**Operating System Course**

Report for Lab2

****

|  |  |
| --- | --- |
| Student Name | Student Number |
| 马洪升 | 20175188 |

2018-12

**Catalogue**

[1. Experiment Basic Information 3](#_Toc532907678)

[1.1 Theme 3](#_Toc532907679)

[1.2 Purpose 3](#_Toc532907680)

[1.3 Round Robin Scheduling Algorithm 3](#_Toc532907681)

[2. Flowchart & Methodology 4](#_Toc532907682)

[2.1 RR Scheduling Flowchart 4](#_Toc532907683)

[2.2 Methodology 4](#_Toc532907684)

[3. Result 6](#_Toc532907685)

[4. Conclusion 8](#_Toc532907686)

[4.1 Analysis: Quantum time works best 8](#_Toc532907687)

[4.2 Analysis: Quantum time more than longest process 9](#_Toc532907688)

[4.3 Summary 9](#_Toc532907689)

[Appendix 10](#_Toc532907690)

## 1. Experiment Basic Information

### 1.1 Theme

Round Robin Scheduling Algorithm Analyses

### 1.2 Purpose

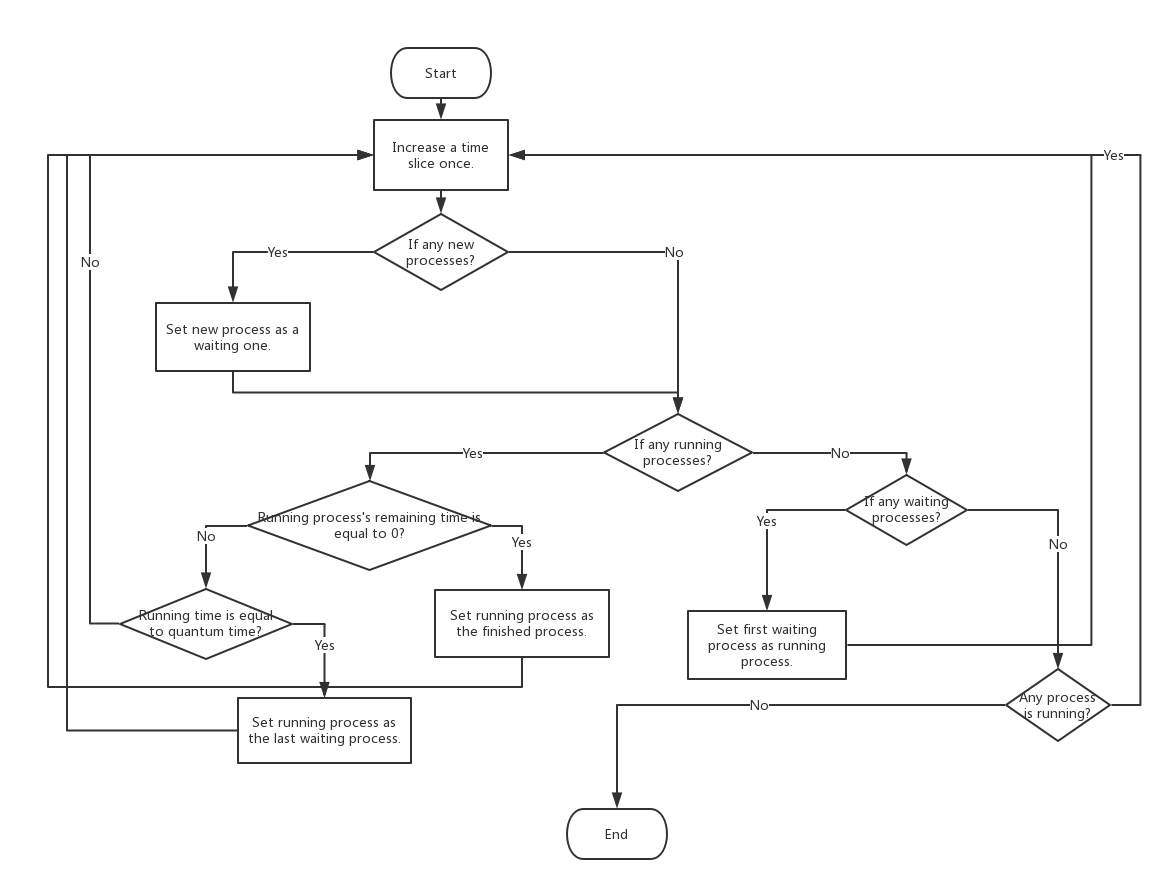
1. Learn about Operating System Uniprocessor Scheduling.
2. Construct a simulation environment.
3. Explore the differences in normalized turn-around time performance while using the round robin scheduling algorithm with various time quantum.
4. Implement a Round Robin scheduler using Java.
5. Find the best quantum with the same processes.

### 1.3 Round Robin Scheduling Algorithm

Round-robin (RR) is one of the algorithms employed by process and network schedulers in computing. As the term is generally used, time quantum is assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive). Round-robin scheduling is simple, easy to implement, and starvation-free. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks.

## 2. Flowchart & Methodology

### 2.1 RR Scheduling Flowchart



### 2.2 Methodology

1. **Achieve Algorithm**

To begin with, I use three queues to store the processes. The queue which stores the new processes is a priority queue. It will sort the process by its arrival time.

From the above flowchart we can see that there are many Judgement sentences, and the whole program is a circle. All of these ensure that the program will run correctly.

Before we start, we will input a time quantum. There are theories such as “The time quantum should be slightly greater than the time required for a typical interaction or process function” and “Round robin degenerates to FCFS if a time quantum is longer than the longest-running process”. In order to find the best performance, we input the time quantum from 1 to 50.

1. **Codes**
2. **Declare 3 queues to store processes in different situation.**

|  |
| --- |
| **static** Queue<Process> *newProcess*; *// We use this Queue to store the process that is new.* **static** Queue<Process> *waitProcess*; *// We use this Queue to store the process that doesn't finish.* **static** Queue<Process> *finishedProcess*; *// We use this Queue to store finished processes.* |

1. **Add all the processes reached by the “arrival time” to the waitProcess queue’s header.**

|  |
| --- |
| **while**(!*newProcess*.isEmpty()) {  **if**(*newProcess*.peek().getArriveTime() <= currTime) {  *waitProcess*.add(*newProcess*.poll());  }  **else** {  **break**;  } } |

1. **If the remain run time of the process is greater than 0. We add it into waitProcess queue's Tail.**

|  |
| --- |
| **if**(currProcess.getLastRunTime() > 0){  *waitProcess*.add(currProcess); } *// When the waitProcess queue is not empty. Poll and run!* **if**(!*waitProcess*.isEmpty()) {  currProcess = *waitProcess*.poll();  currTime = executeProcess(currProcess, currTime); } **else** {  *// There is currently no process execution,  // but there are still processes that arrive, so time jumps directly to the arrival time.* currTime = *newProcess*.peek().getArriveTime(); } |

1. **We also use some statistical method to calculate the time.**

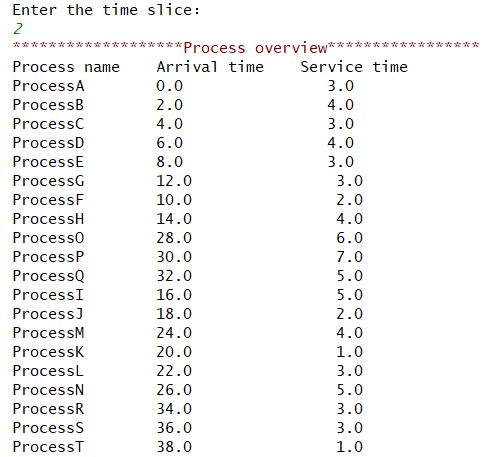
|  |
| --- |
| **private void** calculateTurnaroundTime(Process process) {  process.setTurnaroundTime(process.getOverTime() - process.getArriveTime()); } |

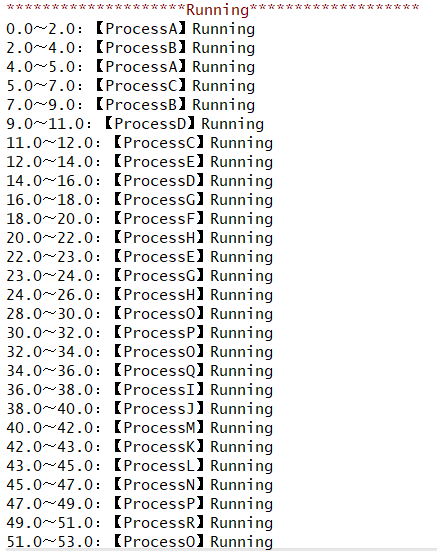
1. **We use such method to print the result.**

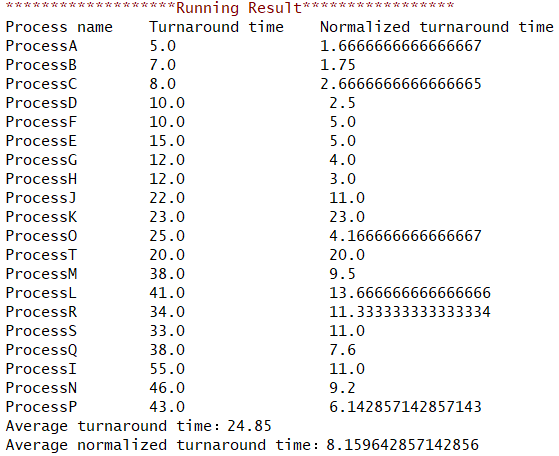
|  |
| --- |
| **public void** showResult() {  System.***out***.print(**"Process name "**);  System.***out***.print(**"Turnaround time "**);  System.***out***.println(**"Normalized turnaround time "**);  Process process;  **while**(!*finishedProcess*.isEmpty()) {  process = *finishedProcess*.poll();  System.***out***.print(**"Process"** + process.getProcessName() + **" "**);  System.***out***.print(**" "** + process.getTurnaroundTime() + **" "**);  System.***out***.println(**" "** + process.getTurnaroundWeightTime() + **" "**);  }  System.***out***.println(**"Average turnaround time："** + **mTotalWholeTime** / (**double**) **processCount**);  System.***out***.println(**"Average normalized turnaround time："** + **mTotalWeightWholeTime** / (**double**) **processCount**); } |

# 3. Result

**We execute the program with time quantum 2. The result is below.**







**Now we show all of running results in the following chart:**

**Shortest time is 8.4555s, when quantum time is 5s.**

## 4. Conclusion

### 4.1 Analysis: Quantum time works best

When quantum time is 5s, we will get the shortest time that is 8.4555s. It proves that the time quantum should be slightly greater than the time required for a typical interaction or process function.

And if we use common “FIFO” scheduling algorithm, the average normalized turnaround time is 8.9032s.

|  |  |
| --- | --- |
| Algorithm | Shortest Average NT Time |
| FIFO | 8.9032s |
| Round Robin(RR) | 8.4555s |

### 4.2 Analysis: Quantum time more than longest process

We find that if the quantum time is more than longest-running process, the average becomes around 8.9032s. It proves that round robin degenerates to FCFS if a time quantum that is longer than the longest-running process.

### 4.3 Summary

The experiment data matches our expectations. We prove RR algorithm is very suitable to short jobs compared to FIFO algorithm.

It’s a very interesting experiment and help us learn about something about CPU scheduling modes using such a good way. Thank you for your teaching!

## Appendix

All of the running results.

|  |  |  |
| --- | --- | --- |
| Ranking | Quantum time | Normalized turn-around time |
| 1 | 1 | 8.872162698 |
| 2 | 2 | 9.066944444 |
| 3 | 3 | 8.691111111 |
| 4 | 4 | 8.654801587 |
| 5 | 5 | 8.455515873 |
| 6 | 6 | 8.878333333 |
| 7 | 7 | 8.975853175 |
| 8 | 8 | 8.839384921 |
| 9 | 9 | 8.901468254 |
| 10 | 10 | 8.561607143 |
| 11 | 11 | 8.613690476 |
| 12 | 12 | 8.90327381 |
| 13 | 13 | 8.90327381 |
| 14 | 14 | 8.90327381 |
| 15 | 15 | 8.90327381 |
| 16 | 16 | 8.90327381 |
| 17 | 17 | 8.90327381 |
| 18 | 18 | 8.90327381 |
| 19 | 19 | 8.90327381 |
| 20 | 20 | 8.90327381 |
| 21 | 21 | 8.90327381 |
| 22 | 22 | 8.90327381 |
| 23 | 23 | 8.90327381 |
| 24 | 24 | 8.90327381 |
| 25 | 25 | 8.90327381 |
| 26 | 26 | 8.90327381 |
| 27 | 27 | 8.90327381 |
| 28 | 28 | 8.90327381 |
| 29 | 29 | 8.90327381 |
| 30 | 30 | 8.90327381 |