COS318 Project 3
Preemptive Scheduling

Project Overview

- 3 targets:
  - Preemptive scheduling: respond to the timer interrupt
  - Blocking sleep: take the sleeping tasks out of the ready queue, put them back after a while
  - Synchronization primitives: implement condition variables, semaphores and barriers
- Your code is based on our provided kernel
- 5 test sets are provided

Preemptive Scheduling

- Tasks are preempted through timer interrupt, which is handled by irq0
- Switch context as what you have done in project 2
- Turn on/off the interrupt properly
  - Safety: prevent race conditions in kernel
  - Liveness: interrupts should be mostly on
- Test sets you can use: test_regs, test_preempt

Blocking sleep

- Maintain your own “wait queue”
- Use “num_ticks” to do the timing
- Wake up the sleeping task as soon as possible
- Carefully handle the case that all tasks are sleeping
- Test set you can use: test_blocksleep

Synchronization Primitives

- The names of all the primitives are provided
- An implementation of locks is also available
- You need to design the data structures and implement the behaviors
- Turn on/off the interrupt properly
- Be careful with the fairness issue
- Test sets you can use: test_barrier, tsk_test (this tests everything)

Extra Credit 1

- Prioritized Task Scheduling
  - Lottery Scheduling is OK
  - But any other algorithms are welcome, as long as you describe clearly
  - Modify the priorities in “test_preempt” set to test your scheduling algorithm
  - Add “#define EC_PRIORITIES” in “common.h”

Extra Credit 2

- Automatic Deadlock Detection
  - Look for cycles in a lock graph
    - simple theory, difficult implementation
  - Only relevant to locks
    - (a restricted use of locks at that – locks released by same process that acquired)
  - Detect sensitively and correctly
  - Recover from the deadlock properly
  - Design your own test cases
  - Add “#define EC_DEADLOCK” to “common.h”

Questions you might want to consider before coding

- When do you need to enter the critical region?
- What would you do if no tasks in the ready queue but some tasks in the sleep queue?
- How to wakeup “ready to be woke up” tasks?
- How to guarantee the fairness of synchronization primitives?

Files you might need to modify (but not limited to)

- common.h : add #define if you touch the extra credits
- entry.S: preemptive context switch
- kernel.c: some more initialization
- queue.c/h: for extra credit
- scheduler.c/h: preemptive scheduling
- sync.c/h: synchronization primitives

Design Review

- irq0_entry: response to the timer interrupt
- Blocking sleep: queue, sleep and wakeup
- Synchronization primitives
  - Data structure
  - Workflow
  - Fairness

Timeline

- Design review = Monday with Yida (not me!)
- Project due: 11:59pm 4 November 2012
  - Codes with necessary comments
  - Readme
    - Less than 500 words
    - Specify what you have done, especially if you touch the extra credits
- Q/A sessions
  - 7:30-8:30pm 10/23