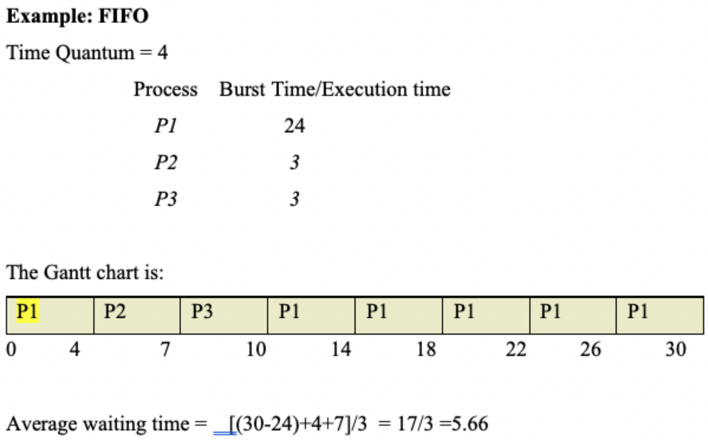
OPERATING SYSTEM LAB 5

Roll No.: K041	Name: Anish Sudhan Nair
Batch No.: A2/K2	Date: 17/01/2022

Aim: To familiarise and implement the round robin algorithm.

1. Example 1



Example: FIFO

Time Quantum = 4

Process	Burst Time/Execution time
P1	24
P2	*3*
P3	*3*

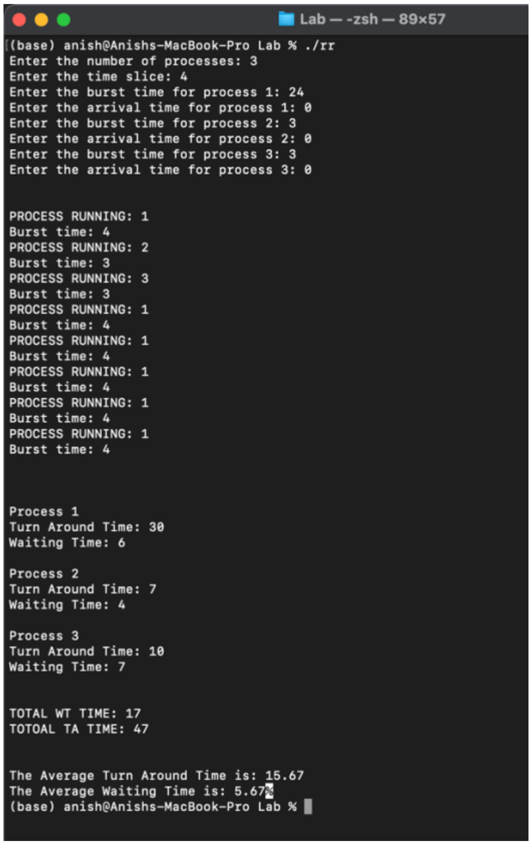
The Gantt chart is:

P1	P2	P3	P1	P1	P1	P1	P1

0 4 7 10 14 18 22 26 30

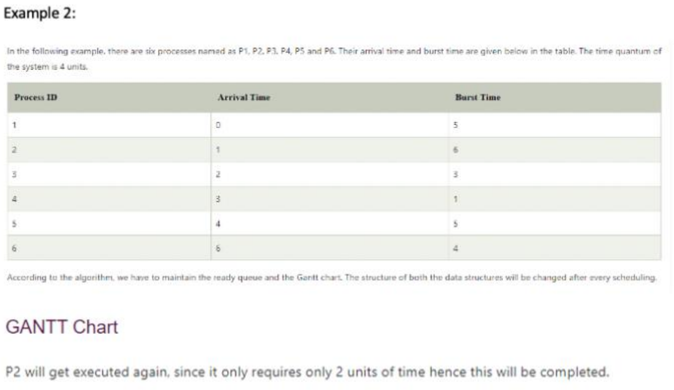
Average waiting time = [(30-24)+4+7]/3 = 17/3 =5.66

Output:



```
[(base) anish@Anishs-MacBook-Pro Lab % ./rr
Enter the number of processes: 3
Enter the time slice: 4
Enter the burst time for process 1: 24
Enter the arrival time for process 1: 0
Enter the burst time for process 2: 3
Enter the arrival time for process 2: 0
Enter the burst time for process 3: 3
Enter the arrival time for process 3: 0


PROCESS RUNNING: 1
Burst time: 4
PROCESS RUNNING: 2
Burst time: 3
PROCESS RUNNING: 3
Burst time: 3
PROCESS RUNNING: 1
Burst time: 4
PROCESS RUNNING: 1
Burst time: 4
PROCESS RUNNING: 1
Burst time: 4
PROCESS RUNNING: 1
Burst time: 4
PROCESS RUNNING: 1
Burst time: 4



Process 1
Turn Around Time: 30
Waiting Time: 6

Process 2
Turn Around Time: 7
Waiting Time: 4

Process 3
Turn Around Time: 10
Waiting Time: 7


TOTAL WT TIME: 17
TOTOAL TA TIME: 47


The Average Turn Around Time is: 15.67
The Average Waiting Time is: 5.67%
(base) anish@Anishs-MacBook-Pro Lab %
```

2. Example 2



Example 2:

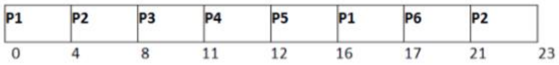
In the following example, there are six processes named as P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table. The time quantum of the system is 4 units.

Process ID	Arrival Time	Burst Time
1	0	5
2	1	6
3	2	3
4	3	1
5	4	5
6	6	4

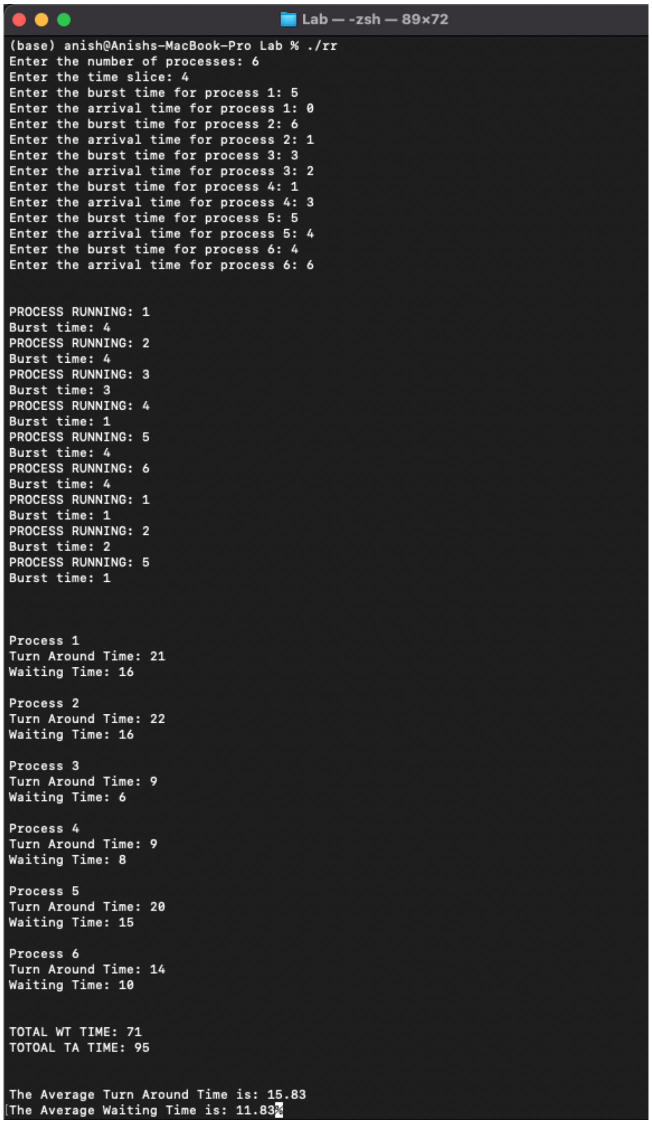
According to the algorithm, we have to maintain the ready queue and the Gantt chart. The structure of both the data structures will be changed after every scheduling.

GANTT Chart

P2 will get executed again, since it only requires only 2 units of time hence this will be completed.



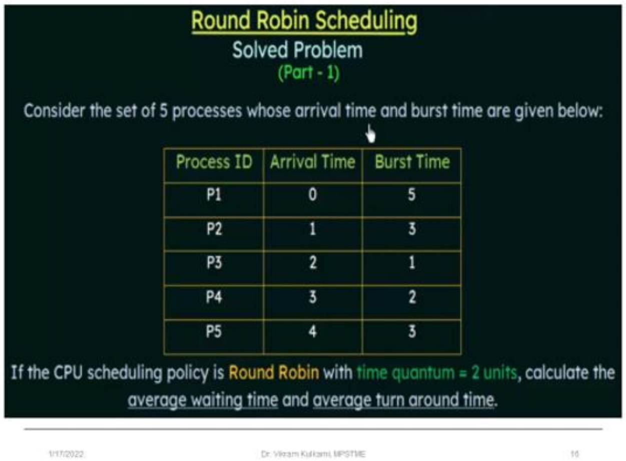
Output:



```
(base) anish@Anishs-MacBook-Pro Lab % ./rr
Enter the number of processes: 6
Enter the time slice: 4
Enter the burst time for process 1: 5
Enter the arrival time for process 1: 0
Enter the burst time for process 2: 6
Enter the arrival time for process 2: 1
Enter the burst time for process 3: 3
Enter the arrival time for process 3: 2
Enter the burst time for process 4: 1
Enter the arrival time for process 4: 3
Enter the burst time for process 5: 5
Enter the arrival time for process 5: 4
Enter the burst time for process 6: 4
Enter the arrival time for process 6: 6


PROCESS RUNNING: 1
Burst time: 4
PROCESS RUNNING: 2
Burst time: 4
PROCESS RUNNING: 3
Burst time: 3
PROCESS RUNNING: 4
Burst time: 1
PROCESS RUNNING: 5
Burst time: 4
PROCESS RUNNING: 6
Burst time: 4
PROCESS RUNNING: 1
Burst time: 1
PROCESS RUNNING: 2
Burst time: 2
PROCESS RUNNING: 5
Burst time: 1



Process 1
Turn Around Time: 21
Waiting Time: 16

Process 2
Turn Around Time: 22
Waiting Time: 16

Process 3
Turn Around Time: 9
Waiting Time: 6

Process 4
Turn Around Time: 9
Waiting Time: 8

Process 5
Turn Around Time: 20
Waiting Time: 15

Process 6
Turn Around Time: 14
Waiting Time: 10


TOTAL WT TIME: 71
TOTOAL TA TIME: 95


The Average Turn Around Time is: 15.83
The Average Waiting Time is: 11.83%
```

3. Example 3

Example-3



Round Robin Scheduling

Solved Problem

(Part - 1)

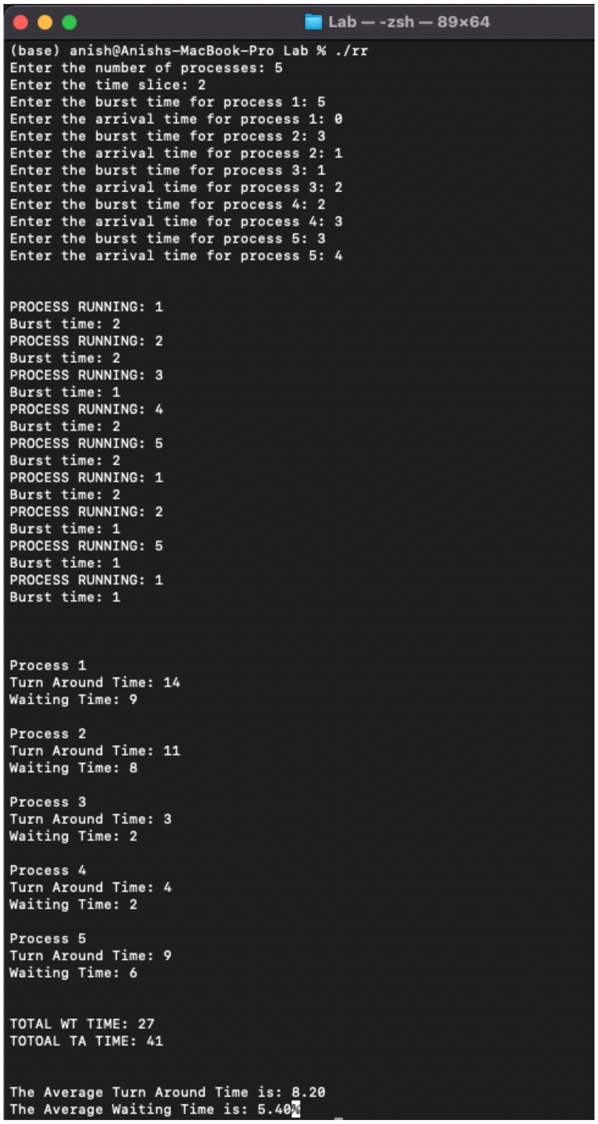
Consider the set of 5 processes whose arrival time and burst time are given below:

Process ID	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

If the CPU scheduling policy is Round Robin with time quantum = 2 units, calculate the average waiting time and average turn around time.

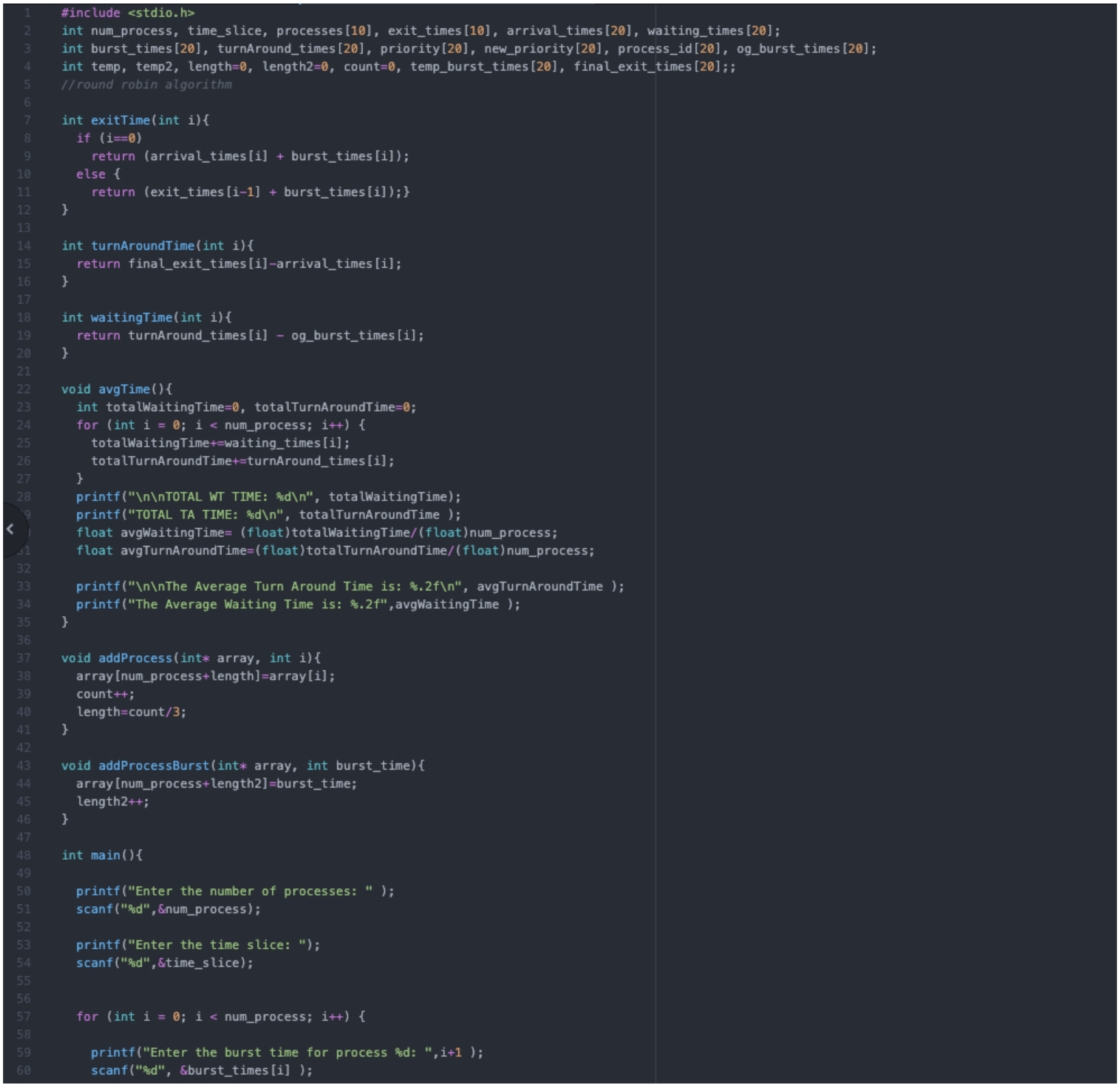
1/17/2022 Dr. Vikram Kulkarni, MPSTME 16

Output:

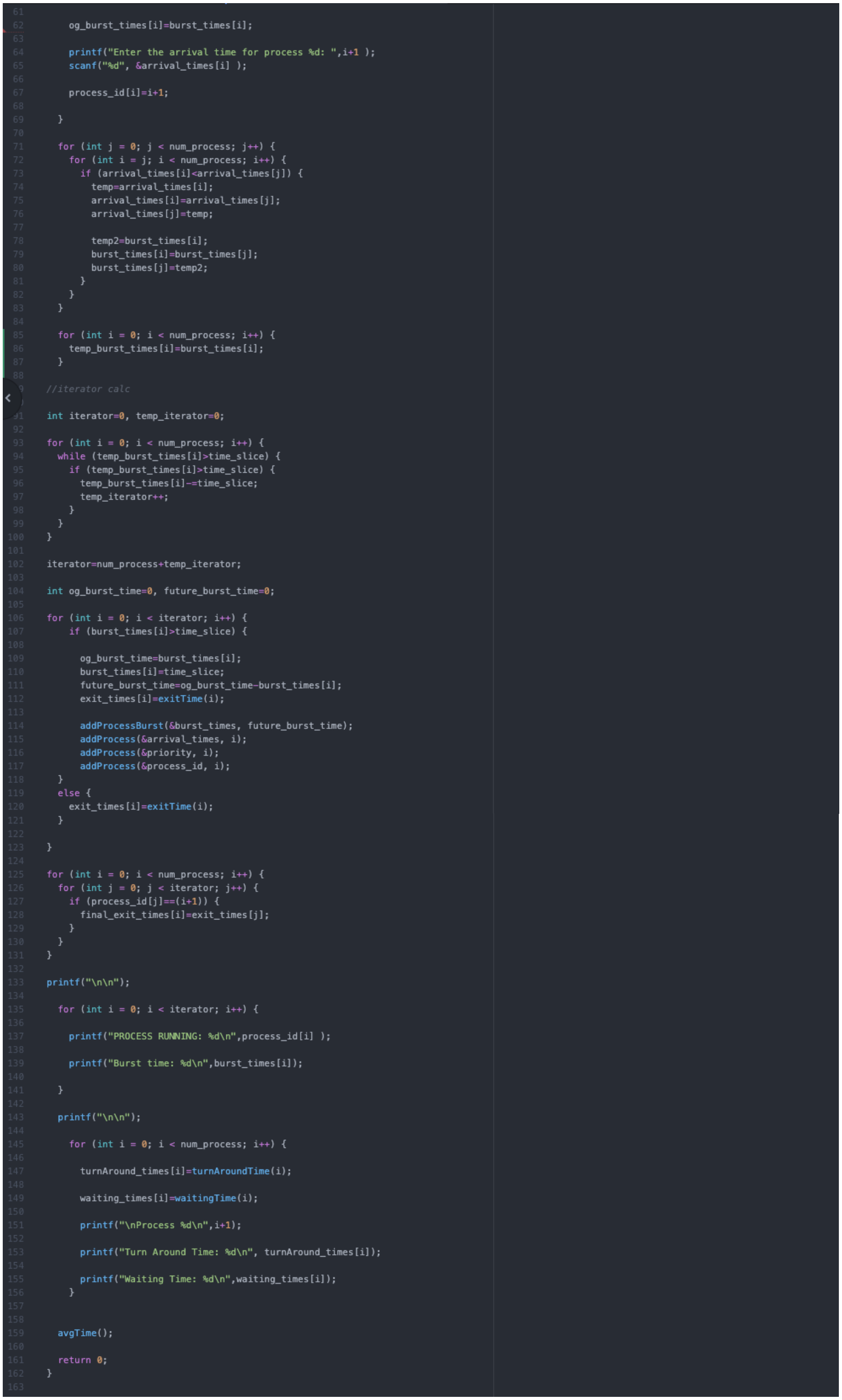


```
(base) anish@Anishs-MacBook-Pro Lab % ./rr
Enter the number of processes: 5
Enter the time slice: 2
Enter the burst time for process 1: 5
Enter the arrival time for process 1: 0
Enter the burst time for process 2: 3
Enter the arrival time for process 2: 1
Enter the burst time for process 3: 1
Enter the arrival time for process 3: 2
Enter the burst time for process 4: 2
Enter the arrival time for process 4: 3
Enter the burst time for process 5: 3
Enter the arrival time for process 5: 4


PROCESS RUNNING: 1
Burst time: 2
PROCESS RUNNING: 2
Burst time: 2
PROCESS RUNNING: 3
Burst time: 1
PROCESS RUNNING: 4
Burst time: 2
PROCESS RUNNING: 5
Burst time: 2
PROCESS RUNNING: 1
Burst time: 2
PROCESS RUNNING: 2
Burst time: 1
PROCESS RUNNING: 5
Burst time: 1
PROCESS RUNNING: 1
Burst time: 1



Process 1
Turn Around Time: 14
Waiting Time: 9

Process 2
Turn Around Time: 11
Waiting Time: 8

Process 3
Turn Around Time: 3
Waiting Time: 2

Process 4
Turn Around Time: 4
Waiting Time: 2

Process 5
Turn Around Time: 9
Waiting Time: 6


TOTAL WT TIME: 27
TOTOAL TA TIME: 41


The Average Turn Around Time is: 8.20
The Average Waiting Time is: 5.40%
```

CODE:



```c
#include <stdio.h>
int num_process, time_slice, processes[10], exit_times[10], arrival_times[20], waiting_times[20];
int burst_times[20], turnAround_times[20], priority[20], new_priority[20], process_id[20], og_burst_times[20];
int temp, temp2, length=0, length2=0, count=0, temp_burst_times[20], final_exit_times[20];;
//round robin algorithm

int exitTime(int i){
  if (i==0)
    return (arrival_times[i] + burst_times[i]);
  else {
    return (exit_times[i-1] + burst_times[i]);}
}

int turnAroundTime(int i){
  return final_exit_times[i]-arrival_times[i];
}

int waitingTime(int i){
  return turnAround_times[i] - og_burst_times[i];
}

void avgTime(){
  int totalWaitingTime=0, totalTurnAroundTime=0;
  for (int i = 0; i < num_process; i++) {
    totalWaitingTime+=waiting_times[i];
    totalTurnAroundTime+=turnAround_times[i];
  }
  printf("\n\nTOTAL WT TIME: %d\n", totalWaitingTime);
  printf("TOTAL TA TIME: %d\n", totalTurnAroundTime );
  float avgWaitingTime= (float)totalWaitingTime/(float)num_process;
  float avgTurnAroundTime=(float)totalTurnAroundTime/(float)num_process;

  printf("\n\nThe Average Turn Around Time is: %.2f\n", avgTurnAroundTime );
  printf("The Average Waiting Time is: %.2f",avgWaitingTime );
}

void addProcess(int* array, int i){
  array[num_process+length]=array[i];
  count++;
  length=count/3;
}

void addProcessBurst(int* array, int burst_time){
  array[num_process+length2]=burst_time;
  length2++;
}

int main(){

  printf("Enter the number of processes: " );
  scanf("%d",&num_process);

  printf("Enter the time slice: ");
  scanf("%d",&time_slice);


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

    printf("Enter the burst time for process %d: ",i+1 );
    scanf("%d", &burst_times[i] );
```



```
      og_burst_times[i]=burst_times[i];

      printf("Enter the arrival time for process %d: ",i+1 );
      scanf("%d", &arrival_times[i] );

      process_id[i]=i+1;

    }

    for (int j = 0; j < num_process; j++) {
      for (int i = j; i < num_process; i++) {
        if (arrival_times[i]<arrival_times[j]) {
          temp=arrival_times[i];
          arrival_times[i]=arrival_times[j];
          arrival_times[j]=temp;

          temp2=burst_times[i];
          burst_times[i]=burst_times[j];
          burst_times[j]=temp2;
        }
      }
    }

    for (int i = 0; i < num_process; i++) {
      temp_burst_times[i]=burst_times[i];
    }

  //iterator calc

  int iterator=0, temp_iterator=0;

  for (int i = 0; i < num_process; i++) {
    while (temp_burst_times[i]>time_slice) {
      if (temp_burst_times[i]>time_slice) {
        temp_burst_times[i]-=time_slice;
        temp_iterator++;
      }
    }
  }

  iterator=num_process+temp_iterator;

  int og_burst_time=0, future_burst_time=0;

  for (int i = 0; i < iterator; i++) {
      if (burst_times[i]>time_slice) {

        og_burst_time=burst_times[i];
        burst_times[i]=time_slice;
        future_burst_time=og_burst_time-burst_times[i];
        exit_times[i]=exitTime(i);

        addProcessBurst(&burst_times, future_burst_time);
        addProcess(&arrival_times, i);
        addProcess(&priority, i);
        addProcess(&process_id, i);
    }
    else {
      exit_times[i]=exitTime(i);
    }

  }

  for (int i = 0; i < num_process; i++) {
    for (int j = 0; j < iterator; j++) {
      if (process_id[j]==(i+1)) {
        final_exit_times[i]=exit_times[j];
      }
    }
  }

  printf("\n\n");

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

      printf("PROCESS RUNNING: %d\n",process_id[i] );

      printf("Burst time: %d\n",burst_times[i]);

    }

    printf("\n\n");

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

        turnAround_times[i]=turnAroundTime(i);

        waiting_times[i]=waitingTime(i);

        printf("\nProcess %d\n",i+1);

        printf("Turn Around Time: %d\n", turnAround_times[i]);

        printf("Waiting Time: %d\n",waiting_times[i]);
      }


    avgTime();

    return 0;
  }

```

CONCLUSION:

In this lab, we were to implement and demonstrate the working of the round robin algorithm. By actually coding the algorithm, it helped to reinforce the working of this process and the manner in which it schedules the processes in a CPU.