

```
1 namespace OS_Prog1 {  
2     public class RR : Scheduler {  
3  
4     }  
5 }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4
5 namespace OS_Prog1 {
6
7
8     public class Sim {
9
10         public static Sim Singleton { get; private set; }
11
12         protected List<Event> eventQueue = new List<Event>
13         >();
14         private Scheduler scheduler;
15
16         List<Process> processes = new List<Process>();
17
18         //initialize fields needed for loop
19         private Process currentProcess;
20         private double clock;
21         private double nextClock, idleTime;
22
23         public Sim(int schedulerType) {
24
25             Singleton = this;
26
27             Init(schedulerType);
28             Run();
29             GenerateReport();
30         }
31
32         private void Init(int schedulerType) {
33
34             //Init Scheduler
35             switch (schedulerType) {
36                 case 1:
37                     scheduler = new FCFS();
38                     break;
39                 case 2:
40                     scheduler = new SRTF();
41                     break;
42                 case 3:
43                     scheduler = new HRRN();
44                     break;
45                 case 4:
```

```

46         scheduler = new RR();
47         break;
48     default:
49         scheduler = new Scheduler(); //shouldn'
        t ever happen
50         break;
51     }
52
53     //generate list of processes
54     scheduler.SetupProcesses();
55     processes = scheduler.processList;
56
57     //start first process
58     currentProcess = processes[0];
59     clock = currentProcess.arrival;
60     nextClock = clock + currentProcess.burst;
61     currentProcess.start = currentProcess.arrival;
62
63 }
64
65 private void Run() {
66
67
68     foreach (Event arrEvent in scheduler.
arrivalEvents) {
69
70         arrEvent.myProc.arrival = arrEvent.time;
71
72         //handle time-slice TODO
73
74         //handle arrival
75         eventQueue.Add(arrEvent);
76         scheduler.EnqueueReady(arrEvent.myProc); //
        process arrived - have scheduler handle
77
78         //handle completion
79         if (arrEvent.time >= nextClock)
80             StartProcess(scheduler.DequeueFromReady
81             ());
82     }
83
84     //handle ready queue once events are finished
        arriving
85     while (!scheduler.ReadyQueueEmpty()) {
86         StartProcess(scheduler.DequeueFromReady());

```

```

87         }
88
89         foreach (Event eve in eventQueue) {
90             //             Debug.Log($"{eve}");
91         }
92
93     }
94
95     private void StartProcess(Process proc) {
96
97         //         proc.start = clock;
98         //         clock = nextClock;
99         //         nextClock = proc.start + proc.burst;
100
101         proc.start = clock > proc.arrival ? clock :
        proc.arrival;
102         clock = nextClock;
103         nextClock = proc.start + proc.burst;
104
105         //enqueue completion event for finished
        process
106         if (currentProcess != null) {
107             currentProcess.completion = clock;
108             currentProcess.turnAround = currentProcess
        .completion - currentProcess.start;
109             eventQueue.Add(new Event(currentProcess,
        EventType.Completion, nextClock));
110         }
111
112
113
114         currentProcess = proc;
115
116     }
117
118     private void GenerateReport() {
119
120         StreamWriter writer = new StreamWriter("./sim.
        data");
121
122         foreach (Process process in processes) {
123             writer.WriteLine(process);
124             //             Debug.Log($"{process}");
125         }
126
127

```

```
128         }  
129  
130  
131     }  
132 }
```

```
1 using System;
2 using System.Collections.Generic;
3
4 namespace OS_Prog1 {
5     public class FCFS : Scheduler{
6         //FCFS is implemented as the default scheduler
7     }
8 }
```

```
1 namespace OS_Prog1 {  
2     public class HRRN : Scheduler{  
3  
4     }  
5 }
```

```
1 namespace OS_Prog1 {  
2     public class SRTF : Scheduler{  
3  
4     }  
5 }
```



```
1 using System;
2
3 namespace OS_Prog1 {
4
5     public enum DebugLevel {
6         Message,
7         Urgent,
8         Error
9     }
10
11     public static class Debug {
12
13         public static DebugLevel debugLevel = DebugLevel.
Message;
14
15         public static void Log(string msg) {
16             Log(msg, DebugLevel.Message);
17         }
18
19         public static void Log(string msg, DebugLevel
msgLevel) {
20
21             if (debugLevel <= msgLevel) {
22                 Console.WriteLine(msg);
23             }
24
25         }
26
27     }
28 }
```

```
1 using System;
2
3 namespace OS_Prog1 {
4     class Program {
5
6         public static float lambda, serviceTime, quantum;
7         static void Main(string[] args) {
8
9             int schedulerType;
10
11             if (!int.TryParse(args[0], out schedulerType
12         )) {
13                 WriteError(args[0], 0);
14             }
15
16             if (!float.TryParse(args[1], out lambda)) {
17                 WriteError(args[1], 1);
18             }
19
20             if (!float.TryParse(args[2], out serviceTime
21         )) {
22                 WriteError(args[2], 2);
23             }
24
25             if (!float.TryParse(args[3], out quantum)) {
26                 WriteError(args[3], 3);
27             }
28
29             Sim sim = new Sim(schedulerType);
30
31             private static void WriteError(object arg, int
32         index) {
33                 Console.WriteLine($"ERROR: failed to parse {arg
34         } from arg[{index}]");
35             }
36 }
```

```
1 using System;
2 using System.Collections.Generic;
3
4 namespace OS_Prog1 {
5
6     public enum EventType {Arrival, Completion, Slice}
7
8     public class Event {
9
10         public Process myProc;
11         public EventType type;
12         public double time;
13
14         public Event(Process myProc, EventType type, double
time) {
15             this.myProc = myProc;
16             this.type = type;
17             this.time = time;
18         }
19
20         public Event(Process myProc, EventType type) {
21             this.myProc = myProc;
22             this.type = type;
23             switch (type) {
24                 case EventType.Arrival:
25                     time = myProc.arrival;
26                     break;
27                 case EventType.Completion:
28                     break;
29                 default:
30
31                     break;
32             }
33         }
34
35         public override string ToString() {
36
37             string typeStr = "";
38             switch (type) {
39                 case EventType.Arrival:
40                     typeStr = "Arrival";
41                     break;
42                 case EventType.Completion:
43                     typeStr = "Completion";
44                     break;
45                 case EventType.Slice:
```

```

46         typeStr = "Slice";
47         break;
48     } //set type str
49
50     string msg = $"\\n=====\\nEvent: {myProc.id}";
51
52     msg += $"\\nMy type: {typeStr}";
53     msg += $"\\nMy time: {time}";
54     msg += "\\n=====\\n";
55
56
57     return msg;
58 }
59 }
60
61 public class Scheduler {
62
63     public List<Event> arrivalEvents = new List<Event
64 >();
65     public List<Process> processList = new List<Process
66 >();
67     public List<Process> readyList = new List<Process
68 >(); //acts like a priority queue - this is my "readyQueue"
69
70     protected float clock;
71     protected ProcessGenerator processGenerator;
72
73     public virtual void EnqueueReady(Process proc) { //
74         insert into priority queue logic
75         readyList.Insert(0, proc);
76     }
77
78     public virtual Process DequeueFromReady() {
79
80         if (readyList.Count <= 0)
81             return null;
82
83         Process end = readyList[^1];
84         readyList.Remove(end);
85         return end;
86     }
87
88     public bool ReadyQueueEmpty() {
89         return (readyList.Count <= 0);
90     }
91 }

```

```

88
89     public virtual void SetupProcesses() {
90
91         processGenerator = new ProcessGenerator(10000
92     );
93         processGenerator.Generate();
94         processList = processGenerator.processes;
95         //         foreach (Process process in processList) {
96         //             Debug.Log($"{process}");
97         //         }
98
99         GetArrivalEvents();
100
101     }
102
103     private void GetArrivalEvents() {
104         for (var i = 0; i < processList.Count; i++) {
105             Process proc = processList[i];
106             Event lastEvent = null;
107             if (arrivalEvents.Count > 0)
108                 lastEvent = arrivalEvents[i-1];
109             double lastArrTime = lastEvent == null ? 0
110 : lastEvent.time;
111             Event arrEvent = new Event(proc, EventType
112 .Arrival, proc.arrival + lastArrTime);
113             arrivalEvents.Add(arrEvent);
114         }
115     }
116
117     public override string ToString() {
118         return processGenerator.ToString();
119     }
120 }

```

```
1 <Project Sdk="Microsoft.NET.Sdk">
2
3   <PropertyGroup>
4     <OutputType>Exe</OutputType>
5     <TargetFramework>netcoreapp3.0</TargetFramework>
6     <RootNamespace>OS_Prog1</RootNamespace>
7   </PropertyGroup>
8
9 </Project>
10
```

```

1 using System;
2 using System.Collections.Generic;
3
4 namespace OS_Prog1 {
5     public class Process {
6         public int id;
7         public double arrival, burst, start, completion,
            turnAround, timeRem;
8
9         public override string ToString() {
10             string msg = $"\\n=====\\nProcess: {id}";
11             msg += $"\\nMy arrival: {arrival}";
12             msg += $"\\nMy burst time: {burst}";
13             msg += $"\\nMy Start time: {start}";
14             msg += $"\\nMy completion time: {completion}";
15             msg += $"\\nMy turn Around: {turnAround}";
16             // msg += $"\\nMy time rem: {timeRem}";
17             msg += "\\n=====\\n";
18
19             return msg;
20         }
21     }
22
23     public class ProcessGenerator {
24
25         private int count;
26         private float lambda, servTime;
27
28         public List<Process> processes = new List<Process
29 >();
30
31         public ProcessGenerator(int count) {
32             this.count = count;
33             lambda = Program.lambda;
34             servTime = Program.serviceTime;
35         }
36
37         public void Generate() {
38
39             for (int i = 0; i < count; i++) {
40
41                 Process proc = new Process();
42                 proc.arrival = RandomExp(lambda);
43                 proc.burst = RandomExp(servTime);
44                 proc.timeRem = proc.burst;
45                 proc.id = i;

```

```

45
46         processes.Add(proc);
47
48     }
49
50 }
51
52 private float RandomExp(float t) {
53
54     /*
55     * float u,x;
56     x = 0;
57     while (x == 0)
58     {
59         u = urand();
60         x = (-1/Lambda)*Log(u);
61     }
62     return(x);
63     */
64     float u = 0, x = 0;
65
66     while (x <= 0) {
67         u = (float) new Random().NextDouble();
68         x = (-1 / t) * MathF.Log(u);
69     }
70
71     return x;
72
73 }
74
75
76 public override string ToString() {
77
78     string msg = $"{\n=====Process Generator:}";
79
80     foreach (Process process in processes) {
81         msg += process.ToString();
82     }
83
84     msg += $"{\n=====}";
85
86     return msg;
87
88 }
89
90 }

```


91

92 }