CSE 421 Recitation

Project 1 Phase 1: Priority Scheduler

Presented by Montana Lee

University at Buffalo

Agenda

1. Phase 1 objectives
2. Understanding the priority scheduler
3. What we need to use to implement the priority scheduler
4. Lists and elems in Pintos
5. Phase 1 files to work in and tests to be passed
6. Ways to succeed

Phase 1 Objectives

- Become familiar and more comfortable with working with/improving upon the Pintos code.
- Understand how the illusion that ‘computers can do multiple things at once’ is created by a sequence of thread state transitions and context switches (handled by the scheduler).
- Understand how a simple priority scheduler works and how it can be used to improve upon a traditional round-robin style of scheduling.

Understanding the Priority Scheduler

Pintos Scheduling

- Pintos originally uses a simple round-robin scheduler
- Why do we want to change this?
  - This means that all threads that are waiting to run are scheduled one after the other, in turn
  - Although seems fair, remember that there are threads that are more important than others
  - Thus, we should run more important ones more frequently than less important ones
  - Round-robin can lead to starvation of critical threads as they wait for their turn
- Therefore, we want to implement a scheduler that can allow these important threads to run as they need to while also allowing less important threads to run as they are able to.

Understanding the Priority Scheduler

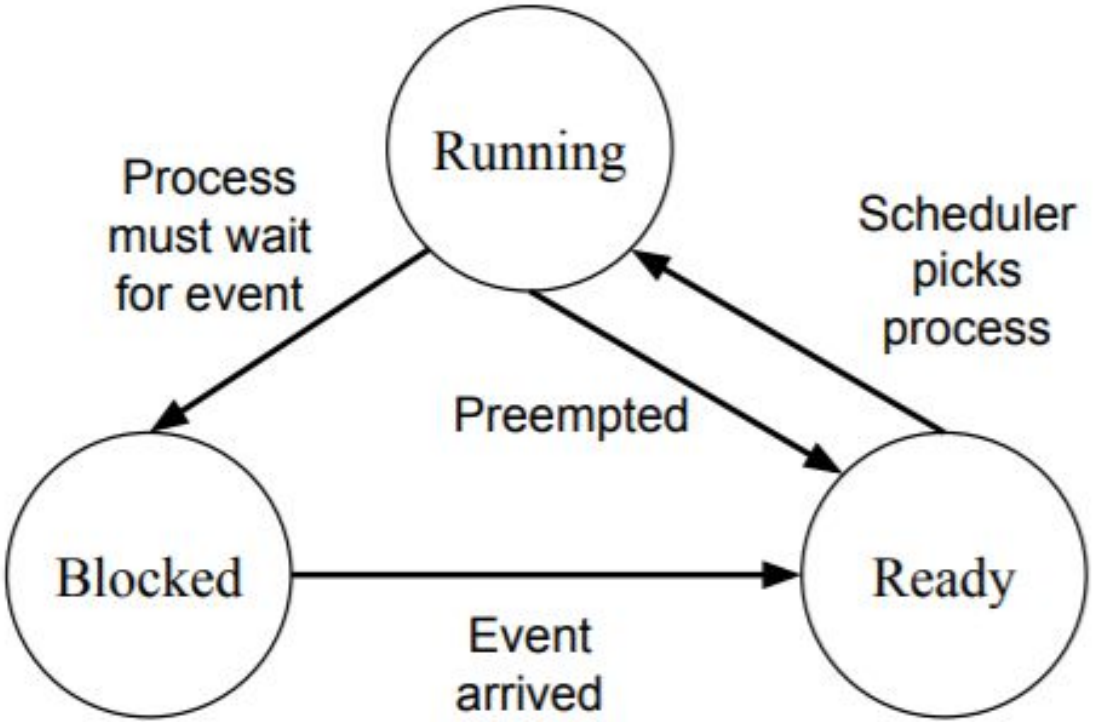
- You will create a priority scheduler in Pintos, which will run threads based on their assigned priority
- Priority in Pintos is defined as 0 being the least important, and 63 being the most important
- A process switch (allowing the next highest priority thread to take over the processor) will occur when thread_yield() is called, the thread block, or on return from interrupt
- If there are threads with equal priorities, then round-robin will be used amongst those threads to choose among them

Simple Priority Scheduler Queue

Single Priority Queue

Implementing the Priority Scheduler

- You will want to use the list that keeps track of threads that are waiting to run (the ready_list)
- When a higher priority thread than the current running one is added to the list, the current thread must yield to the higher priority thread
- If a thread has finished running, then the next thread to run must be selected through having the highest priority in the ready_list



Lists and Elems in Pintos

- Lists and list functions are already set up in Pintos, look at *src/lib/kernel/list.c*
- Notice that these lists do not use the ‘thread’ type- they use struct list_elem *
- These elems serve as a sort of ‘placeholder’ for the thing that they are connected to. The elem is tied to the thread, and is placed in the list, but the thread itself is not placed in the list
- You will want to include another list_elem struct in the thread struct if you create other lists, as each elem can only be in a single list at a time
- When you manipulate the ready_list, you manipulate the elems, not threads

Step-By-Step

- Follow the Pintos code around to get familiar with threads
- Understand the ready_list and how threads are inserted into it
- Figure out how you want to manipulate the ready_list itself or use another list of your own creation in order to help schedule appropriate threads to run
- Ensure that, when a thread yields or otherwise stops running, the next scheduled thread is the one in the ready list with the highest priority
- Run the tests

Files to Work In and Tests

- Much of your work this phase will take place in *src/threads/thread.c*
- The three tests needed to pass to get full credit for this phase are:
    - *src/tests/threads/priority-fifo*
    - *src/tests/threads/priority-change*
    - *src/tests/threads/priority-preempt*
- This phase is *tiny*. Do not use this to gauge the difficulty/length of the rest of the project.

Ways to Succeed in the Project and Course

- Don’t be afraid of the code! Explore everything and read through important files
- Read the documentation!
- Visit office hours and ask questions *early*
- Start *early*. The projects are very time-consuming and require steady, metered work
- Commit often to Github in case you need to revert back
- Communicate with your team member
- Get in contact with staff ASAP if problems arise
- *Don’t cheat*!

Good luck!