Practical No 6

Aim : Develop, debug and Execute a C program to simulate the Round Robin CPU scheduling algorithms to find turnaround time and waiting time.

Apparatus: Computer system with windows installed in it.

Mingw compiler for C/C++, and a text editor for developing C code file (Dev C++).

Theory :

What is RR scheduling?

- RR is short for 'Round Robin' Scheduling algorithm.
- This algorithm is designed especially for time sharing systems.
- It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes.
- A small unit of time, called a time quantum, or time slice, is defined. A time quantum is generally from 10 to 100ms in length.
- To implement RR scheduling, we treat the ready queue as a FIFO queue of processes.
- The performance of the RR algorithm depends heavily on the size of the time quantum.
- Every process is allocated no more than 1 time quantum in a row, unless it is the only process in the ready queue.
- To implement RR scheduling, we treat the ready queue as a FIFO queue of processes.
- One of the two scenarios can arise when interrupt occurs:
    - The process may have a burst time less than that of 1 time quantum, in this case the process itself will release the CPU voluntarily.
    - If the CPU burst of the current running process is longer than 1 time quantum, the timer will go off and cause an interrupt to the operating system. A context switch will be executed, and the process will be pout at the tail of the ready queue.
    - The CPU scheduler will then select the next process in the ready queue.

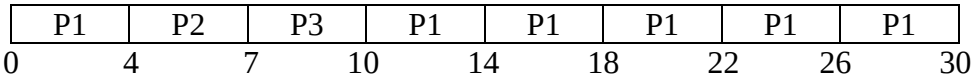
What is a Time Quantum?

- A time quantum is defined as a small unit of time or a time slice.
- A time quantum is generally from 10 to 100ms in length.

- The performance of the RR algorithm depends heavily on the size of the time quantum.
    - If the quantum is extremely large, the RR policy is the same as the FCFS policy.
    - If the quantum is extremely small (e.g. 1ms), the RR approach can result in a large number of context switches.
- We want the time quantum to be large with respect to the context switch time.
- If the context-switch time is approximately 10 percent of the time quantum, then about 10% of the CPU time will be spent in context switching.
- Most computing systems have a time quantum ranging from 10ms to 100ms.
- The time required for a context switch is typically less than 10 microseconds.
- Thus, the context-switch time is a small fraction of the time quantum

Example :

Process	Burst Time
P1	**24**
P2	**3**
P3	**3**



Waiting time for p1 = (10-4) = 6

Waiting time for p2 = 4

Waiting time for p3 = 7

Average waiting time =p1 + p2 + p3 / 3

= (6 + 4 + 7)/ 3

= 17/4

= 5.67ms

Turnaround time for p1 = 30ms

Turnaround time for p2 = 7ms

Turnaround time for p3 = 10ms

Average turnaround time = 30 + 7 + 10 / 3

= 47 / 3

= 15.67ms

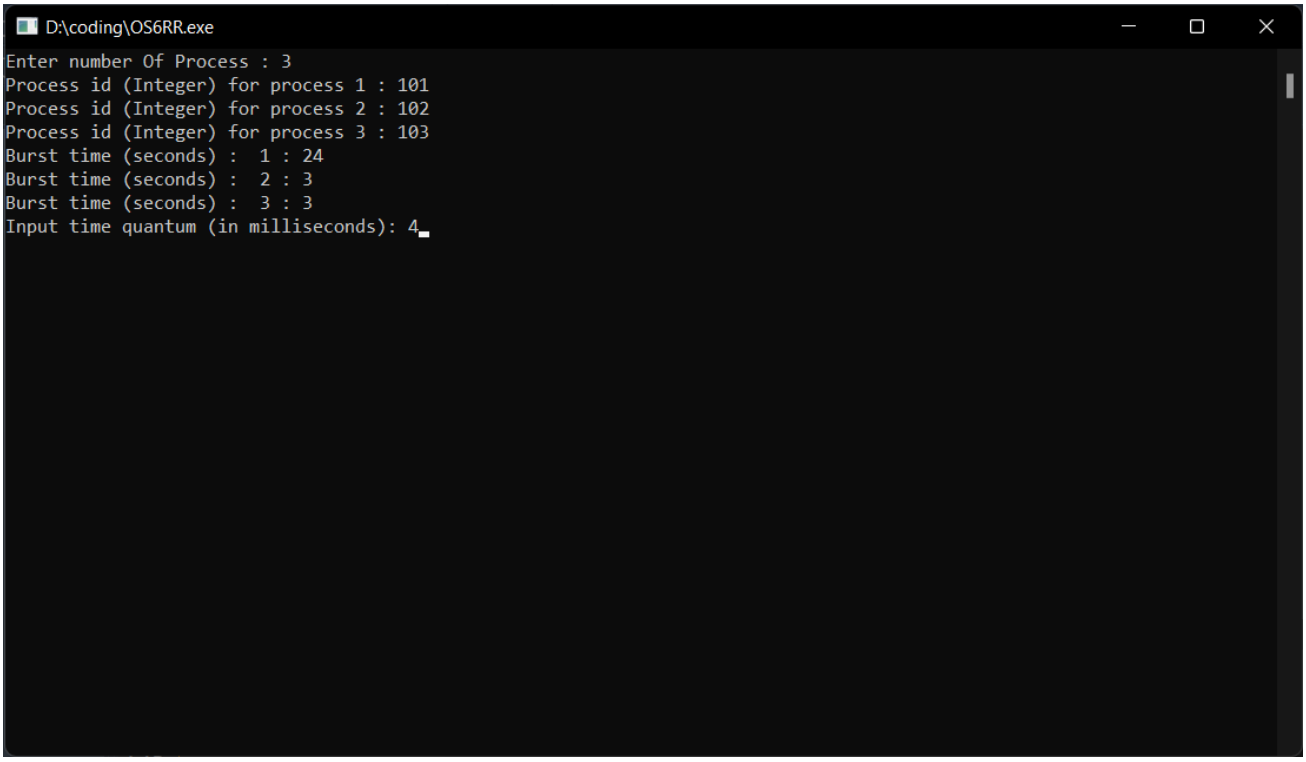
Code:

```
#include<stdio.h>
#define true 1
#define false 0


void CalculateWaitingTime(int size, int burstTime[], int waitingTime[],int
timeQuantum){
  int burstTimeRemaining[size];
  for(int i = 0; i < size; i++){
    burstTimeRemaining[i] = burstTime[i];
  }

  int time = 0;
  int flag = true;
  while(flag){

    int done = true;

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

      if(burstTimeRemaining[i] > 0){
        done = false;

        if(burstTimeRemaining[i] > timeQuantum){
          time += timeQuantum;
          burstTimeRemaining[i] -= timeQuantum;
        }else{
          time = time + burstTimeRemaining[i];
          burstTimeRemaining[i] -= burstTimeRemaining[i]; ///
          waitingTime[i] = time - burstTime[i];
        }

      }

    }

    if(done == true){
```

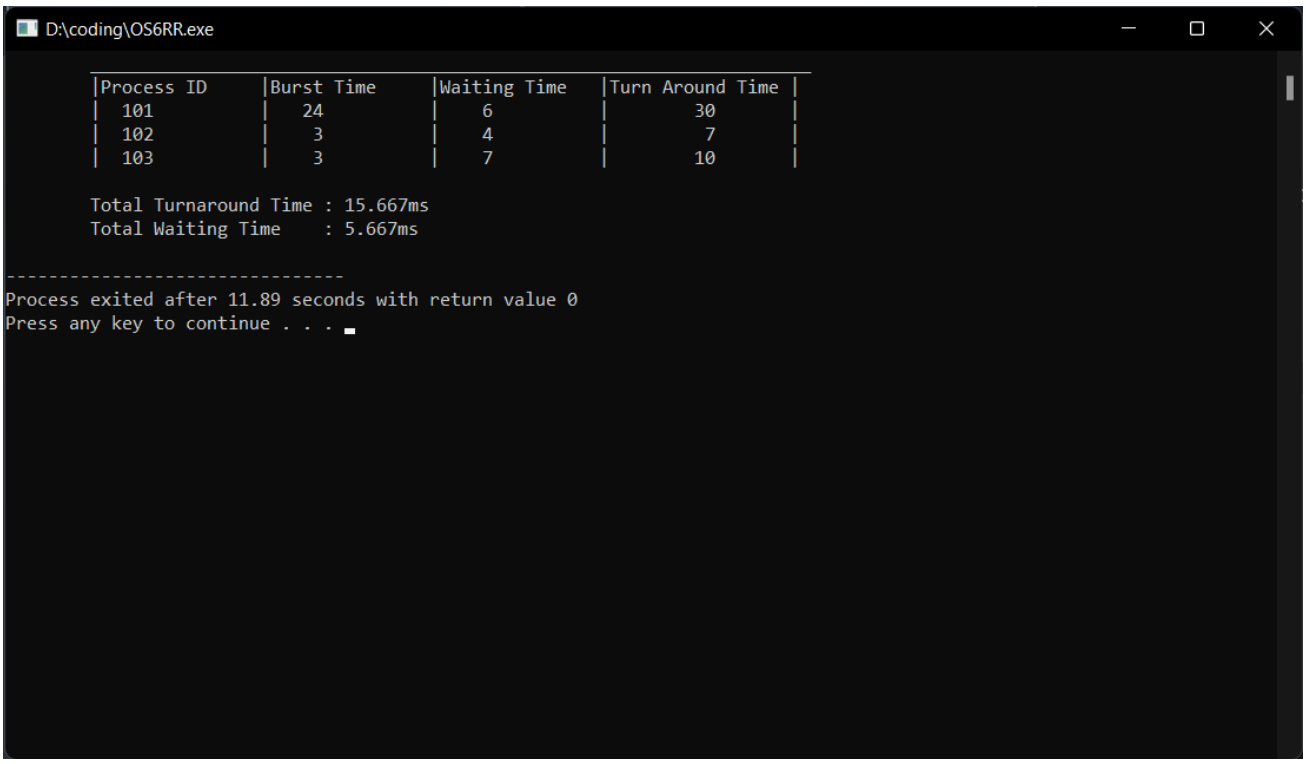
```
      flag = false;
    }

  }

}

void CalculateTurnAroundTime(int turnAroundTime[],int waitingTime[], int
burstTime[],int size){

  for(int i = 0; i <size; i++){
    turnAroundTime[i] = waitingTime[i] + burstTime[i];
  }

}

void println(char ch){
  for(int i = 0; i < 68 ; i++)
    printf("%c",ch);
  printf("\n");
}

void CalculateAverageTime(int process[], int size, int burstTime[], int
timeQuantum){
  int waitingTime[size];
  int turnAroundTime[size];
  float totalWaitingTime = 0, totalTurnAroundTime = 0;

  CalculateWaitingTime(size,burstTime,waitingTime,timeQuantum);

  CalculateTurnAroundTime(turnAroundTime,waitingTime,burstTime,size);

  for(int i = 0; i < size; i++){
    totalWaitingTime += waitingTime[i];
  }

  for(int i = 0; i < size; i++){
    totalTurnAroundTime += turnAroundTime[i];
  }
  system("cls");
  printf("\t");


  println('_');
  printf("\t|Process ID\t|Burst Time\t|Waiting Time\t|Turn Around Time |\n");
  for(int i = 0; i < size; i++ ){
    printf("\t|%5d\t\t|%5d\t\t|%5d\t\t|%10d\t
|\n",process[i],burstTime[i],waitingTime[i],turnAroundTime[i]);
```

```
  }


  totalTurnAroundTime = totalTurnAroundTime/(float) size;
  totalWaitingTime = totalWaitingTime/(float) size;
  printf("\n\tTotal Turnaround Time : %.3fms\n",totalTurnAroundTime);
  printf("\tTotal Waiting Time    : %.3fms\n",totalWaitingTime);
}

void input(int arr[], int n, char* name){

  for(int i = 0; i<n; i++){
    printf("%s %d : ",name,i+1);
    scanf("%d",&arr[i]);
  }

}

int main(){

  int sizeOfProcess;
  int timeQuantum;

  printf("Enter number Of Process : ");
  scanf("%d",&sizeOfProcess);
  int process[sizeOfProcess];
  int burstTime[sizeOfProcess];

  input(process,sizeOfProcess,"Process id (Integer) for process");

  input(burstTime,sizeOfProcess,"Burst time (seconds) : ");

  printf("Input time quantum (in milliseconds): ");
  scanf("%d",&timeQuantum);

  CalculateAverageTime(process,sizeOfProcess,burstTime,timeQuantum);


  return 0;
}
```

Output:



Output 6.1. Inputting Data



Output 6.2. Displaying Result

Conclusion:

Hence, by performing this practical I got to know about the concept of Round Robin Scheduling Algorithm, which is almost like First-come, First-serve scheduling but with Preemption added to it by using Time Quantum for preempting processes after exactly 1 time quantum. I also developed, debugged and executed a c program to simulate the Round Robin CPU scheduling algorithm and found out the turnaround time and the waiting time using the program.