Code Reading Questions

Thread Questions

1. **What happens to a thread when it exits (i.e., calls thread_exit())? What about when it sleeps?**
   When a thread exits it gets detached from its process and all parts of the thread structure that are not actually needed to run are cleaned up. After that, a thread_switch is performed where the status of the thread is set to S_ZOMBIE and the thread is removed from its current threadlist (wchan) and it is assigned to the ZOMBIE threadlist. There, the thread remains until it is destroyed (incl. cleaning up from all left thread structure parts). If it goes to sleep it's status is changed to S_SLEEP and it is removed from its current wait channel and is added to the appropriate (sleep) wait channel.

2. **What function(s) handle(s) a context switch?**
   thread_switch() *(thread.c:563)* is the high level code for thread switching. This function eventually calls the machine code function switchframe_switch() in switch.S where the context of two threads is switched.

3. **What does it mean for a thread to be in each of the possible thread states?**
   The states are as follows:
   - **a.** S_RUN means the thread is currently running,
   - **b.** S_READY means the thread is ready to run,
   - **c.** S_SLEEP means the thread is sleeping, and
   - **d.** S_ZOMBIE means the thread has been exited and is waiting to be destroyed.

4. **What does it mean to turn interrupts off? How is this accomplished? Why is it important to turn off interrupts in the thread subsystem code?**
   You turn off interrupts by calling splhigh(), which sets the thread's priority level to the highest possible value so that it can't be interrupted. Turning off interrupts in thread subsystem code prevents untimely interrupts, such as getting interrupted mid thread switch.

5. **What happens when a thread wakes up another thread? How does a sleeping thread get to run again?**
   the first thread calls wchan_wakeone, which attempts to pull a thread from the head of the wchan's threadlist, and then if the returned thread is not null, calls thread_make_runnable on it, which adds the awoken thread to the cpu's runqueue.

Scheduler Questions

1. **What function(s) choose(s) the next thread to run?**
   thread_switch() (thread.c line 563), schedule() (thread.c line 849 (depends on the implementation of thread_join))

2. **How does it (do they) pick the next thread?**
   thread_switch() pulls the next thread from the head of curcpu's c_runqueue. If there is no thread in the curcpu's c_runqueue thread_switch() idles the processor. The processor sleeps/sits around until it things something interesting may have happened (e.g. interrupt). Before we idle the cpu we have to release the lock of the c_runqueue. And after idling we have to acquire the lock again. This is necessary to make sure that threads can be added to the threadlist (c_runqueue). After getting a new thread to switch with,

6. **What role does the hardware timer play in scheduling? What hardware independent function is called on a timer interrupt?**
   The hardware occasionally calls hardclock(), which calls thread_consider_migration() and schedule().
   thread_consider_migration() decides whether to move threads between cpus.
   schedule() can rearrange threads (wrt. priority) on that cpu that are currently ready.

Synchronization Questions

1. **Describe how wchan_sleep() and wchan_wakeone() are used to implement semaphores.**
   When P is called, if the semaphore's value is still not greater than 0, then it calls wchan_sleep, and is later awoken by wchan_wakeone when someone calls V. Since the semaphore's value must now be greater than 0, it wakes just one thread to decrement the counter it just incremented. If there aren't any waiting threads, the next time P is called, it won't need to go to sleep.

2. **Why does the lock API in OS/161 provide lock_do_i_hold(), but not lock_get_holder()?**
   The lock api doesn't have lock_get_holder() since there’s no guarantee the holder you get back from lock_get_holder() is correct unless you are the one holding the lock. If you don’t hold the lock, it could have switched holders several times between when lock_get_holder() is called and when you actually use that value.