Software Engineering Project Report

CPU Process Scheduling Algorithms Simulator

Iman Aslam
Nabeeha Shahid
Barikah Shafqat
Muhammad Abdullah Wasi

Submitted to

Prof. Dr. Farooq-e-Azam

December 31, 2022

Contents

List of Figures

List of Codes

Chapter 1

Project Overview

Our software is a CPU Process Scheduling Algorithms Simulator, hereinafter referred to as CPSA-Sim. It has been designed as an educational application serving both, a functional and an academic purpose in order to learn more about scheduling algorithms, such as FCFS or SJF.

1.1 Tools

The software has been designed completely in Windows Forms Applications, or WFA, using C# and .NET Framework keeping in mind the latest guidelines and rules of software engineering. The GUI has also been made in the same software.

1.2 Goals

Goals of the CPSA-Sim include but are not limited to:

- To provide users of the application a fully function algorithm simulator.
- Design the interface of the application to be as efficient, functional yet modern in design as possible.
- Provide an easy and practical approach while using the simulator, for users with experience of all degrees.

Chapter 2

Interface Design

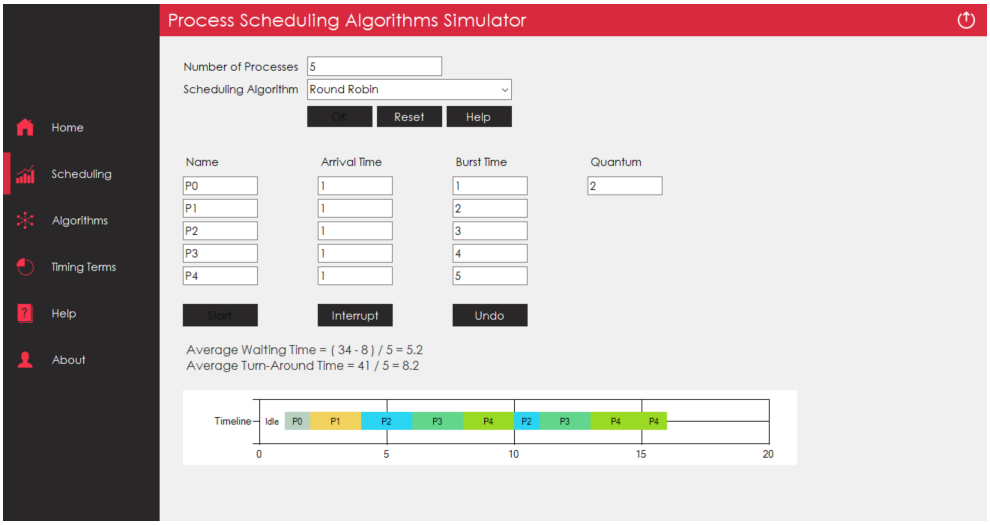

Figure 2.1: UI: The Round Robin Algorithm Screen

Designed completely from scratch, the UI of the CPA-Sim is meant to be as utilitarian as possible while still keeping modernity and minimalism in mind. The homepage is the default start page of the application, as the name suggests. The algorithms and tinning pages serve an educational purpose, as a means to quickly refer to what a certain term means.

The main part of the simulator, however, lies in the Scheduling tab. More on this is explained further below.

Chapter 3

The Application

Following are some visuals depicting the application in use.

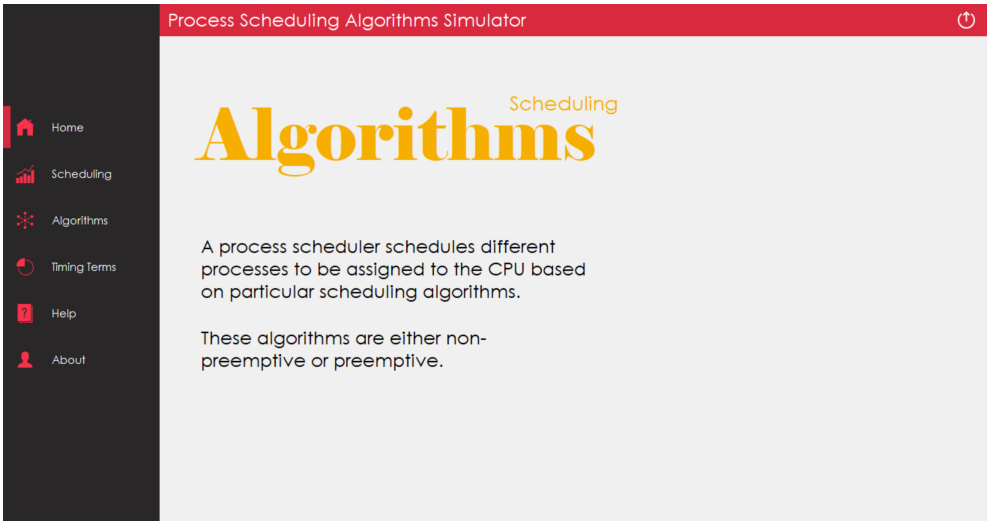

Figure 3.1: The Homepage

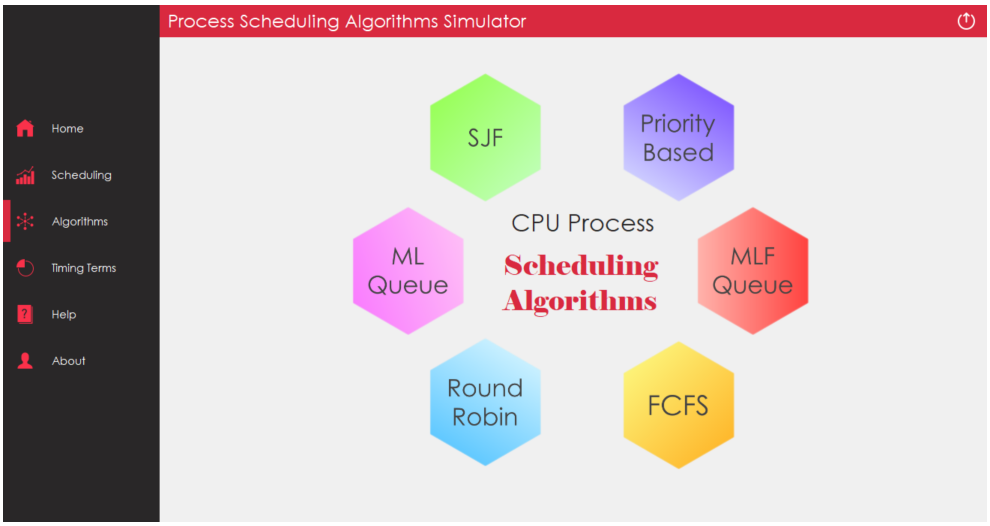

Figure 3.2: Algorithms Tab

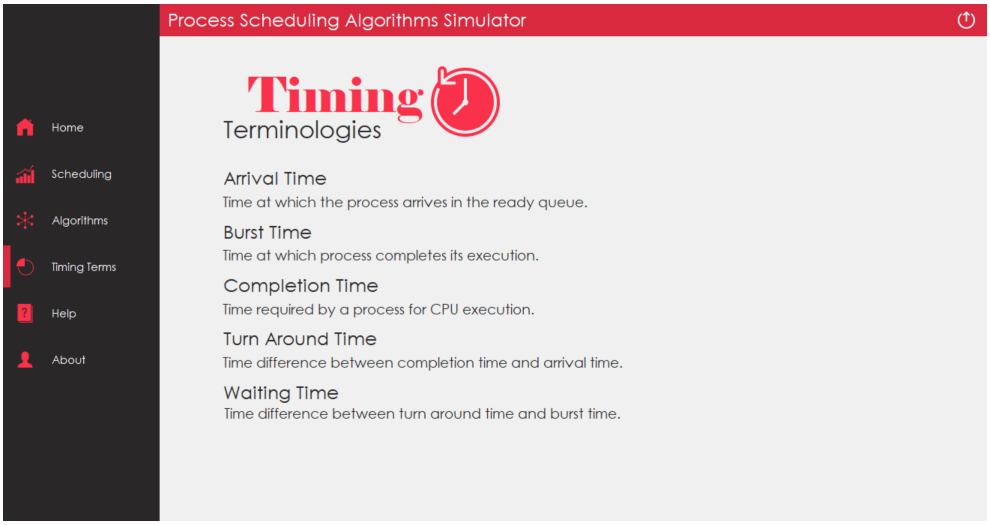

Figure 3.3: Timing Terms Tab

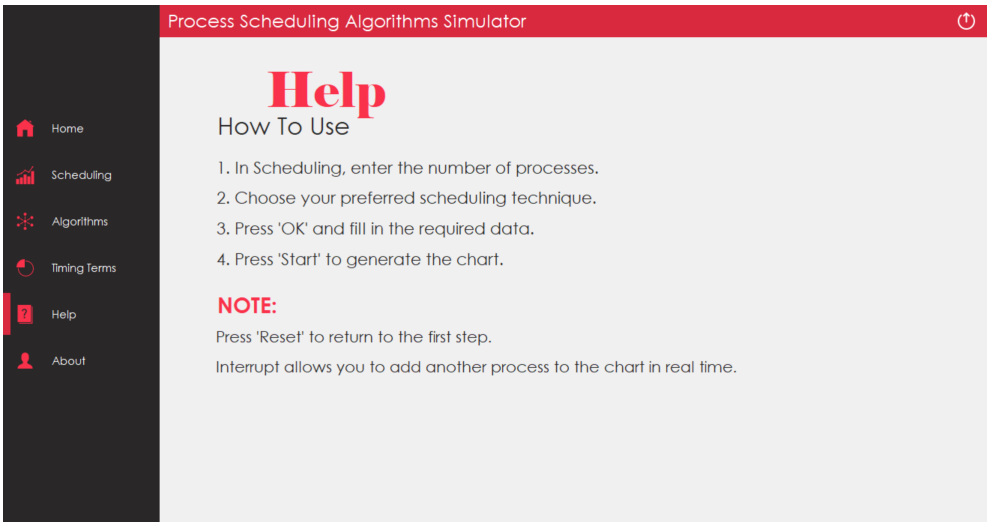

Figure 3.4: Help Tab

Chapter 4

The Simulator

4.1 Class Variables

The Scheduling part of the CPSA-Sim has been achieved using an array of functional commands and looping structures. We declare all necessary variables and structures for each of the scheduling algorithms as class variables.

```
1     int numberOfProcesses;
2     int counter;
3     string type;
4     double q;
5     Button btn_st;
6     TextBox quantum;
7     TextBox interrupt;
8     TextBox[] names;
9     TextBox[] arrivals;
10    TextBox[] bursts;
11    TextBox[] priorities;
12    TextBox newname;
13    TextBox newburst;
14    TextBox newpriority;
15    Chart timeline;
16    Label waiting;
```

Code 4.1: Class Variables

4.2 Try Catches

Before we move further into discussing how the algorithms have been made to work, it is important to note the program catches exceptions such as negative quantum and priority data and displays it to the user. It does this like so:

```
    try
    {
        if (type == "Round Robin")
        {
            q = Convert.ToDouble(quantum.Text);
            if (q <= 0)
                throw new System.Exception("Quantum must be
positive integer");
        }
        fillScheduler();
        scheduleCPU();
        btn_st.Enabled = false;
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
```

Code 4.2: Try Catch Sequence Example

The program first seeks if the "type" variable, referring to the user's selected algorithm sequence, is Round Robin. If so, it then checks the value in the text box meant for the user to put the quantum value in and converts it into a double. Next, if this converted number is less than or equal to zero, the program stop running the algorithm and throws an exception.

4.3 Pre-Processing

In this section, we'll define how we've made the program ready to execute based on the user's choice. Right after the user selects an algorithm, fills in the required details and clicks the "Start" button, the program calculates a number of different things, with the important ones being the earliest arrival time (which is used to define the idle stage), the average burst time (used to create the timeline in the Gantt chart) and the last arrival time. This is shown below:

```
    createWaiting();
    createTimeline();
    Scheduler temp = new Scheduler();
    ...
    Series idle = new Series();
    createSeries(idle, i);
```

Code 4.3: Pre-Processing

4.4 FCFS

The First-Come First-Served algorithm works in a simple manner: whichever process arrives first will be executed first. In other words, it processes and executes queued process requests based on their order of arrival, and is one of the easiest and simplest scheduling algorithms. As shown below, once

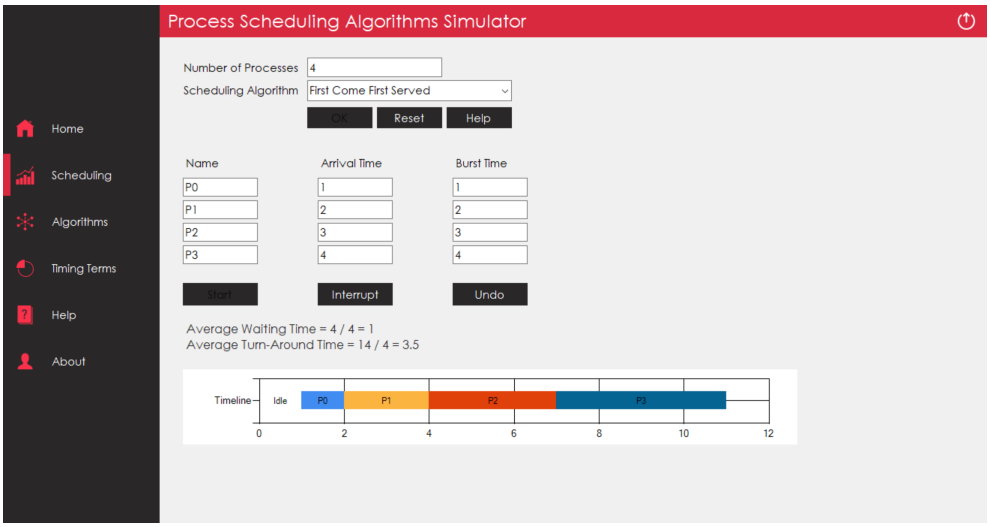

Figure 4.1: First-Come First-Served Algorithm

the FCFS algorithm is selected and the user presses the "OK" button, the program displays a number of text boxes based on the number of processes the user selected (at the very top). This is done by adding label, text box and button creation commands to the "OK" button's on-click event. It works somewhat like so:

```
1     // Label
2     this.Controls.Add(l1);
3     l1.Top = 200;
4     l1.Left = 240;
5     l1.Text = "Name";
6     l1.Font = new Font("Century Gothic", 11);
7
8     // Text Box
9     System.Windows.Forms.TextBox txt1 = new System.Windows.Forms
      .TextBox();
10    names[i] = txt1;
11    this.Controls.Add(txt1);
12    txt1.Top = (i * 30) + 230;
13    txt1.Left = 240;
14    txt1.Text = "P" + (i).ToString();
```

```
15      txt1.Font = new Font("Century Gothic", 11);
16
17      // Button
18      System.Windows.Forms.Button btn = new System.Windows.Forms.
        Button();
19      btn.Name = "btn_start";
20      btn.Font = new Font("Century Gothic", 11, System.Drawing.
        FontStyle.Regular);
21      btn.ForeColor = Color.White;
22      btn.BackColor = Color.FromArgb(41, 39, 40);
23      btn.FlatAppearance.BorderSize = 0;
24      btn.FlatStyle = FlatStyle.Flat;
```

Code 4.4: Creating Labels, Text Boxes and Buttons

4.5 SJF

Next up, we have the Shortest Job First algorithm, which as its name suggests, takes the process with the shortest execution time is chosen to be executed first. In the example below, the process P0 can be seen to arrive together with all other processes, at 1 time unit, but it is last in the Gantt chart since it has the longest burst time.

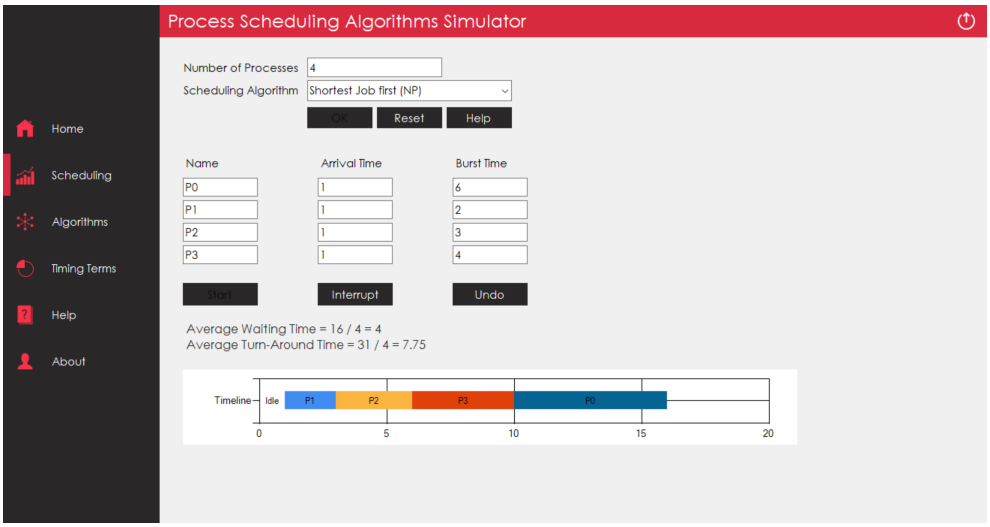

Figure 4.2: Shortest Job First Algorithm

4.6 Priority-Based

Priority-based Scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the given priority. In the application, an extra column is enabled if the user selects any of the two Priority (preemptive or non-preemptive) algorithms, where the user can fill in each process' priority.

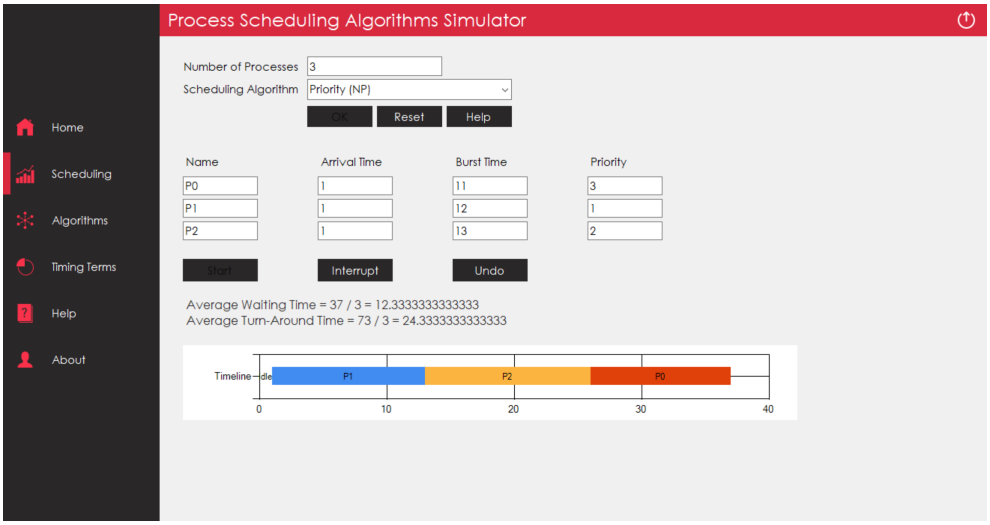

Figure 4.3: Priority-Based Algorithm

4.7 Round-Robin

In Round-robin scheduling, each ready process runs turn by turn only in a cyclic queue for a limited time slice, called a quantum. This algorithm also offers starvation free execution of processes. Like above, the application displays a separate text box dedicated to the quantum value if the selected algorithm is Round-Robin. This, for instance, is done by:

```
1    if (t == "Round Robin")
2        {
3            // Defining properties for the "Quantum Time" label
4            System.Windows.Forms.Label l5 = new System.Windows.
   Forms.Label();
5            this.Controls.Add(l5);
6            l5.Top = 200;
7            l5.Left = 780;
8            l5.Text = "Quantum Time";
9            l5.Font = new Font("Century Gothic", 11);
```

```
10
11              // Creating a text box for user input
12              System.Windows.Forms.TextBox txt5 = new System.
    Windows.Forms.TextBox();
13              quantum = txt5;
14              this.Controls.Add(txt5);
15              txt5.Top = 230;
16              txt5.Left = 780;
17              txt5.Text = "1";
18              txt5.Font = new Font("Century Gothic", 11);
19          }
```

Code 4.5: Implementing Quantum Text Box

Once the user puts in this value and clicks "Start", the program runs the Round-Robin algorithm and displays the result, as shown below:

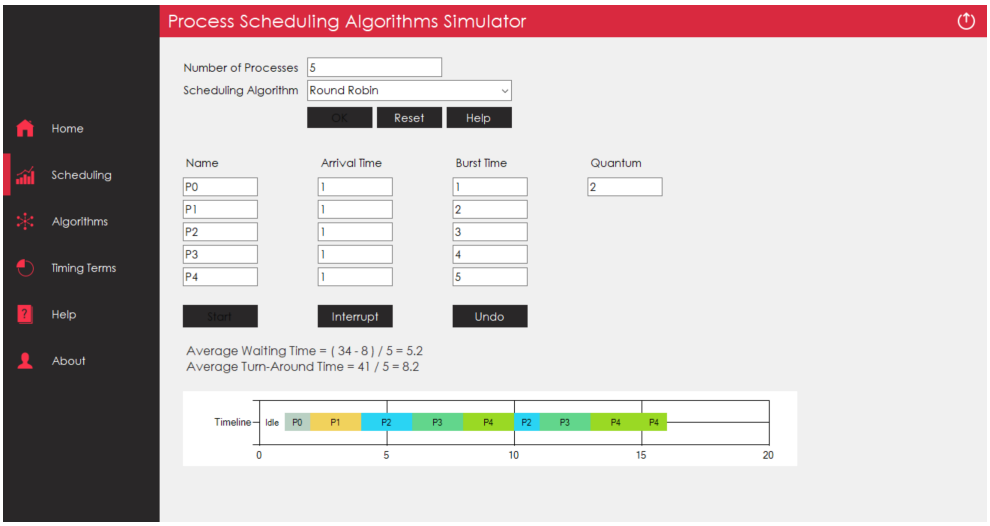

Figure 4.4: Round-Robin Algorithm

4.8 Interrupts

In the application, we can add processes to the already running algorithm in real time by clicking the "Interrupt" button. This temporarily hides the time calculations and the Gantt chart and displays a number of text boxes for user input. The user can now add a process name of their choice, as well as its corresponding arrival and burst time, and press "Add Process". Once the input is taken and the new process is added, the Gantt chart is updated and new calculations are made, as below.

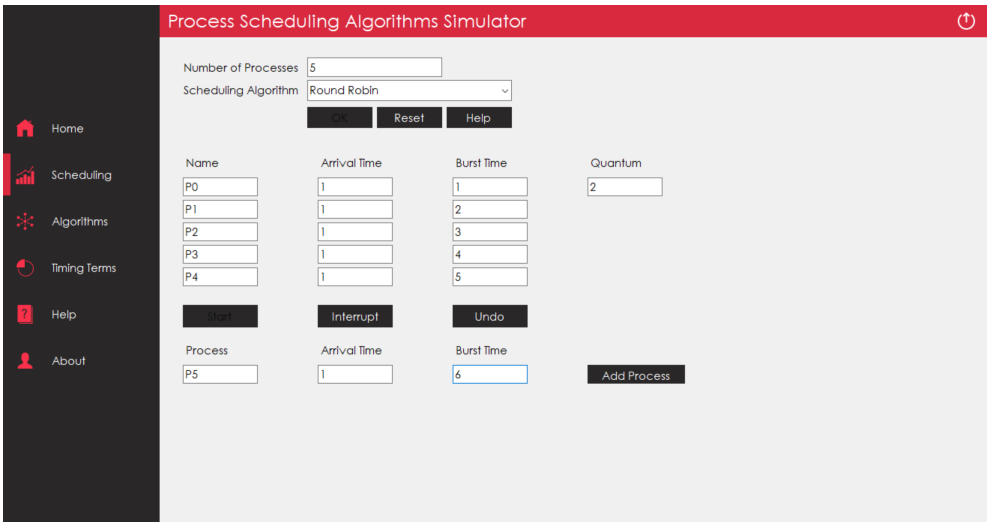

Figure 4.5: Interrupting the RR Algorithm

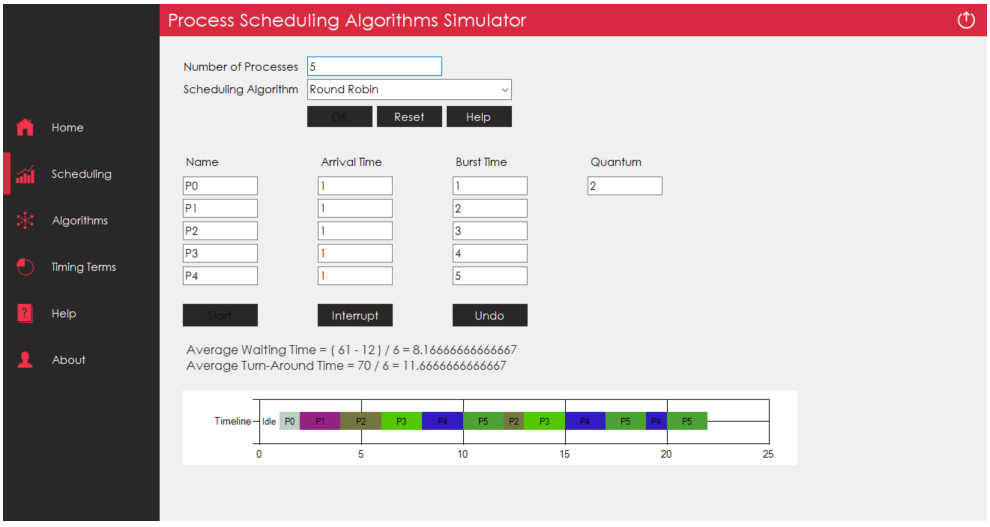

Figure 4.6: The RR Algorithm After Interrupt

Chapter 5

Summary

From the back-end code to the interface design, the application has been designed keeping users of all skill level in mind. It provides a practical approach towards calculating and running scheduling algorithms and displaying all relevant information to the user.

5.1 Improvements

After testing and running the software, we looked at a number of improvements we could make as further update releases in the application. Some of those ideas include:

- Implement MLQ and MLFQ algorithms,
- Enhancing the UI by adding reactive panels and buttons,
- Improving the Gantt chart by adding on-hover information,
- Overall code refining and potential bug fixes, etc.

END