Operating System Course

Report for Lab2

Student Name	Student Number
马洪升	20175188

2018-12

Catalogue

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.

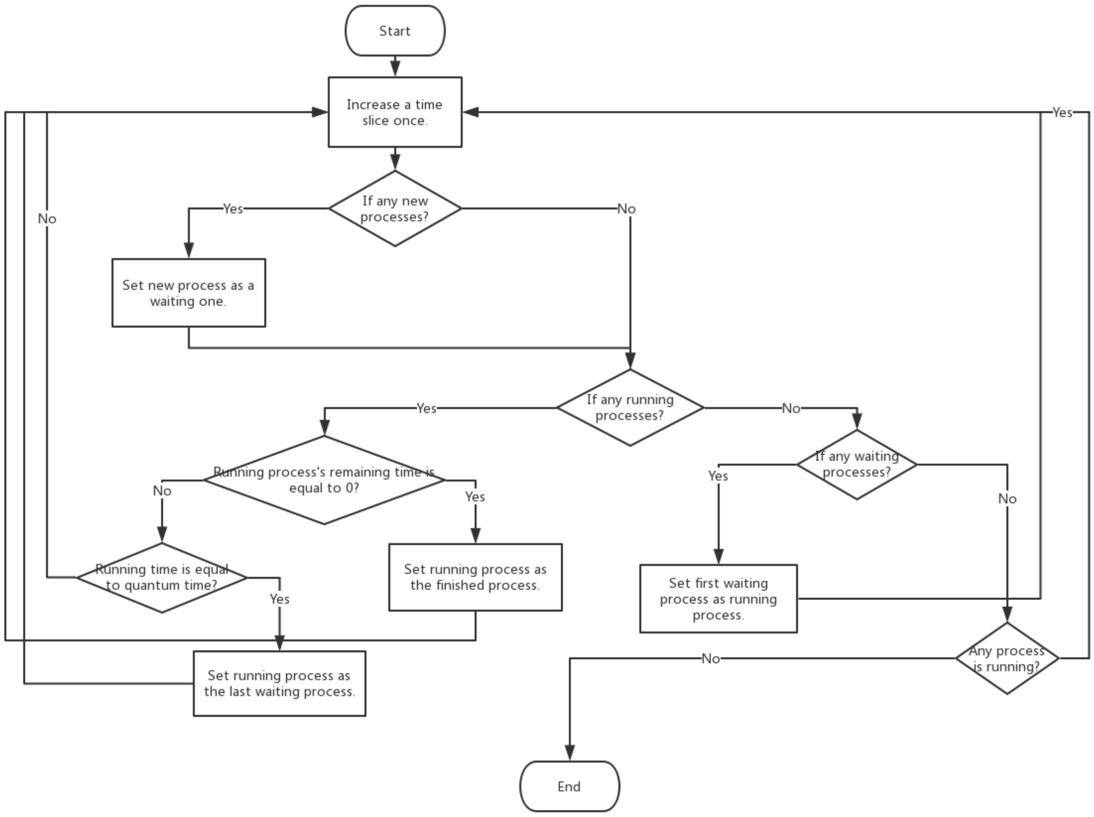
(5) Find the best quantum with the same processes.

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.

(2) Codes

a) 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.
```

b) 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;

    }

}
```

c) 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();
}
```

d) We also use some statistical method to calculate the time.

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

e) 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

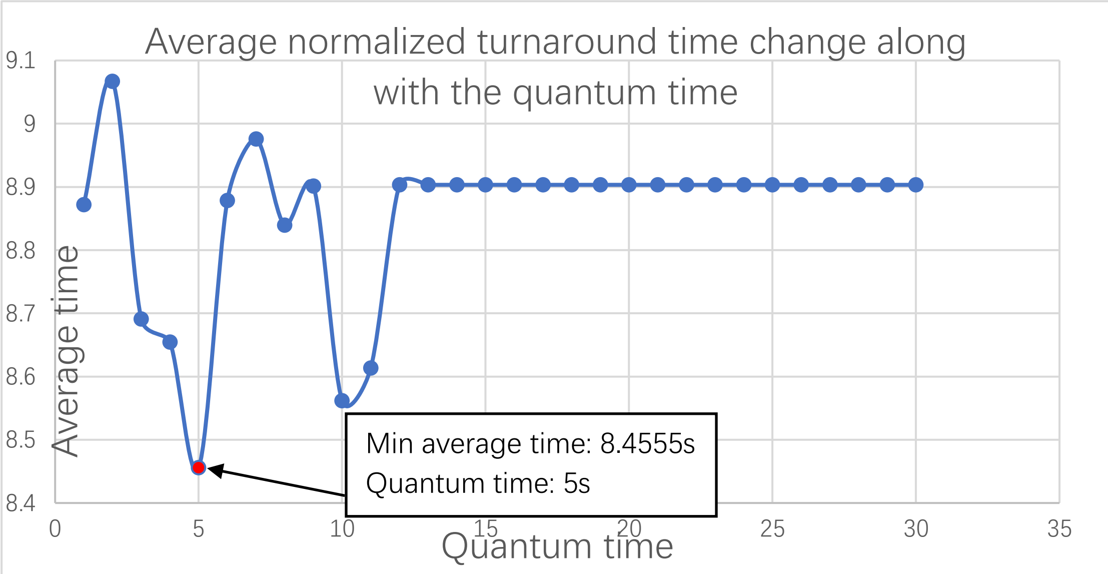
We execute the program with time quantum 2. The result is below.

```
Enter the time slice:
2
*******************Process overview*****************
Process name    Arrival time    Service time
ProcessA        0.0                3.0
ProcessB        2.0                4.0
ProcessC        4.0                3.0
ProcessD        6.0                4.0
ProcessE        8.0                3.0
ProcessG        12.0               3.0
ProcessF        10.0               2.0
ProcessH        14.0               4.0
ProcessO        28.0               6.0
ProcessP        30.0               7.0
ProcessQ        32.0               5.0
ProcessI        16.0               5.0
ProcessJ        18.0               2.0
ProcessM        24.0               4.0
ProcessK        20.0               1.0
ProcessL        22.0               3.0
ProcessN        26.0               5.0
ProcessR        34.0               3.0
ProcessS        36.0               3.0
ProcessT        38.0               1.0
```

```
**********************Running*********************
0.0~2.0: 【ProcessA】Running
2.0~4.0: 【ProcessB】Running
4.0~5.0: 【ProcessA】Running
5.0~7.0: 【ProcessC】Running
7.0~9.0: 【ProcessB】Running
9.0~11.0: 【ProcessD】Running
11.0~12.0: 【ProcessC】Running
12.0~14.0: 【ProcessE】Running
14.0~16.0: 【ProcessD】Running
16.0~18.0: 【ProcessG】Running
18.0~20.0: 【ProcessF】Running
20.0~22.0: 【ProcessH】Running
22.0~23.0: 【ProcessE】Running
23.0~24.0: 【ProcessG】Running
24.0~26.0: 【ProcessH】Running
28.0~30.0: 【ProcessO】Running
30.0~32.0: 【ProcessP】Running
32.0~34.0: 【ProcessO】Running
34.0~36.0: 【ProcessQ】Running
36.0~38.0: 【ProcessI】Running
38.0~40.0: 【ProcessJ】Running
40.0~42.0: 【ProcessM】Running
42.0~43.0: 【ProcessK】Running
43.0~45.0: 【ProcessL】Running
45.0~47.0: 【ProcessN】Running
47.0~49.0: 【ProcessP】Running
49.0~51.0: 【ProcessR】Running
```

```
*******************Running Result*****************
Process name    Turnaround time    Normalized turnaround time
ProcessA        5.0                1.6666666666666667
ProcessB        7.0                1.75
ProcessC        8.0                2.6666666666666665
ProcessD        10.0                2.5
ProcessF        10.0                5.0
ProcessE        15.0                5.0
ProcessG        12.0                4.0
ProcessH        12.0                3.0
ProcessJ        22.0                11.0
ProcessK        23.0                23.0
ProcessO        25.0                4.166666666666667
ProcessT        20.0                20.0
ProcessM        38.0                9.5
ProcessL        41.0                13.666666666666666
ProcessR        34.0                11.333333333333334
ProcessS        33.0                11.0
ProcessQ        38.0                7.6
ProcessI        55.0                11.0
ProcessN        46.0                9.2
ProcessP        43.0                6.142857142857143
Average turnaround time: 24.85
Average normalized turnaround time: 8.159642857142856
```

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