Real-time Scheduling Algorithms:

Preemptive Algorithm Simulation Report

Submitted By:

Name: LUO ZIJIAN

Matric.No: A0224725H

Mobile: 98912483

Supervisor:

Prof. Bharadwaj Veeravalli

Electrical and Computer Engineering, School of Engineering

7 April 2021

Abstract

Preemptive scheduling algorithm is a very meaningful part, especially in real-time problems. Whether it is pre-designed priority or dynamic adjustment of priority, the purpose is to make the scheduling algorithm more fair. In this paper we will discuss the performance of two chosen preemptive scheduling algorithms, Optimized Shortest Job First(OSJF) and Multi-Level Feedback Queue with Intelligent Mitigation(MLFQ-IM). For the purpose of simulation, some basic object(Process, Queue) was created using Python. Performance metrics of average waiting time and turnaround time was calculated in the end. Simulation results illustrate that OSJF perform better in small load and reduce a little context switching. As for MLFQ-IM can get good result in the condition of high input, especially for starvation, it reduce a lot.

1 Introduction

1.1 Problem motivation

In a single processor system, the rule is that only one process can be processed at one time slice, other ready process can be chosen when the current process is finished or its priority is greater than the current process. At the same time, other process should be stored at the waiting queue. Applying queuing theory to this waiting queue, we dynamic adjust the priority of all the waiting process in this waiting list. Every time, only one greatest priority process is executed, and other processes have to wait in their waiting queue. Only when all the process were finished, this whole simulation is done.

1.2 Problem formulation

There are two conditions of this problem. The first one is OSJF condition, as we all know, we just keep the new arriving process into waiting list, and update their remaining priority by steps, and to get the suitable process followed by the basic rule of OSJF. This process is sent to the CPU processor and reduce its remaining service time. So, in this algorithm, priority is based on its remaining service time and that of other competitive processes.

$$Priority = OSJF\_priority(process, other\ competive\ process)$$

And then, we choose the process with greatest priority as current process.

$$current\ process = \max\ (Priority)$$

Another one is MLFQ-IM condition, it seems that the classification of priority is complex, but setting the level of waiting level in advance truly helps the scheduling to chose the right waiting list to get processed. As for the influence factor of priority, here are three parameters: waiting time, current queue level, interrupt. We can update priorities of all the processes based on below formulation.

$$Priority = MLFQ\_IM\_priority(waiting\ time, current\ queue\ level, interrupt)$$

Similarly, after checking the priority of all the process, we choose the most suitable process into working space.

$$current\ process = \max\ (Priority)$$

1.3 Objective function

Comparing the similarity of two algorithms, I found the common metric between them is average waiting time, which can directly show the performance of scheduling and reducing starvation. Waiting time of a process is the sum of waiting periods spent in the waiting queue. For a total system, average waiting time is defined as

$$minimize\ J = \frac{1}{L}\sum_{k=1}^{L} w_k$$

In this paper, we denote the total number of process is L, and the separate process waiting time is w_k. And we can judge the fairness of scheduling algorithm mainly based on the average waiting time.

As for MLFQ-IM algorithm, the starvation and context switching also can be judged by the weighed turnaround time, which is calculated by the turnaround time divided by the actual needed service time.

$$weighed\_turnaround\_time = \frac{turnaround\ time}{service\ time}$$

2 Description of input data

As for the object of process, there are these basic definitions.

Table 1: The definitions of process

Name	Type	Descriptions
process_id	int	The index of the process
arrive_time	int	The arriving time of the process
finish_time	int	The finishing time of the process
left_serve_time	int	The remaining needed service time of the process
service_time	int	The needed service time of the process
waiting_time	int	The waiting time of the process
turnaround_time	int	The time interval from the time of submission of the process to the time of the completion of the process
w_turnaround_time	int	The weighed turnaround time of the process

For OSJF algorithm, all input data are generated from own computer, we just pre-set on the parameters of process_id, arrive_time, service_time.

Table 2: The distribution of process in OSJF list

Process_id	Arrive_time	Service_time
1	0	4
2	2	7
3	5	5
4	6	8
5	8	9

As for MLFQ-IM algorithm, in order to keep the constraints in the simulation, I create these processes to keep same distribution in the paper [3]. In this simulation, I use random.randit() to create a right data set in different situation.

Table 3: The distribution of process in MLFQ-IM list

Process_id	Arrive_time	Service_time	Descriptions
1-54	random.randint(1,16)	random.randint(1,16)	Small load
55	Random.ranint(1,16)	1	I/O Process
56-99	random.randint(16,256)	random.randint(16,256)	Normal load
100	random.randint(256,1256)	random.randint(256,1256)	Large load

Note: In this table, I set I/O task is Process with process-id=55, and its service time is 1(the service time of interrupt is small). As for other processes, the properties show the different situation in the operation system.

I have put these generate functions into the main coding structure, so if you run the code, the process list would be generated automatically.

3 Algorithm description

The first algorithm, Optimized Short Job First (OSJF) Algorithm, [2] is innovated from Short Job First (SJF) Algorithm and Short Remaining Job First (SRJF) Algorithm. It changes the searching method to get the next process. When a new job is coming, the key of OSJF algorithm is to decide whether to continue executing current process.

Obviously, when the remaining needed service time of a new competitive process is smaller than half that of current process, the next process is this new process. But in the converse condition, the next process is still current process.

OSJF Algorithm pseudo code in choosing next_process

if 1/2*current_process.left_serve_time<competitive_process.left_serve_time:
  next_process=current_process
else:
  next_process=competitive_process

The Second algorithm, Multi-Level Feedback Queue with Intelligent Mitigation (MLFQ-IM), [1] is innovated from Multi-Level Feedback Queue (MLFQ) Algorithm. This MLFQ-IM algorithm belongs to the conditional preemptive scheduling algorithm. It adds some update functions of the priority in different queues. **The first one is redirecting the processes in lower level queues into higher level queues in a period of time. The second one is release remaining time in highest queue(Q0) to some processes in lower level queue, which can reduce the wasted time and increase the throughput and utilization.**

Separate time quanta with different queue level are set in this table.

Table 2: Time quanta set in separate queue levels

	Q1	**Q2**	**Q3**	**Q4**	**Q5**
Time_Quanta	**16ms**	**32ms**	**64ms**	**128ms**	**256ms**

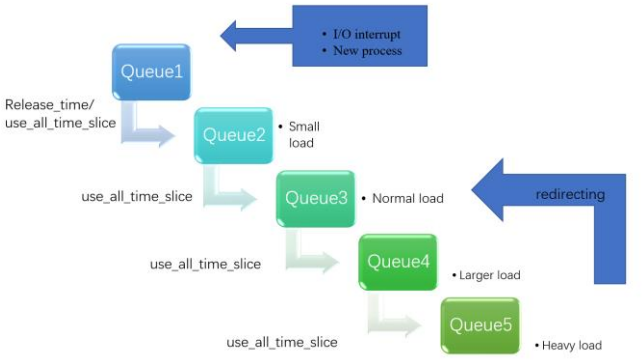

Figure 1: Five Queue level state machine diagram

MLFQ-IM Algorithm pseudo code in updating queue level

```
while(True):
    def redirect_in_S_period(queue1,queu2,queue3,queue4,queue5,S)
        if running_time%S==0:
            queue2.add(queue4.process_list)
            queue4.delete(queue4.process_list)
            queue3.add(queue5.process_lisy)
            queue5.delete(queue5.process_list)
    def release_time(queue1,current_process):
        extra_time=queue1.time_slice-current_process.left_serve_time
        return extra_time
```

4 Simulation Efforts and Results Discussion

Our simulation model is based on the event driven, which means the scheduler choose next process according to current process list and scheduling rules. In this period of time, scheduler update the process list and other storage structures. Until scheduler detect a termination condition (process list is empty), the entire simulation will still be processed. After calculating the final result, the simulation shows the metrics, which can be judge the performance in different algorithms.

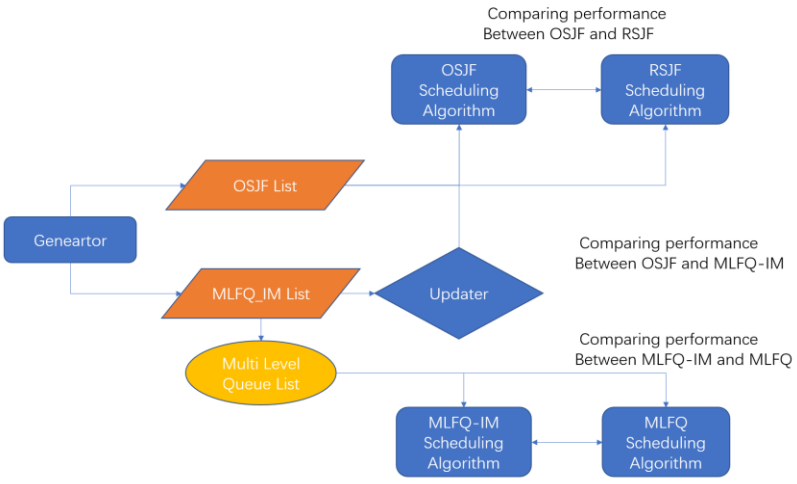

Figure 2: Flow chart of this simulation architecture

4.1 Simulation of OSJF and RSJF

After simulation with OSJF List, we can plot the Gantt graph based on the result.

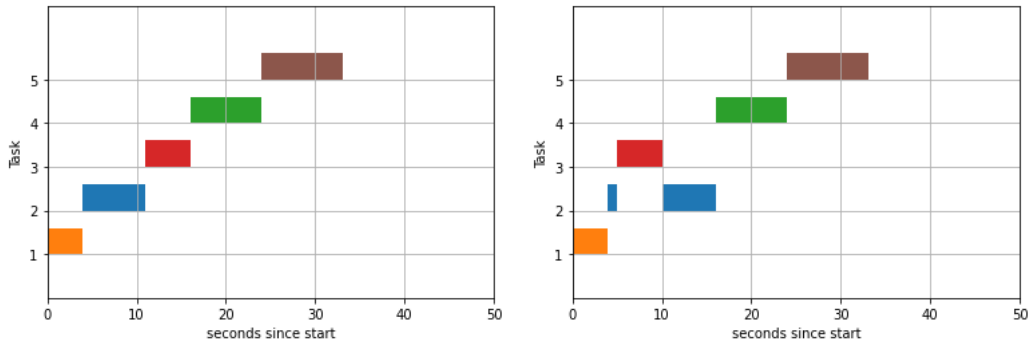

Figure 3: Gantt graph between OSJF algorithm and RSJF algorithm

The trend shown in Fig.3 is due to the OSJF algorithm continuously choose the process 2 at the time lice of 5, but for RSJF algorithm, it changes process 3 as next process. From this graph, we can find that OSJF algorithm is truly reduced the context switching because OSJF algorithm is more inclined to retain current progress. As for this special process list, the average waiting time is 6.8ms for OSJF algorithm, and that of RSJF algorithm is 6.6ms. As a result of more context switching, so the average waiting time of OSJF algorithm increase a little.

4.2 Simulation of MLFQ-IM and MLFQ

After simulation with MLFQ-IM List, we can get this average waiting time and weighted turnaround time table.

Table 5: Comparison between MLFQ-IM and MLFQ

	MLFQ-IM	MLFQ
Average waiting time	1440.76ms	1508.ms
Average weighted turnaround time	21.15	20.33
Total running time	6810ms	6923ms

Here are three statement in this table. The first one is that MLFQ-IM algorithm truly reduce the average waiting time for entire system. The reason is that more context switching can let all processes get more fair time slice to process, which can reduce the waiting time. Second one is that the average weighted turnaround time in MLFQ-IM algorithm is greater than that of MLFQ because it adds redirecting function and releasing time to lower queue level. But the probability of these certain cases is small, so the growth is also small. The last statement is that MLFQ-IM algorithm get less running time in same process list, which means it perform better in higher throughput. MLFQ-IM algorithm can reduce the running time mainly because it release time to lower queue, but for MLFQ algorithm, the extra time is wasted.

4.3 Simulation of MLFQ-IM and OSJF

In order to compare the performance between MLFQ-IM algorithm and OSJF algorithm, in this simulation, I choose same process list test these two algorithms, but for OSJF algorithm, the process list is updated with some changes in pre-processing.

And then I can get the result in below table

Table 6: Comparison between MLFQ-IM and OSJF

	MLFQ-IM	OSJF
Average waiting time	1158.60ms	1124.20.ms
Average weighted turnaround time	19.45	18.87
Total running time	5955ms	5955ms

Based on the difference between them, I found two views. The first one is that MLFQ-IM perform better in reducing starvation and increasing context switching. As a result of redirecting and releasing time to low level queue, it can get more fair scheduling for the entire system. But for the OSJF algorithm, the aim of invention is to keep steady in current process. As a result, the average weighted turnaround time [4] is reduced.

Another interesting view is that the average waiting time of OSJF algorithm is less than that of MLFQ-IM algorithm. I suppose that there are two reasons. Firstly, current process is more likely to keep in OSJF algorithm, so the number of waiting process is less and the number of processes with small waiting time is more. Totally, the average waiting time is less. Secondly, the case of redirecting to low level queue in MLFQ-IM algorithm can lead more processes to keep waiting, so the total waiting time increase.

5 Advantages and Disadvantages

Our simulation is well structed and well processes. To sum up, here are strengths and pitfalls between OSJF and MLFQ-IM algorithm.

5.1 Advantages

By calculating the performance in different test cases, we can get their advantages separately.

For OSJF algorithm, the outstanding advantage is reducing context switching. To some extent, this can reduce the average waiting time. The entire system can benefit a lot from keeping the next process is still the current process. For example, reducing the consuming time in context switching can increase the CPU utilization because less time is wasted.

The first advantage of MLFQ-IM is that it monitors the execution of current process and update the priority of all processes accordingly, instead of pre-setting fixed priority of many processes at the beginning of simulation. And the most advantage of MLFQ-IM is that reducing the probability of starvation cases. It can make the system perform more fairly in reallocating time.

5.2 Disadvantages

After checking the simulation and reading the information in the paper, I found some limits of our simulation method.

Firstly, in OSJF algorithm, the author in the paper [1] truly propose a good scheduling algorithm in reducing the context switching, but he just assumes that the consuming

time in context switching is zero. Obviously, the average waiting time is greater than RSJF algorithm. But if we calculate the actual switching time, maybe the result is different. Secondly, for OSJF algorithm in the paper [1], the test data is so small. In order to compute the performance in higher load, I let OSJF algorithm to run same list as MLFQ-IM algorithm, and get some interesting ideas.

Lastly, for the MLFQ-IM algorithm, our simulation is still so simple and idealistic because the test data is not normal. We just arrange the process list with certain order, but in real-time operation system, the probability is unfixed. In real-time world, all the new arriving processes are event-driven, so there is no certain distribution. In the future work, the model can set more interrupts from I/O response.

6 Conclusions

The result for the comparison of the two algorithms OSJF and MLFQ-IM shows that they perform differently depends on the process list with different load. As for OSJF algorithm, our simulation performs well as the paper [1], so we get same result. However, it is still not easy to counter the consuming time of every context switching. That is to say, the cost is not easy to justify. For the author, this is a place that lacks consideration. For MLFQ-IM algorithm, it increases the context switching, so the entire system can perform better than MLFQ-IM in less average waiting time, which means it reduces the starvation totally. However, when the system load is large, it is very difficult to determine which queue level happens starvation [3]. Therefore, there is a lot of future work to finish.

7 Reference

[1] Akhtar, Muhammad & Hamid, Bushra & Ur-Rehman, Inayat & Humayun, Mamoona & Hamayun, Maryam & Khurshid, Hira. (2015). An Optimized Shortest job first Scheduling Algorithm for CPU Scheduling. J. Appl. Environ. Biol. Sci. , 5(12)42-46, 2015. 5. 42-46.

[2] K. Hoganson and J. Brown, "Real-time scheduling with MLFQ-RT multilevel feedback queue with starvation mitigation," 2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC), Madeira, Portugal, 2017, pp. 155-160, doi: 10.1109/ICE.2017.8279883.

[3] E. J. Brown, "On Intelligent Mitigation of Process Starvation In Multilevel Feedback Queue Scheduling," Kennesaw State University, 2016.

[4] https://stackoverflow.com/questions/9758763/how-to-calculate-average-waiting-time-and-average-turn-around-time-in-sjf-schedu

Coding effort

In this total simulation, the coding effort is 90%. At the beginning of simulation, I plan to use the SimSo simulator to test these algorithms, but I found some problems in this simulator software, so finally I give up this approach. And then, I searched all the possible data structures may be used. I know how to design objects in MLFQ from this GitHub reference website (https://github.com/ananthamapod/Multi-Level-Feedback-Queue-Scheduling-Algorithm). After that, I create my own MLFQ-IM function hand by hand. As for the OSJF algorithm is simple to code, I finish this algorithm program quickly without any reference.

As for the generating data, I use this gist[random] to get the same distribution which satisfy the requirement in the paper [2]. After consulting with professor, I know the least size of this MLFQ-IM simulation is 100. Therefore, I set 100 process with different arriving time and different needed serving time, and put these processes into a process list to do our simulation.

All the steps in this simulation are signed by myself. Here are three simulations, the first one is to compare OSJF and RSJF. The second one is to compare MLFQ-IM and MLFQ. The last one is to compare MLFQ-IM and OSJF.