Project 3

100 Possible Points

11/9/2023

Add Comment

11/12/2023

Details

Problem Statement

You are to implement a simulation of three simple scheduling policies on a single processor: FIFO, SJF (preemptive), and RR with a time slice of 1. The policy to simulate should be selected by a command line argument of -fifo, -sjf, or -rr, respectively. Your simulation must read input from standard input so that I/O redirection can be used to run different test cases.

Discussion

Your program should read pairs of numbers from standard input. The first number is the arrival time of a task, and the second number is the service time of that task. The input pairs will be ordered according to non-decreasing arrival times (non-decreasing since multiple tasks can arrive at the same time). Your program should assign alphabetic task identifiers to each task in arrival order, starting with tid A. You may assume that there will no more than 26 tasks (i.e, A-Z). Note that there may be idle times prior to the first task arriving, idle times between groups of tasks, as well as tasks arriving at the same time. E.g., this is a valid workload:

```
3 4
10 4
10 2
```

Your program should first print a trace of each time unit, labeled as the discrete time value at the beginning of the time unit and showing what task is running on the CPU and what tasks are waiting in the ready queue during that time unit. Any arrivals, preemption, and scheduling occur at the discrete time value that starts a time unit and incur zero overhead. Completions occur at the discrete time value that ends a time unit.

During the trace, the tasks should be identified as the tid concatenated with rst, where tid is the task identifier assigned upon arrival and rst is the current remaining service time evaluated at the start of a time unit.

You should stop the simulation after all tasks have completed. (Note that this means that the processor is empty, the ready queue is empty, and there is no more input.)

Note that for SJF preemptive, a running task is preempted only when an arriving task has less service time. Also, note that for RR, a preempted task goes to the back of the ready queue after any arrivals have been added.

After the simulation ends, your program should print a task summary table, ordered by ascending task identifer and containing the tid, arrival time, service time, completion time, response time, and wait time for each task. Finally, your program should print a table of service time and wait time pairs from the task summary table, ordered by ascending service time.

You may use any programming language that supports command line arguments and I/O redirection on the command line. Because of the queues used in the simulation, you may want to consider using C++ so that you have access to vectors.

If you choose to program in C using explicit linked lists, you might structure tasks as follows:

```
struct task{
  int
    task_id,       /* alphabetic tid can be obtained as 'A'+(task_counter++) */
    arrival_time,
    service_time,
    remaining_time,
    completion_time,
    response_time,
    wait_time;
  struct task *next;
};
```

I anticipate that we will use Gradescope to submit and test your programs.

Pedagogical Rationale

This assignment has the students implement three classic uniprocessor scheduling algorithms.

Guidelines

The code should be written totally by yourself.

You may discuss the project requirements and the concepts with me or with anyone in the class.

However, you should not send code to anyone or receive code from anyone, whether by email, printed listings, photos, visual display on a computer/laptop/cell-phone/etc. screen, or any other method of communication.

Do not post the assignment, or a request for help, or your code on any web sites.

The key idea is that you shouldn't short-circuit the learning process for others once you know the answer. (And you shouldn't burden anyone else with inappropriate requests for code or "answers" and thus short-circuit your own learning process.)

Grading

There will be multiple test cases for which the output must match exactly. The test cases will be equally weighted. Note that a corrupted submission or a submission that does not compile will receive 0 points. Hardcoded answers will also receive 0 points.

Please note that similarity checking will be used to help detect plagiarism. The Clemson policy on Academic Integrity is available **here.** (https://catalog.clemson.edu/content.php?catoid=39&navoid=1230#undergraduate-academic-integrity)

Example Expected Output

Let "in1" be the file containing:

```
3 4
10 4
10 2
```

For the command line "./a.out -fifo < in1", the output would be:

```
FIFO scheduling results

time   cpu   ready queue (tid/rst)
----   ---   ---------------------
  0          --
  1          --
  2          --
  3    A4    --
  4    A3    --
  5    A2    --
  6    A1    --
  7          --
  8          --
  9          --
 10    B4    C2
 11    B3    C2
 12    B2    C2
 13    B1    C2
 14    C2    --
 15    C1    --

     arrival service completion response wait
tid   time    time      time       time   time
---  ------- ------- ---------- -------- ----
 A      3       4         7         4      0
 B     10       4        14         4      0
 C     10       2        16         6      4

service wait
 time   time
```

```
------- ----
   2      4
   4      0
   4      0
```

For the command line "./a.out -sjf < in1", the output would be:

```
SJF(preemptive) scheduling results

time   cpu   ready queue (tid/rst)
----   ---   --------------------
  0          --
  1          --
  2          --
  3    A4    --
  4    A3    --
  5    A2    --
  6    A1    --
  7          --
  8          --
  9          --
 10    C2    B4
 11    C1    B4
 12    B4    --
 13    B3    --
 14    B2    --
 15    B1    --

     arrival service completion response wait
tid   time    time      time      time   time
---  ------- ------- ---------- -------- ----
 A      3       4         7         4      0
 B     10       4        16         6      2
 C     10       2        12         2      0

service wait
 time   time
------- ----
   2      0
   4      0
   4      2
```

For the command line "./a.out -rr < in1", the output would be:

```
RR scheduling results (time slice is 1)

time   cpu   ready queue (tid/rst)
----   ---   --------------------
  0          --
  1          --
  2          --
  3    A4    --
  4    A3    --
  5    A2    --
  6    A1    --
  7          --
  8          --
  9          --
 10    B4    C2
 11    C2    B3
 12    B3    C1
 13    C1    B2
 14    B2    --
 15    B1    --

     arrival service completion response wait
tid   time    time      time      time   time
--- ------- ------- ---------- -------- ----
 A      3       4          7        4      0
 B     10       4         16        6      2
 C     10       2         14        4      2

service wait
 time   time
------- ----
   2      2
   4      0
   4      2
```

Let "in2" be the file containing:

```
2 6
4 4
6 2
```

The expected outputs from the three runs are:

```
% ./a.out -fifo < in2
FIFO scheduling results

time   cpu   ready queue (tid/rst)
----   ---   --------------------
  0          --
  1          --
  2    A6    --
  3    A5    --
  4    A4    B4
  5    A3    B4
  6    A2    B4, C2
  7    A1    B4, C2
  8    B4    C2
  9    B3    C2
 10    B2    C2
 11    B1    C2
 12    C2    --
 13    C1    --

     arrival service completion response wait
tid   time    time      time      time   time
--- ------- ------- ---------- -------- ----
 A      2       6          8        6      0
 B      4       4         12        8      4
 C      6       2         14        8      6

service wait
 time   time
------- ----
   2      6
   4      4
   6      0

% ./a.out -sjf < in2
SJF(preemptive) scheduling results

time   cpu   ready queue (tid/rst)
----   ---   --------------------
```

```
  0         --
  1         --
  2    A6    --
  3    A5    --
  4    A4    B4
  5    A3    B4
  6    A2    C2, B4
  7    A1    C2, B4
  8    C2    B4
  9    C1    B4
 10    B4    --
 11    B3    --
 12    B2    --
 13    B1    --

     arrival service completion response wait
tid   time    time      time      time   time
---  ------- ------- ---------- -------- ----
 A      2       6         8         6      0
 B      4       4        14        10      6
 C      6       2        10         4      2

service wait
 time   time
------- ----
   2      2
   4      6
   6      0

% ./a.out -rr < in2
RR scheduling results (time slice is 1)

time   cpu   ready queue (tid/rst)
----   ---   --------------------
  0         --
  1         --
  2    A6    --
  3    A5    --
  4    B4    A4
  5    A4    B3
  6    B3    C2, A3
  7    C2    A3, B2
  8    A3    B2, C1
  9    B2    C1, A2
 10    C1    A2, B1
 11    A2    B1
 12    B1    A1
 13    A1    --

     arrival service completion response wait
tid   time    time      time      time   time
---  ------- ------- ---------- -------- ----
 A      2       6        14        12      6
 B      4       4        13         9      5
 C      6       2        11         5      3

service wait
 time   time
------- ----
   2      3
   4      5
   6      6
```