The Comparison of CPU Process Scheduling Algorithms

- Operating System Final Project

Jen-Yin Chao

BU MET Collage Computer Science Department,

CS575 Fall 2017 – John Day

Table of Contents

Abstract

It's always a complicated issue about choosing algorithm of CPU process scheduling, including FCFS, SJF, RR, SRTF, and HRRN. By simulating CPU processing various cases with different algorithms, the result comes out as the following: SJF and SRTF have the best performance in any scenario, while RR is always the worst one. Considering the preemptive problem, FCFS is actually a suitable choice for CPU process scheduling, which overall performs better than HRRN in most cases and the mean of average waiting time is quite similar to SJF and SRTF, though with higher variance.

Introduction

There are several algorithms introduced in the lecture of CPU processing, despite the priority, including the followings.

- First-Come, First-Served(FCFS): The algorithm prefers to process the earlier coming job first.
- Shortest-Job First(SJF): The algorithm prefers to process the job with shortest burst time first in a preemptive mode.
- Round-Robin(RR): The algorithm will alternatively allocate CPU source to jobs evenly with a specific time period.

And there are two more algorithms for OS designer to use.

- Shortest-Remaining-Time-First(SRTF): The algorithm prefers to process the job with shortest remaining time first in a preemptive mode.
- Highest-Response-Ratio-Next(HRRN): The algorithm prefers to process the job with highest response ratio first, with a trade-off between performance and preemptive problem solving.

$$Response\ Ratio = (Waiting\ Time + Burst\ Time)/Burst\ Time$$

The features of each algorithm can be easily checked by hand with any example, but what's the overall performance dealing with all kinds of situation? Is it possible that some of them have advantages dealing with some specific situations? Or, do SJF and SRTF have good performance but with significant deviation? So, in this project, I simulate numerous variant cases to different algorithms to compare their performance and behaviors in different environment module setting.

Objective

The objectives are the following:

1. Simulate the algorithms by coding.
2. Compare overall performance between algorithms in random cases.
3. Compare the behavior of algorithms in different case module setting.

Approach

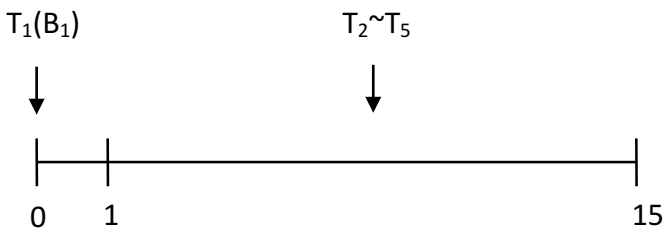

Figure 1. Task Delivering Module

As shown in the figure 1, the setting of the module is the following. Task 1 will arrive at time 0 with its burst time B_1, and task 2 to 5 arrive between time 1 to 15 with their own burst time. The arriving times and burst times are random variables coming from the following distribution.

$$Arriving\ Time \sim U(1,15)\ round\ to\ integer$$

$$Burst\ Time \sim N(5, 2^2)\ round\ to\ integer$$

Accordingly, there are three steps to simulate algorithms to deal with each case. Create a task table as figure 2, with arrive time, burst time, remaining time, and complete time, choose the next processing task depending on the preference of the algorithm, and then allocate CPU source to process the task. Finally, the average waiting time for the algorithm in this case can be computed by the waiting time of each task computed from the variables in the table.

	ArriveTime	BurstTime	RemainTime	CompleteTime
1	0	6	0	11
2	1	3	0	6
3	4	5	0	18
4	8	4	0	20
5	10	2	0	16

Figure 2. Task Table

Coding

R is the software used in this project, and the structure is shown as figure 3. To simulate each case, it creates the task table to initialize the module. In the inner loop, it calls one of algorithm function to return the preference of next processing task, and then allocate CPU source to process it. The outer loops are used to run whole code with setting number of running times and run with each algorithm in turn.

```
Main

#Loop for each algorithm
loop{
#Loop for many turns to simulate
loop{
        Create Task Table()
        #Loop for processing each task until finish
        loop{
          Choose Next Processing Task()
          Process Task
        }
}}
```

Figure 3. Pseudo Code

Result

With the outcome from the code, there are several parts of comparison we can discuss: Comparison of performance between algorithms, performance comparison of each algorithm in different running turns (number of cases), parameter of mean burst time, number of task, and parameter of max arriving time.

Comparison between algorithms

By collecting the average waiting time of each turn from those algorithms, the power density function can be plotted as figure 4, represented in solid lines. It's clear that the curve of SRTF and SJF is thinner with a higher peak than rest of the other algorithms, which means the average waiting time of them both have lower deviation, that is, most of the average waiting

time locate in the period of the peak. On the other hand, FCFS, HRRN, and RR have flatter PDF curve, representing higher deviation, that is, there are more awful cases for them to deal with, especially RR.

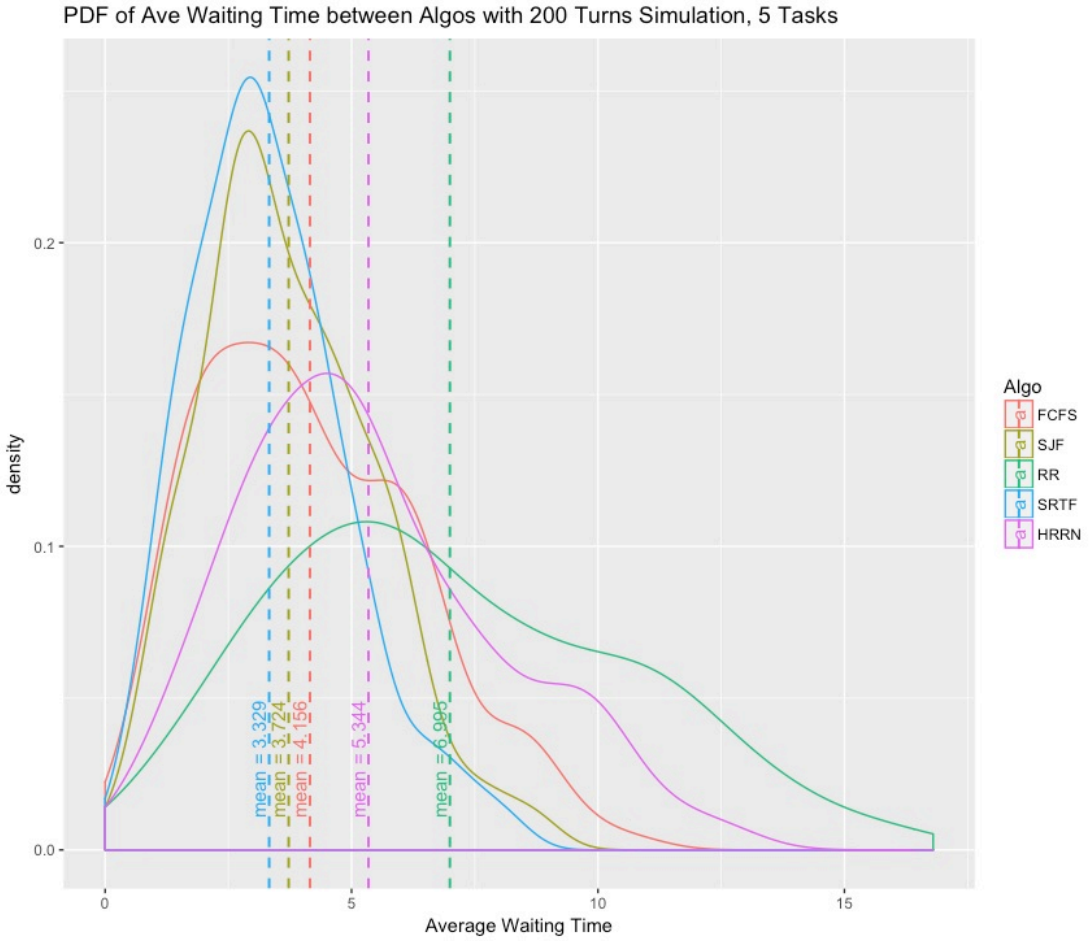

Figure 4. PDF of Average Waiting Time

The dash lines are the mean of the average waiting time for each algorithm. There is not much difference between SRTF, SJF, and FCFS, in the range of 3.2 to 4.2, which is the group of best performance. HRRN is the second-best algorithm with 5.34 mean average waiting time, while RR is the worst one with around 7 mean average waiting time.

Comparison between different turns of simulation

To make the simulation be more real, increasing the turn of running, which means more cases are included, can approach the goal. Accordingly, I increase the turns from 200 to 2000 and compare their difference to check the consistency, shown as figure 5.

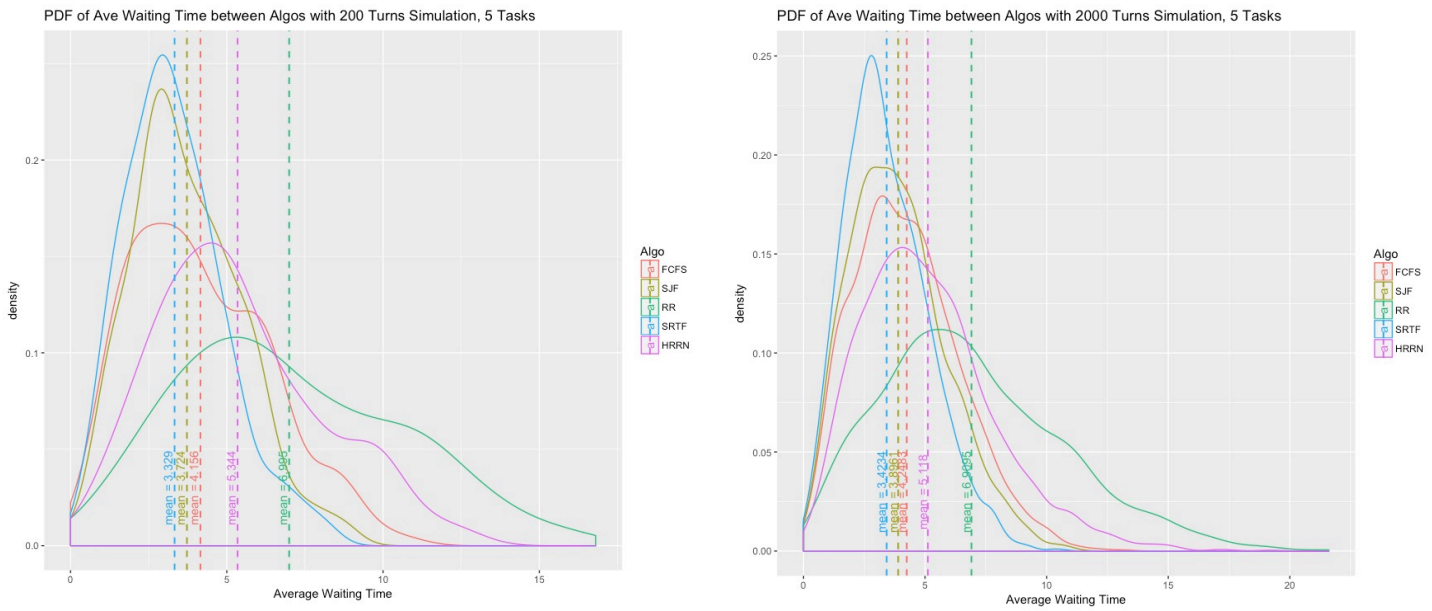

Figure 5. PDF of 200 turns vs. 2000 turns

The result comes out that the relation and order of algorithms remains the same, but the curves of PDF centralize to their mean with less deviation, that is, there are more awful cases included but lower percentage of it that the performance becomes more stable and less influenced by those extreme cases.

Comparison between different parameter of mean burst time

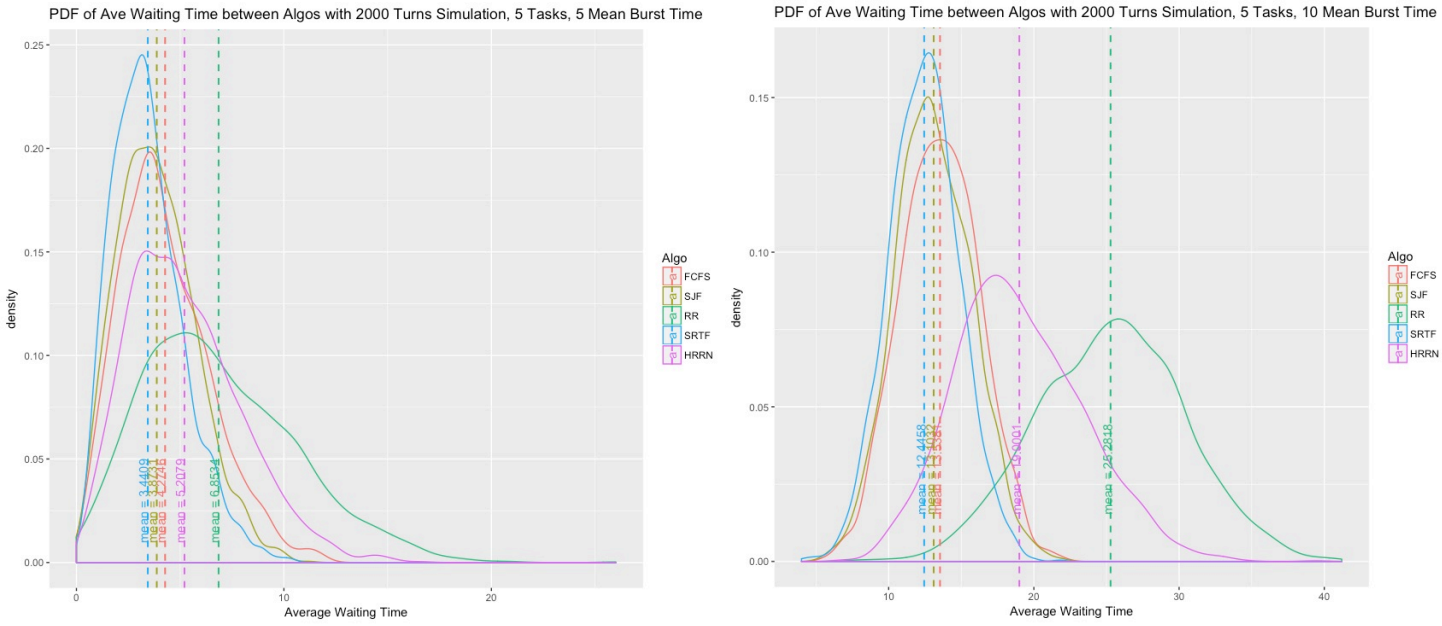

Figure 6. PDF of 5 mean burst time vs. 10 mean burst time

The initial setting of the mean task burst time is 5, and increasing it is a kind of loading increase. By doing so, algorithms may behave differently because of the change depending on the feature of each of them.

Shown as figure 6, the order of the algorithms remains the same. However, the curve of HRRN and RR both right shift dramatically, which means they are not good at dealing this situation, although the means of average waiting time of all algorithms increase because of the heavy loading.

Comparison between different number of task

Increasing the number of the task is also another way of loading increase, and, in this part, I increase the number of task from 5 to 10 to see how they will perform.

Shown as figure 7, in this case of loading increase not only the mean of average waiting time of HRRN and RR increase but also, surprisingly, FCFS right shifts dramatically, which is almost the same to the mean of HRRN, representing that FCFS is not good at this scenario.

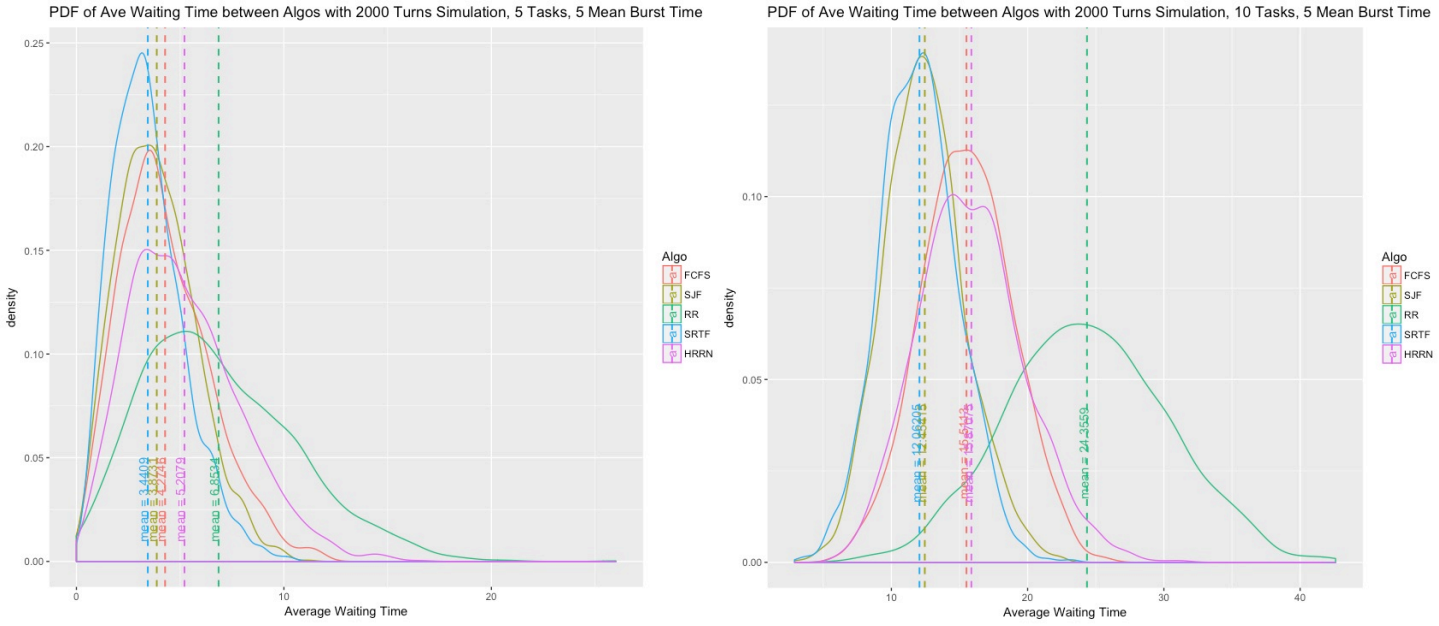

Figure 7. PDF of 5 tasks vs. 10 tasks

Comparison between different parameter of max arriving time

To increase the max arriving time of module, the density of task arriving will decrease while processes same number of task, and then the performance of all algorithms will be better undoubtedly. As the result, to keep the density of task arriving to determine the change only because of the max burst time, the number of task should be increased simultaneously.

As shown in figure 8, except RR, the curves of the rest of four algorithms become more identical, which means HRRN performance better in this case.

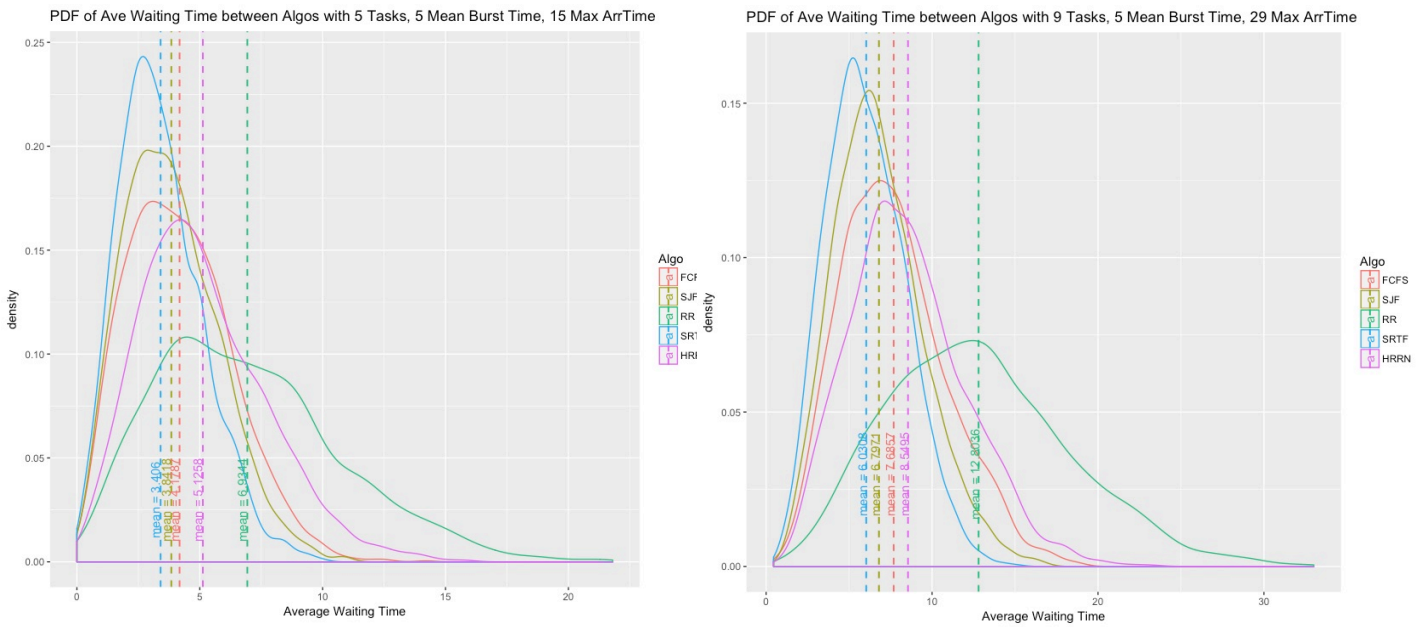

Figure 8. PDF of 15 max arriving time vs. 29 max arriving time

Conclusion

To sum up the comparison of outcome before, the conclusions are the follows.

1. SJF & SRTF have the best performance in any scenario.
2. Considering the preemptive-problem, FCFS is actually a good choice for CPU processing, which always performs better than HRRN.
3. RR is always the worst one.

References and Links

[1] Abraham Silberschatz, Peter Bear Galvin, Greg Ggne*, Operating System Concepts*, 9th Ed., 2013

[2] http://www.geeksforgeeks.org/gate-notes-operating-system-process-scheduling/