094210 – computer organization and operating systems

Version 1.3

version	date	change	comment
1.3	2020-06-09	Fix def of set_priority()	

Assignment 2 –

implement Multi Level Queue (MLQ) scheduler in xv6

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

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.

Your MLQ scheduler must follow these rules:

- There should be four priority levels, numbered from 3 (highest) down to 0 (lowest). At creation, a process starts with priority 2.
- 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, other than level 0, in a round robin fashion. At level 0 it should use a FIFO scheduler.
- 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 the time slice:
3	8
2	16
1	32

At level 0 it executes the process until completion or interrupted by a higher priority process, or the process yields the CPU.

- If a process voluntarily relinquishes the CPU before its time-slice expires at a particular priority level, its time-slice should not be reset; the next time that that process is scheduled, it will continue to use the remainder of its existing time-slice at that priority level.

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..3)

  @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 (and update Makefile) or override its contents.

For the testing code, implement the following function:

```
int wait2(int *retime, int *rutime, int *stime, int* elapsed);
```

This function waits for a child process termination, and updates the following parameters:

retime: The aggregated number of clock ticks during which the process waited (was able to run but did not get CPU)

rutime: The aggregated number of clock ticks during which the process was running

stime: The aggregated number of clock ticks during which the process was waiting for I/O (was not able to run).

elapsed: number of ticks since process creation until death

You can assume that the pointer parameters sent to the function are not NULL.

The function should return the pid of the child process or -1 if it fails for some reason.

Getting the source code to start with:

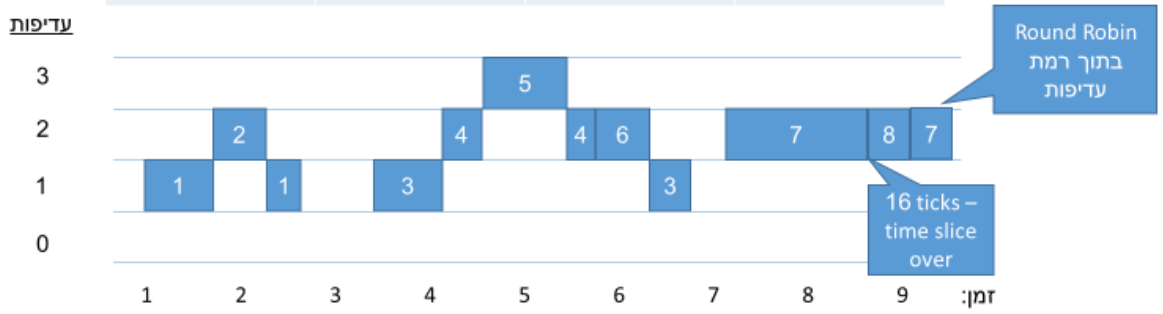
```
mkdir hw5 && cd hw5
git clone https://github.com/noam1023/xv6-public.git
cd xv6-public
git checkout mlq
```

Submit a patch file with all your code in Moodle. (see https://stackoverflow.com/questions/5159185/create-a-git-patch-from-the-changes-in-the-current-working-directory)

Sample run to help understand the process scheduling:

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

תהליך	הגעה	התחלה	סיום
1	1	1	2.7
2	1.65	1.7	2.2
3	3.5	3.5	6.7
4	4.1	4.1	5.7
5	4.6	4.6	5.5
6	4.5	5.7	6.4
7	7.1	7.1	9.2
8	7.2	7.6	8.8

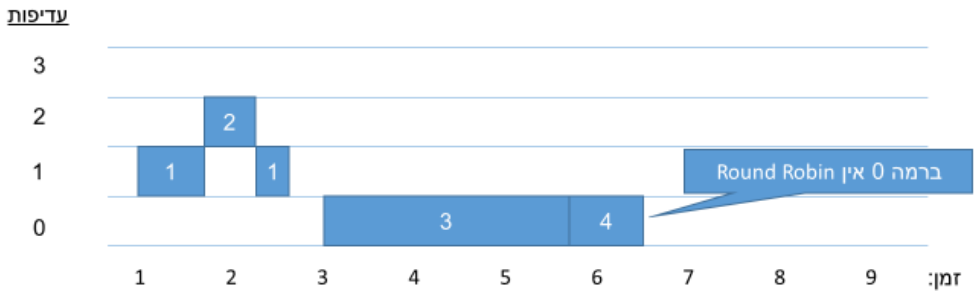


Pay attention to process 2: it arrives at 1.65, which is within a time tick interval. The scheduler will look which process to run at the next tick which is 1.7

דוגמה 2.
Multi-level Queue (MLQ)

הנחות:
זמן בשניות
כל tick = 0.1 שניה.

תהליך	הגעה	התחלה	סיום
1	1	1	2.7
2	1.7	1.7	2.2
3	3	3	5.7
4	3.5	5.7	6.5



Source code to use:

You will use a version of xv6 with special support code for the testing. This code will be given to you on Sunday. A link to the repo will be published in Moodle.

Submission:

Submit a single patch file (same idea as in hw4)

Due date: June 14th