CIS 345/545 Project 2 Fall 2020

(Due Oct. 29)

In this project, your tasks are to extend the functionality of threading system in Pintos (CSU version). These tasks are described below and the corresponding reference material can be found in:

`http://grail.eecs.csuohio.edu/~cis345s/PintosCSU_Ref.pdf`

You will be working primarily in the 'threads' directory for this assignment, with some work in the 'devices' directory on the side. Compilation should be done in the 'threads' directory.

Task 1: Efficient Alarm Clock

In Pintos, threads may call this function to put themselves to sleep:

```
/**
* This function suspends execution of the calling thread until time has
* advanced by at least x timer ticks. Unless the system is otherwise idle, the
* thread need not wake up after exactly x ticks. Just put it on the ready queue
* after they have waited for the right number of ticks. The argument to
* timer_sleep() is expressed in timer ticks, not in milliseconds or any another
* unit. There are TIMER_FREQ timer ticks per second, where TIMER_FREQ is a
* constant defined in devices/timer.h (spoiler: it's 100 ticks per second).
*/
void timer_sleep (int64_t ticks);
```

`timer_sleep()` is useful for threads that operate in real-time (e.g. for blinking the cursor once per second). The current implementation of `timer_sleep()` is inefficient, because it calls `thread_yield()` in a loop until enough time has passed. Your task is to re-implement `timer_sleep()` so that it executes efficiently without any "busy waiting". That is, your implementation has to use an ordered list so that any thread calling `timer_sleep()` will be inserted to the list until its timer expires. The thread's state has be changed to `PINTHR_SLEEP` when it is in the sleeping list.

Task 2: Basic Priority Scheduling

In Pintos, each thread has a priority value ranging from 0 (PRTY_MIN) to 63 (PRTY_MAX). However, the current scheduler does not respect these priority values. You must modify the scheduler so that higher-priority threads always run before lower-priority threads.

You must also modify the two Pintos synchronization primitives (lock and semaphore), so that these shared resources prefer higher-priority threads over lower-priority threads.

A thread may set its own priority by calling `thread_set_priority(int new_priority)` and get its own priority by calling `thread_get_priority()`. If a thread no longer has the highest priority (e.g. it called `thread_set_priority()` with a low value, it released a lock, or incremented semaphore value), it must immediately yield the CPU to the highest-priority thread.

Testing

To completely test your implementation, invoke `make check` from the project 'build' directory. This will build and run each test and print a "pass" or "fail" message for each one. When a test fails, `make check` also prints some details of the reason for failure. After running all the tests, `make check` also prints a summary of the test results.

You can also run individual tests one at a time. A given test `t` writes its output to `'t.output'`, then a script scores the output as "pass" or "fail" and writes the verdict to `'t.result'`. To run and grade a single test, make the `'.result'` file explicitly from the `'build'` directory, e.g.

`make tests/threads/alarm-multiple.result`.

If make says that the test result is up-to-date, but you want to re-run it anyway, either run `make clean` or delete the '.output' file by hand.

You also need to run another two test cases by using the producer/consumer program (in homework 2) with your priority scheduler. In the first case, the producer is assigned a higher priority than the consumer when they are created. In the second case, just reverse their priorities. Take a screenshot of each.

Turning it in

Each group (two students) has to submit your program electronically. Before you submit, you need to use `make clean` to clean all of the object/executable files. Then, on grail, change the directory to the parent directory of the `pintos_csu` dir and use the following command to submit the whole `pintos_csu` dir:

```
turnin -c cis345s -p proj2 pintos_csu
```

Each group also needs to submit a report (i.e. `p2_report.pdf` under the `pintos_csu` dir) which includes the description of your design and implementation, the state transition diagram, experiences in debugging/testing, the project status (working or not), how the team works together, screenshots/explanation of the producer/consumer testing results, etc. The cover page should contain your picture(s) , name(s) and the login id you used to turnin the project. Start on time and good luck. If you have any questions, send e-mail to `sang@cis.csuohio.edu`.