Zhenyu Yan

Report 4

In the report, I conclude the algorithm and implementation of non-aggressive multi-level feedback scheduler , aggressive multi-level feedback scheduler, Preemptive shortest job first scheduler, and lottery scheduler. Also discuss the Correctness Testing , Efficiency testing and compare the policy I implemented.

Non-aggressive Multi-level Feedback Scheduler and aggressive Multi-level Feedback Scheduler

I implement non-aggressive by following this algorithm:

1. Jobs are scheduled according to priority, which ranges from 0 (highest priority) to 7 (lowest priority)
2. When a job starts a burst (that is, when it becomes ready either because it has just started or because it has finished doing I/O), it is assigned priority 0 .
3. The scheduler maintains a (FIFO) queue of jobs for each priority level. The scheduler will always run the first job of the highest priority level available (i.e. lowest-numbered non-empty queue). For example, if queues 0 and 1 are empty but queue 2 is not, the scheduler will run the first job in queue 2.
4. When a job is run, it is assigned a slice, which is a number of quanta based on the priority of the job. A job at priority level 0 has a time slice length of 1 quantum, a job at level 1 has a time slice of 2 quanta, a job at level 2 has a time slice of 4 quanta, and so on. In general, a job with priority i has a time slice of 2***i*** quanta.
5. If a job with priority i uses up its time slice without blocking for I/O or terminating, the scheduler stops it, lowers its priority to i+1, and adds it at the tail of queue i+1, and selects a job as in rule (3). However if the job is already in the lowest priority queue, its priority is unchanged and it returns to the end of the same queue. While it is possible that the same job will be selected again--for example, if it is the only ready job--normally a different job will be given the opportunity to run.
6. This policy is non-aggressive in the following sense: If a job becomes ready while another job is running, it is added to the tail of queue 0, but the running job is not stopped until it terminates, blocks for I/O, or uses up its time slice.

I implement aggressive by following this algorithm:

This version is a modified version of your first version. In this version jobs arriving at the CPU scheduler can preempt running jobs, and the priority of a job is ``remembered'' from one burst until the next. In more detail, rules (2) and (5) are modified as follows:
1. (2)' When a job becomes ready because it has finished doing I/O, it is given priority i-1, where i is the priority it had when it blocked for I/O. There is no level -1,

so if a job finishes a burst at priority 0, it stays at priority 0. Newly created jobs are assigned priority 0.

2. (5)' This policy is **aggressively preemptive** in the following sense: If a job becomes ready while another job is running, it is added to the tail of the appropriate queue as defined by rule (2'), the running job is stopped and has 1 subtracted from its priority (unless it is already at priority 0), it is added to the tail of the appropriate queue, and another job is selected to run as in rule (2).

The major different for aggressive multi-level feedback and non-aggressive multi-level feedback is aggressive multi-level feedback can preempt the current process if priority is higher, and non-aggressive multi-level feedback cannot preempt and must wait till the current process finish.

Below is the code to determine if the scheduler can preempt or not:

```
int scheduler_can_preempt_multilevel( process_t* p ){
   if(SCHEDULER_AGRESSIVE){
      for(int i=0; i < p->priority; i++){
         if(ready_queue[i]->count != 0){
      return 1;
         }
      }
   }
   return 0;
}
```

The result for non – aggressive:

```
Tick     PID Burst Sleep    IO Queue                 IO Completed           Status
=====+=====+=====+=====+=======================+=======================+==================
    0:  123:     9:False:                       :                       :slice expired
    1:  124:    19:False:                       :                       :slice expired
    2:  123:     8:False:                       :                       :still running
    3:  123:     7:False:                       :                       :slice expired
    4:  124:    18:False:                       :                       :still running
    5:  124:    17:False:                       :                       :slice expired
    6:  123:     6: True:                       :                       :sleeping
    7:  124:    16:False:123                    :                       :still running
    8:  124:    15:False:123                    :                       :still running
    9:  124:    14:False:123                    :                       :still running
   10:  124:    13:False:                       :123                    :slice expired
   11:  123:     5:False:                       :                       :slice expired
   12:  123:     4:False:                       :                       :still running
   13:  123:     3:False:                       :                       :slice expired
   14:  123:     2:False:                       :                       :still running
   15:  123:     1:False:                       :                       :still running
   16:  123:     0:False:                       :                       :* exited
   17:  124:    12:False:                       :                       :still running
   18:  124:    11:False:                       :                       :still running
   19:  124:    10:False:                       :                       :still running
   20:  124:     9:False:                       :                       :still running
   21:  124:     8:False:                       :                       :still running
   22:  124:     7:False:                       :                       :slice expired
   23:  124:     6:False:                       :                       :still running
   24:  124:     5:False:                       :                       :still running
   25:  124:     4:False:                       :                       :still running
   26:  124:     3:False:                       :                       :still running
   27:  124:     2:False:                       :                       :still running
   28:  124:     1:False:                       :                       :still running
   29:  124:     0:False:                       :                       :* exited


         |  Response    | Total time   | Total time        | Total time        | Total time      | Penalty
Job#     |    time      | on cpu       | in ready to run   | in sleeping on    | in system       |  Ratio
         |              |              | state             | I/O state         |                 |
=========+==============+==============+===================+===================+=================+==============
  123    |     0        |   10         |    3              |   4               |   17            | 1.70
  124    |     1        |   20         |   10              |   0               |   30            | 1.50


  Non-aggressive Preemptive Multilevel Feedback
         Total simulation time:       30
          Total number of jobs:        2
 Shortest job turn-around time:       17
  Longest job turn-around time:       30
    Number of context switches:       11
         Average response time:     0.50
         Average penalty ratio:     1.60
       Average completion time:    23.50
   Average time in ready queue:     6.50
  Average time sleeping on I/O:     2.00
```

The result for non – aggressive:

```
Tick    PID Burst Sleep    IO Queue                 IO Completed          Status
=====+=====+=====+=====+=====================+=====================+=================
    0:  123:    9:False:                     :                     :slice expired
    1:  124:   19:False:                     :                     :slice expired
    2:  123:    8:False:                     :                     :still running
    3:  123:    7:False:                     :                     :slice expired
    4:  124:   18:False:                     :                     :still running
    5:  124:   17:False:                     :                     :slice expired
    6:  123:    6: True:                     :                     :sleeping
    7:  124:   16:False:123                  :                     :still running
    8:  124:   15:False:123                  :                     :still running
    9:  124:   14:False:123                  :                     :still running
   10:  124:   13:False:                     :123                  :preempted
   11:  123:    5:False:                     :                     :still running
   12:  123:    4:False:                     :                     :slice expired
   13:  124:   12:False:                     :                     :still running
   14:  124:   11:False:                     :                     :slice expired
   15:  123:    3:False:                     :                     :still running
   16:  123:    2:False:                     :                     :still running
   17:  123:    1:False:                     :                     :still running
   18:  123:    0:False:                     :                     :* exited
   19:  124:   10:False:                     :                     :still running
   20:  124:    9:False:                     :                     :still running
   21:  124:    8:False:                     :                     :still running
   22:  124:    7:False:                     :                     :slice expired
   23:  124:    6:False:                     :                     :still running
   24:  124:    5:False:                     :                     :still running
   25:  124:    4:False:                     :                     :still running
   26:  124:    3:False:                     :                     :still running
   27:  124:    2:False:                     :                     :still running
   28:  124:    1:False:                     :                     :slice expired
   29:  124:    0:False:                     :                     :* exited


         |  Response    | Total time    | Total time        | Total time        | Total time     | Penalty
Job#     |    time      | on cpu        | in ready to run   | in sleeping on    | in system      |  Ratio
         |              |               | state             | I/O state         |                |
=========+==============+===============+===================+===================+================+=============
  123    |    0         |   10          |    5              |   4               |   19           | 1.90
  124    |    1         |   20          |   10              |   0               |   30           | 1.50


  Aggressive Preemptive Multilevel Feedback
         Total simulation time:      30
          Total number of jobs:       2
 Shortest job turn-around time:      19
  Longest job turn-around time:      30
    Number of context switches:      12
         Average response time:    0.50
         Average penalty ratio:    1.70
       Average completion time:   24.50
   Average time in ready queue:    7.50
  Average time sleeping on I/O:    2.00
```

compare to the sample output, there is one more switch on the aggressive more than the sample.

Preemptive shortest job first scheduler

I implement sjf scheduler by following this algorithm:

In this strategy the ready queue will consist of one queue ordered according to the time that the scheduler 'thinks' the job needs on the CPU. You will need to calculate this "guess" using exponential averaging (p. 269 in textbook). The weight of the most current value is w and the default weight is 1/2. Suggest using 5 as the default for the initial guess G(1)=5, as we did in lecture.

The most important part is to how to calculate the exponential averaging. In the book,

$$guess = \frac{previous\ burst}{2} + \frac{previous\ guess}{2}$$

the fomular is :

The example in lecture:

Example

- G(1) = 5 as default value
- A(1) = 10.

```
G(2) = 1/2 * G(1) + 1/2 A(1) = 1/2 * 5.00 + 1/2 * 10 = 7.5
G(3) = 1/2 * G(2) + 1/2 A(2) = 1/2 * 7.50 + 1/2 * 10 = 8.75
G(4) = 1/2 * G(3) + 1/2 A(3) = 1/2 * 8.75 + 1/2 * 10 = 9.38
```

As the fomular and the example in our lecture, we can come up the implementation for the brust guess below:

```
p->sjf_burst = 0.5 * p->sjf_burst + 0.5*(clock - p->timeslice_started);
```

(remember, the initial guess is set to 5.0)

result:

```
Tick    PID Burst Sleep    IO Queue                  IO Completed          Status
=====+=====+=====+=====+=====================+=====================+==================
   0:  123:     9:False:                     :                     :still running
   1:  123:     8:False:                     :                     :still running
   2:  123:     7:False:                     :                     :still running
   3:  123:     6:False:                     :                     :still running
   4:  123:     5:False:                     :                     :still running
   5:  123:     4:False:                     :                     :still running
   6:  123:     3: True:                     :                     :sleeping
   7:  124:    19:False:123                  :                     :still running
   8:  124:    18:False:123                  :                     :still running
   9:  124:    17:False:123                  :                     :still running
  10:  124:    16:False:                     :123                  :still running
  11:  124:    15:False:                     :                     :still running
  12:  124:    14:False:                     :                     :still running
  13:  124:    13:False:                     :                     :preempted
  14:  123:     2:False:                     :                     :still running
  15:  123:     1:False:                     :                     :still running
  16:  123:     0:False:                     :                     :* exited
  17:  124:    12:False:                     :                     :still running
  18:  124:    11:False:                     :                     :still running
  19:  124:    10:False:                     :                     :still running
  20:  124:     9:False:                     :                     :still running
  21:  124:     8:False:                     :                     :still running
  22:  124:     7:False:                     :                     :still running
  23:  124:     6:False:                     :                     :still running
  24:  124:     5:False:                     :                     :still running
  25:  124:     4:False:                     :                     :still running
  26:  124:     3:False:                     :                     :still running
  27:  124:     2:False:                     :                     :still running
  28:  124:     1:False:                     :                     :still running
  29:  124:     0:False:                     :                     :* exited


         |  Response    | Total time   | Total time        | Total time        | Total time      | Penalty
Job#     |   time       | on cpu       | in ready to run   | in sleeping on    | in system       |  Ratio
         |              |              | state             | I/O state         |                 |
=========+==============+==============+==================+==================+================+=============
 123     |   0          |  10          |   3              |  4               |  17            | 1.70
 124     |   7          |  20          |  10              |  0               |  30            | 1.50


  Preemptive Shortest Job First
         Total simulation time:      30
          Total number of jobs:       2
 Shortest job turn-around time:      17
  Longest job turn-around time:      30
    Number of context switches:       4
         Average response time:    3.50
         Average penalty ratio:    1.60
       Average completion time:   23.50
   Average time in ready queue:    6.50
  Average time sleeping on I/O:    2.00
```

Compare to the sample, my result is very close excpet there is one less number of context switches and one more seconds in the total time in ready to run states which effect the total time in system by one second too.

Lottery Scheduler

I implement lottery scheduler by following this algorithm:

statistically guarantees a variable fraction of processor time to each runnable process. The concept is much like a lottery. At each scheduling decision, each runnable process is given a number of "lottery tickets". Then a random number is generated, corresponding to a specific ticket. The process with that ticket gets the quantum.

The implementation is below, as you can see, I sum up all the tickets and generate a random number between 0 and the total tickets +1. Then I find which jobs own that particular ticket.(below)

```
process_t* scheduler_get_job_lottery()
{
  process_t* job = NULL;
  if(queue_index == 0){
    return 0;
  }
  int totalTickets = 0;
  for(int i = 0; i < queue_index; i++){
    totalTickets += queue[i]->lottery_tickets;
  }

  int bingo = (mrand() % totalTickets)+1;
  int currentTicket = 0;
  for(int i = 0; i < queue_index; i++){
    currentTicket += queue[i]->lottery_tickets;
    if(currentTicket >= bingo){
      job = queue[i];
      queue_index--;
      queue[i] = queue[queue_index];
      job->timeslice_started = clock;
      break;
    }
  }

  return job;
}
```

Result:

```
Tick    PID Burst Sleep    IO Queue              IO Completed          Status
=====+=====+=====+=====+=====================+=====================+=================
   0:  124:   19:False:                      :                      :still running
   1:  124:   18:False:                      :                      :still running
   2:  124:   17:False:                      :                      :still running
   3:  124:   16:False:                      :                      :still running
   4:  124:   15:False:                      :                      :still running
   5:  124:   14: True:                      :                      :sleeping
   6:  123:    9:False:                      :124                   :still running
   7:  123:    8:False:                      :                      :still running
   8:  123:    7:False:                      :                      :still running
   9:  123:    6:False:                      :                      :still running
  10:  123:    5: True:                      :                      :sleeping
  11:  124:   13:False:123                   :                      :still running
  12:  124:   12:False:123                   :                      :still running
  13:  124:   11:False:123                   :                      :still running
  14:  124:   10:False:123                   :                      :still running
  15:  124:    9:False:123                   :                      :still running
  16:  124:    8:False:123                   :                      :slice expired
  17:  124:    7:False:123                   :                      :still running
  18:  124:    6:False:                      :123                   :still running
  19:  124:    5:False:                      :                      :still running
  20:  124:    4:False:                      :                      :still running
  21:  124:    3:False:                      :                      :still running
  22:  124:    2:False:                      :                      :slice expired
  23:  123:    4:False:                      :                      :still running
  24:  123:    3:False:                      :                      :still running
  25:  123:    2:False:                      :                      :still running
  26:  123:    1:False:                      :                      :still running
  27:  123:    0:False:                      :                      :* exited
  28:  124:    1:False:                      :                      :still running
  29:  124:    0:False:                      :                      :* exited


         |  Response    | Total time   | Total time       | Total time       | Total time      | Penalty
Job#     |   time       | on cpu       | in ready to run  | in sleeping on   | in system       |  Ratio
         |              |              | state            | I/O state        |                 |
=========+==============+==============+==================+==================+=================+=============
  123    |    6         |   10         |   10             |   8              |   28            | 2.80
  124    |    0         |   20         |    9             |   1              |   30            | 1.50


  Lottery
         Total simulation time:       30
          Total number of jobs:        2
Shortest job turn-around time:       28
 Longest job turn-around time:       30
   Number of context switches:        6
         Average response time:     3.00
         Average penalty ratio:     2.15
       Average completion time:    29.00
   Average time in ready queue:     9.50
  Average time sleeping on I/O:     4.50
```

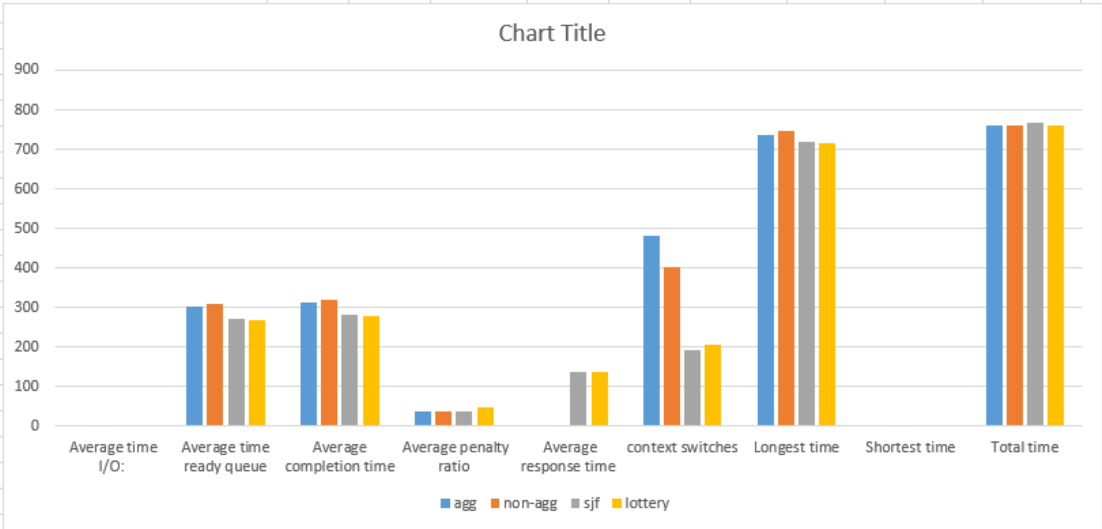
(because there is no lottery scheduler in the binary test, I can not compare)

Comparesion and effciency

type	lottery	sjf	non-agg	agg
Average time I/O:	2.57	2.41	2.23	2.44
Average time ready queue	268.3	271.7	309.09	303.95
Average completion time	278.47	281.71	318.92	313.99
Average penalty ratio	47.72	37.9	36.26	36.15
Average response time	136.3	139.03	1.73	1.37
context switches	207	193	402	483
Longest time	714	719	747	736
Shortest time	2	2	2	2
Total time	761	766	760	760

In comparsion of above table , aggressive and non aggressive number of context switch is much higher than sjf and lottery, but much faster for the response time. Also they ar slighterly higher in ready queue and completion.

Conclustion:



In conclusion, we inplement fou different scheduler and the coordinator to compare their perfromance in different circumstance.

1. Aggressive is better than non aggressive but it require more context switches.
2. Sjf and lottery is faster than aggressitve and non aggressive has lower context switch but it reuqire more response time.
3. Aggreesive require the most context switches.
4. Non-aggressive has the longest system time for job.

(For intrustion, read “README.txt” file and it include an youtube demo video link in it.)