HW5 - Multi Level Queue (MLQ) scheduler

תאריך הגשה: 05.07.2023 בשעה 23:59

נהלי פנייה: בכל שאלה הנוגעת לתרגיל, אנא כתבו בפורום הייעודי במודל.

פניות הנוגעות להארכה בהגשת תרגיל הבית – אנא כתבו מייל מפורט למתרגל שאחראי על התרגיל בצירוף אישורים מתאימים. מייל: mizrahid@campus.technion.ac.il

In this task you will replace the existing process scheduler with an MLQ scheduler. Note that this is a different policy than the 'multilevel feedback queue' that we learned in class. There are diagrams that demonstrate this policy at the bottom of this document.

The objectives:

1. Understand how context switching is done in the kernel
2. Hands-on with a scheduling algorithm

Preparations

Use the correct source code in the linux Virtual Machine

You will use the same Azure virtual machine from previous homework.

The directory `xv6-public` in the virtual machine has to be prepared with the correct version of the source code. **FAILING TO FOLLOW THESE STEPS WILL CAUSE YOUR SUBMISSION TO FAIL.** Write the following commands in the terminal:

```
cd xv6-public
git reset --hard
git clean -f
git remote set-url origin https://github.com/noam1023/xv6-public
git pull
git checkout mlq
make clean qemu-nox
```

Reading

- Read chapter 6 (Scheduling) in the guide: https://pdos.csail.mit.edu/6.828/2019/xv6/book-riscv-rev0.pdf
- Read the current implementation in XV6 of the round-robin scheduler: the function `void scheduler(void)` in the file `proc.c`. This is the main function that you will modify in this exercise. We recommend that before you start doing so, keep on the side the original code, and the original executable, so you will have a reference. (For example, one may think of copying the XV6 folder and working on another one. To do so, change the name of you current XV6 folder, and re-install the xv6-public folder using the command line in the terminal: *git clone https://github.com/noam1023/xv6-public* Make sure that when you write the command, you are in the /home/student/ directory). This is a suggestion, and you can think of any other way. Remember that you need to write again the above commands in the new directory.

Delete a place holder file

Delete the file “set_priority.c” (make sure to delete all declarations regarding this functions).

Need to start over?

If you made a terrible mess and you want to start from a completely new directory, erasing the old directory:

cd /

rm -rf xv6-public # this will delete the directory!

git clone https://github.com/noam1023/xv6-public

Don't forget to run the commands written in the "use the correct source code" section

The Task

Your MLQ scheduler must follow these rules:

- There should be three priority levels, numbered from 2 (highest) down to 0 (lowest). At creation, a process starts with priority 1.
- Whenever the xv6 timer tick occurs (by default this happens every 10 ms), the highest priority process which is ready (‘RUNNABLE’) is scheduled to run. That is, this is a **preemptive scheduler**.
- The highest priority ready process is scheduled to run whenever the previously running process exits, sleeps, or otherwise ‘yields’ the CPU.
- Your scheduler should schedule all the processes at each priority level in a **round robin** fashion.
- When a timer tick occurs, whichever process was currently using the CPU should be considered to have used up an entire timer tick's worth of CPU, even if it did not start at the previous tick (note that a timer tick is different than the time-slice).

- Time-slices:

Priority	Timer ticks in one time slice:
2	8
1	16
0	32

- If a process voluntarily relinquishes the CPU before its time-slice expires at a particular priority level, <u>its time-slice is reset</u>; the next time that that process is scheduled, it will have a new time-slice at that priority level.
- **A process in a given priority completes its time slice before other processes in the same priority can run**. In other words, a new process will not preempt the running process in this priority. The running process will stop running if preempted by a higher priority process, or it relinquish (i.e. give away) the CPU.

Implement a system call to set the current process priority and a user space function that will call the system call:

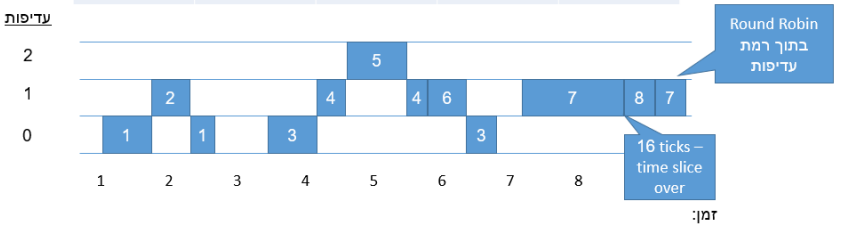
```
/**
  set the current process priority (0..2)
  @return 0 if success, non zero if error
 **/
int set_priority(int new_priority);
```

NOTE: there is currently a placeholder called “set_priority.c” . Delete this file.

Examples

דוגמה 1.
Multi-level Queue (MLQ)
הנחות:
זמן בשניות
כל tick = 0.1 שניה.

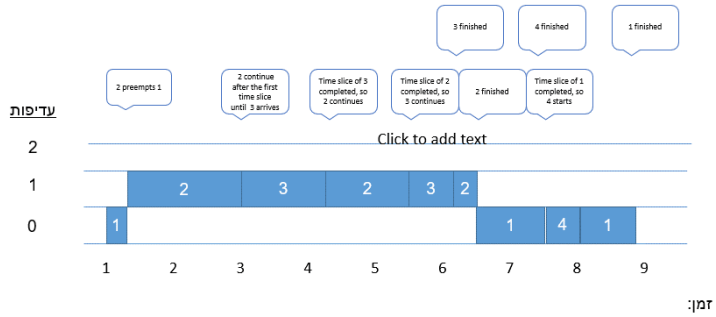
תהליך	עדיפות	הגעה	התחלה	סיום
1	0	1	1	2.7
2	1	1.65	1.7	2.2
3	0	3.5	3.5	6.7
4	1	4.06	4.1	5.7
5	2	4.6	4.6	5.5
6	1	4.5	5.7	6.4
7	1	7.1	7.1	9.2
8	1	7.2	8.7	8.8



Pay attention to process 2: it arrives at 1.65, which is inside a time tick interval. The scheduler will look which process to run at the next tick which is 1.7

דוגמה 2.
Multi-level Queue (MLQ)
הנחות:
זמן בשניות
כל tick = 0.1 שניה.

תהליך	עדיפות	הגעה	התחלה	סיום
1	0	1	1	8.9
2	1	1.2	1.2	6.5
3	1	3.0	3	6.1
4	0	3.5	7.6	8.0



In this example, we show that process 2 runs after it completed its first time slice because nobody with its priority (or higher) requests the CPU. **When Process 3 arrives, it waits until process 2 for completed the current time slice**.

The same goes for processes 1 and 4: process 1 resumes at 6.5 and completes its time slice. Only then, process 4 (who is waiting impatiently since time 3.5) can run. Process 4 finishes at 8.0 and then process 1 continues (because there are no other processes at this or higher priority)

Testing yourself

In the XV6 terminal, write the command `fairness`.

The outputs before the implemented changes and after are:

Initial output

```
xv6$ fairness
parent run at pid 4
Child(5) is setting prio: 1
Child(6) is setting prio: 1
Child(7) is setting prio: 0
Child(8) is setting prio: 0
Child(9) is setting prio: 2
Child(10) is setting prio: 2
Child(11) is setting prio: 1

Child(8) DONE

Child(5) DONE

Child(6) DONE

Child(9) DONE

Child(10) DONE

Child(7) DONE

Child(11) DONE

PARENT finished
```

Desired output

```
xv6$ fairness
parent run at pid 3
Child(4) is setting prio: 1
Child(5) is setting prio: 1
Child(6) is setting prio: 0
Child(7) is setting prio: 0
Child(8) is setting prio: 2

Child(8) DONE
Child(9) is setting prio: 2

Child(9) DONE
Child(10) is setting prio: 1

Child(4) DONE

Child(5) DONE

Child(10) DONE

Child(6) DONE

Child(7) DONE

PARENT finished
```

Read the source code of `check_proc_order.c` to learn why the expected output looks like this.

The Checker link:

(will be published soon)

Submission

1. Clean the directory by running `make clean`.
2. Create a patch file named HW5_ID1_ID2.patch
3. Upload the patch to the test server. Once you are happy with the result, submit to Moodle.