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.

```
0 0 9
1 1 8
2 2 2
3 5 2
3 30 5
1 31 2
2 32 6
0 38 8
2 60 7
0 62 2
1 65 3
3 66 8
1 90 10
0 95 10
2 98 9
3 99 8
```

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	(0, 0) (9, 1) (17, 2) (19, 3) (30, 3) (35, 1) (37, 2) (43, 0) (60, 2) (67, 0) (69, 1) (72, 3) (90, 1) (100, 0) (110, 2) (119, 3)	(0, 0) (68, 3) (2, 1) (70, 2) (4, 2) (72, 1) (6, 3) (73, 3) (8, 0) (75, 2) (10, 1) (76, 3) (12, 0) (78, 3) (14, 1) (90, 1) (16, 0) (92, 1) (18, 1) (94, 1) (20, 0) (96, 0) (30, 3) (98, 2) (32, 1) (100, 3) (34, 2) (102, 1) (36, 3) (104, 0) (38, 2) (106, 2) (40, 0) (108, 3) (42, 3) (110, 1) (43, 2) (112, 0) (45, 0) (114, 2) (47, 0) (116, 3) (49, 0) (118, 0) (60, 2) (120, 2) (62, 0) (122, 3) (64, 2) (124, 0) (66, 1) (126, 2)	(0, 0) (2, 2) (4, 0) (5, 3) (7, 0) (13, 1) (30, 3) (31, 1) (33, 3) (37, 2) (43, 0) (60, 2) (62, 0) (64, 2) (65, 1) (68, 2) (72, 3) (90, 1) (100, 3) (108, 2) (117, 0)	(0, 0) (9, 1) (17, 2) (19, 3) (30, 3) (35, 2) (41, 1) (43, 0) (60, 2) (67, 1) (70, 3) (78, 0) (90, 1) (100, 0) (110, 2) (119, 3)
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);

2. Created Simulator

2.1 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 some testing using different Q for RR and α for SJF, we find the best value for the parameter to get optimal average waiting time for RR and SJF.

- The output of some tested values is as following table for scheduler RR and

SJF.

Table 3. output of waiting time using different Q for RR

RR	
Q	**Average Waiting Time**
1	8.94
2	8.56
3	9.13
4	9.31
5	8.93
6	9.5
7	9.6
8	9.69
9	8.88
10(max burst time)	6.44

Table 4. output of waiting time using different alpha

SJF	
α	**Average Waiting Time**
0.0	7.12
0.1	7.12
0.2	7.12
0.3	7.12
0.4	7.12
0.5	7.12
0.6	7.43
0.7	7.43
0.8	7.25
0.9	7.25
1.0	6.44

- For RR, when Q=maximum of burst time(e.g. 10 for given input), it is same with FCFS and get minimal waiting time.
- For SJF, when α =0.5, the initial predicted time for all process are same, so it is same with FCFS and get minimal waiting time.

Table 5. Output of RR and SJF

Scheduling Algorithm	RR(Q=10)	SJF(α =1, T1=5)
Output	(0, 0) (9, 1) (17, 2) (19, 3) (30, 3) (35, 1) (37, 2) (43, 0) (60, 2) (67, 0) (69, 1) (72, 3) (90, 1) (100, 0) (110, 2) (119, 3)	(0, 0) (9, 1) (17, 2) (19, 3) (30, 3) (35, 1) (37, 2) (43, 0) (60, 2) (67, 0) (69, 1) (72, 3) (90, 1) (100, 0) (110, 2) (119, 3)
AVG Waiting Time	6.44	6.44

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

Table 6. Input of different processes.

All Short Processes (**input2.txt**)	Interleaved Very Short and Very Long Processes (**input3.txt**)
0 0 4 1 1 3 2 2 2 3 5 1	0 0 1 1 1 100 2 2 103 3 5 1 3 7 101 1 9 1

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

Table 7. 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	(0, 0) (2, 1) (4, 2) (6, 3) (7, 0) (9, 1)	(0, 0) (4, 2) (6, 3) (7, 1)	(0, 0) (4, 1) (7, 2) (9, 3)	(0, 0) (4, 1) (7, 2) (9, 3)	(0, 0) (4, 1) (7, 2) (9, 3)
AVG Waiting Time	3.5	2.25	3.0	3.0	3.0

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

Table 8. 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	(0, 0) (1, 1) (3, 2) (5, 3) (…) (301, 2) (303, 3) (305, 2) (306, 3)	(0, 0) (1, 1) (5, 3) (6, 1) (9, 1) (10, 1) (103, 3) (204, 2)	(0, 0) (1, 1) (101, 3) (202, 2) (305, 3) (306, 1)	(0, 0) (1, 1) (11, 2) (21, 3) (…) (283, 2) (293, 3) (303, 2) (306, 3)	(0, 0) (1, 1) (101, 2) (204, 3) (205, 3) (306, 1)
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.

2.3 Multilevel Queue Scheduling

Intuition of Multilevel Queue Scheduling

a. Jobs under this scheme cannot switch from queue to queue. Once they are assigned a queue, they will finish using the assigned queue.
b. When process can be readily categorized, then can create multiple separate queues, each implementing whatever scheduling algorithm is most appropriate for that type of job, and/or with different parametric adjustments.
c. 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.

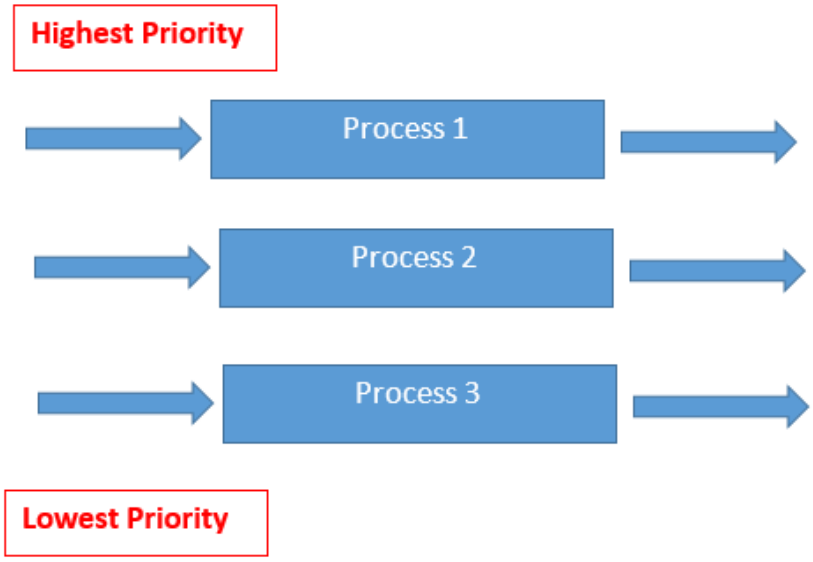

Figure 2. Multi-level Queue Scheduler

2.4 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.

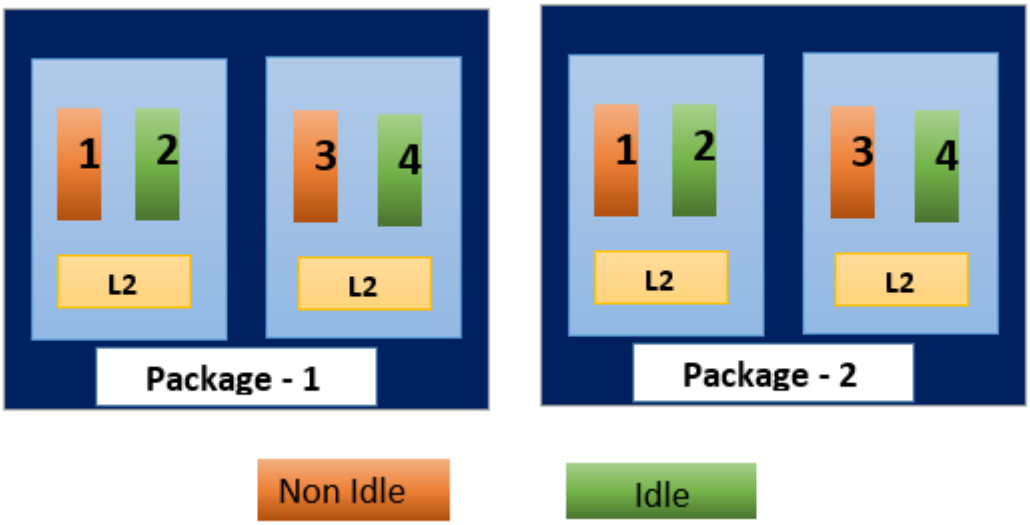

Figure 3. Scheduler on multi-processor

Part B: Spin Lock

The implemented code of spin shows in figure 1.
Assumptions:

a. All threads id will be greater than 0.
b. The spin lock will not used within interrupt handler.
c. The spin lock holder will not be delayed.

```
void spin_lock_init(int *lock_owner)
{
    // 0 means the lock is not acquired by any thread
    *lock_owner = 0;
}

void spin_lock(int thread_id)
{
    while(1)
    {
        // check if lock is avaiable
        if(cas(lock_owner, 0, thread_id))
            break;
    }
}

void spin_unlock(int *lock_owner)
{
    // release lock
    *lock_owner = 0;
}
```

Figure 4. C code of Spin lock