Small Kernal

Operating Systems Project

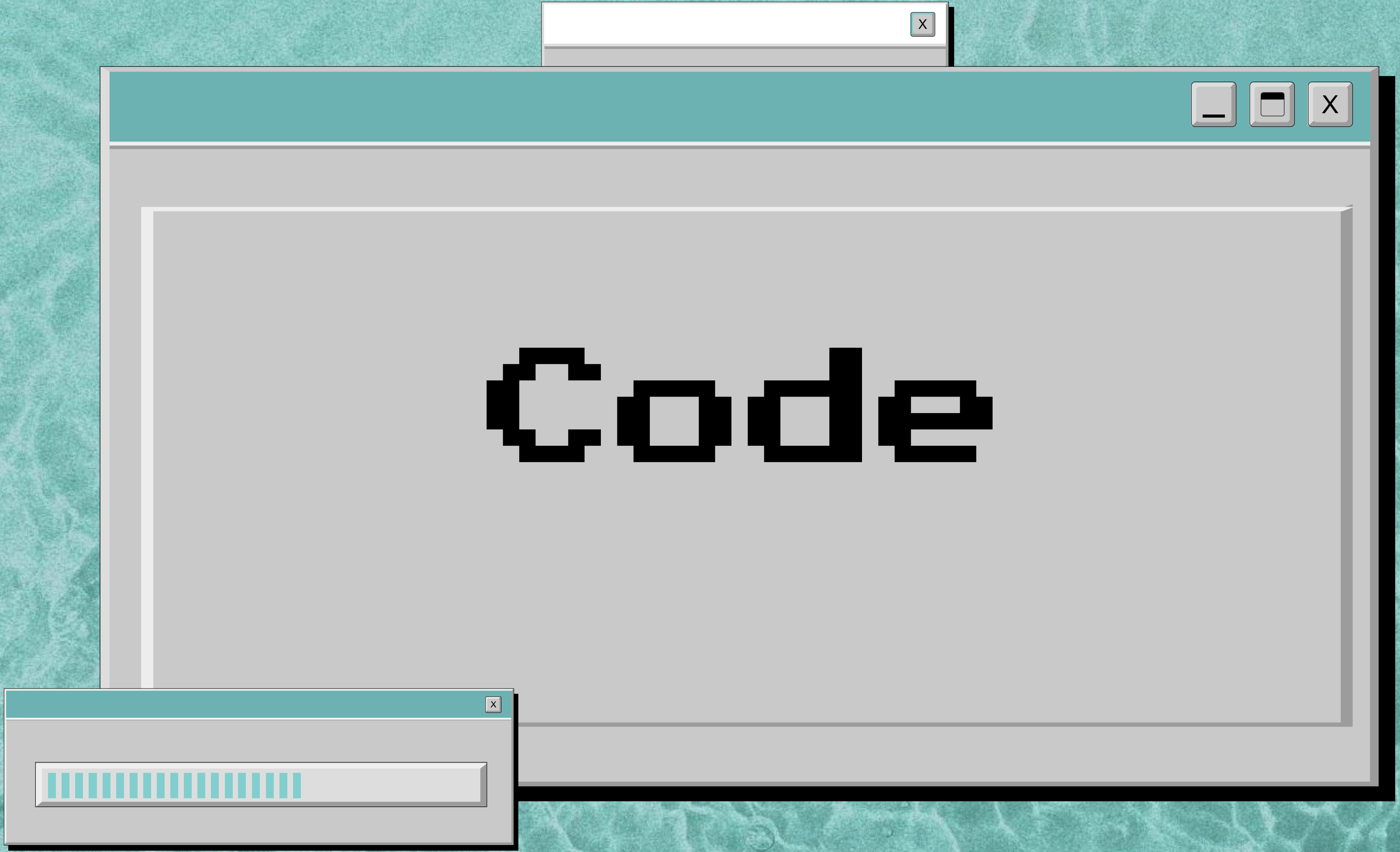
Contents:

Code

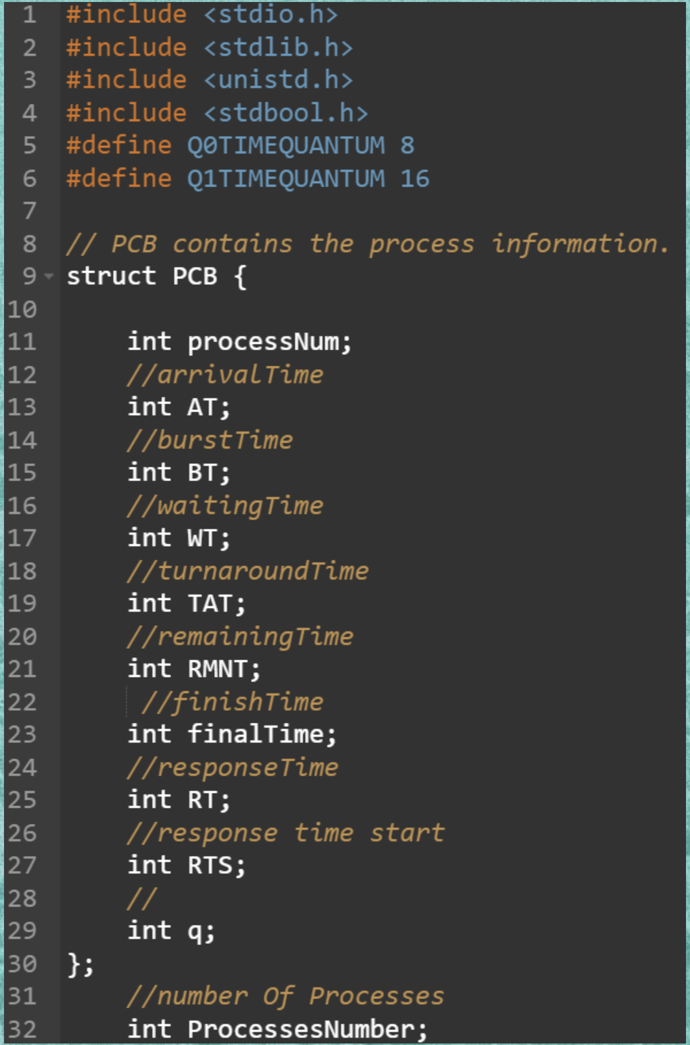
Output of code

Features and capabilities of your project

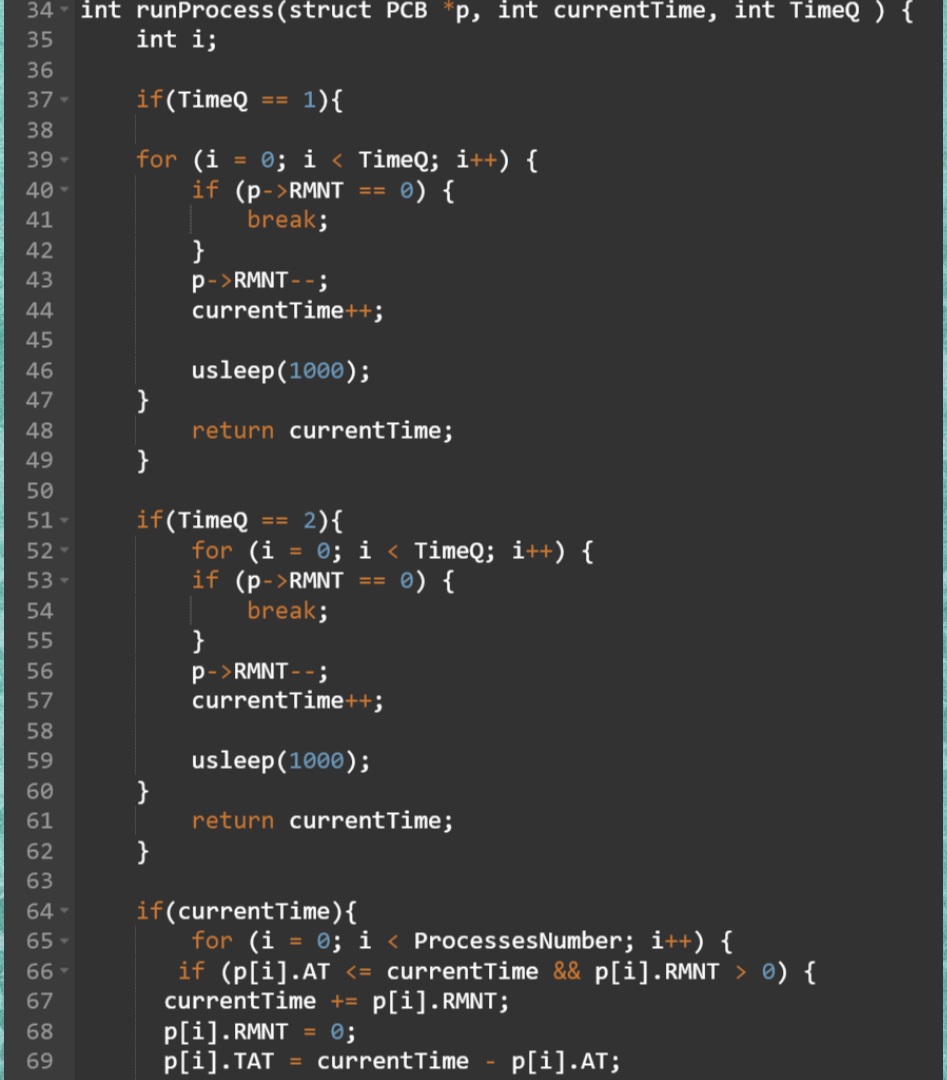
A simple user manual instructing a new user on how to use your program



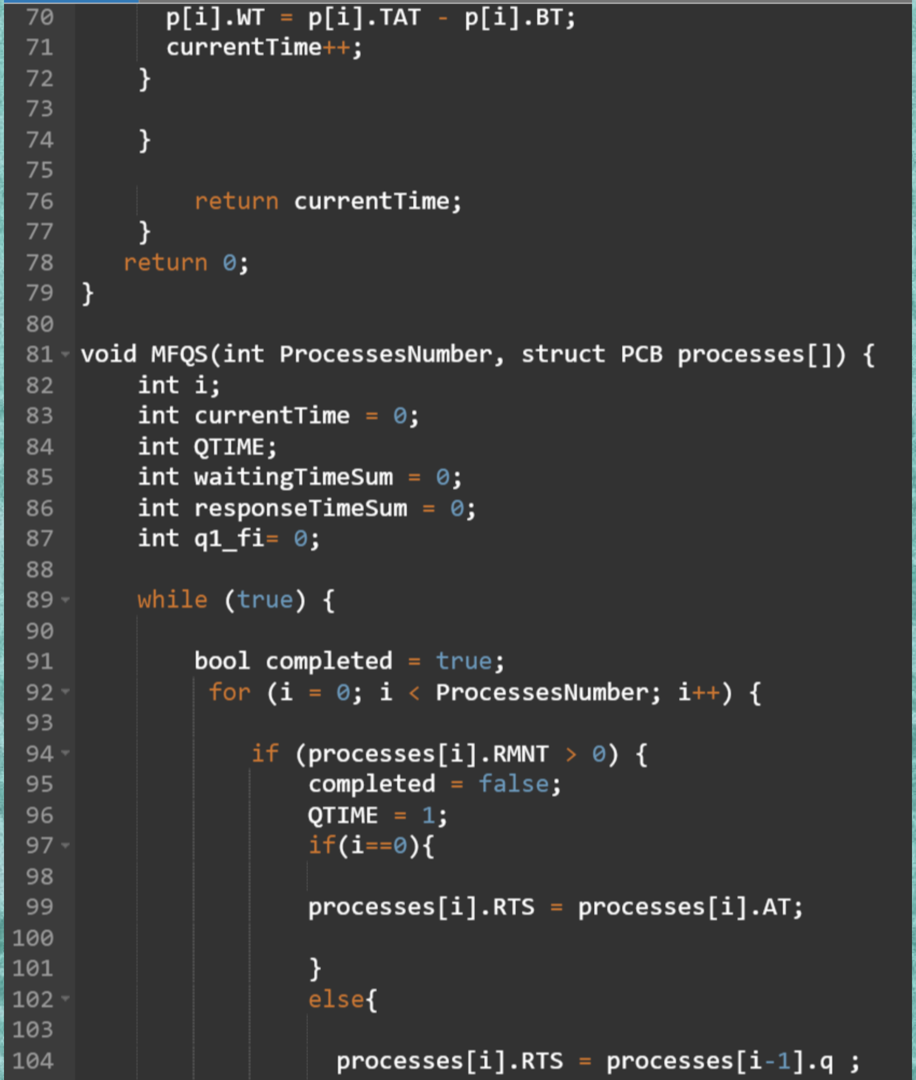
Code



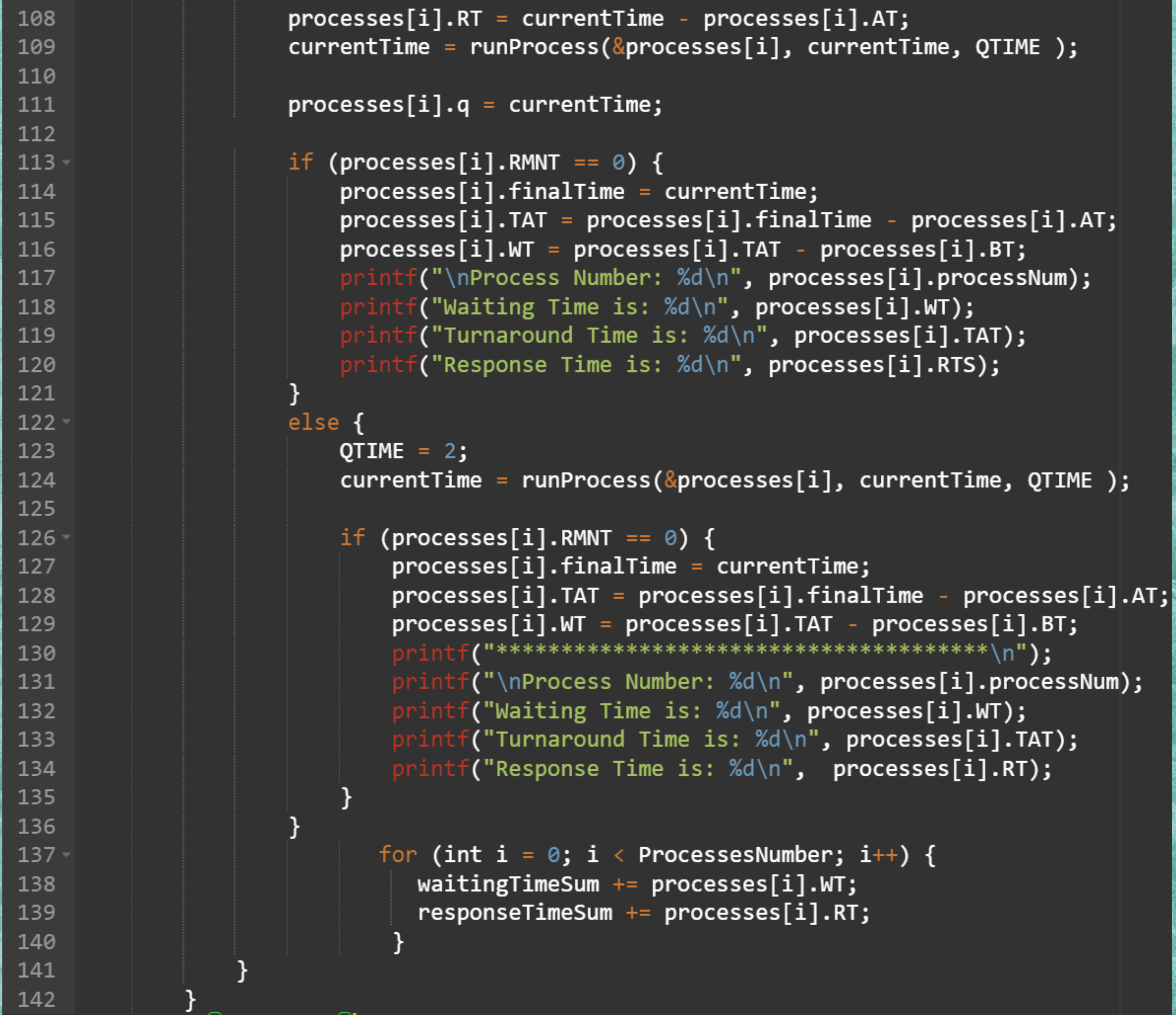
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdbool.h>
#define Q0TIMEQUANTUM 8
#define Q1TIMEQUANTUM 16

// PCB contains the process information.
struct PCB {

    int processNum;
    //arrivalTime
    int AT;
    //burstTime
    int BT;
    //waitingTime
    int WT;
    //turnaroundTime
    int TAT;
    //remainingTime
    int RMNT;
     //finishTime
    int finalTime;
    //responseTime
    int RT;
    //response time start
    int RTS;
    //
    int q;
};
    //number Of Processes
    int ProcessesNumber;
```



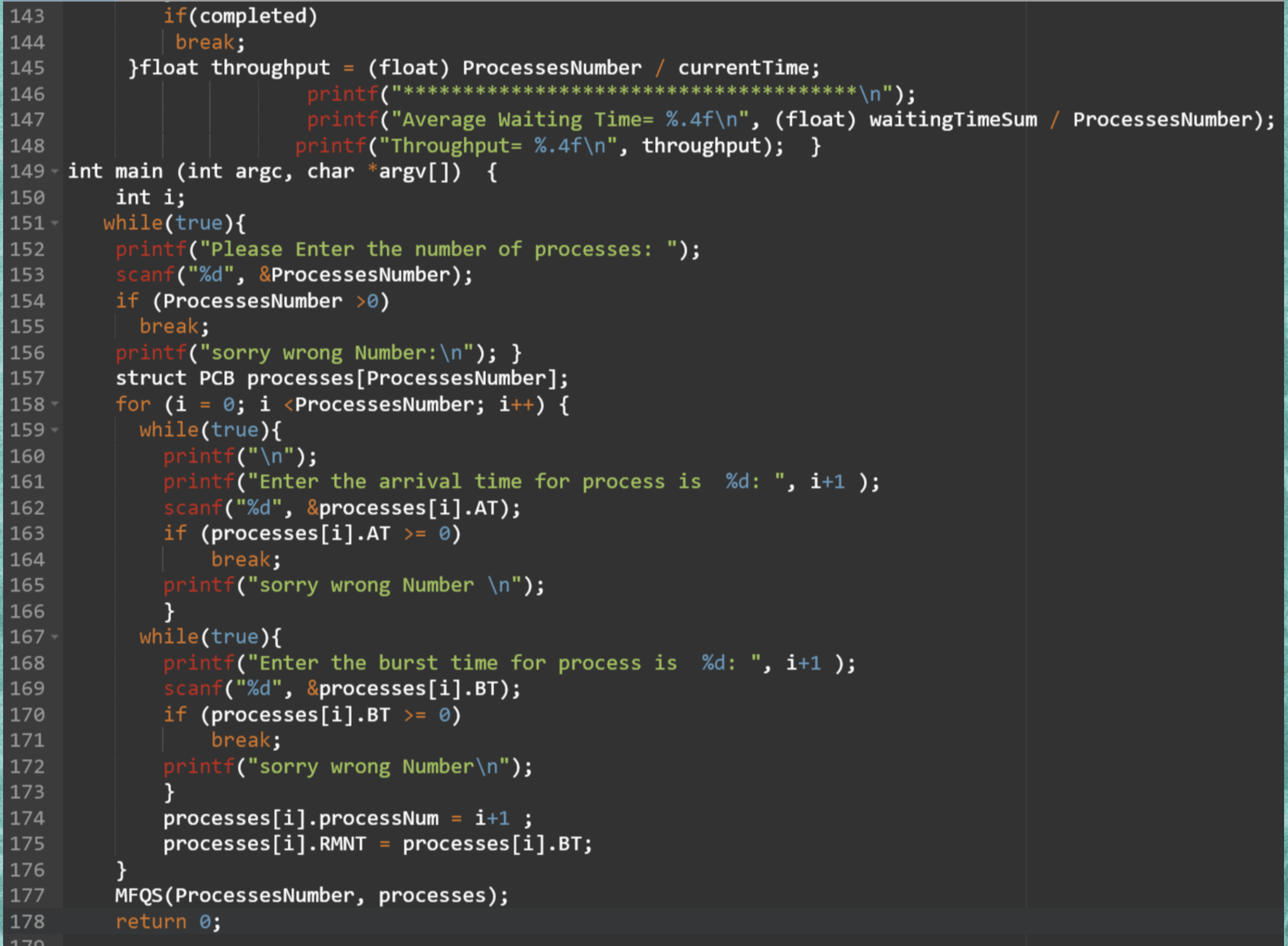
```
int runProcess(struct PCB *p, int currentTime, int TimeQ ) {
    int i;

    if(TimeQ == 1){

    for (i = 0; i < TimeQ; i++) {
        if (p->RMNT == 0) {
            break;
        }
        p->RMNT--;
        currentTime++;

        usleep(1000);
    }
        return currentTime;
    }

    if(TimeQ == 2){
        for (i = 0; i < TimeQ; i++) {
        if (p->RMNT == 0) {
            break;
        }
        p->RMNT--;
        currentTime++;

        usleep(1000);
    }
        return currentTime;
    }

    if(currentTime){
        for (i = 0; i < ProcessesNumber; i++) {
       if (p[i].AT <= currentTime && p[i].RMNT > 0) {
      currentTime += p[i].RMNT;
      p[i].RMNT = 0;
      p[i].TAT = currentTime - p[i].AT;
```



```
      p[i].WT = p[i].TAT - p[i].BT;
      currentTime++;
    }

    }

        return currentTime;
    }
   return 0;
}

void MFQS(int ProcessesNumber, struct PCB processes[]) {
    int i;
    int currentTime = 0;
    int QTIME;
    int waitingTimeSum = 0;
    int responseTimeSum = 0;
    int q1_fi= 0;

    while (true) {

        bool completed = true;
         for (i = 0; i < ProcessesNumber; i++) {

            if (processes[i].RMNT > 0) {
                completed = false;
                QTIME = 1;
                if(i==0){

                processes[i].RTS = processes[i].AT;

                }
                else{

                  processes[i].RTS = processes[i-1].q ;
```

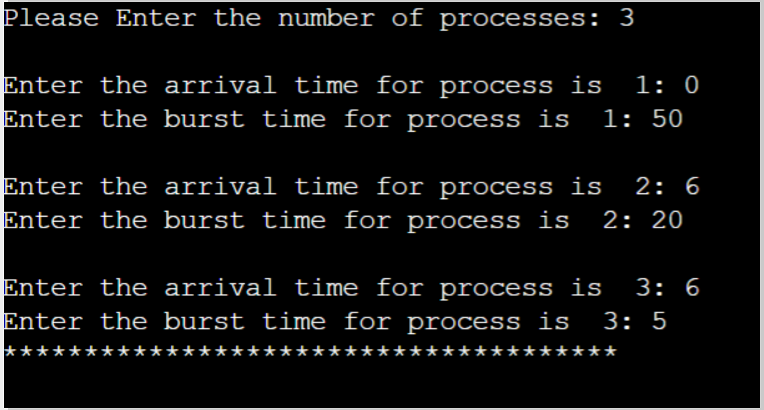


```c
        processes[i].RT = currentTime - processes[i].AT;
        currentTime = runProcess(&processes[i], currentTime, QTIME );

        processes[i].q = currentTime;

        if (processes[i].RMNT == 0) {
            processes[i].finalTime = currentTime;
            processes[i].TAT = processes[i].finalTime - processes[i].AT;
            processes[i].WT = processes[i].TAT - processes[i].BT;
            printf("\nProcess Number: %d\n", processes[i].processNum);
            printf("Waiting Time is: %d\n", processes[i].WT);
            printf("Turnaround Time is: %d\n", processes[i].TAT);
            printf("Response Time is: %d\n", processes[i].RTS);
        }
        else {
            QTIME = 2;
            currentTime = runProcess(&processes[i], currentTime, QTIME );

            if (processes[i].RMNT == 0) {
                processes[i].finalTime = currentTime;
                processes[i].TAT = processes[i].finalTime - processes[i].AT;
                processes[i].WT = processes[i].TAT - processes[i].BT;
                printf("**************************************\n");
                printf("\nProcess Number: %d\n", processes[i].processNum);
                printf("Waiting Time is: %d\n", processes[i].WT);
                printf("Turnaround Time is: %d\n", processes[i].TAT);
                printf("Response Time is: %d\n",  processes[i].RT);
            }
        }
               for (int i = 0; i < ProcessesNumber; i++) {
                  waitingTimeSum += processes[i].WT;
                  responseTimeSum += processes[i].RT;
                }
    }
}
```

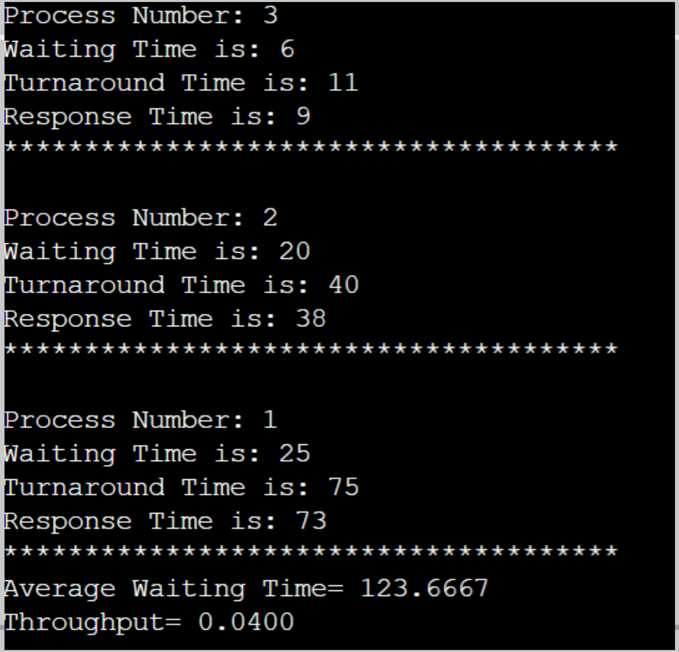


```c
        if(completed)
         break;
     }float throughput = (float) ProcessesNumber / currentTime;
                    printf("**************************************\n");
                    printf("Average Waiting Time= %.4f\n", (float) waitingTimeSum / ProcessesNumber);
                   printf("Throughput= %.4f\n", throughput);  }
int main (int argc, char *argv[])  {
    int i;
   while(true){
    printf("Please Enter the number of processes: ");
    scanf("%d", &ProcessesNumber);
    if (ProcessesNumber >0)
      break;
    printf("sorry wrong Number:\n"); }
    struct PCB processes[ProcessesNumber];
    for (i = 0; i <ProcessesNumber; i++) {
      while(true){
        printf("\n");
        printf("Enter the arrival time for process is  %d: ", i+1 );
        scanf("%d", &processes[i].AT);
        if (processes[i].AT >= 0)
            break;
        printf("sorry wrong Number \n");
        }
      while(true){
        printf("Enter the burst time for process is  %d: ", i+1 );
        scanf("%d", &processes[i].BT);
        if (processes[i].BT >= 0)
            break;
        printf("sorry wrong Number\n");
        }
        processes[i].processNum = i+1 ;
        processes[i].RMNT = processes[i].BT;
    }
    MFQS(ProcessesNumber, processes);
    return 0;
```

Output:



```
Please Enter the number of processes: 3

Enter the arrival time for process is  1: 0
Enter the burst time for process is  1: 50

Enter the arrival time for process is  2: 6
Enter the burst time for process is  2: 20

Enter the arrival time for process is  3: 6
Enter the burst time for process is  3: 5
**************************************
```



```
Process Number: 3
Waiting Time is: 6
Turnaround Time is: 11
Response Time is: 9
**************************************

Process Number: 2
Waiting Time is: 20
Turnaround Time is: 40
Response Time is: 38
**************************************

Process Number: 1
Waiting Time is: 25
Turnaround Time is: 75
Response Time is: 73
**************************************
Average Waiting Time= 123.6667
Throughput= 0.0400
```

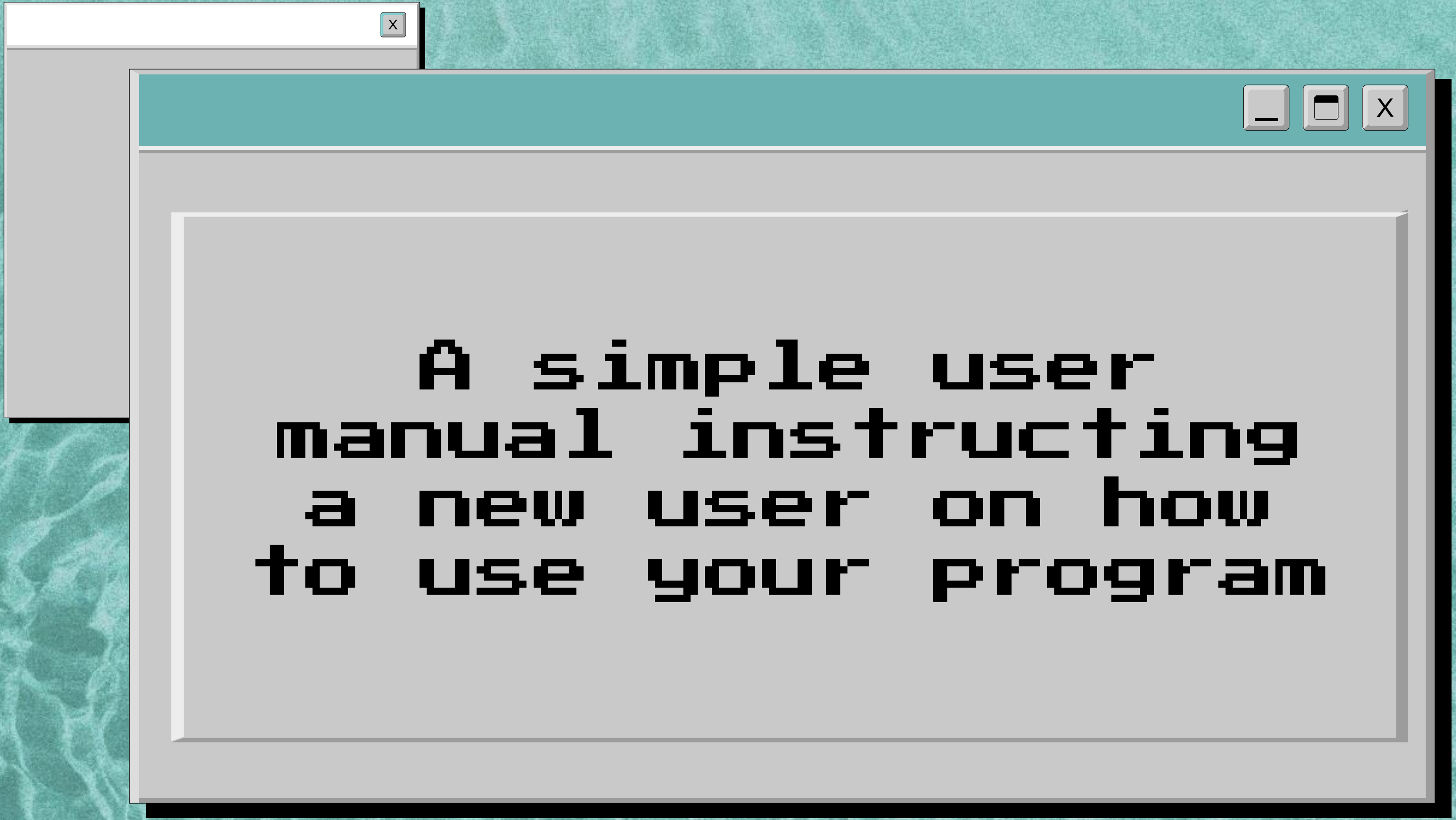
Features and capabilities of your project

Features

1-It is more flexible.

2-It allows different processes to move between different queues.

3-It prevents starvation by moving a process that waits too long for the lower priority queue to the higher priority queue.



A simple user manual instructing a new user on how to use your program

How to use the program?

Step 1

Run the program

Step 2

Request to enter the processor number

Step 3

Request to enter an arrival time and burst time

Step 4

Count RR with time quantum 8 milliseconds

Step 5

Cout RR with time quantum 16 milliseconds

Step 6

Count first come first serval (FCFS)

Step 7

Finally, each processer are displayed details