CS 550 Operating Systems, Spring 2018
Programming Project 2

Out: 3/15/2018, Thursday
Due date: 4/8/2018, Sunday 23:59:59

Overview: in this project, you are going to build a special **Multi-Level Feedback Queue (MLFQ)** scheduler in xv6. This MLFQ scheduler has two queues:

- the first queue has the higher priority, and uses round robin scheduling for processes within the queue;
- the second queue has the lower priority, and uses priority-based scheduling for processes within the same queue.

A process in the first queue is demoted to the second queue when its accumulated running time in the first queue exceeds a configured value. A process in the second queue is promoted to the first queue when its accumulated wait time in the second queue exceeds a configured value.

1 Baseline xv6 source code

ATTENTION: before you proceed, read the whole spec carefully and make sure you understand all the requirements, especially those highlighted parts in this section, the submission requirements (Section 3), and the grading guidelines (Section 4).

- Log into your GitHub account whose username is the same your BU username.
- Go to the link at the end of this section to accept the assignment. This will create a private repository of your own on the instructor's teaching organization, and start importing the baseline code into your private repository.

  Note: sometimes the import process may take a long time. Do not click the "Cancel" button, which will lead to an empty repository. You can just close the page and come back later.

  If you have clicked the "Cancel" button or it already took really long (e.g., more than two hours), contact the instructor and the TAs so we can delete your repository, and you can click the assignment link to import again.
- Once the import process finishes, you can clone or download the repository into your computer and start working.
- Start early! The instructors and the TAs will be less responsive during the last several days before the due date, and will not respond to any email inquiries regarding the project in the last day.
- Refer back to the spec of PROJ0 and PROJ1 for details of running xv6.

Assignment link: `https://classroom.github.com/a/zzsc2D1n`

2 Building an MLFQ scheduler (100 points)

The original scheduler of xv6 adopts a round-robin (RR) policy. In this part, you are going to implement a special MLFQ scheduler, which adopts the following scheduling policies.

2.1 The MLFQ Scheduling Policies

1. We only consider the single CPU case. The Makefile in the base code has been set up accordingly. You don't need to do anything on this.
2. The MLFQ has two queues — the first queue (i.e., Queue 0) has the higher priority, and the second queue (i.e., Queue 1) has the lower priority.
3. **Queue 0 (round-robin)**
   - When a job/process enters the system, it is always placed at the end of Queue 0.
   - Queue 0 has higher priority than Queue 1. Specifically, at any given point in time, if there are ready/runnable processes in Queue 0, they have higher priority to be scheduled than processes in Queue 1.
   - If there are multiple processes in Queue 0, these process are scheduled in a round robin manner.
   - Each process in Queue 0 has a run time counter (named as *running_tick*), which records the process's running time in timer ticks since joining Queue 0. Therefore, this counter should be set to 0 when the process is initially inserted to Queue 0, and every time when it is promoted from Queue 1.
   - When the scheduler function is invoked (i.e., when a timer tick[1] occurs), the process that is scheduled to run next is considered to consume a timer tick, and therefore its *running_tick* value should be incremented by one.
   - **Demotion:** Queue 0 is associated with a configurable value (named as *RUNNING_THRESHOLD*), which is the timer ticks quota of Queue 0. Once a process's *running_tick* exceeds *RUNNING_THRESHOLD*, this process is moved to Queue 1.
   - The first two *system* processes "init" and xv6's "shell" (with the pid of 1 and 2 separately) should always stay in Queue 0.
4. **Queue 1 (priority-based)**
   - If there are multiple processes in Queue 1, these process are scheduled in a priority-based manner. The priorities of the processes in Queue 1 are determined as follows.
   - Each process in Queue 1 has a wait time counter (named as *waiting_tick*), which records the process's waiting time for the CPU in timer ticks since joining Queue 1. Therefore, this counter should be set to 0 every time when the process is demoted from Queue 1.
   - When the scheduler function is invoked (i.e., when a timer tick occurs), all the processes in Queue 1 that are not to be scheduled to run next are considered to spend one timer tick in waiting for the CPU, and therefore the *waiting_tick* values of all these processes should be incremented by one.

[1]Here a "timer tick" is a "clock tick" in xv6's term. It essentially a timer interrupt. Hint: check `trap()` in `trap.c` to see how a timer interrupt leads up to the process scheduler function.

- The *waiting_tick* values of the processes in Queue 1 serve as the priorities of these processes in Queue 1: if there are multiple processes in Queue 1 while none in Queue 0, the process in Queue 1 with the largest *waiting_tick* value has the highest priority to be scheduled.
- **Promotion:** Queue 1 is associated with a configurable value (named as *WAITING_THRESHOLD*), which is promotion threshold. Once a process's *waiting_tick* exceeds *WAITING_THRESHOLD*, this process is moved to Queue 0.

5. Pinning processes to Queue 0. We need to allow some processes to always stay in Queue 0. For this purpose, a system call is needed to pin processes to Queue 0 (see details below).

2.2 What To Do?

1. Implement an MLFQ scheduler that strictly follows the above scheduling policies presented in Section 2.1.
2. Switch between xv6's default RR (round robin) scheduler and the MLFQ scheduler.

   - Your MLFQ code should not overwrite xv6's defautl RR scheduler. Instead, you should provide an option to switch between these two schedulers. You only need to choose the desired scheduler before xv6 runs.
   - Place an integer variable (named as "`sched_policy`") at the top of "proc.c":

     ```
     int sched_policy = 1;
     ```

     Use this variable in your code to determine which scheduling policy is active: if the value of `sched_policy` is 0, the default RR policy is adopted. If the value of `sched_policy` is 1, the MLFQ scheduler is used.

     Notice that, your default scheduler should be MLFQ (i.e., set the default value of `sched_policy` to 1). We will test your code based on this assumption. Failure to comply will lead to point deduction.

3. Implement a system call that allows user programs to set the *RUNNING_THRESHOLD* for Queue 0.

   - Write the corresponding system call user space wrapper function, and declare it in "user.h". The wrapper function's prototype should be:

     ```
     int setrunningticks(int time_allotment);
     ```

     This user-level wrapper function takes one positive integer argument: the value of *RUNNING_THRESHOLD* to be set. The return value indicates whether this functions succeeds (i.e., 0) or not (i.e., 1).

     Notice that, the default time of *RUNNING_THRESHOLD* should be 2.

4. Implement a system call that allows user programs to set the *WAITING_THRESHOLD* for Queue 1.

   - Write the corresponding system call user space wrapper function, and declare it in "user.h". The wrapper function's prototype should be:

     ```
     int setwaitingticks(int waiting_thres);
     ```

This user-level wrapper function takes one positive integer argument: the value of *WAITING_THRESHOLD* to be set. The return value indicates whether this functions succeeds (i.e., 0) or not (i.e., 1).

Notice that, the default time of *WAITING_THRESHOLD* should be 4.

5. Implement a system call that allows user program to pin a process in Queue 0.

   - Write the corresponding system call user space wrapper function, and declare it in "user.h". The wrapper function's prototype should be:

     ```
     int setpriority(int pid, int priority);
     ```

     This user-level wrapper function takes two integer arguments. The first argument is the process's pid. The second is the priority value — 0 indicates that the process stays in Queue 0 all the time, a non-zero value indicates that the process should be scheduled according to the MLFQ policy described above. The return value indicates whether this functions succeeds (i.e., 0) or not (i.e., 1)

     Notice that, the default priority for processes should be NON-ZERO.

2.3 Scheduling tracing functionality

To help you implement and debug, a simple scheduling tracing functionality has been added to the base code. When this tracing functionality is enabled, the kernel prints the PID of the currently running process every time before the CPU is given back to the scheduler in the timer interrupt handler.

With this scheduling tracing functionality enabled, you can see the sequence of processes that the scheduler schedules. To enable it, you simply put "1" to the following variable in "proc.c":

```
int sched_trace_enabled = 1;
```

2.4 Hints

- Most of the code of xv6 scheduler is located in `proc.c`. The header file `proc.h` is also useful and you may need to modify accordingly. In addition, you may want to understand the default RR scheduler before implementing the MLFQ scheduler.
- Do not mess up xv6's default RR scheduler. Thus, you will always have a working scheduler for the system.
- Build and test one function at a time. For example, in the beginning, you could only implement a round robin queue for Queue 0. Then, you could use round robin queues for both Queue 0 and Queue 1. Last, you replace the round robin queue with a queue that realizes the priority-based policy for Queue 1.
- In addition to the test driver program (in Section 2.3, you would probably need other tracing points to track the status of each process and the behaviors of the scheduler (using cprintf) for debugging your code — you will probably want to trace the processes in each queue, the current running process, the consumed ticks (before a process will be demoted to Queue 1), the waiting ticks (before a process will be promoted to Queue 0), etc.

- Please DISABLE all your debugging print functions and output information, but only leave the test driver program enabled (i.e., int sched_trace_enabled = 1) before submitting your code. (You will lose points if you keep your debugging information on, and/or not enable the test driver program).

Note: Your implementation should keep the patch that fixes the always-100% CPU utilization problem (hint: to understand this patch, pay attention to how the local variable "`ran`" in `scheduler(void)` (in proc.c) is used). If your code causes the problem to re-occur, 10 points off.

2.5 Test Cases

We have included 3 test cases (test1 $\sim$ test3) included the BASE code, which cover some basic test scenarios. They are not included in the compilation — you may need to compile them after you correctly implement all required system call functions.

Obviously, they do not cover all test scenarios. Thus, first make sure your code work properly under these test cases. Then come up with as many test cases as possible to test your code extensively. Our final test cases are not limited to the three cases.

- Test 1: This test case tests the round robin scheduling in Queue 0.
  In the test program, we set the *RUNNING_THRESHOLD* to a large number, so that processes in Queue 0 (round robin) will not get demoted to Queue 1 (priority based).

  Three processes, A, B, and C are created (we use pipe to synchronize them to make sure they start running almost at the same time).

  As these three processes reside in the round robin queue, they should run by turns in a round robin manner. Hence, the expected results should be like:

  .....CBACBACBACBACBA.....

  Of course, it could be:

  ...ABCABC... or, ...ACBACB..., etc.

  Notice that, you need to enable the scheduling tracing functionality (see Section 2.3), and the output will be the pid of process A, B, and C.
- Test 2: This test case tests the priority-based scheduling in Queue 1.
  In the test program, we set the *WAITING_THRESHOLD* to a large number, so that once a process is demoted, it will stay in Queue 1.

  — Three processes, A, B, and C, are created. At the beginning, C is pinned to stay in Queue 0 using the `setpriority()` system call. During the time when C is pinned in Queue 0, A and B are first demoted to Queue 1, and start accumulating their *waiting_tick* values.

  — After some time, C is unpinned from Queue 0 (using the `setpriority()` system call), and thus it is demoted to Queue 1 shortly.

  — After C is demoted to Queue 1, all the three processes A, B, and C are in Queue 1 and there is no process in Queue 0. In this case, it is expected that only process A and B should get scheduled at the beginning. This is because, when C is demoted, A and B has a large *waiting_tick* value, and C's *waiting_tick* value is 0. Therefore A and B have higher priority than C to get scheduled.

— But while C is waiting in Queue 1, its *waiting_tick* value increased faster than A and B's. So eventually C's *waiting_tick* will catch up with A and B's. In this case, A, B, and C are scheduled as if it is round robin.

— With the above explanation, the expected results after C is demoted into Queue 0 should be:

.....ABABABABABAB.....ABCABCABC.....

or something like:

.....AABBAABBAABB.....ABCCBAABCCBA.....

depending on how you deal wilt two processes with the same priority value. In other words, you are fine as long as the scheduling pattern of your implementation matches what's expected.

- Test 3: This test case tests demotion and promotion.
  In the test program, we set both *RUNNING_THRESHOLD* and *WAITING_THRESHOLD* to a reasonable value (e.g., 10 and 20 separately).

  Two processes, A and B, are created. We pin process B to the Queue 0.

  So, we except that:

  (1) A and B should first execute in the round robin queue for a while:

  .....ABABABABABAB.....

  (2) Then A gets demoted to the Queue 1. As B still stays in the Queue 0, B will be executing during this period:

  .....BBBBBBBBBBBB.....

  (3) During this period, A accumulates waiting ticks and gets promoted to Queue 0. So the output should look like:

  .....ABABABABAB.....

  (4) So on and so forth

  Overall, the expected results should be:

  .....ABABABABABAB.....BBBBBBBBBB.....ABABABABAB.....

3 Submit your work

If you use the GitHub website upload function to commit code to the GitHub server, remember to "`make clean`" before uploading. Uploading intermediate files, such as `.o`, `.d` files will lead to points taken (see grading section for details).

A submission link be available on the MyCourses website. **Once your code in your GitHub private repository is ready for grading, submit a file named "DONE", which should include the following info**, to that link:

- Your name and B-number.
- **The SHA1 value (a 40-char string) of the latest commit.** You can obtain it from the GitHub website or from the "`git log`" command. (Note: this is similar to PROJ0.)

Additionally, you can include the following info in this file:

- The status of your implementation (especially, if not fully complete).
- A description of how your code works, if that is not completely clear by reading the code (note that this should not be necessary, ideally your code should be self-documenting).
- Possibly a log of test cases which work and which don't work.
- Any other material you believe is relevant to the grading of your project.

Suggestion: Test your code thoroughly on a CS machine before submitting.

4 Grading

The following are the general grading guidelines for this and all future projects.

(1) **The code in your repository will not be graded until a "DONE" file is submitted to MyCourses**.

(2) The submission time of the "DONE" file shown on the MC system will be used to determine if your submission is on time or to calculate the number of late days. Late penalty is 10% of the points scored for each of the first two days late, and 20% for each of the days thereafter.

(3) Your GitHub username should be the same as your BU username. If your BU username has been used by someone else, contact the instructor for the solution.

If you have multiple GitHub accounts, please make sure you log into the one which has your BU username as the GitHub username before you accept the assignment.

Once you accept the assignment, you can check what account you used to accept the assignment. For example, if your the GitHub account you use to accept the assignment is "abc", the name of your private repository will be "cs550-18s-projX-abc" (with X=0, 1, 2, 3, or 4). If you use a GitHub account whose username is different from you BU username, our script will not be able to locate your repository.

If you made the mistake, you can correct it by logging into the correct GitHub account, and go to the assignment link and accept again.

Any grading issues caused by using incorrect GitHub account will lead to 10 points off.

(4) Uploading intermediate files (e.g., `.o`, `.d` files) will lead to 5 points off ("`make clean`" before uploading can avoid this).

(5) No SHA1 value of the latest commit in the "DONE" file, 5 points off.

(6) If you are to compile and run the xv6 system on the department's remote cluster, remember to use baseline xv6 source code provided by our GitHub classroom. Compiling and running xv6 source code downloaded elsewhere can cause 100% CPU utilization on QEMU.

Removing the patch code from the baseline code will also cause the same problem. So make sure you understand the code before deleting them.

If you are reported by the system administrator to be running QEMU with 100% CPU utilization on QEMU, 10 points off.

(7) If the submitted patch cannot successfully patched to the baseline source code, or the patched code does not compile:

```
1   TA will try to fix the problem (for no more than 3 minutes);
2   if (problem solved)
3     1%-10% points off (based on how complex the fix is, TA's discretion);
4   else
5     TA may contact the student by email or schedule a demo to fix the problem;
6     if (problem solved)
7       11%-20% points off (based on how complex the fix is, TA's discretion);
8     else
9       All points off;
```

So in the case that TA contacts you to fix a problem, please respond to TA's email promptly or show up at the demo appointment on time; otherwise the line 9 above will be effective.

(8) If the code is not working as required in the project spec, the TA should take points based on the assigned full points of the task and the actual problem.

(9) Lastly but not the least, stick to the collaboration policy stated in the syllabus: you may discuss with you fellow students, but code should absolutely be kept private. Any kind of cheating will result in zero point on the project, and further reporting.

References

[1] XV6 a simple, Unix-like teaching operating system. https://pdos.csail.mit.edu/6.828/2014/xv6/book-rev8.pdf.