Aaron Reich

Computer Operating Systems

CPU Scheduler

Table of Contents

Introduction

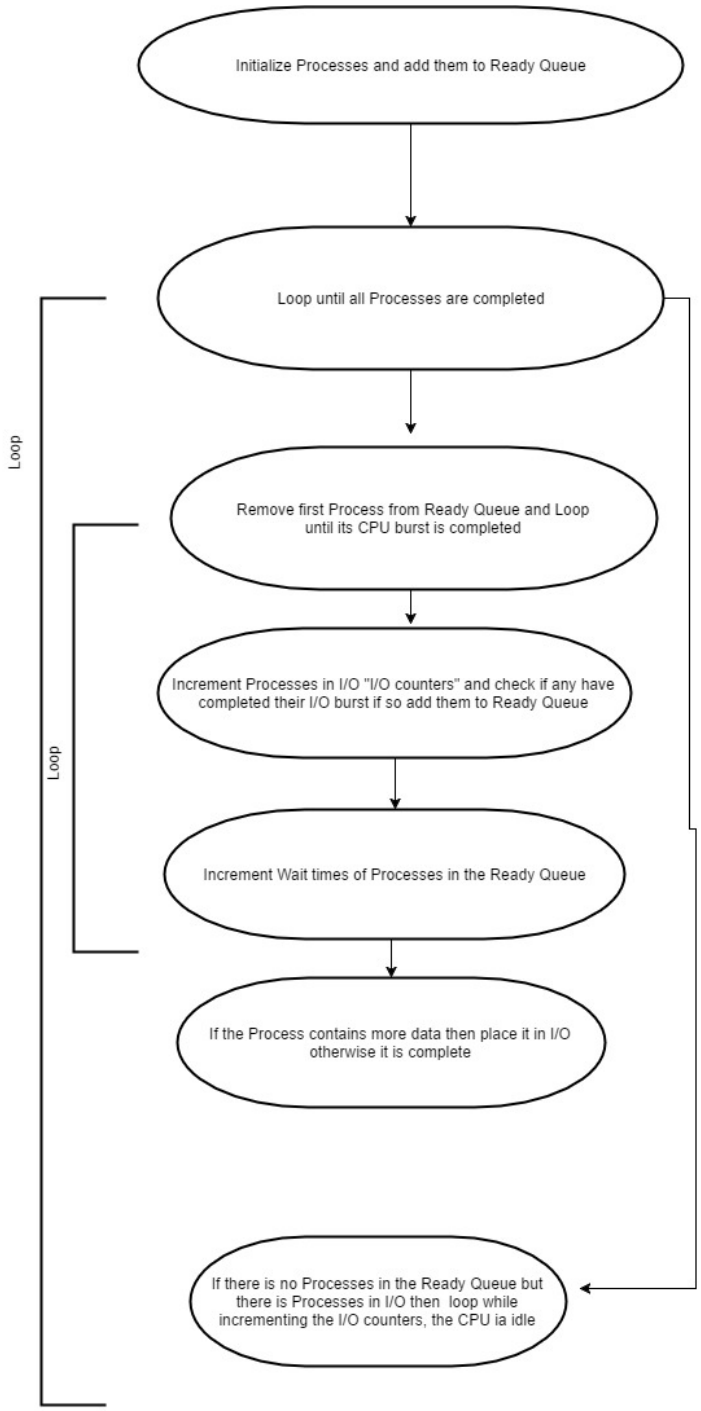
For this project I implemented three different CPU Scheduler algorithms. The algorithms consisted of FCFS (First Come First Serve), SJF (Shortest Job First), and MLFQ (Multi Level Feedback Queue). Through the simulation programs I built I was able to calculate the average wait time, turnaround time, CPU utilization, and response time. After each Process completes a CPU burst, it then goes in I/O and this will also factor into the results. The final results of the scheduling algorithms can be used to conclude which of the three algorithms is the most efficient.

Each algorithm has different criteria for selecting the Process to execute and for how long to allow it to execute. FCFS selects the first Process in the Ready Queue and executes the process for the full length of its CPU burst. SJF selects the Process in the Ready Queue with the smallest CPU burst and also executes the process for the full length of its CPU burst. MLFQ consists of three different queue's each with different priority levels and scheduling algorithms. Queue 1 uses round robin with a time quanum of 6 as its algorithm and has the highest priority level. Queue2 also uses round robin with a time quanum of 11 as its algorithm and has the second highest priority level. Queue3 uses FCFS as its algorithm and has the lowest priority level. MLFQ also uses preemption in order to stop a execute a Process in a higher priority queue halting the execution of Processes in lower queues.

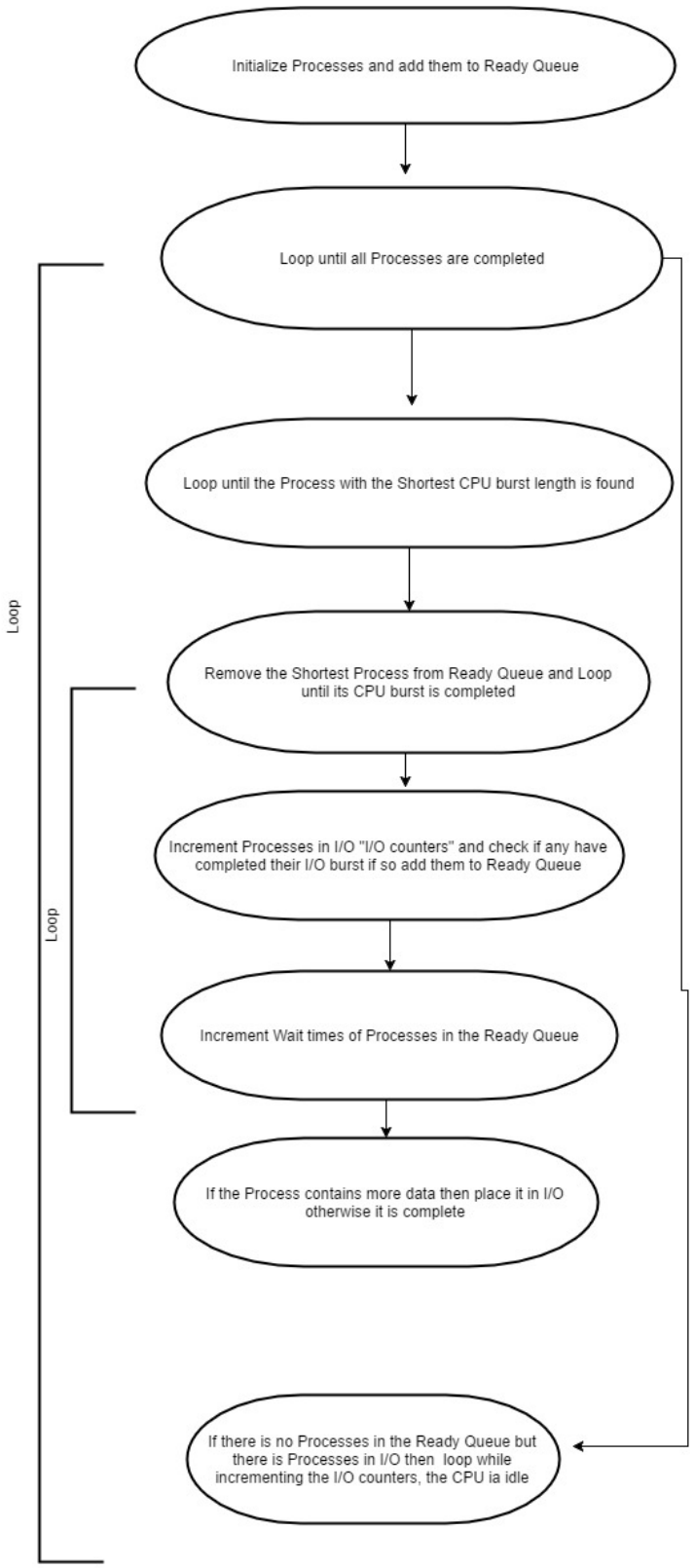
I used C# as the programming language for the simulation programs. I used the Queue class for my ready queue in FCFS and the List class for keeping a collection of the Processes in I/O. I used the List class both the ready queue and the collection of the Processes in I/O for SJF. I created a

MLFQ_Queue class inheriting from the Queue class for the three queues in MLFQ. Each MLFQ_Queue has a reference to its lower priority queue except for Queue3. I created a Process class for three algorithms to hold its CPU and I/O burst data along with its own timing data.

FCFS Logic Flow Chart



SJF Logic Flow Chart



MLFQ Logic Flow Chart

Tables and Discussion

	FCFS CPU Utilization: 86.94%		
	WT	TT	RT
P1	320	672	0
P2	285	758	4
P3	349	571	22
P4	204	741	28
P5	278	907	45
P6	314	611	50
P7	226	596	60
P8	335	700	81
Avg	288	694.5	36.25

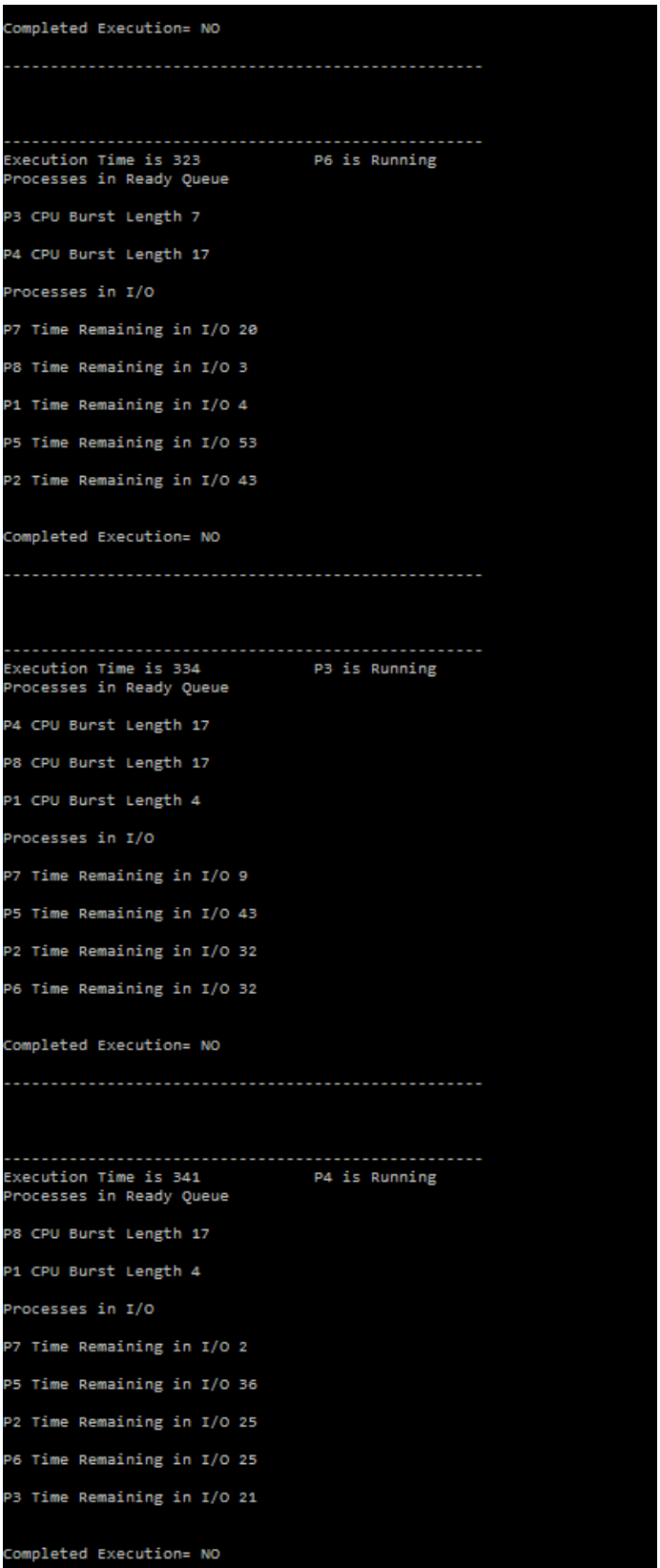
	SJF CPU Utilization: 84.55%		
	WT	TT	RT
P1	75	427	0
P2	281	754	81
P3	83	305	9
P4	270	807	45
P5	67	696	4
P6	63	360	15
P7	430	800	403
P8	154	519	25
Avg	177	583.5	72.75

	MLFQ CPU Utilization: 79.03%		
	WT	TT	RT
P1	70	384	0
P2	358	660	4
P3	128	298	10
P4	310	716	16
P5	22	610	22
P6	203	414	27
P7	325	600	33
P8	243	495	39
Avg	207	522.125	18.875

From analyzing the data resulting from the simulation programs, varying observations can be made. SJF has the smallest average waiting time and the smallest average turnaround time compared to the other algorithms. FCFS has a very high average wait time of 288 and average turnaround time of 694.5 which are relatively high compared to the other algorithms. FCFS is 111 time intervals higher than SJF when it comes to average turnaround time. MLFQ does happen to have close results to SJF in terms of average wait time. When it comes to average response time, MLFQ has the best results followed by FCFS. SJF has the worst results having an average of 72.75. An observation can be made that MLFQ is the most efficient by comparing most of the results with the other two algorithms. Since the MLFQ algorithm does have the highest degree of degree of complexity of the three algorithms, its efficiency is not a surprise.

Dynamic Execution Program Output

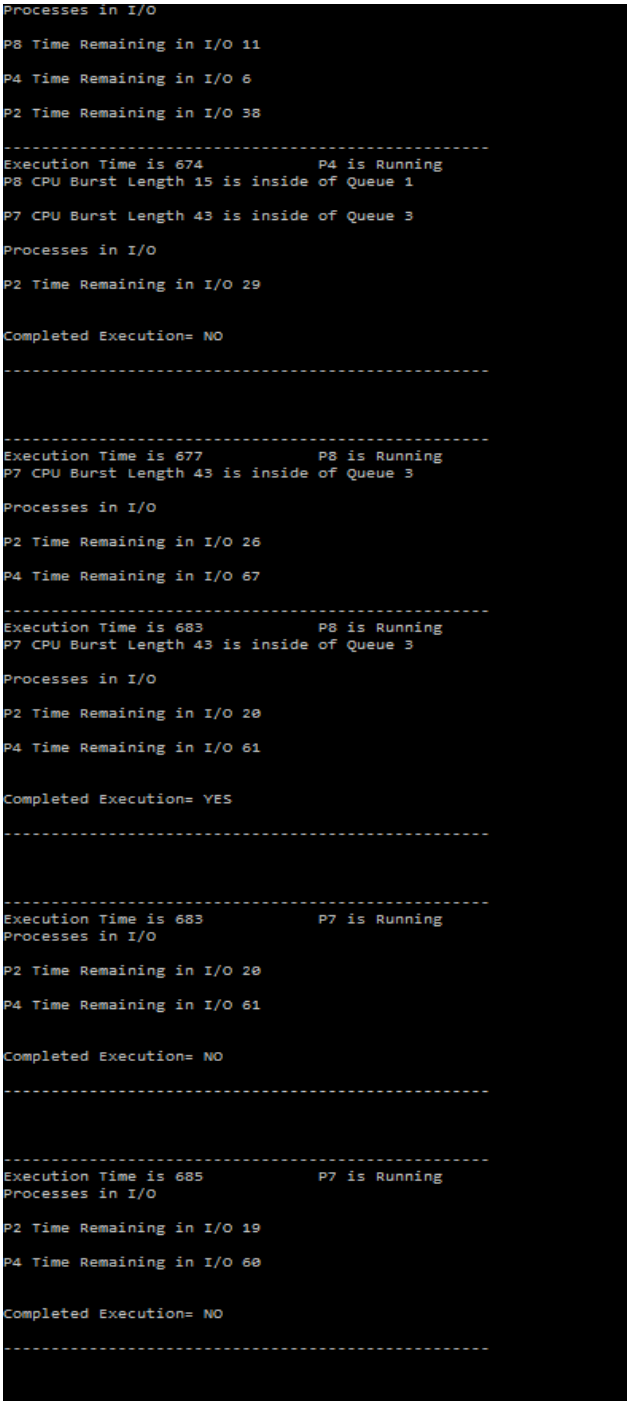
FCFS



SJF

```
--------------------------------------------------
Execution Time is 200           P1 is Running
Processes in Ready Queue

P7 CPU Burst Length 21

P2 CPU Burst Length 19

P5 CPU Burst Length 5

P3 CPU Burst Length 8

Processes in I/O

P4 Time Remaining in I/O 19

P8 Time Remaining in I/O 21

P6 Time Remaining in I/O 32


Completed Execution= NO

--------------------------------------------------



--------------------------------------------------
Execution Time is 204           P5 is Running
Processes in Ready Queue

P7 CPU Burst Length 21

P2 CPU Burst Length 19

P3 CPU Burst Length 8

Processes in I/O

P4 Time Remaining in I/O 15

P8 Time Remaining in I/O 17

P6 Time Remaining in I/O 28

P1 Time Remaining in I/O 33


Completed Execution= NO

--------------------------------------------------



--------------------------------------------------
Execution Time is 209           P3 is Running
Processes in Ready Queue

P7 CPU Burst Length 21

P2 CPU Burst Length 19

Processes in I/O

P4 Time Remaining in I/O 10

P8 Time Remaining in I/O 12

P6 Time Remaining in I/O 23

P1 Time Remaining in I/O 28

P5 Time Remaining in I/O 71


Completed Execution= NO
```

MLFQ

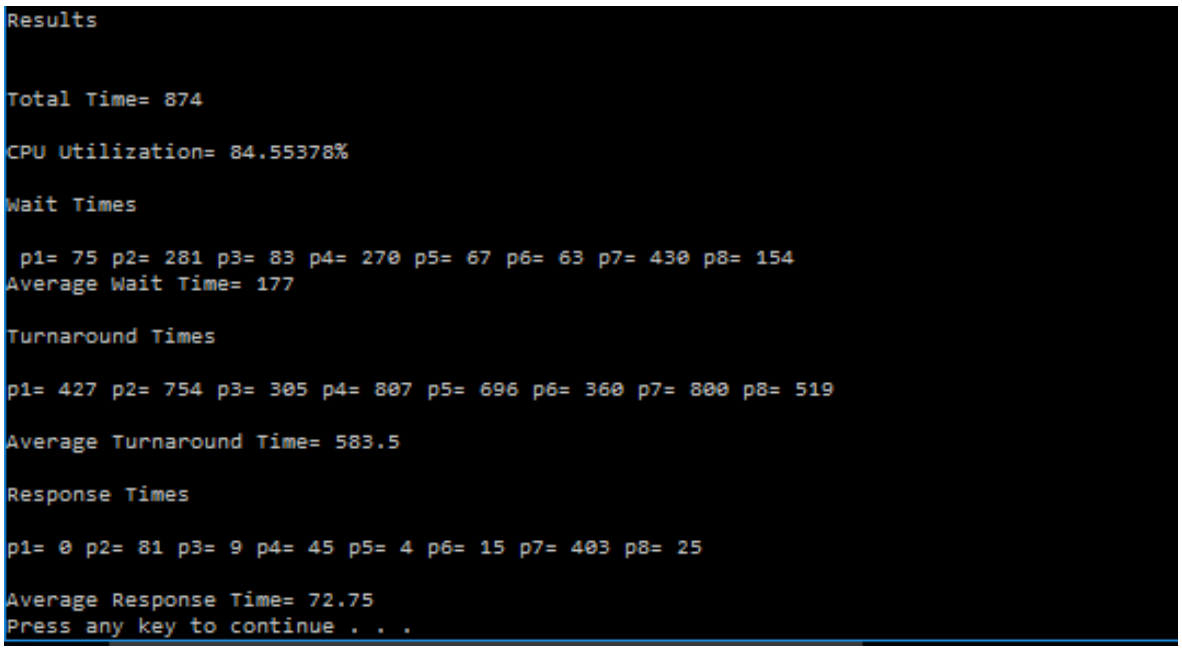

Processes in I/O
P8 Time Remaining in I/O 11
P4 Time Remaining in I/O 6
P2 Time Remaining in I/O 38
Execution Time is 674 P4 is Running
P8 CPU Burst Length 15 is inside of Queue 1
P7 CPU Burst Length 43 is inside of Queue 3
Processes in I/O
P2 Time Remaining in I/O 29
Completed Execution= NO
Execution Time is 677 P8 is Running
P7 CPU Burst Length 43 is inside of Queue 3
Processes in I/O
P2 Time Remaining in I/O 26
P4 Time Remaining in I/O 67
Execution Time is 683 P8 is Running
P7 CPU Burst Length 43 is inside of Queue 3
Processes in I/O
P2 Time Remaining in I/O 20
P4 Time Remaining in I/O 61
Completed Execution= YES
Execution Time is 683 P7 is Running
Processes in I/O
P2 Time Remaining in I/O 20
P4 Time Remaining in I/O 61
Completed Execution= NO
Execution Time is 685 P7 is Running
Processes in I/O
P2 Time Remaining in I/O 19
P4 Time Remaining in I/O 60
Completed Execution= NO

End of Simulation Results Output

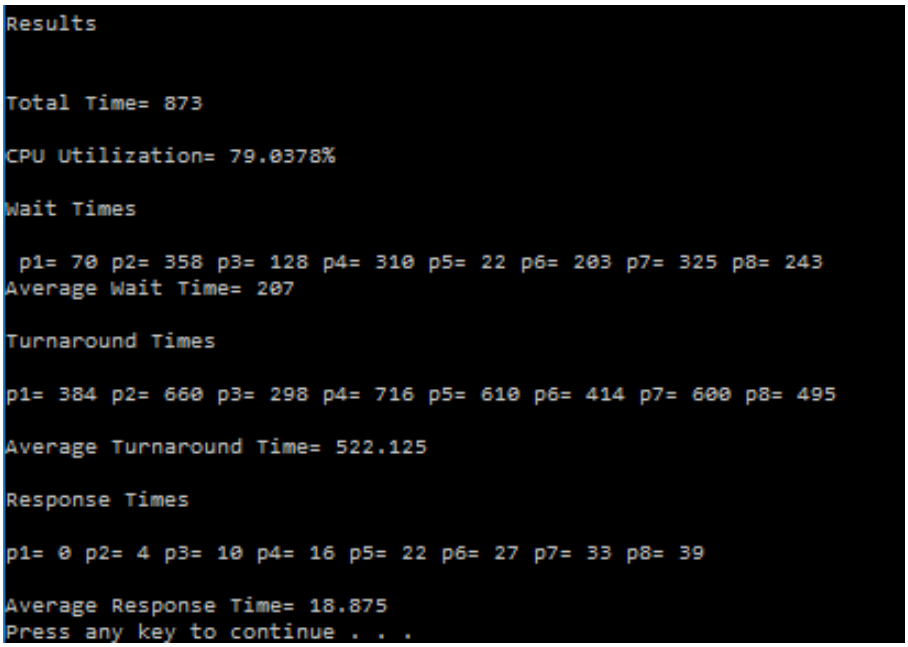
FCFS

```
Results


Total Time= 850

CPU Utilization= 86.94118%

Wait Times

 p1= 320 p2= 285 p3= 349 p4= 204 p5= 278 p6= 314 p7= 226 p8= 335
Average Wait Time= 288

Turnaround Times

p1= 672 p2= 758 p3= 571 p4= 741 p5= 907 p6= 611 p7= 596 p8= 700

Average Turnaround Time= 694.5

Response Times

p1= 0 p2= 4 p3= 22 p4= 28 p5= 45 p6= 50 p7= 60 p8= 81

Average Response Time= 36.25
Press any key to continue . . .
```

SJF


Results
Total Time= 874
CPU Utilization= 84.55378%
Wait Times
p1= 75 p2= 281 p3= 83 p4= 270 p5= 67 p6= 63 p7= 430 p8= 154
Average Wait Time= 177
Turnaround Times
p1= 427 p2= 754 p3= 305 p4= 807 p5= 696 p6= 360 p7= 800 p8= 519
Average Turnaround Time= 583.5
Response Times
p1= 0 p2= 81 p3= 9 p4= 45 p5= 4 p6= 15 p7= 403 p8= 25
Average Response Time= 72.75
Press any key to continue . . .

MLFQ


Results
Total Time= 873
CPU Utilization= 79.0378%
Wait Times
p1= 70 p2= 358 p3= 128 p4= 310 p5= 22 p6= 203 p7= 325 p8= 243
Average Wait Time= 207
Turnaround Times
p1= 384 p2= 660 p3= 298 p4= 716 p5= 610 p6= 414 p7= 600 p8= 495
Average Turnaround Time= 522.125
Response Times
p1= 0 p2= 4 p3= 10 p4= 16 p5= 22 p6= 27 p7= 33 p8= 39
Average Response Time= 18.875
Press any key to continue . . .

Source Code

FCFS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FCFS_ProcessScheduler
{
   class Program
   {
      static void Main(string[] args)
      {
         Process p1 = new Process("P1", new List<int> { 4, 24, 5, 73, 3, 31, 5, 27, 4, 33, 6, 43, 4, 64, 5,
19, 2 });
         Process p2 = new Process("P2", new List<int> { 18, 31, 19, 35, 11, 42, 18, 43, 19, 47, 18, 43,
17, 51, 19, 32, 10 });
         Process p3 = new Process("P3", new List<int> { 6, 18, 4, 21, 7, 19, 4, 16, 5, 29, 7, 21, 8, 22, 6,
24, 5 });
         Process p4 = new Process("P4", new List<int> { 17, 42, 19, 55, 20, 54, 17, 52, 15, 67, 12, 72,
15, 66, 14 });
         Process p5 = new Process("P5", new List<int> { 5, 81, 4, 82, 5, 71, 3, 61, 5, 62, 4, 51, 3, 77, 4,
61, 3, 42, 5 });
         Process p6 = new Process("P6", new List<int> { 10, 35, 12, 41, 14, 33, 11, 32, 15, 41, 13, 29,
11 });
         Process p7 = new Process("P7", new List<int> { 21, 51, 23, 53, 24, 61, 22, 31, 21, 43, 20 });
         Process p8 = new Process("P8", new List<int> { 11, 52, 14, 42, 15, 31, 17, 21, 16, 43, 12, 31,
13, 32, 15 });
         readyqueue = new Queue<Process>();
         //Add all processes to readyqueue
         readyqueue.Enqueue(p1);
         readyqueue.Enqueue(p2);
         readyqueue.Enqueue(p3);
         readyqueue.Enqueue(p4);
         readyqueue.Enqueue(p5);
```

```
readyqueue.Enqueue(p6);
readyqueue.Enqueue(p7);
readyqueue.Enqueue(p8);
int cpuburst = 0;
processesinio = new List<Process>();
int numprocesscomplete = 0;
while (numprocesscomplete != 8)//Run until all processes complete
{

   if (readyqueue.Count != 0)
   {
      Process p = readyqueue.Dequeue();//Remove Process from readyqueue
      if (p.firstrun == true)
      {
         p.responsetime = p.wait;
         p.firstrun = false;
      }


      cpuburst = p.data[0];

      p.totalcpu = p.totalcpu + cpuburst;
      p.data.RemoveAt(0);

      printexecution(p);

      for (int i = 0; i < cpuburst; i++)//Loop until CPU Burst Completes
      {
         totalallcpu++;
         totaltime++;
         if (totaltime == 400)
         {
            int r = 0;
         }
         for (int k = 0; k < processesinio.Count; k++)//Increment Processes in I/O Burst counters
         {
            processesinio.ElementAt(k).iocounter++;


            if (processesinio.ElementAt(k).iocounter == processesinio.ElementAt(k).ioburst)
//Check to see if a Process has completed its I/O burst and if so then it is added to the readyqueue and
removed from the I/O list
            {
               processesinio.ElementAt(k).ioburst = 0;
               processesinio.ElementAt(k).iocounter = 0;
```

```
                    readyqueue.Enqueue(processesinio.ElementAt(k));
                    processesinio.RemoveAt(k);

                }

            }

            for (int j = 0; j < readyqueue.Count; j++)//Increments Processes' Wait times
            {
                readyqueue.ElementAt(j).wait++;

            }

        }

        if (p.data.Count != 0)//Checks to see if the process has not completed and if so puts it in
I/O
        {
            p.ioburst = p.data[0];
            p.totalio = p.totalio + p.ioburst;
            p.data.RemoveAt(0);
            processesinio.Add(p);
            Console.WriteLine("\nCompleted Execution= NO");
            Console.WriteLine("\n-----------------------------------------------------\n\n\n");
        }
        else//Checks to see if the process has completed
        {
            p.turnaroundtime = p.totalcpu + p.totalio + p.wait;
            numprocesscomplete++;
            Console.WriteLine("\nCompleted Execution= YES");
            Console.WriteLine("\n-----------------------------------------------------\n\n\n");
        }

    }
    else if (processesinio.Count != 0)//Handles CPU Idle incrementing Processes in I/O Burst
counters
    {
        int n = 0;
        while (processesinio.ElementAt(n).iocounter < processesinio.ElementAt(n).ioburst)
        {
            totaltime++;
            processesinio.ElementAt(n).iocounter++;
            if (n == processesinio.Count - 1)
            {
                n = 0;
```

```
                }
                else
                {
                    n++;
                }

            }
            readyqueue.Enqueue(processesinio.ElementAt(n));
            processesinio.RemoveAt(n);
        }
      }
      printresults(p1, p2, p3, p4, p5, p6, p7, p8);

    }
    public static Queue<Process> readyqueue;
    public static List<Process> processesinio;
    public static float totaltime = 0;
    public static float totalallcpu = 0;
    public static void printexecution(Process p)
    {
      Console.WriteLine("--------------------------------------------------");
      Console.WriteLine("Execution Time is " + totaltime + "          " + p.name + " is Running");
      Console.WriteLine("Processes in Ready Queue\n");
      for (int j = 0; j < readyqueue.Count; j++)
      {
        Console.WriteLine(readyqueue.ElementAt(j).name + " CPU Burst Length " +
readyqueue.ElementAt(j).data.First() + "\n");
      }
      int timeremaining = 0;
      Console.WriteLine("Processes in I/O\n");
      for (int y = 0; y < processesinio.Count; y++)
      {
        timeremaining = processesinio.ElementAt(y).ioburst - processesinio.ElementAt(y).iocounter;
        Console.WriteLine(processesinio.ElementAt(y).name + " Time Remaining in I/O " +
timeremaining + "\n");
      }

    }
    public static void printresults(Process p1, Process p2, Process p3, Process p4, Process p5, Process
p6, Process p7, Process p8)
    {

      int totalwait = p1.wait + p2.wait + p3.wait + p4.wait + p5.wait + p6.wait + p7.wait + p8.wait;
      double waitave = totalwait / 8;
      Console.WriteLine("\n\n\n----------------------------------------\nResults\n\n");
```

```
            Console.WriteLine("Total Time= " + totaltime);
            float cpuutilization = totalallcpu / totaltime;
            cpuutilization = cpuutilization * 100;
            Console.WriteLine("\nCPU Utilization= " + cpuutilization + "%");
            Console.WriteLine("\nWait Times\n");
            Console.WriteLine(" p1= " + p1.wait + " p2= " + p2.wait + " p3= " + p3.wait + " p4= " +
p4.wait + " p5= " + p5.wait + " p6= " + p6.wait + " p7= " + p7.wait + " p8= " + p8.wait);
            Console.WriteLine("Average Wait Time= " + waitave);


            float totalturnaroundtime = p1.turnaroundtime + p2.turnaroundtime + p3.turnaroundtime +
p4.turnaroundtime + p5.turnaroundtime + p6.turnaroundtime + p7.turnaroundtime +
p8.turnaroundtime;
            float turnaroundave = totalturnaroundtime / 8;
            Console.WriteLine("\nTurnaround Times\n");
            Console.WriteLine("p1= " + p1.turnaroundtime + " p2= " + p2.turnaroundtime + " p3= " +
p3.turnaroundtime + " p4= " + p4.turnaroundtime + " p5= " + p5.turnaroundtime + " p6= " +
p6.turnaroundtime + " p7= " + p7.turnaroundtime + " p8= " + p8.turnaroundtime);
            Console.WriteLine("\nAverage Turnaround Time= " + turnaroundave);


            Console.WriteLine("\nResponse Times\n");
            Console.WriteLine("p1= " + p1.responsetime + " p2= " + p2.responsetime + " p3= " +
p3.responsetime + " p4= " + p4.responsetime + " p5= " + p5.responsetime + " p6= " + p6.responsetime
+ " p7= " + p7.responsetime + " p8= " + p8.responsetime);
            float totalresponsetime = p1.responsetime + p2.responsetime + p3.responsetime +
p4.responsetime + p5.responsetime + p6.responsetime + p7.responsetime + p8.responsetime;
            float responseave = totalresponsetime / 8;
            Console.WriteLine("\nAverage Response Time= " + responseave);




        }
    }

}
class Process
{
    public Process(String n, List<int> d)
    {
        name = n;
        data = d;
    }
```

```
    public String name;
    public List<int> data;
    public int wait = 0;
    public int ioburst = 0;
    public int iocounter = 0;

    public bool firstrun = true;
    public int responsetime = 0;

    public int turnaroundtime = 0;
    public int totalcpu = 0;
    public int totalio = 0;

}
```

SJF

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SJF_ProcessScheduler
{
    class Program
    {
        static void Main(string[] args)
        {
            Process p1 = new Process("P1", new List<int> { 4, 24, 5, 73, 3, 31, 5, 27, 4, 33, 6, 43, 4, 64, 5,
19, 2 });
            Process p2 = new Process("P2", new List<int> { 18, 31, 19, 35, 11, 42, 18, 43, 19, 47, 18, 43,
17, 51, 19, 32, 10 });
            Process p3 = new Process("P3", new List<int> { 6, 18, 4, 21, 7, 19, 4, 16, 5, 29, 7, 21, 8, 22, 6,
24, 5 });
            Process p4 = new Process("P4", new List<int> { 17, 42, 19, 55, 20, 54, 17, 52, 15, 67, 12, 72,
15, 66, 14 });
            Process p5 = new Process("P5", new List<int> { 5, 81, 4, 82, 5, 71, 3, 61, 5, 62, 4, 51, 3, 77, 4,
61, 3, 42, 5 });
            Process p6 = new Process("P6", new List<int> { 10, 35, 12, 41, 14, 33, 11, 32, 15, 41, 13, 29,
11 });
            Process p7 = new Process("P7", new List<int> { 21, 51, 23, 53, 24, 61, 22, 31, 21, 43, 20 });
            Process p8 = new Process("P8", new List<int> { 11, 52, 14, 42, 15, 31, 17, 21, 16, 43, 12, 31,
13, 32, 15 });
```

```
        readyqueue = new List<Process>();
        //Add all processes to readyqueue
        readyqueue.Add(p1);
        readyqueue.Add(p2);
        readyqueue.Add(p3);
        readyqueue.Add(p4);
        readyqueue.Add(p5);
        readyqueue.Add(p6);
        readyqueue.Add(p7);
        readyqueue.Add(p8);
        processesinio = new List<Process>();

        bool shortest;

        int k = 1; ;
        while (numprocesscomplete != 8)//Run until all processes complete
        {
           //Find the Process with the Shortest CPU Burst Length
           if (readyqueue.Count != 0)
           {

              Process shortestprocess = readyqueue.ElementAt(0);

              k = 1;
              shortest = false;
              while (!shortest)
              {

                 if (readyqueue.Count == 1)//If there is one Process in the readyqueue than that is the
Shortest Process
                 {
                    shortest = true;
                    shortestprocess = readyqueue.ElementAt(0);
                    readyqueue.RemoveAt(0);

                 }
                 else if (k != readyqueue.Count - 1 && shortestprocess.data.First() >
readyqueue.ElementAt(k).data.First())//If the Process is smaller than the "Shortest Process" then
reassign the "Shortest Process"
                 {
                    shortest = false;
                    shortestprocess = readyqueue.ElementAt(k);

                 }
```

```
                else if (k == readyqueue.Count - 1 && shortestprocess.data.First() <
readyqueue.ElementAt(k).data.First())//If it is comparing the last Process in the ready queue to the
"Smallest Process" and the "Smallest Process" is smaller then it is the smallest
                {
                   shortest = true;
                }
                else if (k == readyqueue.Count - 1 && shortestprocess.data.First() >
readyqueue.ElementAt(k).data.First())//If it is comparing the last Process in the ready queue to the
"Smallest Process" and the "Smallest Process" is bigger then it is the last Process is the smallest
                {
                   shortest = true;
                   shortestprocess = readyqueue.ElementAt(k);
                }
                else if (k != readyqueue.Count - 1 && shortestprocess.data.First() ==
readyqueue.ElementAt(k).data.First())//If there is a tie than the Process that entered the readyqueue
first is the "Smallest Process"
                {
                   if (shortestprocess.timeenteringqueue > readyqueue.ElementAt(k).timeenteringqueue)
                   {
                      shortest = false;
                      shortestprocess = readyqueue.ElementAt(k);
                   }
                }
                else if (k == readyqueue.Count - 1 && shortestprocess.data.First() ==
readyqueue.ElementAt(k).data.First())//If there is a tie than the Process that entered the readyqueue
first is the "Smallest Process", this condition is for comparing with the last element in the readyqueue
                {
                   if (shortestprocess.timeenteringqueue > readyqueue.ElementAt(k).timeenteringqueue)
                   {
                      shortest = true;
                      shortestprocess = readyqueue.ElementAt(k);

                   }
                   else
                   {
                      shortest = true;

                   }

                }

                k++;
             }
             if (shortestprocess.firstrun == true)
             {
```

```
                shortestprocess.responsetime = shortestprocess.wait;
                shortestprocess.firstrun = false;
            }

            readyqueue.Remove(shortestprocess);
            executeprocess(shortestprocess);//Execute the Shortest Process
        }
        else if (processesinio.Count != 0)//Handles CPU Idle incrementing Processes in I/O Burst
counters
        {
            int n = 0;
            while (processesinio.ElementAt(n).iocounter < processesinio.ElementAt(n).ioburst)
            {
                cpuidle++;
                totaltime++;
                processesinio.ElementAt(n).iocounter++;
                if (n == processesinio.Count - 1)
                {
                    n = 0;
                }
                else
                {
                    n++;
                }

            }
            if (processesinio.ElementAt(n).data.Count == 0)
            {
                processesinio.ElementAt(n).turnaroundtime = processesinio.ElementAt(n).totalcpu +
processesinio.ElementAt(n).totalio + processesinio.ElementAt(n).wait;
                numprocesscomplete++;
                processesinio.RemoveAt(n);

            }
            else
            {
                readyqueue.Add(processesinio.ElementAt(n));
                processesinio.RemoveAt(n);
            }
        }
    }
    printresults(p1, p2, p3, p4, p5, p6, p7, p8);


}
```

```
    public static void printexecution(Process p)
    {
        Console.WriteLine("-------------------------------------------------");
        Console.WriteLine("Execution Time is " + totaltime + "           " + p.name + " is Running");
        Console.WriteLine("Processes in Ready Queue\n");
        for (int j = 0; j < readyqueue.Count; j++)
        {
            Console.WriteLine(readyqueue.ElementAt(j).name + " CPU Burst Length " +
readyqueue.ElementAt(j).data.First() + "\n");
        }
        int timeremaining = 0;
        Console.WriteLine("Processes in I/O\n");
        for (int y = 0; y < processesinio.Count; y++)
        {
            timeremaining = processesinio.ElementAt(y).ioburst - processesinio.ElementAt(y).iocounter;
            Console.WriteLine(processesinio.ElementAt(y).name + " Time Remaining in I/O " +
timeremaining + "\n");
        }

    }
    public static void printresults(Process p1, Process p2, Process p3, Process p4, Process p5, Process
p6, Process p7, Process p8)
    {

        int totalwait = p1.wait + p2.wait + p3.wait + p4.wait + p5.wait + p6.wait + p7.wait + p8.wait;
        double waitave = totalwait / 8;
        Console.WriteLine("\n\n\n----------------------------------------\nResults\n\n");
        Console.WriteLine("Total Time= " + totaltime);
        float cpuutilization = totalallcpu / totaltime;
        cpuutilization = cpuutilization * 100;
        Console.WriteLine("\nCPU Utilization= " + cpuutilization + "%");
        Console.WriteLine("\nWait Times\n");
        Console.WriteLine(" p1= " + p1.wait + " p2= " + p2.wait + " p3= " + p3.wait + " p4= " +
p4.wait + " p5= " + p5.wait + " p6= " + p6.wait + " p7= " + p7.wait + " p8= " + p8.wait);
        Console.WriteLine("Average Wait Time= " + waitave);


        float totalturnaroundtime = p1.turnaroundtime + p2.turnaroundtime + p3.turnaroundtime +
p4.turnaroundtime + p5.turnaroundtime + p6.turnaroundtime + p7.turnaroundtime +
p8.turnaroundtime;
        float turnaroundave = totalturnaroundtime / 8;
        Console.WriteLine("\nTurnaround Times\n");
        Console.WriteLine("p1= " + p1.turnaroundtime + " p2= " + p2.turnaroundtime + " p3= " +
p3.turnaroundtime + " p4= " + p4.turnaroundtime + " p5= " + p5.turnaroundtime + " p6= " +
p6.turnaroundtime + " p7= " + p7.turnaroundtime + " p8= " + p8.turnaroundtime);
```

```
        Console.WriteLine("\nAverage Turnaround Time= " + turnaroundave);

        Console.WriteLine("\nResponse Times\n");
        Console.WriteLine("p1= " + p1.responsetime + " p2= " + p2.responsetime + " p3= " +
p3.responsetime + " p4= " + p4.responsetime + " p5= " + p5.responsetime + " p6= " + p6.responsetime
+ " p7= " + p7.responsetime + " p8= " + p8.responsetime);
        float totalresponsetime = p1.responsetime + p2.responsetime + p3.responsetime +
p4.responsetime + p5.responsetime + p6.responsetime + p7.responsetime + p8.responsetime;
        float responseave = totalresponsetime / 8;
        Console.WriteLine("\nAverage Response Time= " + responseave);




    }
    public static void executeprocess(Process p)
    {
        int cpuburst = p.data.First();
        p.totalcpu = p.totalcpu + cpuburst;
        p.data.RemoveAt(0);
        printexecution(p);
        for (int i = 0; i < cpuburst; i++)//Loop until CPU Burst Completes
        {
            totalallcpu++;
            totaltime++;

            for (int k = 0; k < processesinio.Count; k++)//Increment Processes in I/O Burst counters
            {
                processesinio.ElementAt(k).iocounter++;


                if (processesinio.ElementAt(k).iocounter == processesinio.ElementAt(k).ioburst)//Check
to see if a Process has completed its I/O burst
                {
                    if (processesinio.ElementAt(k).data.Count == 0)//If I/O was the last action needed to do
by the Process than the Process has completed
                    {
                        p.turnaroundtime = p.totalcpu + p.totalio + p.wait;
                        numprocesscomplete++;
                        processesinio.RemoveAt(k);
                        Console.WriteLine("\nCompleted Execution= YES");
                        Console.WriteLine("\n-------------------------------------------------\n\n\n");

                    }
```

```
                else//If I/O was not the last action needed to do by the Process then it is added to the readyqueue and removed from the I/O list
                {

                    processesinio.ElementAt(k).ioburst = 0;
                    processesinio.ElementAt(k).iocounter = 0;
                    processesinio.ElementAt(k).timeenteringqueue = totaltime;

                    readyqueue.Add(processesinio.ElementAt(k));

                    processesinio.RemoveAt(k);
                }
            }

        }

        for (int j = 0; j < readyqueue.Count; j++)//Increments Processes' Wait times
        {
            readyqueue.ElementAt(j).wait++;

        }

    }
    if (p.data.Count != 0)//Checks to see if the process has not completed and if so puts it in I/O
    {

        p.ioburst = p.data[0];
        p.totalio = p.totalio + p.ioburst;
        p.data.RemoveAt(0);
        processesinio.Add(p);
        Console.WriteLine("\nCompleted Execution= NO");
        Console.WriteLine("\n-------------------------------------------------\n\n\n");
    }
    else//Checks to see if the process has completed
    {
        p.turnaroundtime = p.totalcpu + p.totalio + p.wait;
        numprocesscomplete++;
        Console.WriteLine("\nCompleted Execution= YES");
        Console.WriteLine("\n-------------------------------------------------\n\n\n");
    }
}

public static List<Process> processesinio;
public static List<Process> readyqueue;
public static int totaltime = 0;
```

```
        public static int numprocesscomplete = 0;
        public static int round = 0;
        public static float totalallcpu = 0;
        public static float cpuidle = 0;
    }
    class Process
    {
        public Process(String n, List<int> d)
        {
            name = n;
            data = d;
        }
        public String name;
        public List<int> data;
        public int wait = 0;
        public int ioburst = 0;
        public int iocounter = 0;
        public int timeenteringqueue = 0;

        public bool firstrun = true;
        public int responsetime = 0;

        public int turnaroundtime = 0;
        public int totalcpu = 0;
        public int totalio = 0;
    }
}
```

MLFQ

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MLFQ_ProcessScheduler
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
        Process p1 = new Process("P1", new List<int> { 4, 24, 5, 73, 3, 31, 5, 27, 4, 33, 6, 43, 4, 64, 5,
19, 2 });
        Process p2 = new Process("P2", new List<int> { 18, 31, 19, 35, 11, 42, 18, 43, 19, 47, 18, 43,
17, 51, 19, 32, 10 });
        Process p3 = new Process("P3", new List<int> { 6, 18, 4, 21, 7, 19, 4, 16, 5, 29, 7, 21, 8, 22, 6,
24, 5 });
        Process p4 = new Process("P4", new List<int> { 17, 42, 19, 55, 20, 54, 17, 52, 15, 67, 12, 72,
15, 66, 14 });
        Process p5 = new Process("P5", new List<int> { 5, 81, 4, 82, 5, 71, 3, 61, 5, 62, 4, 51, 3, 77, 4,
61, 3, 42, 5 });
        Process p6 = new Process("P6", new List<int> { 10, 35, 12, 41, 14, 33, 11, 32, 15, 41, 13, 29,
11 });
        Process p7 = new Process("P7", new List<int> { 21, 51, 23, 53, 24, 61, 22, 31, 21, 43, 20 });
        Process p8 = new Process("P8", new List<int> { 11, 52, 14, 42, 15, 31, 17, 21, 16, 43, 12, 31,
13, 32, 15 });
        MLFQ_Queue queue1 = new MLFQ_Queue(1);
        MLFQ_Queue queue2 = new MLFQ_Queue(2);
        MLFQ_Queue queue3 = new MLFQ_Queue(3);
        //Initialize queue references to each other and set lower priority queue references
        queue1.queue1 = queue1;
        queue1.queue2 = queue2;
        queue1.queue3 = queue3;
        queue1.lowerpriority = queue2;
        queue2.queue1 = queue1;
        queue2.queue2 = queue2;
        queue2.queue3 = queue3;
        queue2.lowerpriority = queue3;
        queue3.queue1 = queue1;
        queue3.queue2 = queue2;
        queue3.queue3 = queue3;
        queue3.lowerpriority = null;
        //Add all processes to queue1
        queue1.Enqueue(p1);
        queue1.Enqueue(p2);
        queue1.Enqueue(p3);
        queue1.Enqueue(p4);
        queue1.Enqueue(p5);
        queue1.Enqueue(p6);
        queue1.Enqueue(p7);
        queue1.Enqueue(p8);
        processesinio = new List<Process>();
        bool finished = false;
        while (finished == false)
        {
```

```
            executequeue1(queue1);//Execute the Processes in queue1 using Round Robin with TQ 6
            executequeue2(queue2);//Execute the Processes in queue2 using Round Robin with TQ 11
            executequeue3(queue3);//Execute the Processes in queue3 using FCFS
            if (queue1.Count == 0 && queue2.Count == 0 && queue3.Count == 0 &&
numprocesscomplete != 8 && processesinio.Count != 0)//Execute if CPU is idle
            {
               cpuidle(queue1);
            }
            if (queue1.Count == 0 && queue2.Count == 0 && queue3.Count == 0 &&
numprocesscomplete != 8 && processesinio.Count == 0)//Condition if all processes have completed
            {
               printresults(p1, p2, p3, p4, p5, p6, p7, p8);
               finished = true;
            }
         }

      }
      public static void executequeue1(MLFQ_Queue queue)
      {

         reverttohigherpriority = 0;//Reset priority variable
         while (queue.Count != 0)
         {

            Process p = queue.Dequeue();
            printexecution(p, queue, queue.queue2, queue.queue3);
            if (p.firstrun == true)
            {
               p.responsetime = p.wait;
               p.firstrun = false;
            }
            roundrobin(queue, p);//Execute the Processes in queue1 using Round Robin with TQ 6


         }
      }
      public static void executequeue2(MLFQ_Queue queue)
      {
         while (queue.Count != 0 && reverttohigherpriority != 1)//Run while queue1 does not have any
processes
         {
            Process p = null;

            p = queue.Dequeue();
            printexecution(p, queue.queue1, queue, queue.queue3);
```

```
            if (p.firstrun == true)
            {
                p.responsetime = p.wait;
                p.firstrun = false;
            }
            roundrobin(queue, p);//Execute the Processes in queue2 using Round Robin with TQ 11

        }
    }
    public static void executequeue3(MLFQ_Queue queue)
    {
        while (queue.Count != 0 && reverttohigherpriority != 1 && reverttohigherpriority != 2)//Run
while queue1 and queue2 do not have any processes
        {
            Process p = queue.Dequeue();
            printexecution(p, queue.queue1, queue.queue2, queue);
            if (p.firstrun == true)
            {
                p.responsetime = p.wait;
                p.firstrun = false;
            }
            fcfs(queue, p);//Execute the Processes in queue3 using FCFS

        }

    }
    public static void cpuidle(MLFQ_Queue queue1)//Handles CPU Idle incrementing Processes in
I/O Burst counters
    {
        if (processesinio.Count != 0)
        {
            int n = 0;
            while (processesinio.ElementAt(n).iocounter < processesinio.ElementAt(n).ioburst)
            {
                cpu_idle++;

                totaltime++;

                processesinio.ElementAt(n).iocounter++;
                if (n == processesinio.Count - 1)
                {
                    n = 0;
                }
                else
```

```
            {
               n++;
            }

         }
         queue1.Enqueue(processesinio.ElementAt(n));
         processesinio.RemoveAt(n);
      }

   }
   public static void fcfs(MLFQ_Queue queue, Process p)
   {
      int cpuburst = 0;
      if (p.cpuburstleft == 0)
      {
         cpuburst = p.data[0];
         p.data.RemoveAt(0);
      }
      for (int i = cpuburst; i < cpuburst; i++)//Loop until CPU Burst Completes
      {
         p.cpuburstleft = cpuburst - i;
         totalallcpu++;
         totaltime++;

         for (int k = 0; k < processesinio.Count; k++)//Increment Processes in I/O Burst counters
         {
            processesinio.ElementAt(k).iocounter++;


            if (processesinio.ElementAt(k).iocounter == processesinio.ElementAt(k).ioburst)//Check
to see if a Process has completed its I/O burst
            {
               processesinio.ElementAt(k).ioburst = 0;
               processesinio.ElementAt(k).iocounter = 0;
               queue.queue1.Enqueue(processesinio.ElementAt(k));
               processesinio.RemoveAt(k);

            }


         }

         incrementwaittimes(queue);
```

```
        }

        if (p.data.Count != 0)//Checks to see if the process has not completed and if so puts it in I/O
        {
            p.ioburst = p.data[0];
            p.totalio = p.totalio + p.ioburst;
            p.data.RemoveAt(0);
            processesinio.Add(p);
            Console.WriteLine("\nCompleted Execution= NO");
            Console.WriteLine("\n-------------------------------------------------\n\n\n");
        }
        else//Checks to see if the process has completed
        {
            p.turnaroundtime = p.totalcpu + p.totalio + p.wait;
            numprocesscomplete++;
            Console.WriteLine("\nCompleted Execution= YES");
            Console.WriteLine("\n-------------------------------------------------\n\n\n");
        }
        if (queue.queue1.Count != 0)//Causes queue3 to stop executing its Processes because queue1
has a process
        {


            reverttohigherpriority = 1;
            return;
        }
        else if (queue.queue2.Count != 0)//Causes queue3 to stop executing its Processes because
queue2 has a process
        {


            reverttohigherpriority = 2;
            return;
        }

    }
    public static void roundrobin(MLFQ_Queue queue, Process p)
    {
        int cpuburst = 0;
        if (p.data.Count == 0)
        {
            p.turnaroundtime = p.totalcpu + p.totalio + p.wait;
            numprocesscomplete++;
        }
        else if (p.cpuburstleft == 0)//If there is no CPU burst left from a previous execution
```

```
        {
            cpuburst = p.data[0];
            p.data.RemoveAt(0);
        }
        else//Continue the CPU burst where the process left off
        {

            cpuburst = p.cpuburstleft;
            p.cpuburstleft = 0;
        }
        int i = 0;

        int tq = 0;
        int rununtil = 0;
        if (queue.queuenumber == 1)//Assign TQ to 6 if this is queue1
        {
            tq = 6;

        }
        else if (queue.queuenumber == 2)//Assign TQ to 11 if this is queue2
        {
            tq = 11;
        }
        if (tq >= cpuburst)//If CPU Burst is less than TQ only run until CPU burst length
        {
            rununtil = cpuburst;
            p.cpuburstleft = 0;
        }
        else//Otherwise run until TQ
        {
            rununtil = tq;
            p.cpuburstleft = cpuburst - tq;
        }
        for (i = 0; i < rununtil; i++)//Loop until CPU Burst Completes or until TQ finishes depending on
which is less
        {


            totalallcpu++;
            totaltime++;

            for (int k = 0; k < processesinio.Count; k++)//Increment Processes in I/O Burst counters
            {
                processesinio.ElementAt(k).iocounter++;
```

```
            if (processesinio.ElementAt(k).iocounter == processesinio.ElementAt(k).ioburst)//Check
to see if a Process has completed its I/O burst
            {
                processesinio.ElementAt(k).ioburst = 0;
                processesinio.ElementAt(k).iocounter = 0;
                queue.queue1.Enqueue(processesinio.ElementAt(k));
                processesinio.RemoveAt(k);

            }


        }

        incrementwaittimes(queue);

    }
    if (p.cpuburstleft != 0)//If the process did not finish before TQ ended, move it to a lower priority
queue
    {
        queue.lowerpriority.Enqueue(p);
    }
    else if (p.data.Count != 0 && p.cpuburstleft == 0)// Checks to see if the process has not
completed and if so puts it in I/O
    {
        p.ioburst = p.data[0];
        p.totalio = p.totalio + p.ioburst;
        p.data.RemoveAt(0);
        processesinio.Add(p);
        Console.WriteLine("\nCompleted Execution= NO");
        Console.WriteLine("\n--------------------------------------------------\n\n\n");
    }
    else if (p.data.Count == 0 && p.cpuburstleft == 0)
    {
        p.turnaroundtime = p.totalcpu + p.totalio + p.wait;
        numprocesscomplete++;
        Console.WriteLine("\nCompleted Execution= YES");
        Console.WriteLine("\n--------------------------------------------------\n\n\n");
    }
    if (queue.queuenumber == 2)
    {
        if (queue.queue1.Count != 0)//Causes queue2 to stop executing its Processes because queue1
has a process
        {
```

```
                reverttohigherpriority = 1;
                return;
            }
        }
    }
    public static void printexecution(Process p, MLFQ_Queue queue1, MLFQ_Queue queue2,
MLFQ_Queue queue3)
    {
        Console.WriteLine("--------------------------------------------------");
        Console.WriteLine("Execution Time is " + totaltime + "           " + p.name + " is Running");
        if (queue1.Count != 0)
        {
            for (int j = 0; j < queue1.Count; j++)
            {
                if (queue1.ElementAt(j).data.Count != 0)
                {
                    Console.WriteLine(queue1.ElementAt(j).name + " CPU Burst Length " +
queue1.ElementAt(j).data.First() + " is inside of Queue 1" + "\n");
                }
            }
        }
        if (queue2.Count != 0)
        {
            for (int k = 0; k < queue2.Count; k++)
            {
                if (queue2.ElementAt(k).data.Count != 0)
                {
                    Console.WriteLine(queue2.ElementAt(k).name + " CPU Burst Length " +
queue2.ElementAt(k).data.First() + " is inside of Queue 2" + "\n");
                }
            }
        }
        if (queue3.Count != 0)
        {
            for (int f = 0; f < queue3.Count; f++)
            {
                if (queue3.ElementAt(f).data.Count != 0)
                {
                    Console.WriteLine(queue3.ElementAt(f).name + " CPU Burst Length " +
queue3.ElementAt(f).data.First() + " is inside of Queue 3" + "\n");
                }
            }
        }
        int timeremaining = 0;
        Console.WriteLine("Processes in I/O\n");
```

```
        for (int y = 0; y < processesinio.Count; y++)
        {
           timeremaining = processesinio.ElementAt(y).ioburst - processesinio.ElementAt(y).iocounter;
           Console.WriteLine(processesinio.ElementAt(y).name + " Time Remaining in I/O " +
timeremaining + "\n");
        }

      }
      public static void printresults(Process p1, Process p2, Process p3, Process p4, Process p5, Process
p6, Process p7, Process p8)
      {

        int totalwait = p1.wait + p2.wait + p3.wait + p4.wait + p5.wait + p6.wait + p7.wait + p8.wait;
        double waitave = totalwait / 8;
        Console.WriteLine("\n\n\n----------------------------------------\nResults\n\n");
        Console.WriteLine("Total Time= " + totaltime);
        float cpuutilization = totalallcpu / totaltime;
        cpuutilization = cpuutilization * 100;
        Console.WriteLine("\nCPU Utilization= " + cpuutilization + "%");
        Console.WriteLine("\nWait Times\n");
        Console.WriteLine(" p1= " + p1.wait + " p2= " + p2.wait + " p3= " + p3.wait + " p4= " +
p4.wait + " p5= " + p5.wait + " p6= " + p6.wait + " p7= " + p7.wait + " p8= " + p8.wait);
        Console.WriteLine("Average Wait Time= " + waitave);


        float totalturnaroundtime = p1.turnaroundtime + p2.turnaroundtime + p3.turnaroundtime +
p4.turnaroundtime + p5.turnaroundtime + p6.turnaroundtime + p7.turnaroundtime +
p8.turnaroundtime;
        float turnaroundave = totalturnaroundtime / 8;
        Console.WriteLine("\nTurnaround Times\n");
        Console.WriteLine("p1= " + p1.turnaroundtime + " p2= " + p2.turnaroundtime + " p3= " +
p3.turnaroundtime + " p4= " + p4.turnaroundtime + " p5= " + p5.turnaroundtime + " p6= " +
p6.turnaroundtime + " p7= " + p7.turnaroundtime + " p8= " + p8.turnaroundtime);
        Console.WriteLine("\nAverage Turnaround Time= " + turnaroundave);

        Console.WriteLine("\nResponse Times\n");
        Console.WriteLine("p1= " + p1.responsetime + " p2= " + p2.responsetime + " p3= " +
p3.responsetime + " p4= " + p4.responsetime + " p5= " + p5.responsetime + " p6= " + p6.responsetime
+ " p7= " + p7.responsetime + " p8= " + p8.responsetime);
        float totalresponsetime = p1.responsetime + p2.responsetime + p3.responsetime +
p4.responsetime + p5.responsetime + p6.responsetime + p7.responsetime + p8.responsetime;
        float responseave = totalresponsetime / 8;
        Console.WriteLine("\nAverage Response Time= " + responseave);
```

```
}

public static void incrementwaittimes(MLFQ_Queue queue)
{


   if (queue.queuenumber == 1)
   {
      for (int j = 0; j < queue.Count; j++)
      {
         queue.ElementAt(j).wait++;
      }
      for (int j = 0; j < queue.queue2.Count; j++)
      {

         queue.queue2.ElementAt(j).wait++;
      }
      for (int j = 0; j < queue.queue3.Count; j++)
      {
         queue.queue3.ElementAt(j).wait++;
      }
   }
   else if (queue.queuenumber == 2)
   {
      for (int j = 0; j < queue.Count; j++)
      {
         queue.ElementAt(j).wait++;
      }
      for (int j = 0; j < queue.queue1.Count; j++)
      {

         queue.queue1.ElementAt(j).wait++;
      }
      for (int j = 0; j < queue.queue3.Count; j++)
      {
         queue.queue3.ElementAt(j).wait++;
      }
   }
   else if (queue.queuenumber == 3)
   {
      for (int j = 0; j < queue.Count; j++)
      {
```

```
            queue.ElementAt(j).wait++;
        }
        for (int j = 0; j < queue.queue1.Count; j++)
        {

            queue.queue1.ElementAt(j).wait++;
        }
        for (int j = 0; j < queue.queue2.Count; j++)
        {
            queue.queue2.ElementAt(j).wait++;
        }
    }

  }
  public static List<Process> processesinio;
  public static float totaltime = 0;
  public static float totalallcpu = 0;
  public static int reverttohigherpriority = 0;
  public static int numprocesscomplete = 0;
  public static float cpu_idle = 0;

}
class MLFQ_Queue : Queue<Process>
{
  public int queuenumber;
  public MLFQ_Queue queue1;
  public MLFQ_Queue queue2;
  public MLFQ_Queue queue3;
  public MLFQ_Queue lowerpriority;
  public MLFQ_Queue(int number)
  {
    queuenumber = number;
  }

}
class Process
{
  public Process(String n, List<int> d)
  {
    name = n;
    data = d;
  }
  public String name;
  public List<int> data;
  public int cpuburstleft = 0;
```

```
        public int wait = 0;
        public int ioburst = 0;
        public int iocounter = 0;

        public bool firstrun = true;
        public int responsetime = 0;

        public int turnaroundtime = 0;
        public int totalcpu = 0;
        public int totalio = 0;

    }
}
```