16 October: Project 3

- Scheduler
  - Lots of GeekOS integration
  - Your algorithm can be simple or complex
- Semaphores
  - Not much GeekOS integration—more like pipes
  - Some care is needed on P and V
- Both: You choose your design

Scheduling (1)

Thread A is running along. It's been on the CPU for a long time.

- Timer interrupt! The handler runs. (More on this, next slide.)
  - Each timer interrupt is one "tick"
  - `Timer_Interrupt_Handler` in timer.c
  - Don't worry about timer events
- kthread structure records the number of ticks it's had the cpu
- If numTicks >= quantum, set g_needReschedule

Scheduling (2)

The handler is actually called from lowlevel.asm, Handle_Interrupt

- Save context for g_currentThread
- Run the actual handler (e.g., Timer_Interrupt_Handler)
- Handler returns. If g_needReschedule:
  - Reset numTicks to zero
  - `Make_Runnable` on g_currentThread
    - puts it on the run queue
  - `Get_Next_Runnable` to choose the next thread
    - chooses thread, removes it from the run queue, returns it
    - lowlevel.asm does the assignment to g_currentThread
  - clear g_needReschedule
- It then calls signal handling code and resumes the new g_currentThread.

Scheduling (3)

Thread A was CPU bound. What about B, which just asked for a file from disk?

- B calls Wait(someWaitQueue)
  - adds B to whatever wait queue it indicated
  - calls Schedule()
- Schedule switches to some new thread
  - Get_Next_Runnable to choose a thread
  - Switch_To_Thread to start it

Switch_To_Thread is in lowlevel.asm, does much the same stuff as Handle_Interrupt.

Round Robin Scheduler

Key RR data structures and fields

- s_runQueue
- kthread.priority

s_allThreadList is not a scheduling structure; you should keep it as is

Key RR functions

- Find_Best

What do you need to do? (1)

2 parts: the scheduler machinery, and the syscall that sets policy.

Scheduler:

- Make a flag that indicates what scheduler is in use
- If you need a different data structure, define it
- Set up your new scheduler in Init_Scheduler
- Change Get_Next_Runnable (and perhaps Make_Runnable) so their behavior depends on the scheduler flag
- May add to the timer interrupt handler if you need to do anything when a thread finishes its quantum

What do you need to do? (2)

Set_Scheduling_Policy:

- Set the flag
- If you are using a different data structure, when the policy changes, you must make sure kthreads are on the appropriate structure

If you're not using s_runQueue, you might want to change your Sys_PS implementation so you can still see what's runnable. However, we will not test this.

Algorithm Choice

What scheduler will you implement?

- Multilevel feedback queue
- Shortest job first
- Any other thoughts?
- Is it worth implementing a fancier data structure, like a priority queue?

Semaphores

Much less GeekOS integration than the scheduler.

You'll define and manipulate your own data structure. You may need to declare initialization functions, etc. in sem.h.

Wait (P) with `Wait`

Signal (V) with `Wake_Up` or `Wake_Up_One`

`P` and `V` do need some care.