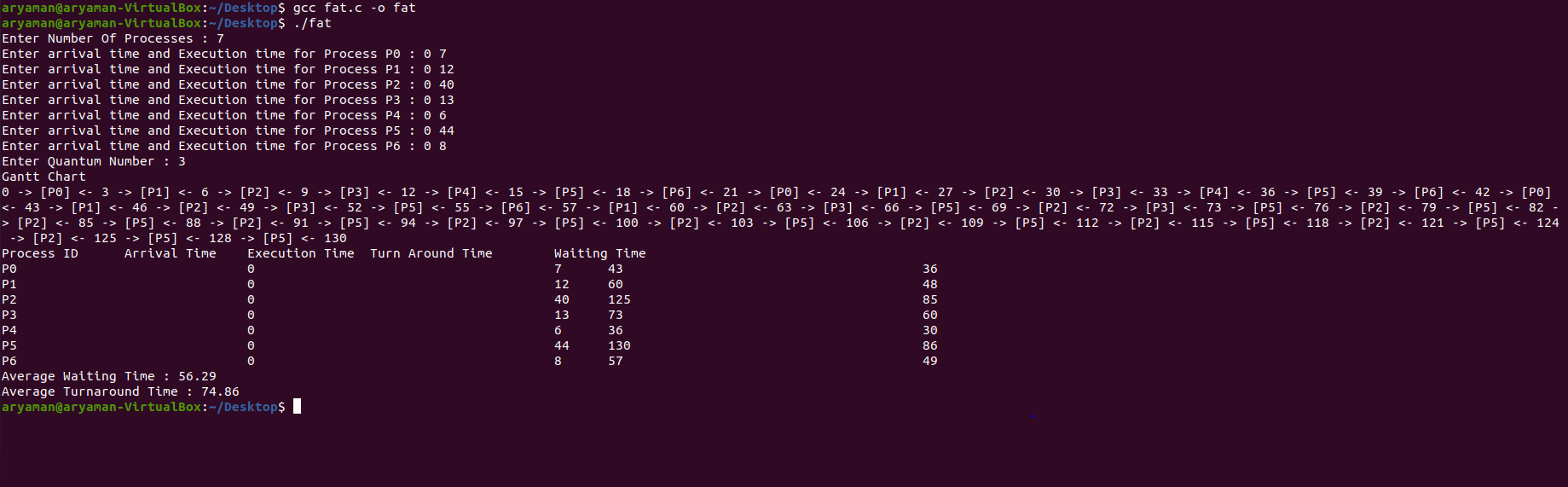
avgtt = avgtt/pro;

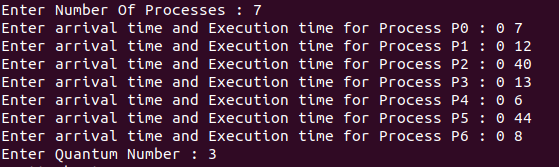
printf("Average Waiting Time : %.2f\n",avgwt);

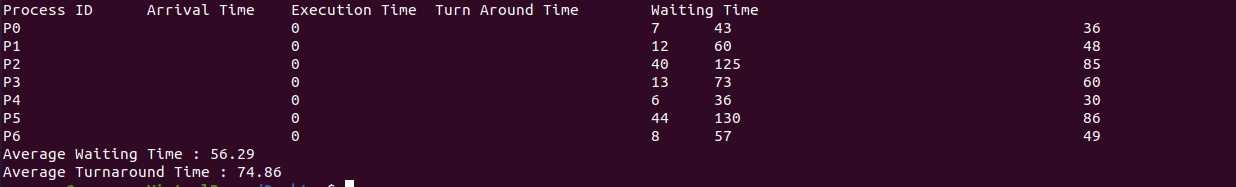
printf("Average Turnaround Time : %.2f\n",avgtt);

return 0;

}

**OBSERVATIONS:**





**Result:**

**Gantt chart and Average Waiting time has been achieved.**