System Software Course Design Programming Test Instructions Document

February 23, 2022

1 Instructions To Stride Scheduling

In pintos, we read the FCFS algorithm which shipped with pintos and implemented priority scheduling, MLFQ scheduling by hand. However, neither of these two algorithms can control the ratio of time running between processes. In the following, you are asked to implement the stride scheduling algorithm in the basic state of pintos.

1.1 Algorithm Steps

1. Set a current `stride` for each process, which indicates the 'length' that the process has been running. In addition, set its corresponding `pass` value (as far as the priority of the process is concerned), which indicates the accumulation value that the stride needs to perform after the process is scheduled.

2. *every time scheduling is needed*, selects the process with the smallest `stride` from among the processes in the current ready state and schedules it. For the process P that gets scheduled, the corresponding `pass` is added to its `stride`.

3. After one time slice, go back to the previous step and reschedule the process with the smallest `stride`.

It can be shown that if we make P.pass $= \frac{\text{BigStride}}{\text{P.priority}}$, where P.priority denotes the priority of the process (greater than 1) and BigStride denotes a predefined large constant, the time allocated to each process by this scheduling scheme will be proportional to its priority proportional to its priority. We omit the proof process here, and interested students can find the relevant information on the Internet.

1.2 Implementation Details

- `stride` scheduling requires *process priority* $p \geq 2$, so setting process priority $p \leq 1$ will result in an error.
- The initial `stride` of process is set to 0.

1.3 Notes

In engineering practice, we use fixed-size data types (e.g. `int32_t`) to store `stride`, and naturally, we will encounter overflow problems. Your algorithm should be able to correctly handle the comparison of `stride` after overflow under *implementation details listed above*, ensuring that the process with the largest `stride` when not overflowing can be selected each time.

2 Test Requirements

2.1 Task Description

You need to implement stride scheduling on *modified pintos we provided*. The required constants and variables are already defined (`BIG_STRIDE` and `stride`). You need to reuse the priority variables of pintos itself, used to implement priority scheduling (`struct thread.priority`) and ensure that `thread_set_priority` and `thread_get_priority` work properly.

You should not modify the definition of `BIG_STRIDE` in `threads.h`.

2.2 Tests Description

There are 4 tests in the test, and you can run `make check` in the `threads` folder to run those tests. *Tests shipped with original pintos has been removed.* Two of the tests are *hidden*. We will add the hidden tests during grading, for now, running the test will get a fixed 'test failed'.

Table 1: Test content description

Test name	Test detail	Hidden
stride-one	Dose your code work properly with one process	false
stride-two	Does your code work properly with two processes and whether the runtime is proportional to the priority	false
stride-multiple	Does your code work properly with multiple processes and whether the runtime is proportional to the priority	true
stride-overflow	Does your code work properly with multiple processes, considering overflow, and whether the runtime is proportional to the priority	true