Scheduling Algorithms
Thompson Rivers University
COMP 3410 – Operating Systems
Uail Zhukenov
T00645104

Description:

The following code represents an implementation of four scheduling algorithms, both pre-emptive and non-pre-emptive. The four scheduling algorithms are First Come First Serve, Shortest Job First, Priority Scheduling and Round-Robin. The program is asking for general input such as:

1. number of processes
2. arrival time
3. burst time

After which some particular input may be asked, such as priority or time quantum, based on which algorithm is being chosen.

The output that is being produced is Table, which contains calculations of completion, turnaround, waiting time for each process. Also, the chart representation is presents, as well as average waiting time.

Note:

Shortest Job First does not support pre-emption.
Priority Scheduling does not support pre-emption.

Code in Java:

```
/* Name: Uail Zhukenov
Id: T00645104 */

import java.util.*;

public class Processes {


    public static void main(String[] args) {

        int number_processes = 0;

        Scanner scan = new Scanner(System.in); // Create a Scanner object
        System.out.println("Enter number of processes: ");
        number_processes = scan.nextInt();

        double[] burst_time = new double[number_processes];
        double[] arrival_time = new double[number_processes];
        int position[] = new int[number_processes];
        int priority[] = new int[number_processes];


        System.out.println("Enter arrival time: ");
        for (int i = 0; i < number_processes; i++) {
            arrival_time[i] = scan.nextDouble();
            System.out.println("Arrival time for process " + i + " is: " + arrival_time[i]);
            position[i] = i;

        }

        System.out.println("Enter burst time: ");
        for (int i = 0; i < number_processes; i++) {
            burst_time[i] = scan.nextDouble();
            System.out.println("Burst time for process " + i + " is: " + burst_time[i]);
        }

        System.out.println("Which algorithm to use? \nEnter 1 for FCFS, 2 for SJF, 3 for Priority, 4
for Round Robin: ");
        int option = 0;
        option = scan.nextInt();

        if (option == 1) {
            System.out.println("You chose: " + option + " which is First Come First Serve");
            sortProcessesBasedOnArrivalTime(number_processes, arrival_time, burst_time,
position);
            firstComefirstServe(arrival_time, burst_time, number_processes, position);

        }
        else if (option == 2) {
            System.out.println("You chose: " + option + " which is SJF");
            sortProcessesBasedOnArrivalTime(number_processes, arrival_time, burst_time,
position);
            SJF(arrival_time, burst_time, number_processes, position);
        }
        else if (option == 3) {

            System.out.println("You chose: " + option + " Priority Scheduling");
            System.out.println("\n Please enter priority: ");
            for (int i=0; i < number_processes; i++)
            {
                priority[i] = scan.nextInt();
            }

            sortProcesses(number_processes, arrival_time, burst_time, position, priority);
            priority(arrival_time, burst_time, number_processes, position, priority);
        }
        else if (option == 4) {
            System.out.println("You chose " + option + " which is Round Robin");
            System.out.println("\n Please enter time quantum: ");
            int timeQuantum = scan.nextInt();

            roundRobin(arrival_time, burst_time, number_processes, position, timeQuantum);

        }
        else
            System.out.println("\n The number you put is not correct, TRY AGAIN");


    }


    public static void sortProcessesBasedOnArrivalTime(int number_processes, double[]
arrival_time, double[] burst_time, int[] position) {



        for (int i = 0; i < number_processes-1; i++) {
            for (int j = 0; j < number_processes-i-1; j++) {
                if (arrival_time[j] > arrival_time[j+1] || (arrival_time[j] == arrival_time[j+1] &&
burst_time[j] > burst_time[j+1])) {
                    // Swap processes
                    double temp = arrival_time[j];
                    arrival_time[j] = arrival_time[j+1];
                    arrival_time[j+1] = temp;

                    double temp1 = burst_time[j];
                    burst_time[j] = burst_time[j+1];
                    burst_time[j+1] = temp1;

                    int temp2 = position[j];
                    position[j] = position[j+1];
                    position[j+1] = temp2;
                }
            }
        }
```

```
  }


  public static void sortProcesses(int number_processes, double[] arrival_time, double[]
burst_time, int[] position, int[] priority) {

    // Sort processes based on priority and arrival times
    for (int i = 0; i < number_processes-1; i++) {
      for (int j = 0; j < number_processes-i-1; j++) {
        if (priority[j] > priority[j+1] || (priority[j] == priority[j+1] && arrival_time[j] >
arrival_time[j+1]) || (priority[j] == priority[j+1] && arrival_time[j] == arrival_time[j+1])) {
          // Swap processes
          int temp = priority[j];
          priority[j] = priority[j+1];
          priority[j+1] = temp;

          double temp1 = arrival_time[j];
          arrival_time[j] = arrival_time[j+1];
          arrival_time[j+1] = temp1;

          double temp2 = burst_time[j];
          burst_time[j] = burst_time[j+1];
          burst_time[j+1] = temp2;

          int temp3 = position[j];
          position[j] = position[j+1];
          position[j+1] = temp3;
        }
      }
    }

  }



    public static void firstComefirstServe(double[] arrival_time, double[] burst_time, int
number_processes, int[] position) {

    double completion_time[] = new double[number_processes];
    double waiting_time[] = new double[number_processes];
    double total_waiting_time = 0;


    completion_time[0] = arrival_time[0] + burst_time[0];
    for (int i = 1; i < number_processes; i++) {
      if (completion_time[i - 1] <= arrival_time[i]) {
```

```
                completion_time[i] = arrival_time[i] + burst_time[i];
            } else {
                completion_time[i] = completion_time[i - 1] + burst_time[i];
            }
        }
        double[] turnaround_time = new double[number_processes];

        for (int i = 0; i < number_processes; i++) {
            waiting_time[i] = completion_time[i] - arrival_time[i] - burst_time[i];
            total_waiting_time += waiting_time[i];
            turnaround_time[i] = completion_time[i] - arrival_time[i];
        }


        double average_waiting_time = total_waiting_time / number_processes;


        //gantt chart implementation
        System.out.println("\n\nProcess execution sequence:");
        for (int i = 0; i < number_processes; i++) {
            if (i==0)
            {
                System.out.print("P" + (position[i]) + " started at " + arrival_time[i] + " completed at "
+ completion_time[i] + "\n");
            }
            else if (i == number_processes - 1) {
                System.out.print("P" + (position[i]) + " started at " +
completion_time[number_processes - 2] + " completed at " + completion_time[i] + "\n");
            }
            else {
                System.out.print("P" + (position[i]) + " started at " + completion_time[i-1] + "
completed at " + completion_time[i] + "\n");
            }
        }
        System.out.println();


        System.out.println("Process\t\tArrival Time\t\tBurst Time\t\tCompletion Time\t\tWaiting
Time\t\tTurnaround Time");
        for (int i = 0; i < number_processes; i++) {
            System.out.println(position[i] + "\t\t\t" + arrival_time[i] + "\t\t\t\t\t\t" + burst_time[i] +
"\t\t\t\t\t"
                    + completion_time[i] + "\t\t\t\t" + waiting_time[i] + "\t\t\t\t" + turnaround_time[i]);
        }

        System.out.println("\nAverage waiting time: " + average_waiting_time);
```

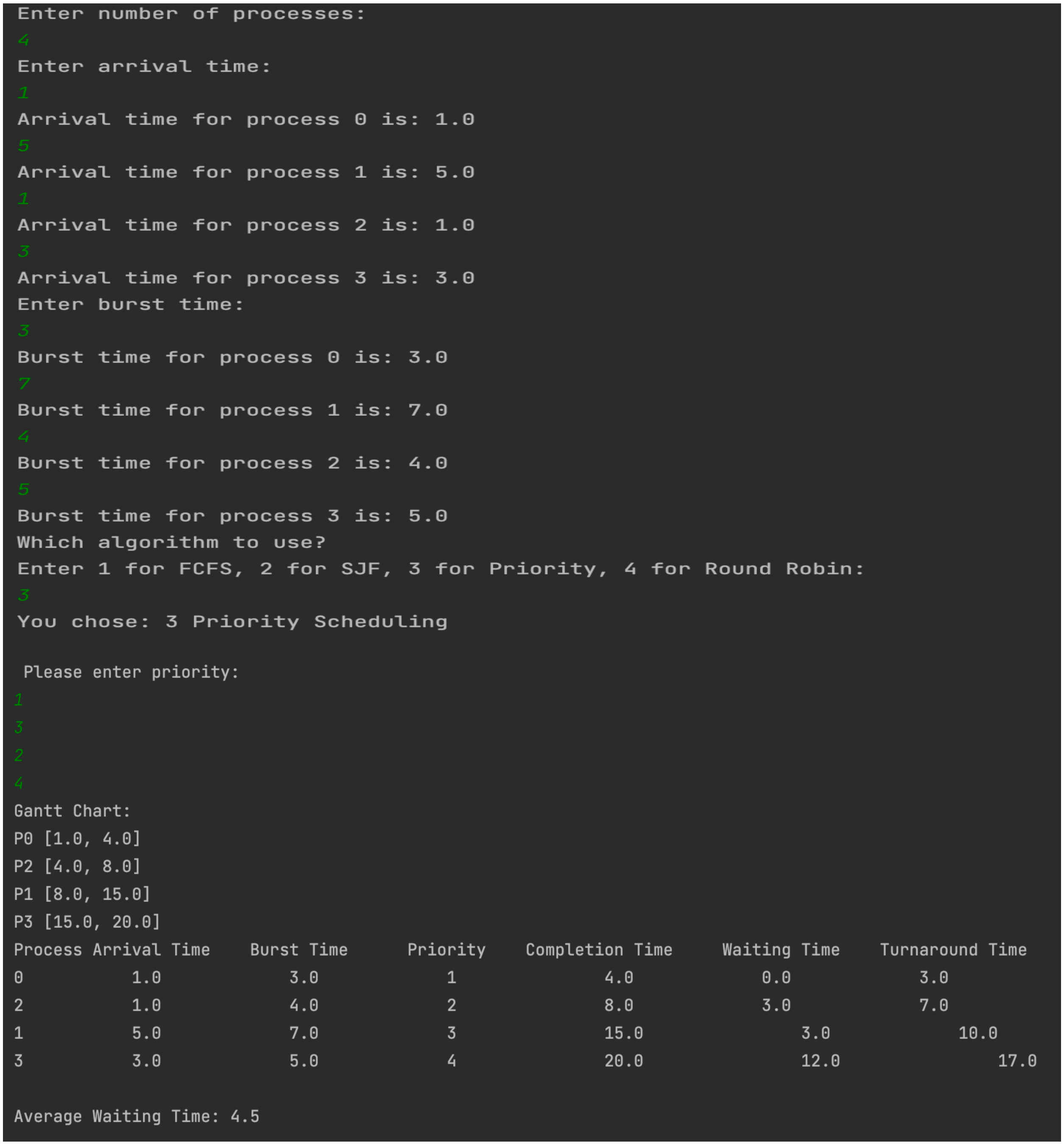
```
    }


    public static void SJF(double[] arrival_time, double[] burst_time, int number_processes, int[]
position) {


        int completed_processes = 0; //number of processes completed
        double current_time = 0.0; //keeps track of current time
        double completion_time[] = new double[number_processes];
        boolean process_completed[] = new boolean[number_processes];
        double remaining_burst_time[] = new double[number_processes]; //keeps track of
remaining burst for preempted processes
        int running_process = -1;
        double turnaround_time[] = new double[number_processes];
        double waiting_time[] = new double[number_processes];

        for (int i = 0; i < number_processes; i++) {
            remaining_burst_time[i] = burst_time[i];
        }

        while (completed_processes < number_processes) { //executes until all processes completed
            int process = -1;
            double min_burst_time = Double.MAX_VALUE;
            for (int i = 0; i < number_processes; i++) {
                if (!process_completed[i] && remaining_burst_time[i] < min_burst_time &&
arrival_time[i] <= current_time) {
                    process = i;
                    min_burst_time = remaining_burst_time[i];
                }



            }

            if (process == -1) {
                // No process is ready to run yet, move time forward to the earliest arrival time of un-
completed processes
                double earliest_arrival_time = Double.MAX_VALUE;
                for (int i = 0; i < number_processes; i++) {
                    if (!process_completed[i] && arrival_time[i] < earliest_arrival_time) {
                        earliest_arrival_time = arrival_time[i];
                    }
                }
                current_time = earliest_arrival_time;
                continue;
            }

            if (running_process == -1 || remaining_burst_time[process] <
remaining_burst_time[running_process]) {
                // A new process with a shorter burst time has arrived
                if (running_process != -1) {
                    // Save the remaining burst time of the previous process
                    remaining_burst_time[running_process] -= (current_time -
arrival_time[running_process]);
                }

                running_process = process;
                current_time = arrival_time[process];
            }

            current_time += remaining_burst_time[process];
            completion_time[process] = current_time;
            process_completed[process] = true; //process is done
            completed_processes++; //controls while loop
            remaining_burst_time[process] = 0;

            waiting_time[process] = completion_time[process] - arrival_time[process] -
burst_time[process];
            turnaround_time[process] = completion_time[process] - arrival_time[process];
        }



        //gantt chart implementation
        System.out.println("\n\nProcess execution sequence:");
        for (int i = 0; i < number_processes; i++) {
            if (i==0)
            {
                System.out.print("P" + (position[i]) + " started at " + arrival_time[i] + " completed at "
+ completion_time[i] + "\n");
            }
            else if (i == number_processes - 1) {
                System.out.print("P" + (position[i]) + " started at " +
completion_time[number_processes - 2] + " completed at " + completion_time[i] + "\n");
            }
            else {
                System.out.print("P" + (position[i]) + " started at " + completion_time[i-1] + "
completed at " + completion_time[i] + "\n");
            }
        }
        System.out.println();



        // Display Table
        System.out.println("Process\tArrival Time\tBurst Time\tCompletion Time\t\tWaiting
Time\tTurnaround Time");
        for (int i = 0; i < number_processes; i++) {
            System.out.println(position[i] + "\t\t\t" + arrival_time[i] + "\t\t\t\t" + burst_time[i] +
"\t\t\t" + completion_time[i] + "\t\t\t\t" + waiting_time[i] + "\t\t\t\t" + turnaround_time[i]);
        }

        double avg_wait = 0.0;
        for (int i=0; i<number_processes; i++)
        {
            avg_wait += waiting_time[i];
        }

        avg_wait = avg_wait/number_processes;

        System.out.println("\n Average Waiting Time: " + avg_wait);


    }

    public static void priority(double[] arrival_time, double[] burst_time, int number_processes,
int[] position, int[] priority) {

        int completed_processes = 0;
        double current_time = 0.0; //keeps track of current time
        boolean process_completed[] = new boolean[number_processes]; //contains boolean if
process is done or not
        double remaining_burst_time[] = new double[number_processes]; //keeps track of
remaining burst time
        double turnaround_time[] = new double[number_processes];
        double waiting_time[] = new double[number_processes];
        double completion_time[] = new double[number_processes];


        for (int i = 0; i < number_processes; i++) {
            remaining_burst_time[i] = burst_time[i];
        }

        List<String> chart = new ArrayList<>(); // array list for gantt chart
        //keeps going until all processes are done
        while (completed_processes < number_processes) {
            int selected_process = -1;
            for (int i = 0; i < number_processes; i++) {
                if (!process_completed[i] && arrival_time[i] <= current_time) {
                    if (selected_process == -1) {
                        selected_process = i;
                    } else if (priority[i] < priority[selected_process]) {
                        selected_process = i;
                    }
                }
            }

            // If there are no processes with an arrival time less than or equal to the current time,
            // then we set the current time to the earliest arrival time among the processes.
            if (selected_process == -1) {
                double min_arrival_time = Double.MAX_VALUE;
                for (int i = 0; i < number_processes; i++) {
                    if (!process_completed[i] && arrival_time[i] < min_arrival_time) {
                        min_arrival_time = arrival_time[i];
                    }
                }
                current_time = min_arrival_time;
            } else {
                process_completed[selected_process] = true; //process is completed
                chart.add("P" + position[selected_process] + " [" + current_time + ", " + (current_time
+ remaining_burst_time[selected_process]) + "]");
                current_time += remaining_burst_time[selected_process];
                completion_time[selected_process] = current_time;
                waiting_time[selected_process] = completion_time[selected_process] -
arrival_time[selected_process] - burst_time[selected_process];
                turnaround_time[selected_process] = completion_time[selected_process] -
arrival_time[selected_process];
                completed_processes++; //increments number of completed processes to control while
loop
            }
        }



        System.out.println("Gantt Chart: ");
        for (String process : chart) {
            System.out.println(process);
        }



        // Display results
        System.out.println("Process\tArrival Time\tBurst Time\t\tPriority\tCompletion
Time\t\tWaiting Time\tTurnaround Time");
        for (int i = 0; i < number_processes; i++) {
            System.out.println(position[i] + "\t\t\t" + arrival_time[i] + "\t\t\t\t" + burst_time[i] +
"\t\t\t\t" + priority[i] + "\t\t\t\t" + completion_time[i] + "\t\t\t\t" + waiting_time[i] + "\t\t\t\t" +
turnaround_time[i]);
        }

        double avg_wait = 0.0;

        for(int i=0; i < number_processes ;i++)
        {
            avg_wait += waiting_time[i];
        }

        avg_wait = avg_wait/number_processes;

        System.out.println("\nAverage Waiting Time: " + avg_wait );


    }

    public static void roundRobin(double[] arrival_time, double[] burst_time, int
number_processes, int[] position, int timeQuantum) {

        double current_time = 0;
        double sum_waiting_time = 0;

        int completed_processes = 0;
       double remaining_time[] = new double[number_processes];
        double waiting_time[] = new double[number_processes];
        double turnaround_time[] = new double[number_processes];

        for (int i=0; i<number_processes; i++) {
            remaining_time[i] = burst_time[i];
        }

        // Create an array to store the completion time for each process
        double[] completion_time = new double[number_processes];
        // Create an ArrayList to store the execution sequence of processes
        List<Integer> execution_sequence = new ArrayList<Integer>();

        // Run the Round Robin algorithm
        while (completed_processes < number_processes) {
            boolean process_completed = true;
            for (int i = 0; i < number_processes; i++) {
```

```
        if (remaining_time[i] > 0 && arrival_time[i] <= current_time) {
            process_completed = false;
            execution_sequence.add(i);
            if (remaining_time[i] > timeQuantum) {
                remaining_time[i] -= timeQuantum;
                current_time += timeQuantum;
            } else {
                current_time += remaining_time[i];
                waiting_time[i] = current_time - arrival_time[i] - burst_time[i];
                remaining_time[i] = 0;
                completed_processes++;
                turnaround_time[i] = current_time - arrival_time[i];
                sum_waiting_time += waiting_time[i];

                completion_time[i] = current_time;
            }
        }
    }
    if (process_completed) {
        current_time++;
    }
}

// Calculate and print the average waiting time
double avg_waiting_time = sum_waiting_time / number_processes;
System.out.println("Average Waiting Time: " + avg_waiting_time);



// Display Chart
System.out.print("Gain Chart: ");
int prev_process_id = -1;
double current_execution_time = 0;
for (int i = 0; i < execution_sequence.size(); i++) {
    int process_id = execution_sequence.get(i);
    if (process_id != prev_process_id) {
        if (prev_process_id != -1) {
            System.out.print(String.format("%.1f", current_execution_time) + " ");
        }
        System.out.print("P" + process_id + " executed for ");
        prev_process_id = process_id;
        current_execution_time = 0;
    }
    current_execution_time += timeQuantum;
}
System.out.print(String.format("%.1f", current_execution_time));
```

```
        System.out.println("\n");

        // Display results
        System.out.println("Process\tArrival Time\tBurst Time\tCompletion Time\t\tWaiting
Time\tTurnaround Time");
        for (int i = 0; i < number_processes; i++) {
            System.out.println(position[i] + "\t\t\t" + arrival_time[i] + "\t\t\t\t" + burst_time[i] +
"\t\t\t" + completion_time[i] + "\t\t\t\t" + waiting_time[i] + "\t\t\t\t" + turnaround_time[i]);
        }


    }

    }
```

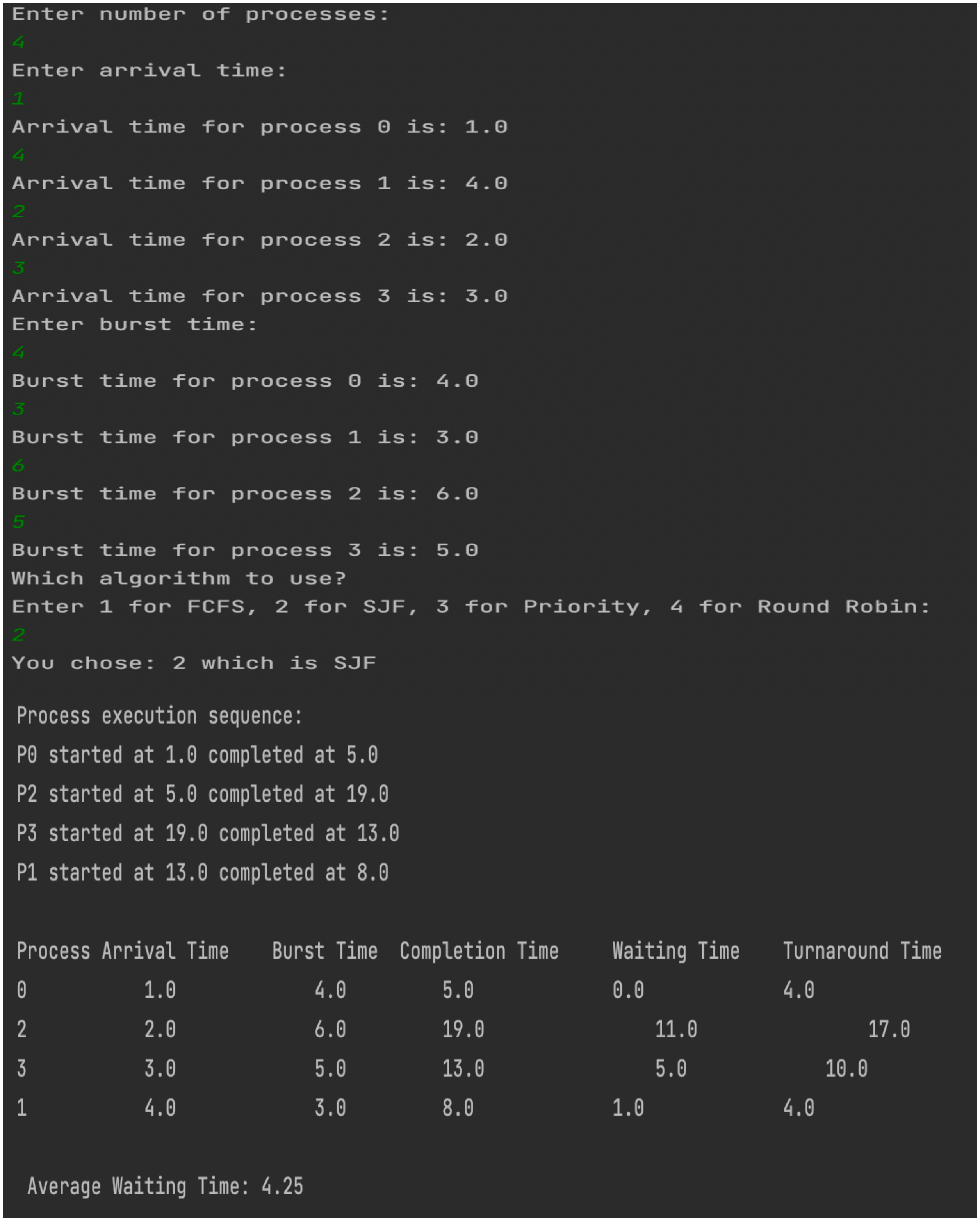
OUTPUT:

1. Priority Scheduling



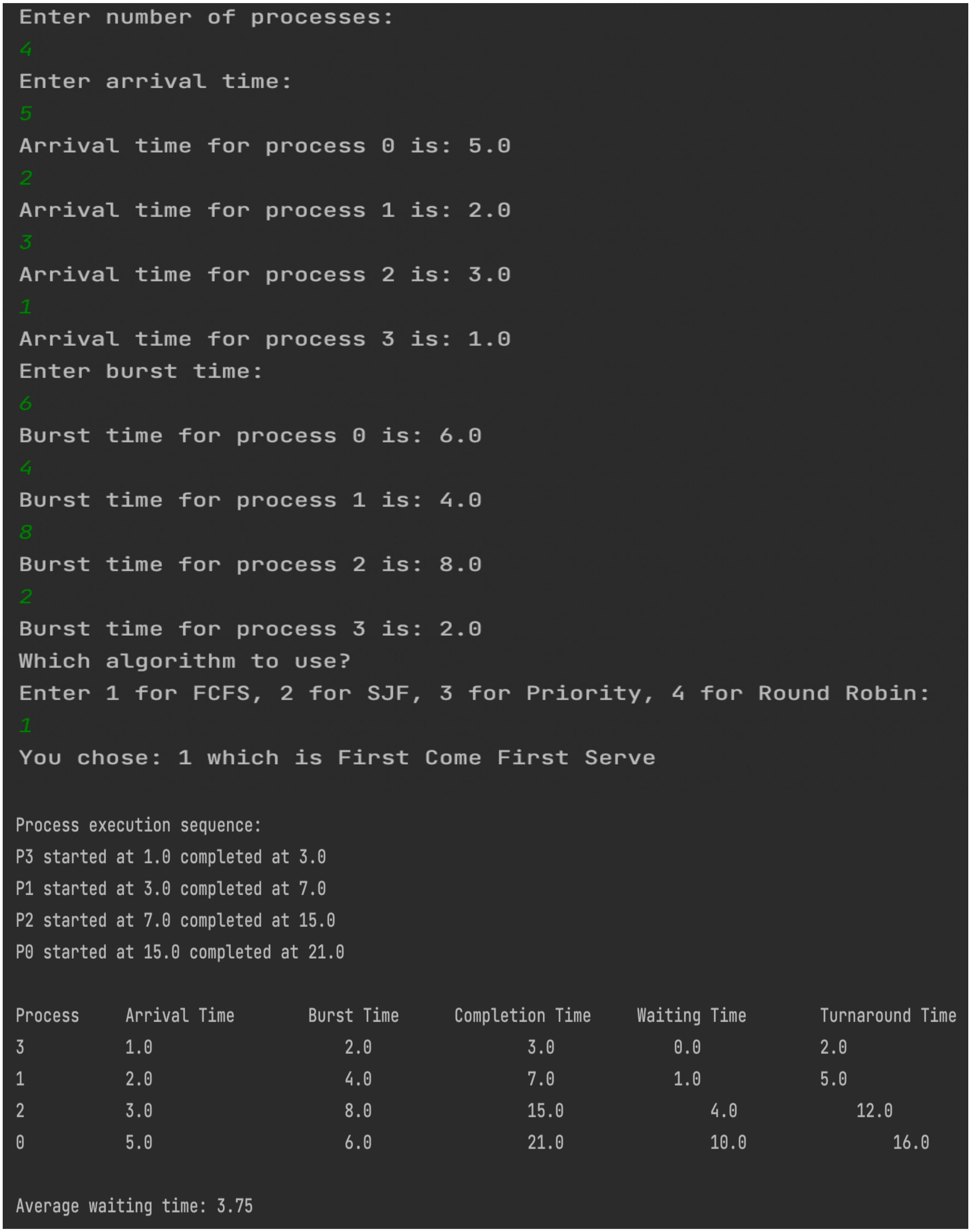
```
Enter number of processes:
4
Enter arrival time:
1
Arrival time for process 0 is: 1.0
5
Arrival time for process 1 is: 5.0
1
Arrival time for process 2 is: 1.0
3
Arrival time for process 3 is: 3.0
Enter burst time:
3
Burst time for process 0 is: 3.0
7
Burst time for process 1 is: 7.0
4
Burst time for process 2 is: 4.0
5
Burst time for process 3 is: 5.0
Which algorithm to use?
Enter 1 for FCFS, 2 for SJF, 3 for Priority, 4 for Round Robin:
3
You chose: 3 Priority Scheduling

 Please enter priority:
1
3
2
4
Gantt Chart:
P0 [1.0, 4.0]
P2 [4.0, 8.0]
P1 [8.0, 15.0]
P3 [15.0, 20.0]
Process Arrival Time    Burst Time      Priority    Completion Time     Waiting Time    Turnaround Time
0           1.0             3.0             1               4.0             0.0             3.0
2           1.0             4.0             2               8.0             3.0             7.0
1           5.0             7.0             3               15.0                3.0             10.0
3           3.0             5.0             4               20.0                12.0                17.0

Average Waiting Time: 4.5
```

2. Shortest Job First



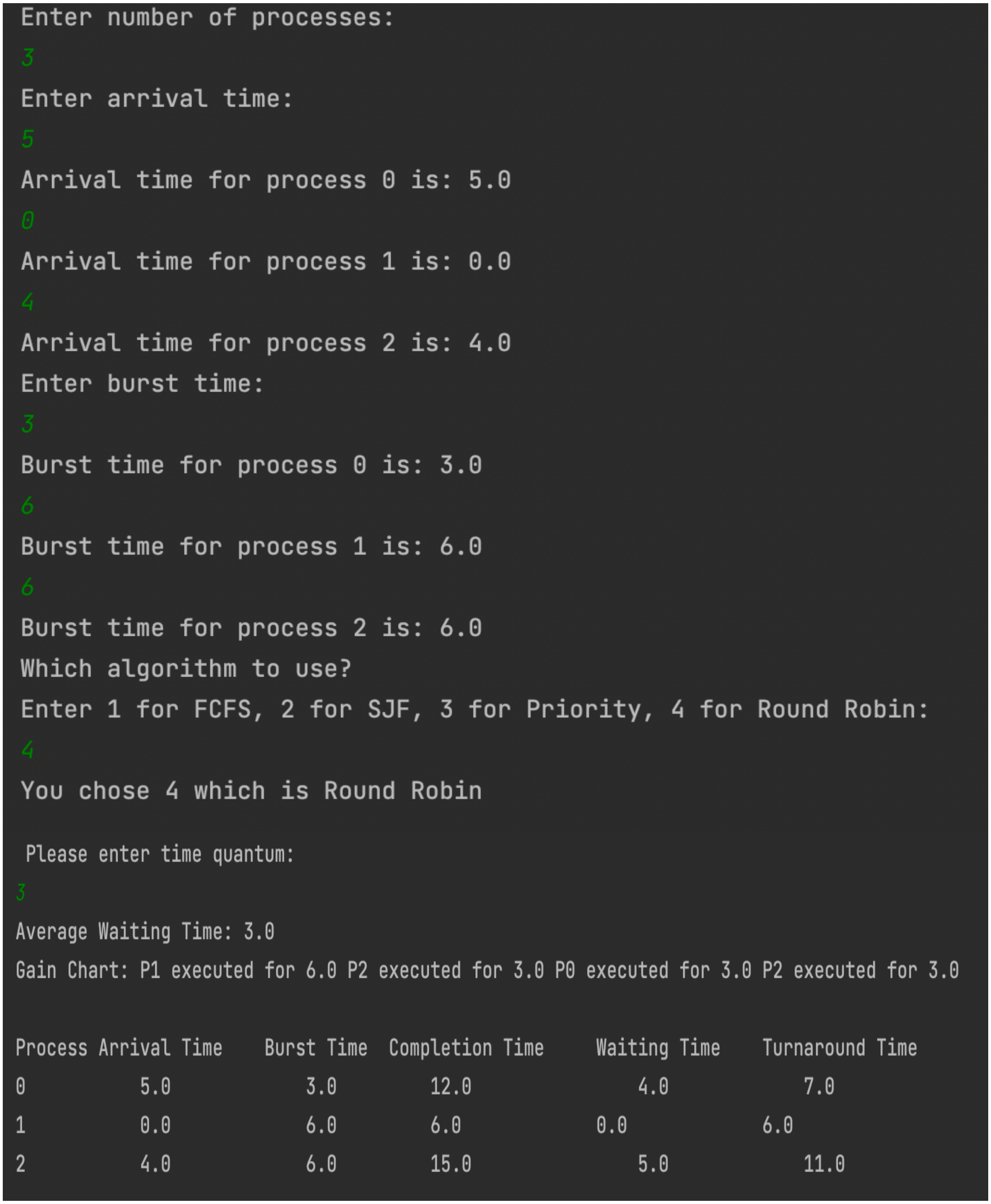
```
Enter number of processes:
4
Enter arrival time:
1
Arrival time for process 0 is: 1.0
4
Arrival time for process 1 is: 4.0
2
Arrival time for process 2 is: 2.0
3
Arrival time for process 3 is: 3.0
Enter burst time:
4
Burst time for process 0 is: 4.0
3
Burst time for process 1 is: 3.0
6
Burst time for process 2 is: 6.0
5
Burst time for process 3 is: 5.0
Which algorithm to use?
Enter 1 for FCFS, 2 for SJF, 3 for Priority, 4 for Round Robin:
2
You chose: 2 which is SJF

Process execution sequence:
P0 started at 1.0 completed at 5.0
P2 started at 5.0 completed at 19.0
P3 started at 19.0 completed at 13.0
P1 started at 13.0 completed at 8.0


Process Arrival Time    Burst Time  Completion Time     Waiting Time    Turnaround Time
0           1.0             4.0         5.0             0.0             4.0
2           2.0             6.0         19.0                11.0                17.0
3           3.0             5.0         13.0                5.0             10.0
1           4.0             3.0         8.0             1.0             4.0


 Average Waiting Time: 4.25
```

3. First Come First Serve



```
Enter number of processes:
4
Enter arrival time:
5
Arrival time for process 0 is: 5.0
2
Arrival time for process 1 is: 2.0
3
Arrival time for process 2 is: 3.0
1
Arrival time for process 3 is: 1.0
Enter burst time:
6
Burst time for process 0 is: 6.0
4
Burst time for process 1 is: 4.0
8
Burst time for process 2 is: 8.0
2
Burst time for process 3 is: 2.0
Which algorithm to use?
Enter 1 for FCFS, 2 for SJF, 3 for Priority, 4 for Round Robin:
1
You chose: 1 which is First Come First Serve

Process execution sequence:
P3 started at 1.0 completed at 3.0
P1 started at 3.0 completed at 7.0
P2 started at 7.0 completed at 15.0
P0 started at 15.0 completed at 21.0

Process     Arrival Time        Burst Time      Completion Time     Waiting Time        Turnaround Time
3           1.0                     2.0                 3.0             0.0             2.0
1           2.0                     4.0                 7.0             1.0             5.0
2           3.0                     8.0                 15.0                4.0             12.0
0           5.0                     6.0                 21.0                10.0                16.0

Average waiting time: 3.75
```

4. Round Robin



```
Enter number of processes:
3
Enter arrival time:
5
Arrival time for process 0 is: 5.0
0
Arrival time for process 1 is: 0.0
4
Arrival time for process 2 is: 4.0
Enter burst time:
3
Burst time for process 0 is: 3.0
6
Burst time for process 1 is: 6.0
6
Burst time for process 2 is: 6.0
Which algorithm to use?
Enter 1 for FCFS, 2 for SJF, 3 for Priority, 4 for Round Robin:
4
You chose 4 which is Round Robin

 Please enter time quantum:
3
Average Waiting Time: 3.0
Gain Chart: P1 executed for 6.0 P2 executed for 3.0 P0 executed for 3.0 P2 executed for 3.0


Process Arrival Time    Burst Time  Completion Time     Waiting Time    Turnaround Time
0           5.0            3.0        12.0               4.0             7.0
1           0.0            6.0        6.0            0.0             6.0
2           4.0            6.0        15.0               5.0             11.0
```