Lab 3 - Recitation

18-648

Lab 2 Feedback

- Cleaning up properly - look for `do_exit` instead of overriding `exit_group`
- When dealing with existing data structures, use APIs that are already present within the kernel
  - Prevents duplication which can cause divergence later
  - Examples - Setting realtime priority, RCU locking
- Timers every 1 ms on each core
  - Error condition: Context switch in after timer fires and context switch out before timer fires

Lab 2 Feedback

- Child process of a reserved thread should not have a reservation or do anything related to your framework
  - Forking does not result in `task_struct` going through the usual initialization
  - Parent's `task_struct` is copied & thread specific data changed
- Synchronization
  - Holding a `spin_lock` without `irq_save` and getting interrupted, and having a locking in context switch or in any timer
  - Lifetime of pointers that you retrieve
    - Example: Pointer to a `task_struct` from reading the tasklist might not be valid when the task dies

Scheduler

- The kernel system timer invokes `scheduler_tick()` on every timer interrupt
  - The kernel then communicates to the scheduler that the task that is currently running needs to be replaced by another task.
- Thread that is suspended must not be awoken until its period boundary. This means, that it should not be marked runnable by the kernel until then.
  - Look into `task_struct->state`
  - What happens if a task is not marked as `TASK_RUNNING`
- Note that when the computation and sleep can be not in sync with the period boundary
- `wake_up_process()` will restore a suspended thread back to the running state, and place it in the run queue of the core that it is on.
  - Lost wakeup problem - new reserve on existing reserved thread

Admission Testing

- When assigning priorities while determining schedulability, make sure to use the RMS priority that you would assign to the task by doing regular RMS.
- Do not use the priority that you assigned it in Lab 2, but you can leave it at that realtime priority

Hrtimers

- You might find the need to make it so that hrtimers work similarly to the interval timers under the `ITIMER_VIRTUAL` configuration.
  - You might want to revisit the `hrtimer` API -- `hrtimer_get_remaining`
- You can pin an `hrtimer` down to the core by starting it on that core.
- Be careful with timer callbacks
  - Which thread context is it running on?

General Tips

- Lost Wake Up: http://www.linuxjournal.com/article/8144 -- This is an interesting article about a classic race condition in Operating Systems!
- Other useful links:
  - http://www.linuxjournal.com/article/8144