CPU Scheduling Implementation

By:

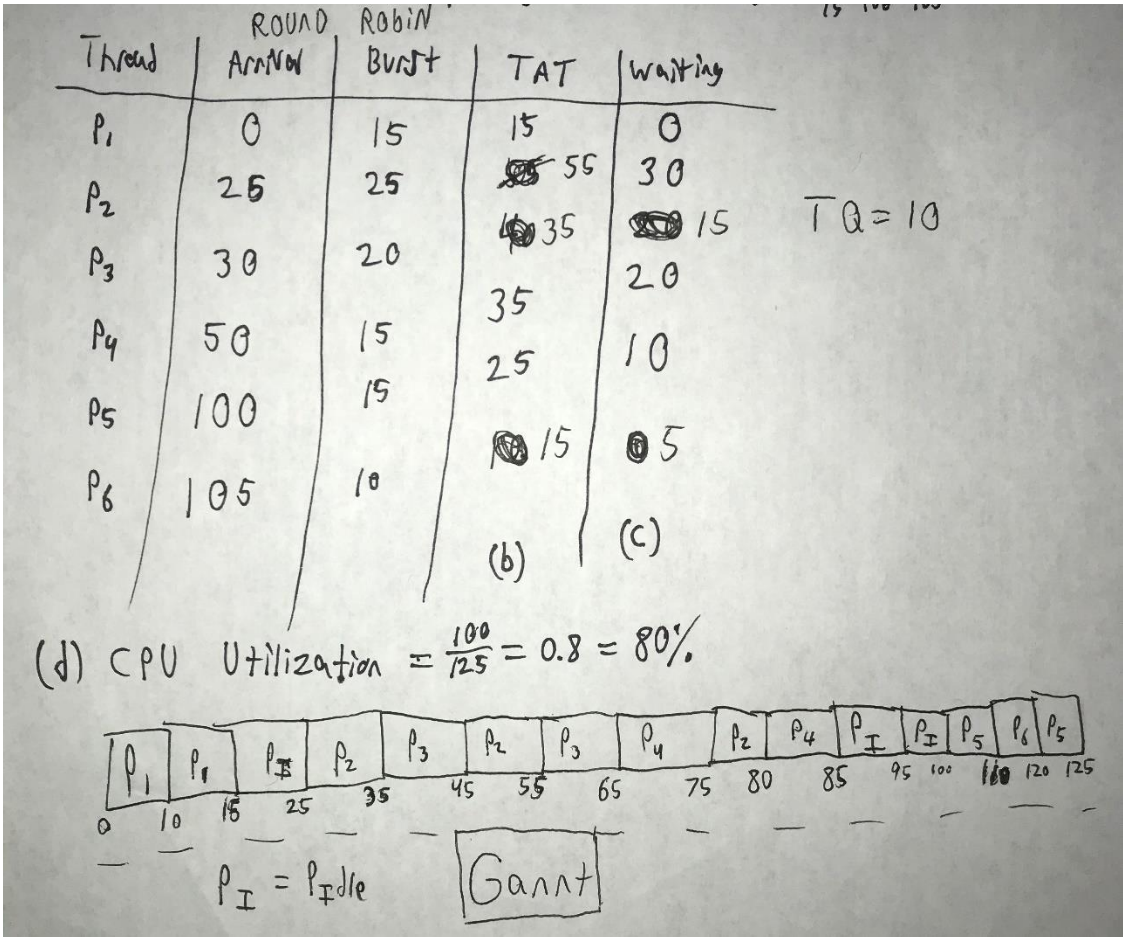
Kyler Finn, Morgan Houston, Haley Walston, & Jordan Wright

1. Introduction

In this project, we analyze the round robin and multilevel queue scheduling schemes. By scheduling these process' we can see their turnaround times and waiting times. In our program, you can change the time quantum by clicking the '+' and '-' buttons. You will see the time quantum label change by a value of 1. You can then calculate the average waiting and turnaround times for that time quantum. You can reset the program and calculate another time quantum to see the difference.

2. Scheduling Techniques

We used the Round Robin scheduling with preemptive priority to schedule the first set of processes.



ROUND Robin

Thread	Arrival	Burst	TAT	waiting
P_1	0	15	15	0
P_2	25	25	55	30
P_3	30	20	35	15
P_4	50	15	35	20
P_5	100	15	25	10
P_6	105	10	15	5
			(b)	(c)

TQ = 10

(d) CPU Utilization $= \frac{100}{125} = 0.8 = 80\%$

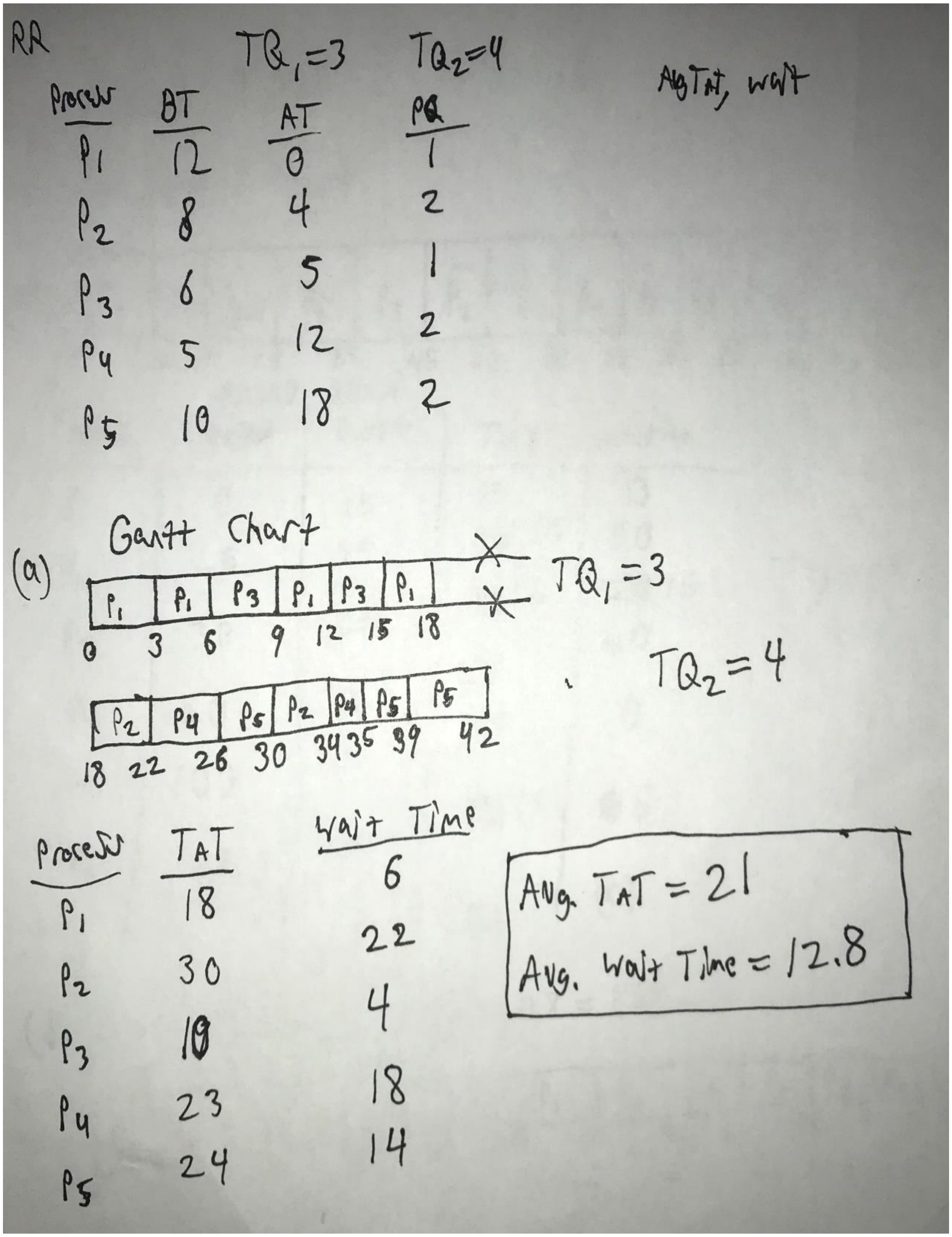
P_1	P_1	P_I	P_2	P_3	P_2	P_3	P_4	P_2	P_4	P_I	P_I	P_5	P_6	P_5

0 10 15 25 35 45 55 65 75 80 85 95 100 110 120 125

$P_I = P_{Idle}$

Gannt

The next scheduling technique we used is multilevel queue with two queues, both using round robin scheduling.



RR

$TQ_1=3$ $TQ_2=4$

AvgTAT, wait

Process	BT	AT	PQ
P_1	12	0	1
P_2	8	4	2
P_3	6	5	1
P_4	5	12	2
P_5	10	18	2

Gantt Chart

(a)

P_1	P_1	P_3	P_1	P_3	P_1	
0	3	6	9	12	15	18

$TQ_1=3$

P_2	P_4	P_5	P_2	P_4	P_5	P_5	
18	22	26	30	34	35	39	42

$TQ_2=4$

Process	TAT	Wait Time
P_1	18	6
P_2	30	22
P_3	10	4
P_4	23	18
P_5	24	14

Avg. TAT = 21

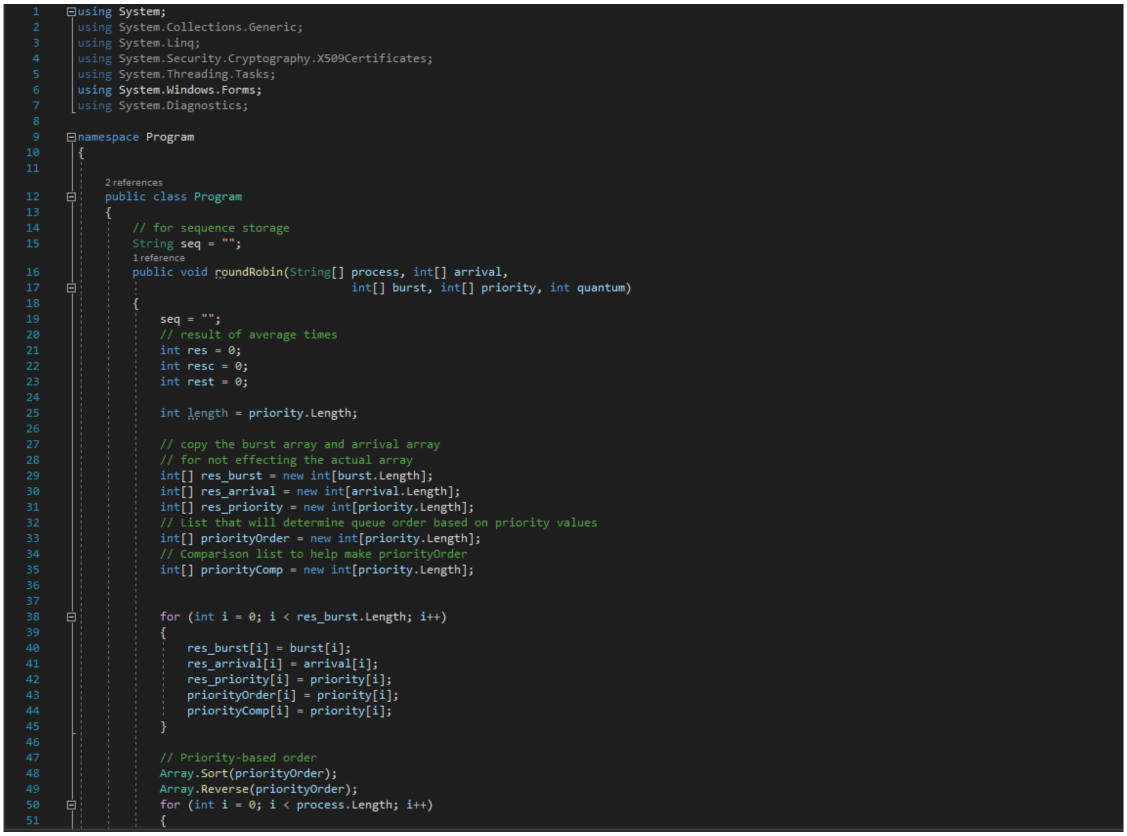
Avg. Wait Time = 12.8

3. Running the program

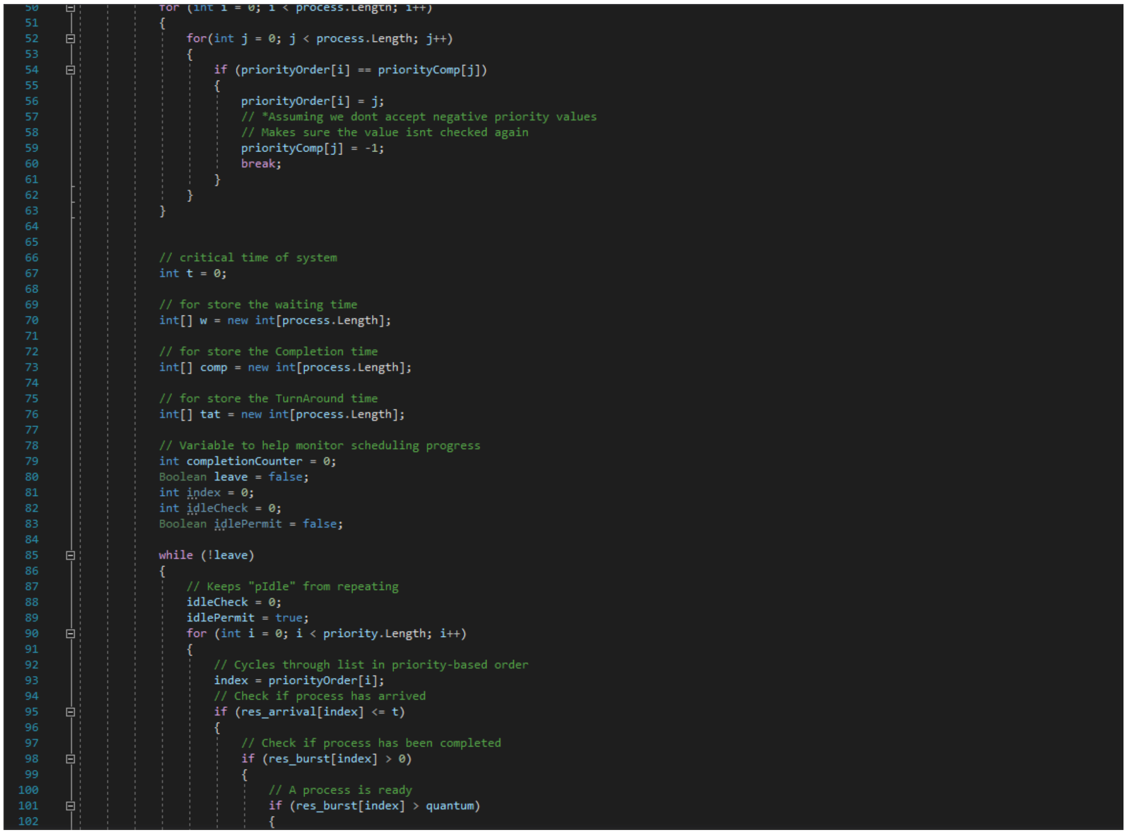
When the program starts, a list of processes and their arrival times, burst times, and priority is displayed for each scheduling technique. We use Round Robin and Multilevel Queue scheduling schemes to schedule these processes. You can change the time quantum for either, by using the '+' and '-' buttons on the left. Press the calculate button to see the Gannt chart, average turnaround and the average waiting time of the process list. Reset button will clear the Gannt chart and average data.

4. Source Code

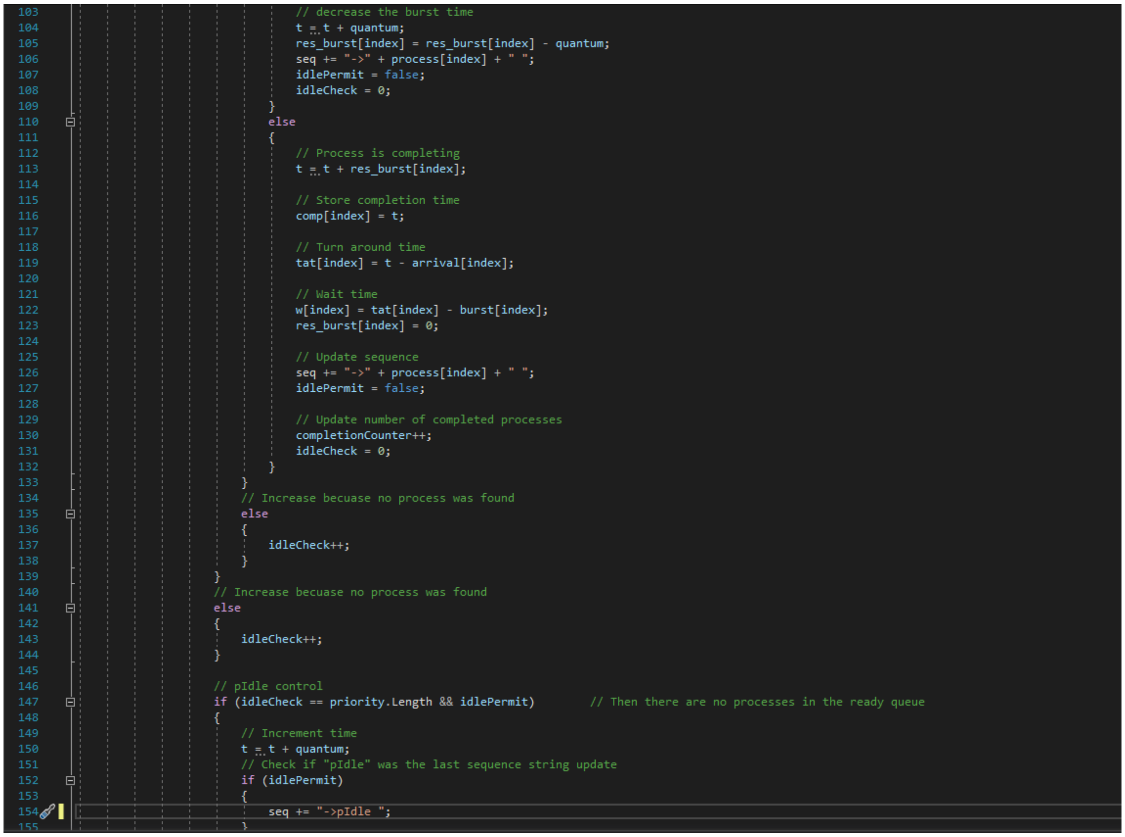
https://github.com/mmhousto/Program



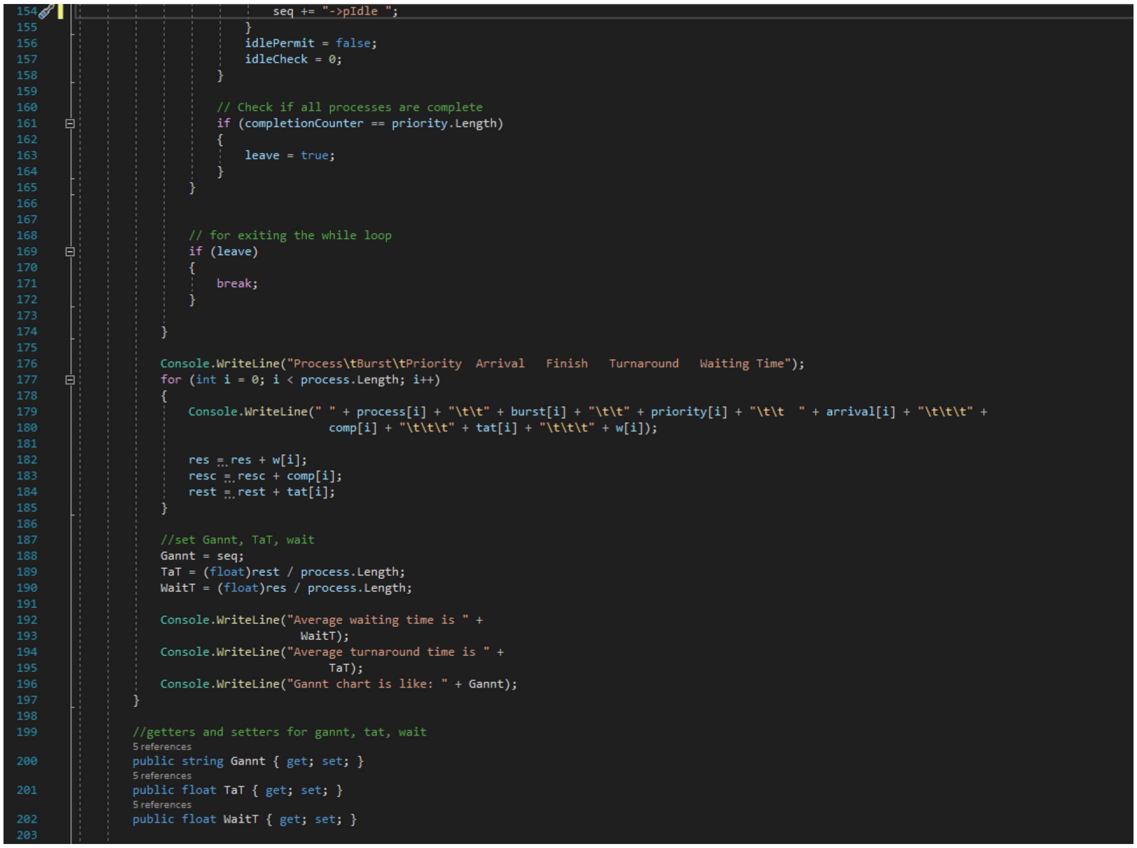
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;

namespace Program
{

    2 references
    public class Program
    {
        // for sequence storage
        String seq = "";
        1 reference
        public void roundRobin(String[] process, int[] arrival,
                                        int[] burst, int[] priority, int quantum)
        {
            seq = "";
            // result of average times
            int res = 0;
            int resc = 0;
            int rest = 0;

            int length = priority.Length;

            // copy the burst array and arrival array
            // for not effecting the actual array
            int[] res_burst = new int[burst.Length];
            int[] res_arrival = new int[arrival.Length];
            int[] res_priority = new int[priority.Length];
            // List that will determine queue order based on priority values
            int[] priorityOrder = new int[priority.Length];
            // Comparison list to help make priorityOrder
            int[] priorityComp = new int[priority.Length];


            for (int i = 0; i < res_burst.Length; i++)
            {
                res_burst[i] = burst[i];
                res_arrival[i] = arrival[i];
                res_priority[i] = priority[i];
                priorityOrder[i] = priority[i];
                priorityComp[i] = priority[i];
            }

            // Priority-based order
            Array.Sort(priorityOrder);
            Array.Reverse(priorityOrder);
            for (int i = 0; i < process.Length; i++)
            {
```



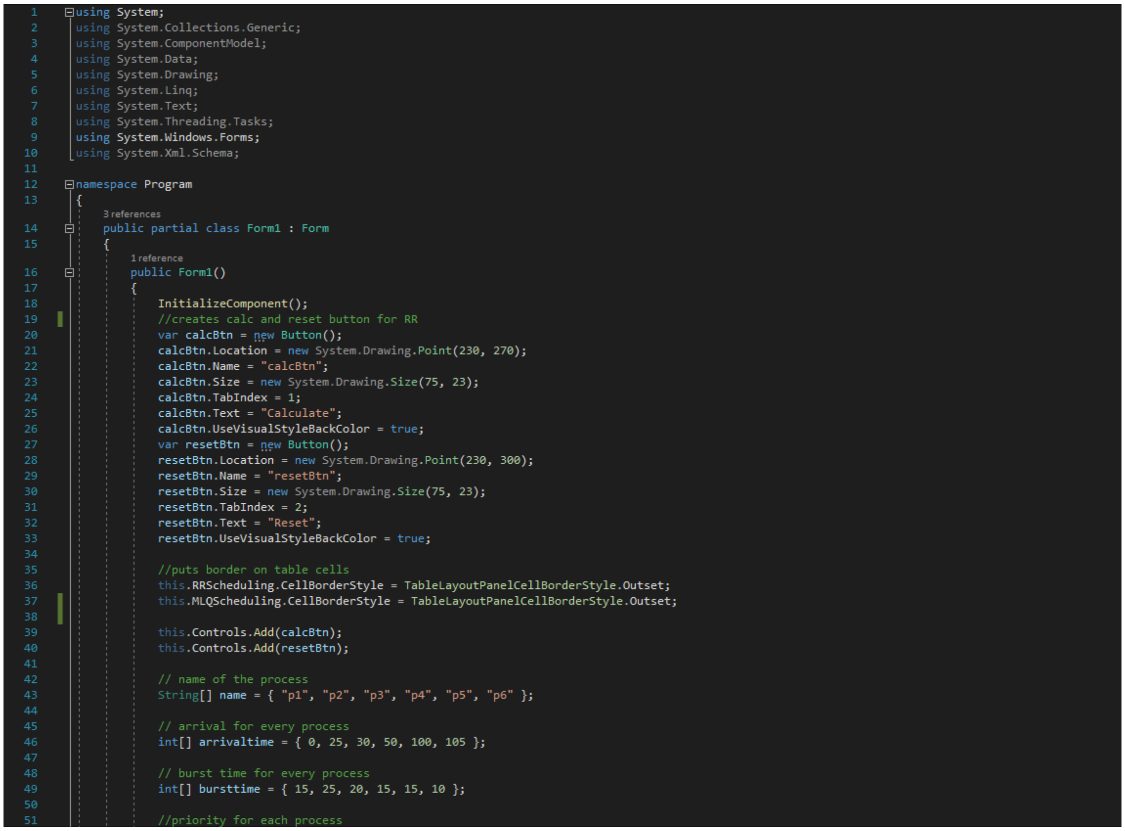
```
            for (int i = 0; i < process.Length; i++)
            {
                for(int j = 0; j < process.Length; j++)
                {
                    if (priorityOrder[i] == priorityComp[j])
                    {
                        priorityOrder[i] = j;
                        // *Assuming we dont accept negative priority values
                        // Makes sure the value isnt checked again
                        priorityComp[j] = -1;
                        break;
                    }
                }
            }


            // critical time of system
            int t = 0;

            // for store the waiting time
            int[] w = new int[process.Length];

            // for store the Completion time
            int[] comp = new int[process.Length];

            // for store the TurnAround time
            int[] tat = new int[process.Length];

            // Variable to help monitor scheduling progress
            int completionCounter = 0;
            Boolean leave = false;
            int index = 0;
            int idleCheck = 0;
            Boolean idlePermit = false;

            while (!leave)
            {
                // Keeps "pIdle" from repeating
                idleCheck = 0;
                idlePermit = true;
                for (int i = 0; i < priority.Length; i++)
                {
                    // Cycles through list in priority-based order
                    index = priorityOrder[i];
                    // Check if process has arrived
                    if (res_arrival[index] <= t)
                    {
                        // Check if process has been completed
                        if (res_burst[index] > 0)
                        {
                            // A process is ready
                            if (res_burst[index] > quantum)
                            {
```



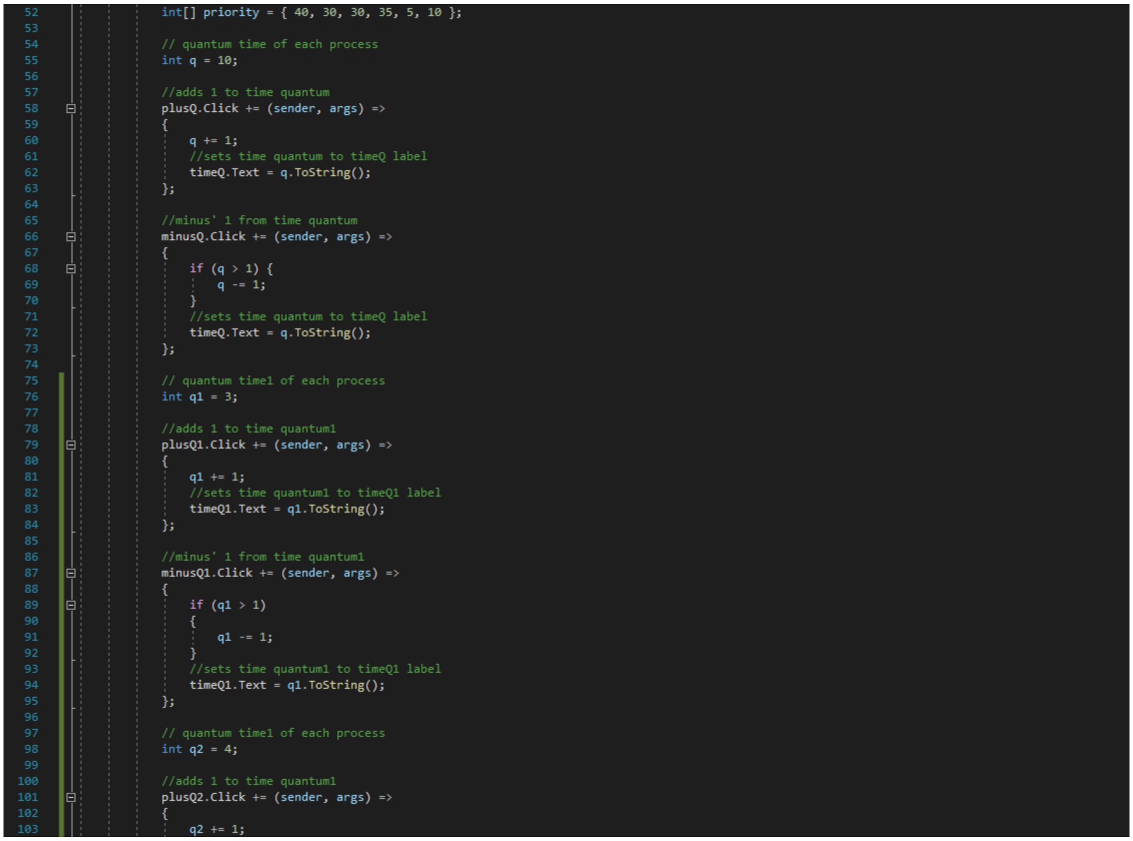
```
                                // decrease the burst time
                                t = t + quantum;
                                res_burst[index] = res_burst[index] - quantum;
                                seq += "->" + process[index] + " ";
                                idlePermit = false;
                                idleCheck = 0;
                            }
                            else
                            {
                                // Process is completing
                                t = t + res_burst[index];

                                // Store completion time
                                comp[index] = t;

                                // Turn around time
                                tat[index] = t - arrival[index];

                                // Wait time
                                w[index] = tat[index] - burst[index];
                                res_burst[index] = 0;

                                // Update sequence
                                seq += "->" + process[index] + " ";
                                idlePermit = false;

                                // Update number of completed processes
                                completionCounter++;
                                idleCheck = 0;
                            }
                        }
                        // Increase becuase no process was found
                        else
                        {
                            idleCheck++;
                        }
                    }
                    // Increase becuase no process was found
                    else
                    {
                        idleCheck++;
                    }

                    // pIdle control
                    if (idleCheck == priority.Length && idlePermit)        // Then there are no processes in the ready queue
                    {
                        // Increment time
                        t = t + quantum;
                        // Check if "pIdle" was the last sequence string update
                        if (idlePermit)
                        {
                            seq += "->pIdle ";
                        }
```



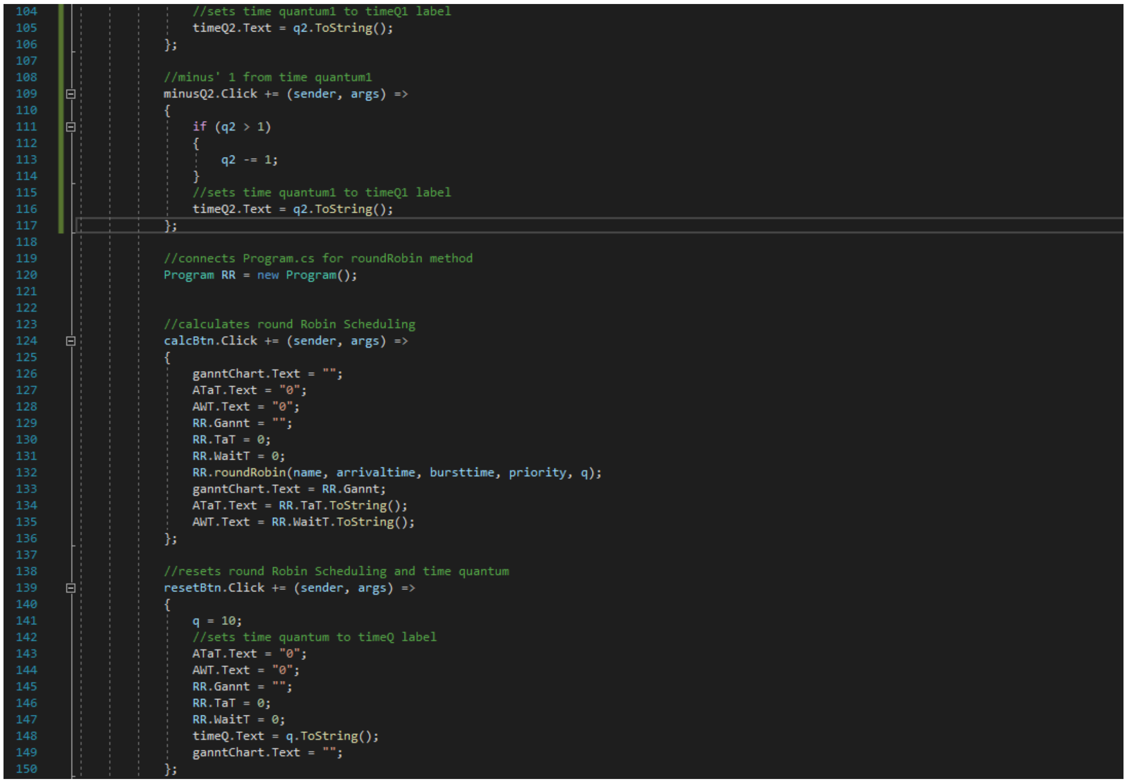
```
                            seq += "->pIdle ";
                        }
                        idlePermit = false;
                        idleCheck = 0;
                    }

                    // Check if all processes are complete
                    if (completionCounter == priority.Length)
                    {
                        leave = true;
                    }
                }


                // for exiting the while loop
                if (leave)
                {
                    break;
                }

            }

            Console.WriteLine("Process\tBurst\tPriority  Arrival   Finish   Turnaround   Waiting Time");
            for (int i = 0; i < process.Length; i++)
            {
                Console.WriteLine(" " + process[i] + "\t\t" + burst[i] + "\t\t" + priority[i] + "\t\t  " + arrival[i] + "\t\t\t" +
                                    comp[i] + "\t\t\t" + tat[i] + "\t\t\t" + w[i]);

                res = res + w[i];
                resc = resc + comp[i];
                rest = rest + tat[i];
            }

            //set Gannt, TaT, wait
            Gannt = seq;
            TaT = (float)rest / process.Length;
            WaitT = (float)res / process.Length;

            Console.WriteLine("Average waiting time is " +
                                WaitT);
            Console.WriteLine("Average turnaround time is " +
                                    TaT);
            Console.WriteLine("Gannt chart is like: " + Gannt);
        }

        //getters and setters for gannt, tat, wait
        5 references
        public string Gannt { get; set; }
        5 references
        public float TaT { get; set; }
        5 references
        public float WaitT { get; set; }

```



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml.Schema;

namespace Program
{
    3 references
    public partial class Form1 : Form
    {
        1 reference
        public Form1()
        {
            InitializeComponent();
            //creates calc and reset button for RR
            var calcBtn = new Button();
            calcBtn.Location = new System.Drawing.Point(230, 270);
            calcBtn.Name = "calcBtn";
            calcBtn.Size = new System.Drawing.Size(75, 23);
            calcBtn.TabIndex = 1;
            calcBtn.Text = "Calculate";
            calcBtn.UseVisualStyleBackColor = true;
            var resetBtn = new Button();
            resetBtn.Location = new System.Drawing.Point(230, 300);
            resetBtn.Name = "resetBtn";
            resetBtn.Size = new System.Drawing.Size(75, 23);
            resetBtn.TabIndex = 2;
            resetBtn.Text = "Reset";
            resetBtn.UseVisualStyleBackColor = true;

            //puts border on table cells
            this.RRScheduling.CellBorderStyle = TableLayoutPanelCellBorderStyle.Outset;
            this.MLQScheduling.CellBorderStyle = TableLayoutPanelCellBorderStyle.Outset;

            this.Controls.Add(calcBtn);
            this.Controls.Add(resetBtn);

            // name of the process
            String[] name = { "p1", "p2", "p3", "p4", "p5", "p6" };

            // arrival for every process
            int[] arrivaltime = { 0, 25, 30, 50, 100, 105 };

            // burst time for every process
            int[] bursttime = { 15, 25, 20, 15, 15, 10 };

            //priority for each process
```

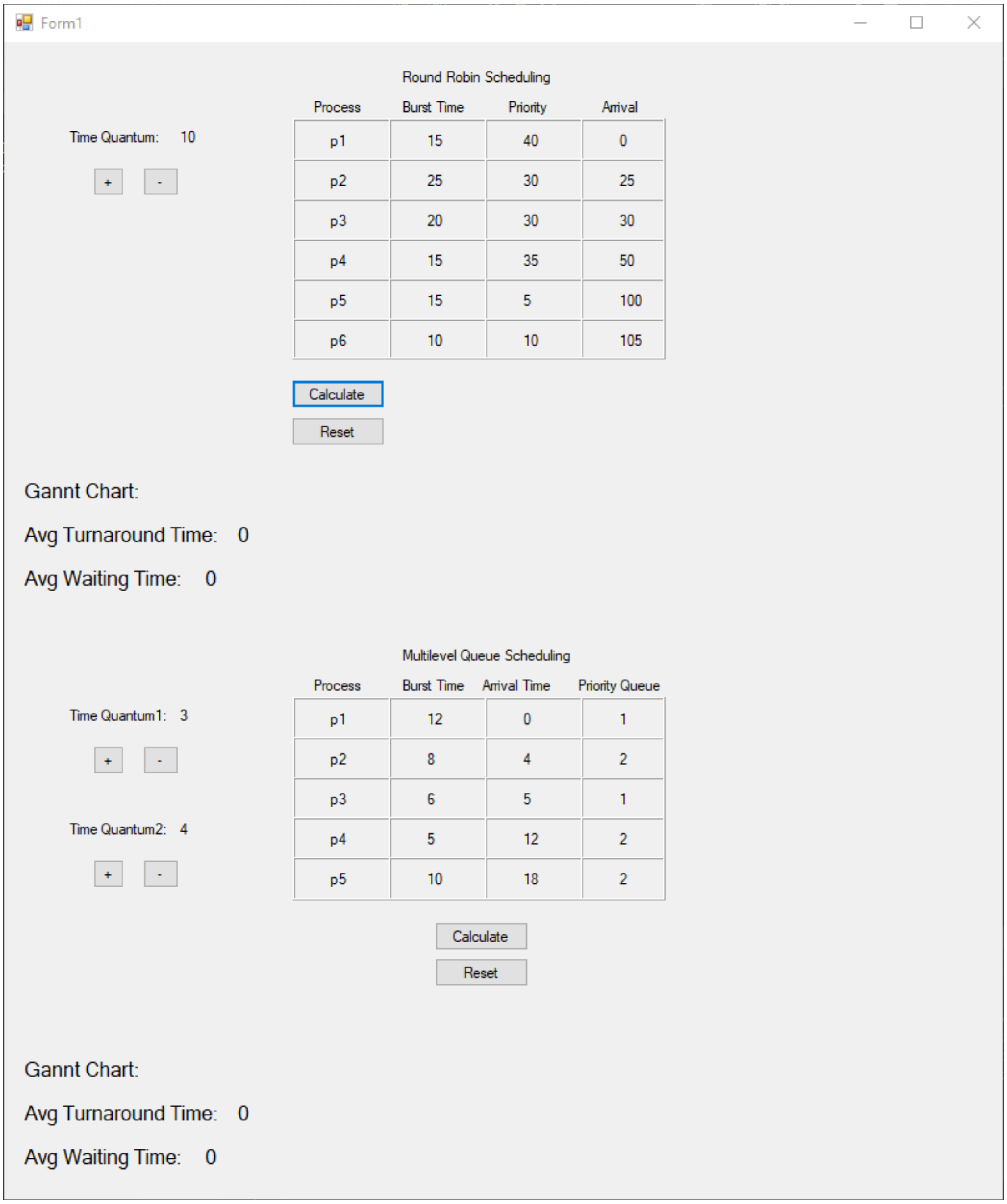


```
int[] priority = { 40, 30, 30, 35, 5, 10 };

// quantum time of each process
int q = 10;

//adds 1 to time quantum
plusQ.Click += (sender, args) =>
{
    q += 1;
    //sets time quantum to timeQ label
    timeQ.Text = q.ToString();
};

//minus' 1 from time quantum
minusQ.Click += (sender, args) =>
{
    if (q > 1) {
        q -= 1;
    }
    //sets time quantum to timeQ label
    timeQ.Text = q.ToString();
};

// quantum time1 of each process
int q1 = 3;

//adds 1 to time quantum1
plusQ1.Click += (sender, args) =>
{
    q1 += 1;
    //sets time quantum1 to timeQ1 label
    timeQ1.Text = q1.ToString();
};

//minus' 1 from time quantum1
minusQ1.Click += (sender, args) =>
{
    if (q1 > 1)
    {
        q1 -= 1;
    }
    //sets time quantum1 to timeQ1 label
    timeQ1.Text = q1.ToString();
};

// quantum time1 of each process
int q2 = 4;

//adds 1 to time quantum1
plusQ2.Click += (sender, args) =>
{
    q2 += 1;
```



```
    //sets time quantum1 to timeQ1 label
    timeQ2.Text = q2.ToString();
};

//minus' 1 from time quantum1
minusQ2.Click += (sender, args) =>
{
    if (q2 > 1)
    {
        q2 -= 1;
    }
    //sets time quantum1 to timeQ1 label
    timeQ2.Text = q2.ToString();
};

//connects Program.cs for roundRobin method
Program RR = new Program();


//calculates round Robin Scheduling
calcBtn.Click += (sender, args) =>
{
    ganntChart.Text = "";
    ATaT.Text = "0";
    AWT.Text = "0";
    RR.Gannt = "";
    RR.TaT = 0;
    RR.WaitT = 0;
    RR.roundRobin(name, arrivaltime, bursttime, priority, q);
    ganntChart.Text = RR.Gannt;
    ATaT.Text = RR.TaT.ToString();
    AWT.Text = RR.WaitT.ToString();
};

//resets round Robin Scheduling and time quantum
resetBtn.Click += (sender, args) =>
{
    q = 10;
    //sets time quantum to timeQ label
    ATaT.Text = "0";
    AWT.Text = "0";
    RR.Gannt = "";
    RR.TaT = 0;
    RR.WaitT = 0;
    timeQ.Text = q.ToString();
    ganntChart.Text = "";
};
```

5. Output



Round Robin Scheduling

Time Quantum: 10

Process	Burst Time	Priority	Arrival
p1	15	40	0
p2	25	30	25
p3	20	30	30
p4	15	35	50
p5	15	5	100
p6	10	10	105

Gannt Chart:

Avg Turnaround Time: 0

Avg Waiting Time: 0

Multilevel Queue Scheduling

Time Quantum1: 3

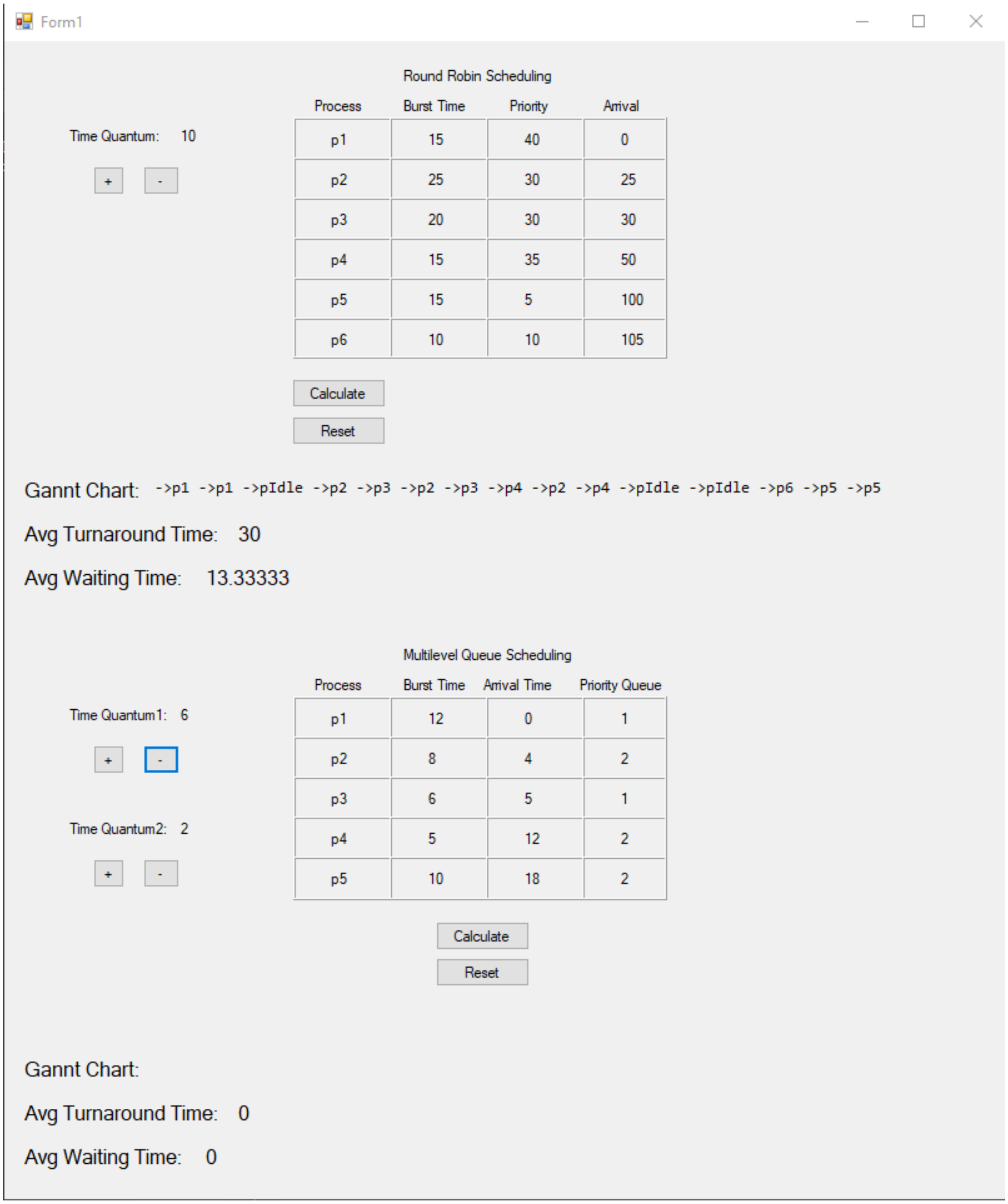
Time Quantum2: 4

Process	Burst Time	Arrival Time	Priority Queue
p1	12	0	1
p2	8	4	2
p3	6	5	1
p4	5	12	2
p5	10	18	2

Gannt Chart:

Avg Turnaround Time: 0

Avg Waiting Time: 0



Round Robin Scheduling

Time Quantum: 10

Process	Burst Time	Priority	Arrival
p1	15	40	0
p2	25	30	25
p3	20	30	30
p4	15	35	50
p5	15	5	100
p6	10	10	105

```
Gannt Chart:  ->p1 ->p1 ->pIdle ->p2 ->p3 ->p2 ->p3 ->p4 ->p2 ->p4 ->pIdle ->pIdle ->p6 ->p5 ->p5
```

Avg Turnaround Time: 30

Avg Waiting Time: 13.33333

Multilevel Queue Scheduling

Time Quantum1: 6

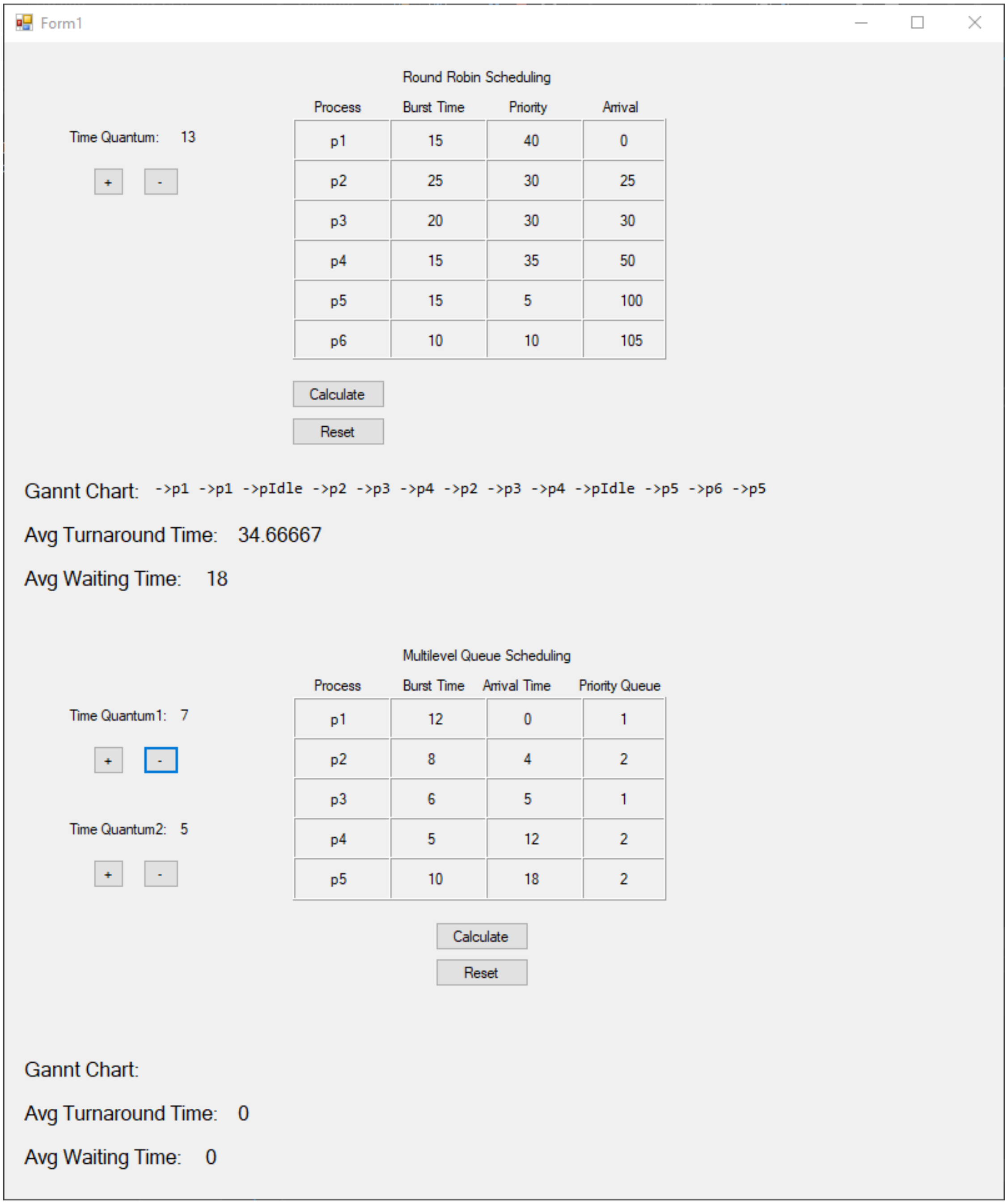
Time Quantum2: 2

Process	Burst Time	Arrival Time	Priority Queue
p1	12	0	1
p2	8	4	2
p3	6	5	1
p4	5	12	2
p5	10	18	2

Gannt Chart:

Avg Turnaround Time: 0

Avg Waiting Time: 0



Round Robin Scheduling

Time Quantum: 13

Process	Burst Time	Priority	Arrival
p1	15	40	0
p2	25	30	25
p3	20	30	30
p4	15	35	50
p5	15	5	100
p6	10	10	105

Gannt Chart: `->p1 ->p1 ->pIdle ->p2 ->p3 ->p4 ->p2 ->p3 ->p4 ->pIdle ->p5 ->p6 ->p5`

Avg Turnaround Time: 34.66667

Avg Waiting Time: 18

Multilevel Queue Scheduling

Time Quantum1: 7

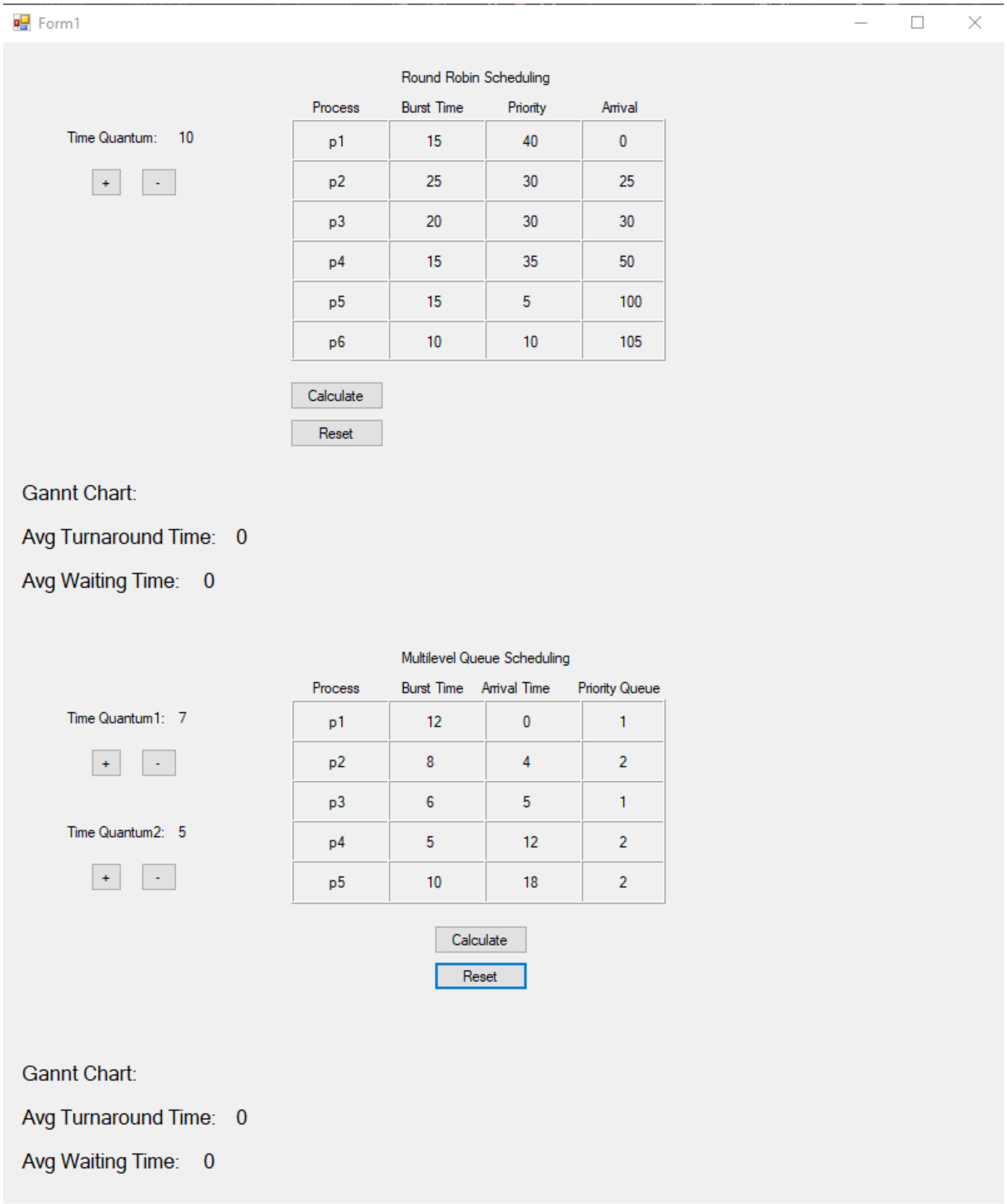
Time Quantum2: 5

Process	Burst Time	Arrival Time	Priority Queue
p1	12	0	1
p2	8	4	2
p3	6	5	1
p4	5	12	2
p5	10	18	2

Gannt Chart:

Avg Turnaround Time: 0

Avg Waiting Time: 0



Round Robin Scheduling

Time Quantum: 10

Process	Burst Time	Priority	Arrival
p1	15	40	0
p2	25	30	25
p3	20	30	30
p4	15	35	50
p5	15	5	100
p6	10	10	105

Gannt Chart:

Avg Turnaround Time: 0

Avg Waiting Time: 0

Multilevel Queue Scheduling

Time Quantum1: 7

Time Quantum2: 5

Process	Burst Time	Arrival Time	Priority Queue
p1	12	0	1
p2	8	4	2
p3	6	5	1
p4	5	12	2
p5	10	18	2

Gannt Chart:

Avg Turnaround Time: 0

Avg Waiting Time: 0

References

https://www.geeksforgeeks.org/round-robin-scheduling-with-different-arrival-times/?ref=rp

Operating System Concepts 9th Edition – Abraham Silberschatzk, Peter Baer Galvin, Greg Gagne

Work Done

Kyler Finn: Fixed the round robin function to display better data. Worked on the multilevel queue scheduling algorithm.

Morgan Houston: Created base project file, set up tables, found starting round robin function, made readme file.

Haley Walston: Sent the initial email, to the professor, of who was in our group. Couldn’t get the project to work on Mac.

Jordan Wright: Created canvas group and a rough draft project report. Also, could not get the project to work on Mac.