**CS5250 – Advanced Operating Systems**

**Assignment 4**

AY2018/2019 Semester 2

Name: SHI Jingli

Student ID: A0163341N

**[Assignment Answer]**

**Part A:**

1. **Simulator**

All the simulators code are in the following GitHub address.

<https://github.com/SHIJINGLI0206/CS5250-CA4>

The input for 4 simulator is as below.

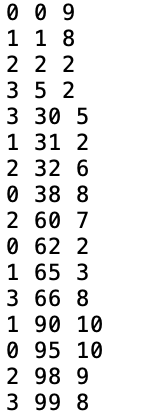


Figure 1. Input for simulator

The output of 4 simulator is as below table.

Table 1. Output of FCFS, RR, SRTF and SJF scheduling scheme

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scheduling**  **Algorithm** | **FCFS** | **RR(Q=2)** | **SRTF** | **SJF(α =0.5, T1=5)** |
| **Output** |  |  |  |  |
| **AVG Waiting Time** | 6.44 | 8.56 | 4.5 | 7.12 |

* The output format is (time, process\_id), for example, for RR(Q=2), (0,0),(2,1),(4,2) means pid=0 arrive at t=0 and takes 2 time unit, pid=1 arrive at t=2 and takes 2 time unit, pid=2 arrive at t=4 and takes 2 time unit. Other output has same meaning with RR one.
* From the table, we know RR(Q=2) gives the largest average\_waiting\_time(8.56 unit), whereas SRTF gives the least average\_waiting\_time(4.5 unit);

1. **Created Simulator**
   1. Using the default value for RR and SJF parameter, the average waiting time comparison of 3 implemented scheduling schemes is as below.

From the below table, we know SRTF gives the least average waiting time.

Table 2. Output of average waiting time of RR, SRTF and SJF

|  |  |  |
| --- | --- | --- |
| **RR(Q=2)** | **SRTF** | **SJF(α =0.5, T1=5)** |
| 8.56 | 4.5 | 7.12 |

After optimize Q for RR and α for SJF, the output and average time is show in the below table.

* For RR, when Q=max(burst time), it is same with FCFS.
* For SJF, when α =0.5, the initial predicted time for all process are same, so it is same with FCFS.

Table 3. Output of RR and SJF

|  |  |  |
| --- | --- | --- |
| **Scheduling**  **Algorithm** | **RR(Q=10)** | **SJF(α =1, T1=5)** |
| **Output** |  |  |
| **AVG Waiting Time** | 6.44 | 6.44 |

* 1. By using two group inputs sample shown in the below table, we get the output and average waiting time.

Table 4. Input of different processes.

|  |  |
| --- | --- |
| All Short Processes (**input2.txt**) | Interleaved Very Short and Very Long Processes (**input3.txt**) |
|  |  |

For input2.txt, we get the output as below.

Table 5. Output of RR, SRTF, SJF using input2.txt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Scheduling**  **Algorithm** | **RR(Q=2)** | **SRTF** | **SJF(α =0.5, T1=5)** | **RR(Q=10)** | **SJF(α =1, T1=5)** |
| **Output** |  |  |  |  |  |
| **AVG Waiting Time** | 3.5 | 2.25 | 3.0 | 3.0 | 3.0 |

For **input3.txt**, we the below output.

Table 6. Output of RR, SRTF, SJF using input3.txt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Scheduling**  **Algorithm** | **RR(Q=2)** | **SRTF** | **SJF(α =0.5, T1=5)** | **RR(Q=10)** | **SJF(α =1, T1=5)** |
| **Output** | **(…)** |  |  | **(…)** |  |
| **AVG Waiting Time** | 99.83 | 50 | 148.5 | 103.5 | 132.17 |

From above testing, we know:

(1) SRTF will be the optimal schedule, which give minimum average waiting time.

(2) As one extreme, the Q for RR is very large and α =1 for SJF, they ae same as the FCFS. However, the performance of FCFS is worse than SRTF, so SRTF will be the optimal schedule.

* 1. Multilevel Queue Scheduling

Intuition of Multilevel Queue Scheduling

1. Jobs under this scheme cannot switch from queue to queue. Once they are assigned a queue, they will finish using the assigned queue.
2. When process can be readily categorized, then can create multiple separate queues, each mplementing whatever scheduling algorithm is most appropriate for that type of job, and/or with different parametric adjustments.
3. Scheduling must also be done between queues, which is scheduling one queue to get time relative to other queues. No queue in a lower priority queue runs until all higher priority queues are not available.

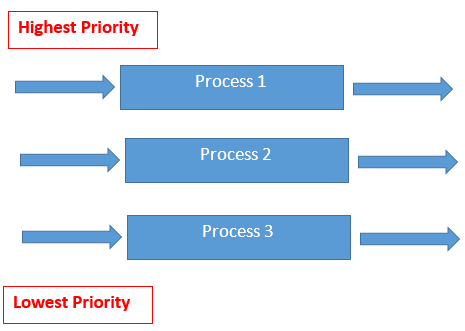


Figure 2. Multi-level Queue Scheduler

* 1. Scheduler with Multi-core CPU

1. Process running on N CPU cores will make the scheduler more complicated because a) it will increase shared case misses, b) it is hard to decide to choose which process run concurrently on a core since cache contention and bus traffic can impact process performance, c) some processes may be share data heavily.
2. How to extend scheduler to multi-processor system.

The current Linux kernel scheduler distributes the running tasks equally among the available last-level caches in an SMP domain. Within logical CPUs that share the last-level cache, the scheduler distributes the load equally, first among the available CPU cores and then among the available logical thread siblings. For example, consider a dual package SMP platform with Core 2 quad processors with four running processes. The multi-core-aware Linux process scheduler distribute these four running tasks among the four L2’s that are available in the system as show the below figure.

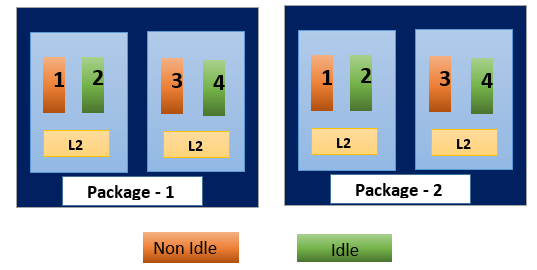


Figure 3. Scheduler on multi-processor

**Part B: Spin Lock**

The implemented code of spin shows in figure 1.

Assumptions:

1. All threads id will be greater than 0.
2. The spin lock will not used within interrupt handler.
3. The spin lock holder will not be delayed.

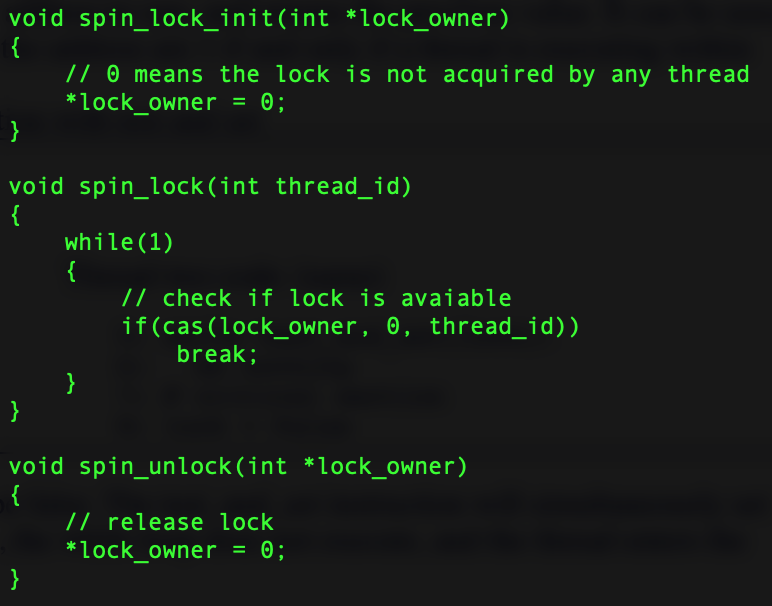


Figure 4. C code of Spin lock