COMP301

Operating Systems

Section: C

Assignment#: 1

Title: Implementation of the Shortest Remaining Time First Algorithm (SRTF)

Dated: November 6, 2014

Submitted To: Prof. Muhammad Rauf Butt

Submitted by: Waleed Ahmed

Roll#: 16-10876

Abstract:

The purpose of this code is to simulate the Shortest Remaining Time First Algorithm (SRTF) which is the pre-emptive version of SJF (Shortest Job First) algorithm which was non-pre-emptive. This algorithm is used in batch operating systems. Before this FCFS algorithm was in practice which had a few issues. It favored the long processes and processor bound operations. SJF solved this issue but it did not have the facility to interrupt the processor during the processing and hence the processor used to stay idle during the process swaps. These issues are resolved in this version. It not only allows the process with minimum burst time to be processed but also it allows the processor to be disturbed by pre-empting the present running process if the new arrived process has a less amount of Burst time as compared to the remaining burst time of the process which in being processed. In this way not only we enhance the processing by reducing the average waiting time but also we avoid the convoy effect. This algorithm is considered to be the best for the non-time sharing Operating Systems.

The basic steps in this algorithm are:

1. Find the total number of processes that have been added in the given interval.

2. Find the one with the shortest Remaining Burst Time.

3. Process it (by decreasing the remaining time).

4. Repeat until all the processes have a Remaining Time of zero.

Introduction:

The code on the following page is an implementation of the SRTF (Shortest Remaining Time First) algorithms. It takes two arrays from the user comprising of the Arrival Times and Burst Times respectively as specified by the user. The process enqueued at the instant zero is processed until another process with less amount of Burst Time arrives in the meanwhile. In this case, the previous process in pre-empted from the processor and the new process is allowed to be processed until another process with even less burst time arrives and the same process of pre-emption is repeated. If the new process is having more amount of the burst time than all the remaining burst times of the processes on the queue then the current process is allowed to execute until another pre-emption is required. Finish Time in this case depends on the instant on the time line where the processor has completely executed a specific process. That time is marked as the Finish Time. The average time calculation differs a little in this case. It is the difference of the Finishing Time and the sum of Arrival Time and the Burst Time. The turn-around time does not affect our calculations. The only issue that is sometime disturbs is that if the process pre-empted had a higher priority.

Code:

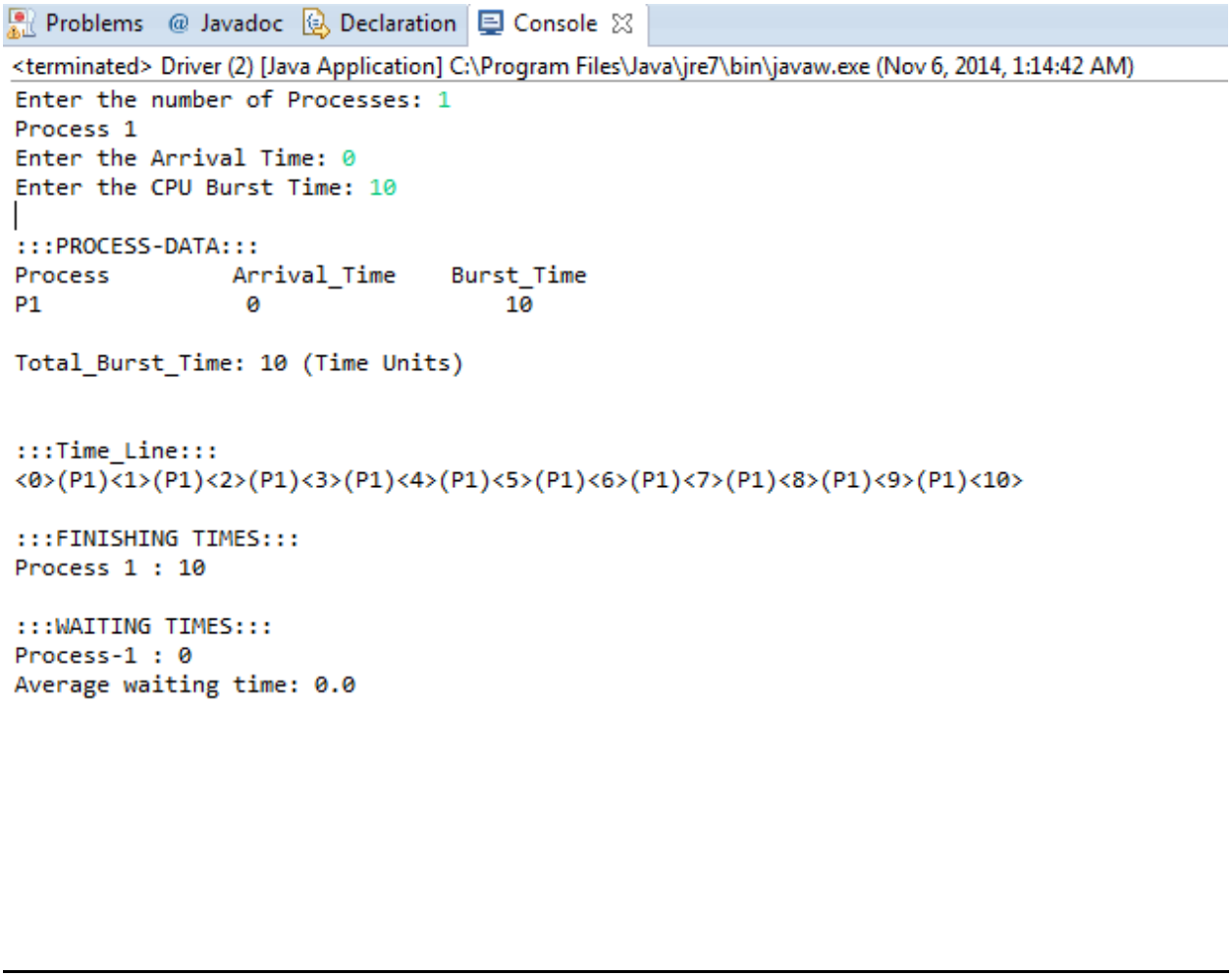
```
import java.util.*;

public class Driver {



	public static void SRTF(int []AT, int []BT, int []RT, int []FT, int []WT, int
n){
		int TT = 0; // total burst time required

		for (int i = 0; i < n; i++) {
			TT += BT[i];
		}
		System.out.println(":::PROCESS-DATA:::");
		System.out.println("Process		Arrival_Time Burst_Time");
		for (int i = 0; i < n; i++) {
			System.out.println("P" + (i + 1) + "		" + AT[i]
					+ "		" + BT[i]);
		}

		System.out.println("\nTotal_Burst_Time: " + TT + " (Time Units)\n\n");

		int timer = 0;
		int present=0;
		int smallest=0;
		int prev_smallest=0;

		System.out.println(":::Time_Line:::");
		while(timer < TT){

				System.out.print("<"+(timer)+">");

			//1. Finding total processes that have been enqueued
			int i=0;
			while(i<AT.length && i<=timer){
				if(AT[i]==timer){
					present++;
				}
				i++;
			}

			//2. Finding the one with the Shortest Remaining Time amongst the
enqueued processes

			smallest=0;
			for(int j=0;j<present;j++){

				//ones who's time is greater than zero are considered
				if( RT[j]>0 && (RT[j]<=RT[smallest]) ){
					smallest=j;
				}
			}
```

```
                        System.out.print("(P"+(smallest+1)+")");

                        //3. Consumption of the present smallest by one unit time
                        RT[smallest]--;

                        if(RT[smallest]==0){      //if that process has been consumed
                              FT[smallest]=timer+1;//+1;       //1 is being added since
the timer starts counting on 0

                        }
                        timer++;
                  }

                  System.out.print("<"+(timer)+">");

                  System.out.println("\n\n:::FINISHING TIMES:::");

                  for(int t=0;t<n;t++){
                        System.out.println("Process "+(t+1)+" : "+FT[t]);
                  }


                  int total_waiting_time=0;
                  for(int t=0;t<n;t++){
                        WT[t]=FT[t]-(BT[t]+AT[t]);       //waiting time formula for Pre-
emptive Scheduling
                        total_waiting_time+=WT[t];
                  }


                  System.out.println("\n:::WAITING TIMES:::");
                  for(int i=0;i<WT.length;i++){
                        System.out.println("Process-"+(i+1)+" : "+WT[i]);
                  }

                  double temp_ave=total_waiting_time/n;

                  System.out.println("Average waiting time: "+temp_ave);

      }//end_Driver



      public static void main(String[] args) {
            Scanner scan = new Scanner(System.in);

            System.out.print("Enter the number of Processes: ");
            int n = scan.nextInt();

            int[] AT = new int[n];     //Arrival Times
            int[] BT = new int[n];     //Burst Times
            int[] RT = new int[n];     //Remaining Time
            int[] FT = new int[n];     //Finish Time
            int[] WT = new int[n];     //Waiting Times
```

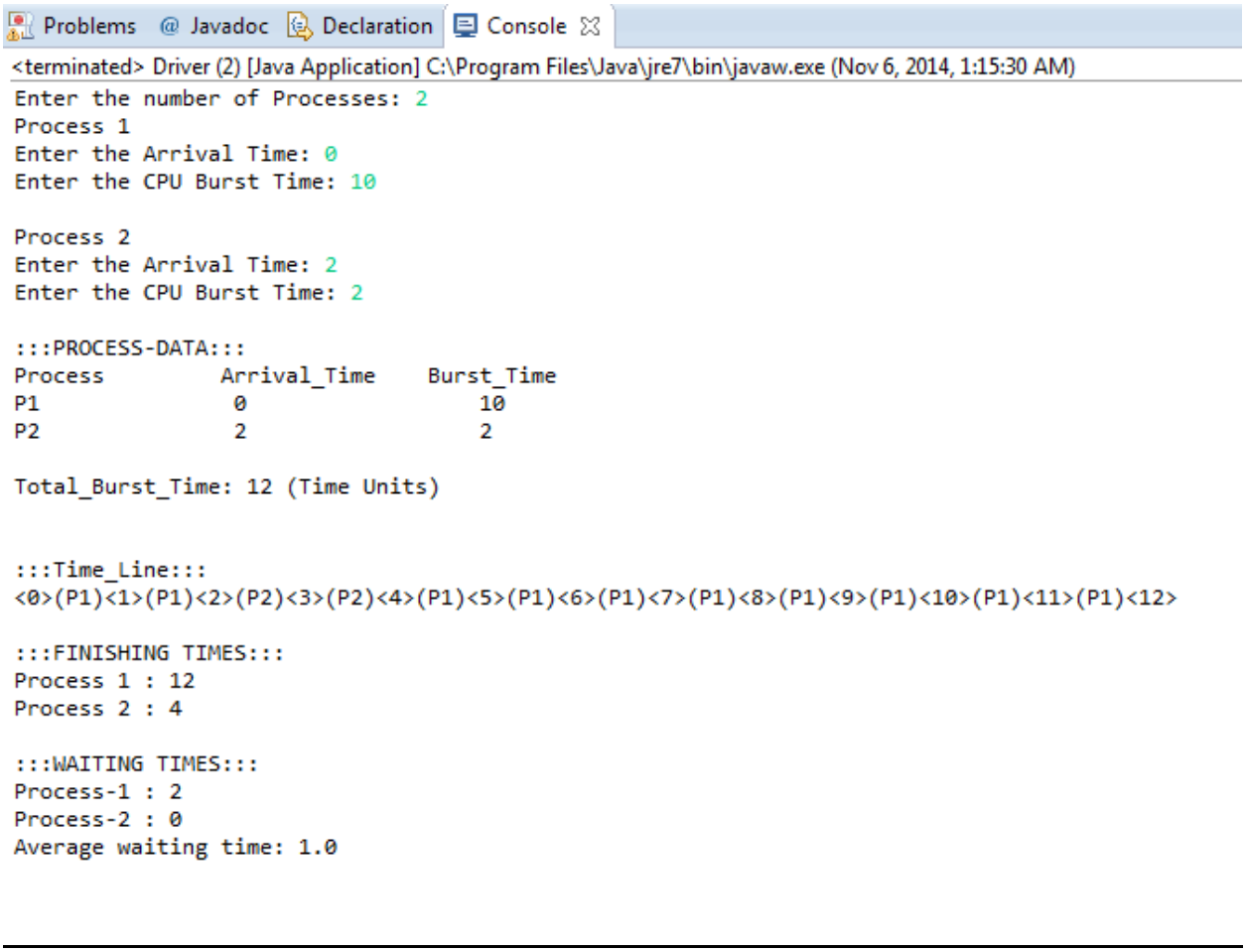
```
            for (int i = 0; i < n; i++) {// initialization of FT array
                  FT[i] = 0;
            }

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



                  System.out.println("Process " + (i + 1));
                  System.out.print("Enter the Arrival Time: ");
                  AT[i] = scan.nextInt();

                  System.out.print("Enter the CPU Burst Time: ");
                  BT[i] = scan.nextInt();

                  RT[i]=BT[i]; //Assignment of the initial Remaining Times

                  System.out.println();
            }

            SRTF( AT, BT, RT, FT, WT, n);
      }

}//end_main()
```

Output Samples:

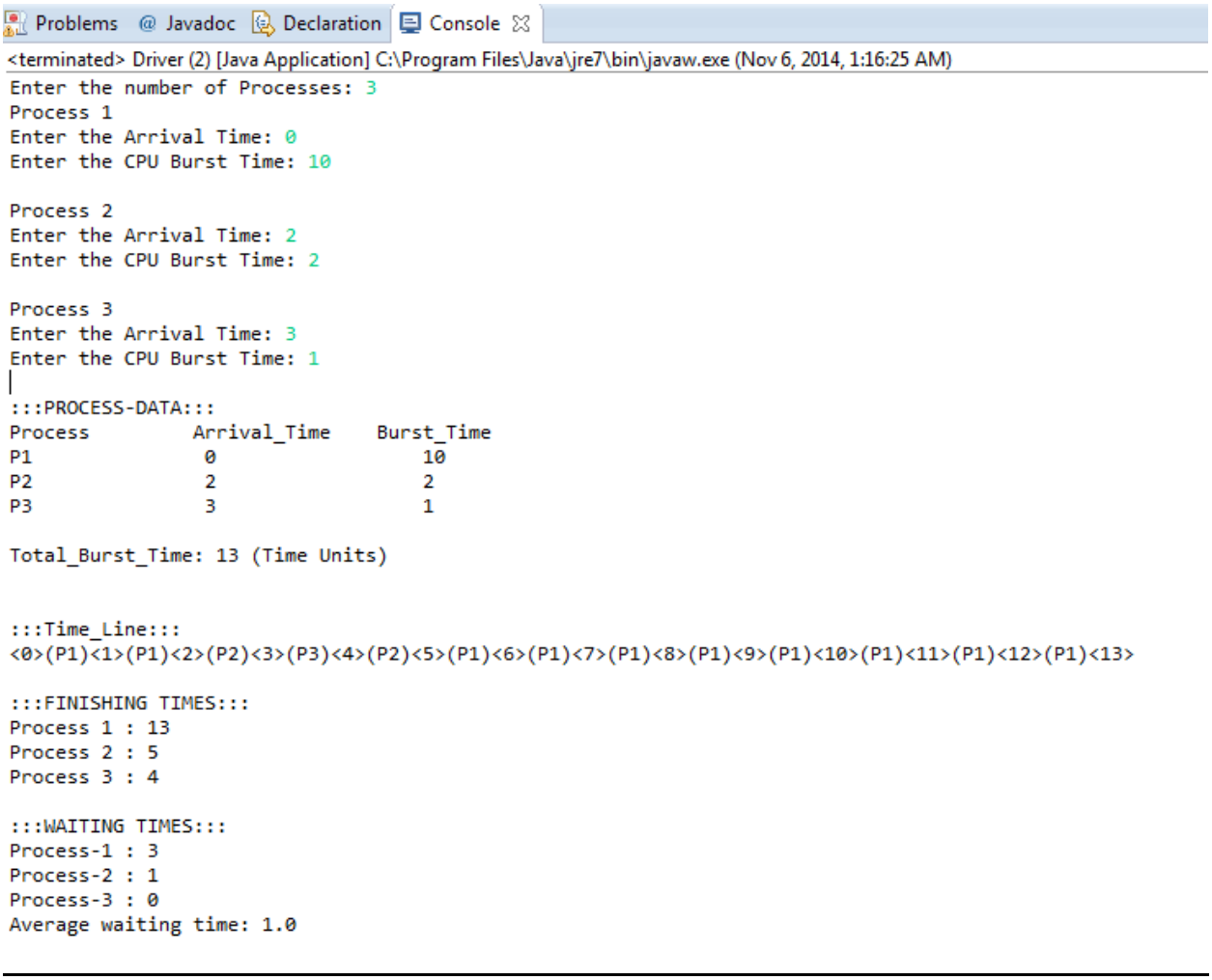
Sample-1



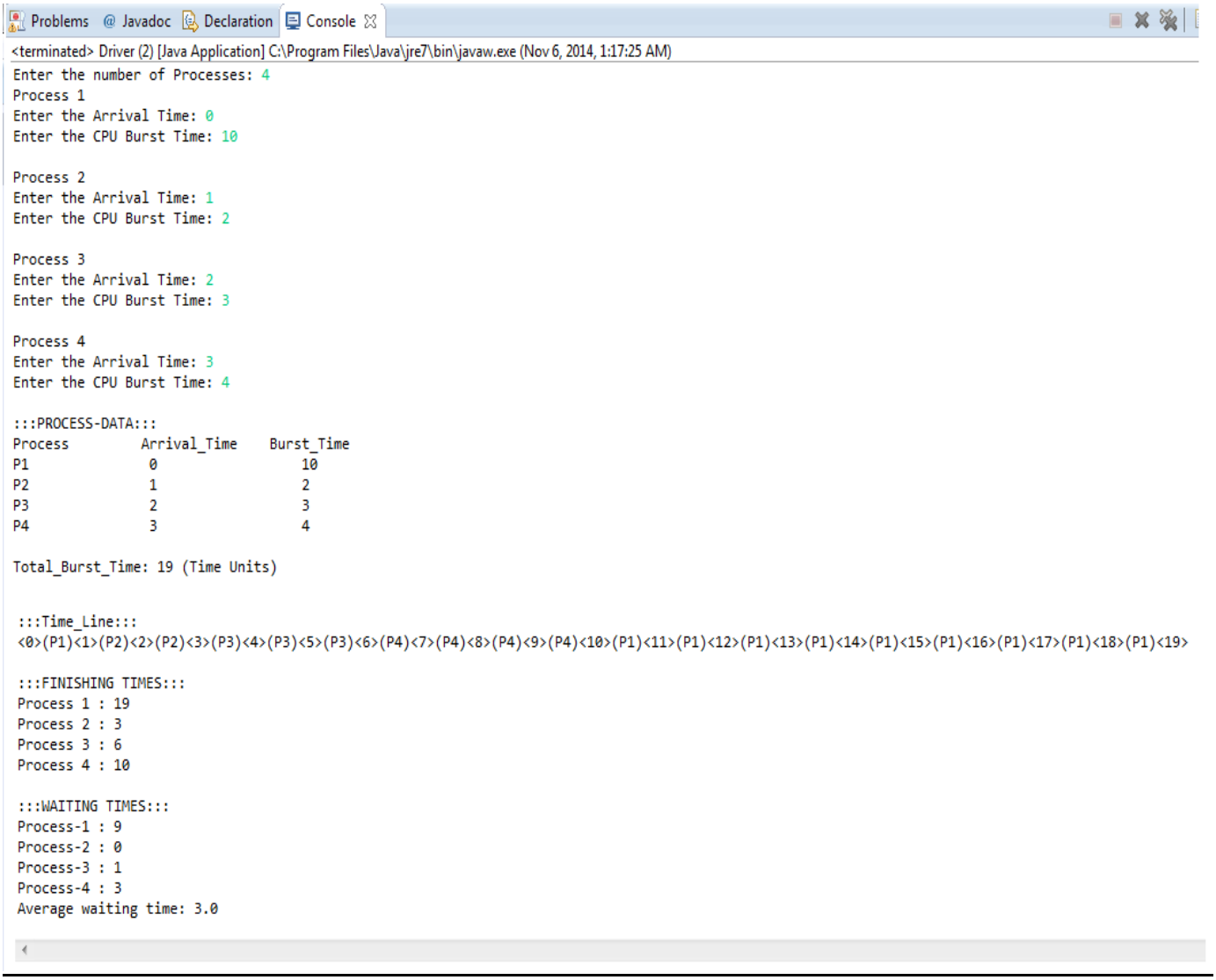
Sample-2


Problems
Javadoc
Declaration
Console
<terminated> Driver (2) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 6, 2014, 1:15:30 AM)
Enter the number of Processes: 2
Process 1
Enter the Arrival Time: 0
Enter the CPU Burst Time: 10
Process 2
Enter the Arrival Time: 2
Enter the CPU Burst Time: 2
:::PROCESS-DATA:::
Process Arrival_Time Burst_Time
P1 0 10
P2 2 2
Total_Burst_Time: 12 (Time Units)
:::Time_Line:::
<0>(P1)<1>(P1)<2>(P2)<3>(P2)<4>(P1)<5>(P1)<6>(P1)<7>(P1)<8>(P1)<9>(P1)<10>(P1)<11>(P1)<12>
:::FINISHING TIMES:::
Process 1 : 12
Process 2 : 4
:::WAITING TIMES:::
Process-1 : 2
Process-2 : 0
Average waiting time: 1.0

Sample-3


Problems
Javadoc
Declaration
Console
<terminated> Driver (2) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 6, 2014, 1:16:25 AM)
Enter the number of Processes: 3
Process 1
Enter the Arrival Time: 0
Enter the CPU Burst Time: 10
Process 2
Enter the Arrival Time: 2
Enter the CPU Burst Time: 2
Process 3
Enter the Arrival Time: 3
Enter the CPU Burst Time: 1
:::PROCESS-DATA:::
Process Arrival_Time Burst_Time
P1 0 10
P2 2 2
P3 3 1
Total_Burst_Time: 13 (Time Units)
:::Time_Line:::
<0>(P1)<1>(P1)<2>(P2)<3>(P3)<4>(P2)<5>(P1)<6>(P1)<7>(P1)<8>(P1)<9>(P1)<10>(P1)<11>(P1)<12>(P1)<13>
:::FINISHING TIMES:::
Process 1 : 13
Process 2 : 5
Process 3 : 4
:::WAITING TIMES:::
Process-1 : 3
Process-2 : 1
Process-3 : 0
Average waiting time: 1.0

Sample-4

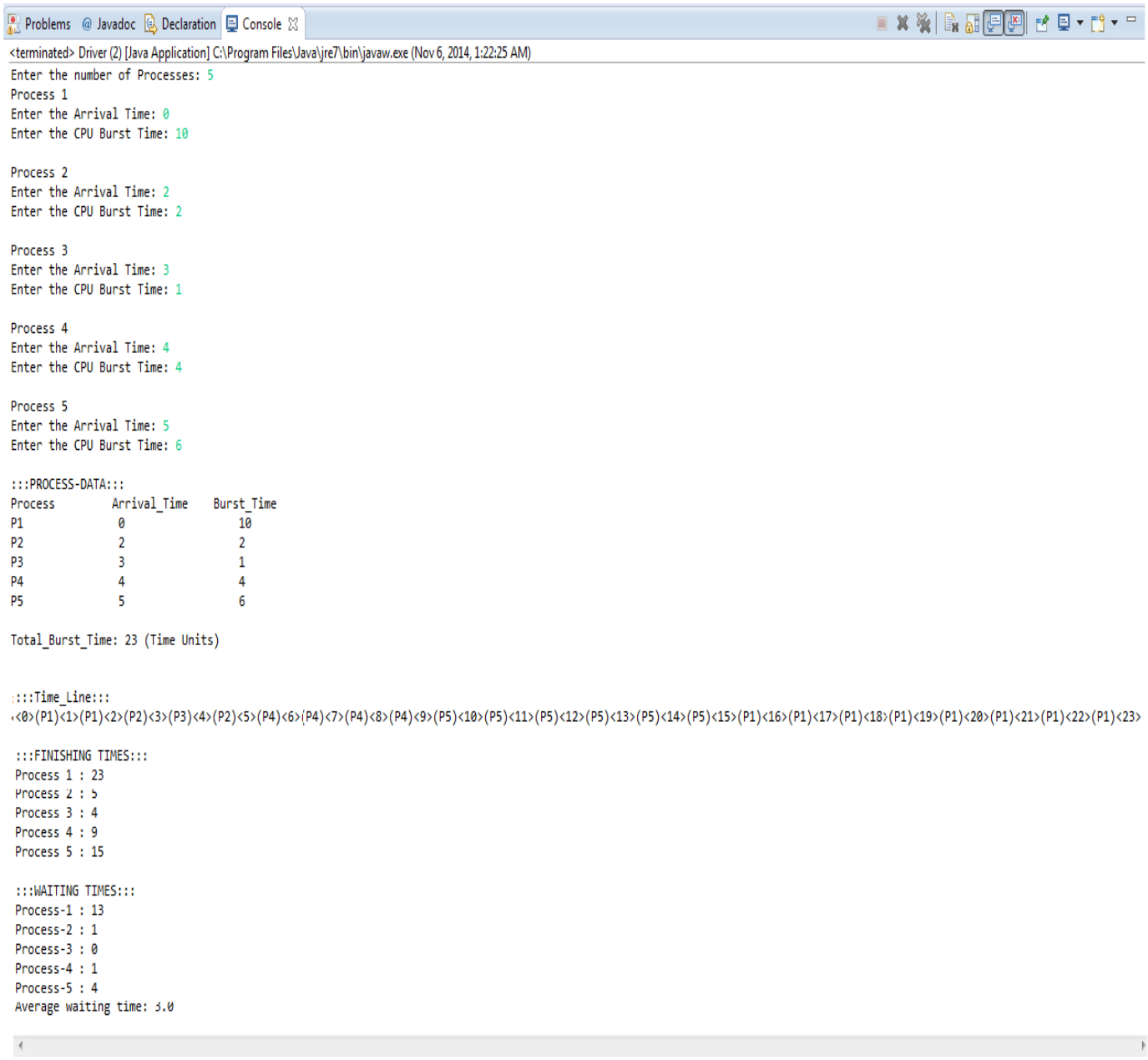


Problems Javadoc Declaration Console

<terminated> Driver (2) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 6, 2014, 1:17:25 AM)

```
Enter the number of Processes: 4
Process 1
Enter the Arrival Time: 0
Enter the CPU Burst Time: 10

Process 2
Enter the Arrival Time: 1
Enter the CPU Burst Time: 2

Process 3
Enter the Arrival Time: 2
Enter the CPU Burst Time: 3

Process 4
Enter the Arrival Time: 3
Enter the CPU Burst Time: 4

:::PROCESS-DATA:::
Process         Arrival_Time    Burst_Time
P1               0                  10
P2               1                  2
P3               2                  3
P4               3                  4

Total_Burst_Time: 19 (Time Units)


:::Time_Line:::
<0>(P1)<1>(P2)<2>(P2)<3>(P3)<4>(P3)<5>(P3)<6>(P4)<7>(P4)<8>(P4)<9>(P4)<10>(P1)<11>(P1)<12>(P1)<13>(P1)<14>(P1)<15>(P1)<16>(P1)<17>(P1)<18>(P1)<19>

:::FINISHING TIMES:::
Process 1 : 19
Process 2 : 3
Process 3 : 6
Process 4 : 10

:::WAITING TIMES:::
Process-1 : 9
Process-2 : 0
Process-3 : 1
Process-4 : 3
Average waiting time: 3.0
```

Sample-5



References:

- http://programersparadise.tk/c-program-for-shortest-remaining-time-next-srtn/