Università della Svizzera italiana
Facoltà di scienze informatiche

Operating Systems **Spring - 2024**

Prof. Fernando Pedone
TAs: Eliã Batista, Lorenzo Martignetti, Nenad Milosevic

Project 3 - Schedulers PintOS

In class, we discussed how to build a simple priority scheduler in pintOS, as well as an advanced scheduler, in which, CPU-hungry threads lose priority to avoid starvation. In the second case, the user sets a $nice$ value. Priority is then automatically adjusted by the OS based on the thread's niceness.

Your task is to implement both schedulers, and you have to do it in such a way that one could choose in the PintOS initialization which scheduler will be used.

Priority Scheduler

- A $priority$ is an integer ranging from 0 to 63. In the current implementation of PintOS we can already find the variables: $PRI\_MIN = 0$; $PRI\_DEFAULT = 31$; $PRI\_MAX = 63$.
- The next thread to run is that with the highest priority. There is already a field $priority$ in the thread's struct in the ***"threads/threads.h"*** file, and functions to set and get this value in the ***"threads/threads.c"*** file.
- If multiple threads have the same highest priority, the scheduler must alternate them in round-robin.

Functions to be modified

$next\_thread\_to\_run()$	Make it return the thread with highest priority.
$thread\_set\_priority()$	Make the current thread yield if the new priority is not the highest.
$thread\_create()$	Make the current thread yield when a higher priority thread is created.

Hints

- You will only need to modify the ***"threads/threads.c"*** file for the priority scheduler.
- The functions above represent the main locations where you could implement your solution. However, depending on your implementation, you could modify other existing functions instead. You may also need to create other auxiliary functions not mentioned above.

Advanced Scheduler

- Each thread has a $nice$ value (ranging from -20 to 20, with 0 as default).
- Each thread has a $recent\_cpu$ value (indicating how much cpu the thread has used recently).
- Global variable $load\_avg$ stores the system load (in number of ready+running threads per second in the last minute)

Functions to be implemented

$thread\_set\_nice(int)$	Update the thread $nice$, recalculate its $priority$ based on that and yield if the new priority is not the highest anymore.
$thread\_get\_nice()$	Return the thread's $nice$ value.
$thread\_get\_load\_avg()$	Return $load\_avg * 100$.
$thread\_get\_recent\_cpu()$	Return current thread's $recent\_cpu * 100$.

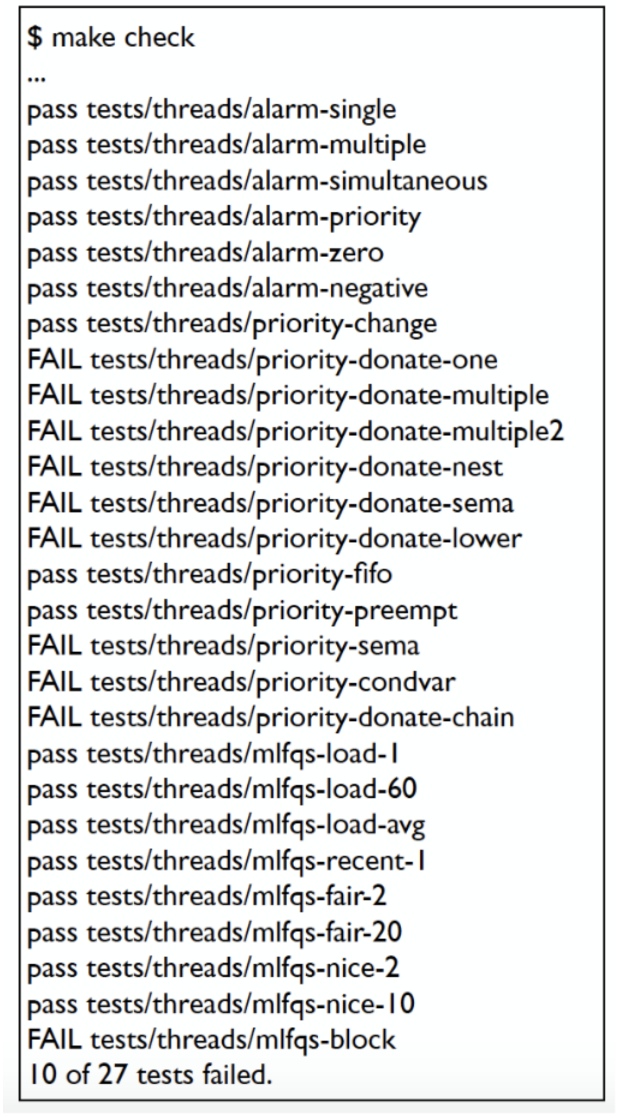
*The current thread must yield if a thread with higher $priority$ (based on its $nice$ value) is created.

Required variables

$nice$ type: int **(per thread)**	Initial value is set by the user. User can change with $thread\_set\_nice()$.
$load\_avg$ type: $fixed - point\ real$ **(global)**	Initial value is 0. Update every second: $load\_avg = (59/60) * load\_avg + (1/60) * ready\_or\_running\_threads$
$recent\_cpu$ type: $fixed - point\ real$ **(per thread)**	Initial value is 0. On every tick increase $recent\_cpu$ by 1 for the current thread. Update every second: $recent\_cpu = (2 * load\_avg)/(2 * load\_avg + 1) * recent\_cpu + nice$
$priority$ type: int **(per thread)**	Initial value is calculated based on the initial value of $nice$. Update every 4 ticks: $priority = PRI\_MAX - (recent\_cpu/4) - (nice * 2)$

Tests

A correct implementation results in a decrease in the number of failed tests to 10 out of 27, and an execution of `make check` should print the following result:



```
$ make check
...
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
pass tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
pass tests/threads/priority-change
FAIL tests/threads/priority-donate-one
FAIL tests/threads/priority-donate-multiple
FAIL tests/threads/priority-donate-multiple2
FAIL tests/threads/priority-donate-nest
FAIL tests/threads/priority-donate-sema
FAIL tests/threads/priority-donate-lower
pass tests/threads/priority-fifo
pass tests/threads/priority-preempt
FAIL tests/threads/priority-sema
FAIL tests/threads/priority-condvar
FAIL tests/threads/priority-donate-chain
pass tests/threads/mlfqs-load-1
pass tests/threads/mlfqs-load-60
pass tests/threads/mlfqs-load-avg
pass tests/threads/mlfqs-recent-1
pass tests/threads/mlfqs-fair-2
pass tests/threads/mlfqs-fair-20
pass tests/threads/mlfqs-nice-2
pass tests/threads/mlfqs-nice-10
FAIL tests/threads/mlfqs-block
10 of 27 tests failed.
```

Switch schedulers

In your implementation check the $bool$ variable $thread\_mlfqs$, you can find it in ***"threads/thread.c"*** file. It tells whether the option $-mlfqs$ was passed to the kernel. If true, use the advanced scheduler. If false, use the simple priority scheduler.

Fixed-point real

Variables $load\_avg$ and $recent\_cpu$ are real numbers. Usually, there is no float operations in the kernel. Fixed-point real variables are a solution implemented using integers. You can use the header file ***"fpr_arith.h"*** available on iCorsi.

Readings

- PintOS documentation
  - Chapter 2.2.3
  - Chapter 2.2.4
  - Chapter 2.3
  - Appendix B