Lab 3: Batch Scheduling in Pintos

Group 43

Shivneshwar Velayutham and Madhumitha Venkatesan

Data structures

The following are the data structure changes

1. **occupied_slots** stores the number of slots being used.
2. **current_direction** is the direction of task/s being executed at the moment.
3. **waiters** contains the number of tasks waiting based on direction and priority.
4. **the_lock** is the lock used for synchronization.
5. **condition_var** is the condition variable for each direction and each priority of the tasks.

Algorithms

The algorithm for each of the below methods is explained underneath.

getSlot(task_t task)

When a task is trying get a slot, it checks if the number of **occupied_slots** is equal to the number of slots and waits if that's the case. It also waits when there are other tasks being executed on the other direction. But even if there are slots available and the tasks running are in the same direction, the task will wait if there are high priority tasks in the opposite direction that are waiting.

Otherwise, it executes. If there are no slots being used, then it tries to wake up the other tasks in the same direction based on priority using broadcast on the appropriate condition variable.

leaveSlot(task_t task)

When a task is exiting, it first removes itself from the slots. It wakes up another task in the same direction (based on priority) using signal of the appropriate condition variable if there are other tasks being executed in the same direction. If there are high priority tasks waiting in the opposite direction then the normal priority tasks will not be woken up until the high priority tasks complete. Once the tasks currently in the slots complete executing, then the direction will switch for the high priority tasks to begin and finish.

When 0 slots are being used, then it broadcasts the condition variable to wake up the tasks. Below shows which takes higher priority in being woken up:

1. same direction, high priority
2. opposite direction, high priority
3. same direction, normal priority
4. opposite direction, normal priority

Synchronization

Synchronization is achieved using locks and condition variables. Locks help with mutual exclusion when executing the critical section. We have a condition variable for each type of task (ie. for each direction and priority) that tells the thread if it can execute or not and using this logic we are able to make sure the right decisions are made.

Rationale

We went with condition variables so that threads need not be doing a spinlock and go to sleep instead.