Computer Science & Engineering
University Of Dhaka

Code Reading Assignment 1

Submitted By:
Md.Muhaimin Shah Pahalovi & Roll: 47

Submitted To:
Mr.Mahmood Jasim

April 18, 2016

1. What happens to a thread when it exits (i.e., calls thread_exit())? What about when it sleeps?

When a thread exits, its address space gets destroyed and it removes itself from the global thread count. It doesn't actually get destroyed itself until thread_destroy is called from exorcise.

When a thread sleeps, it assigns itself a "sleep address" and stops being processed, freeing the CPU to handle another process. When it wakes up, it must be awoken from the same sleep address.

2.What function(s) handle(s) a context switch?

thread_switch() performs a context switch using switchframe_switch().

3.How many thread states are there? What are they?

4 state. S_RUN , S_READY , S_SLEEP , S_ZOMBIE

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? It means that no other thread can replace the current thread through a context switch. This is done by setting the SPL to high using splhigh(). We turn interrupts off in thread subsystem code so that context switches can complete properly or threads can properly exit.

5.What happens when a thread wakes up another thread? How does a sleeping thread get to run again?

A thread wakes up another thread by calling threadlist_remhead() to grab the next thread on the waiting channel and makes it ready/runnable. Then that thread is added to the CPU's run queue and waits for a context switch.

6.What function is responsible for choosing the next thread to run?

scheduler()

7.How does that function pick the next thread?

If we implement that function it will follow our rule , or by default it it follow round-robin fashion.

8.What role does the hardware timer play in scheduling? What hardware independent function is called on a timer interrupt?

The hardware timer influences rtclock, a generic clock interface. hardclock is called from the timer interrupt a set number of times per second.

9.What is a wait channel? Describe how wchan_sleep() and wchan_wakeone() are used to implement semaphores.

wait channel is the place in the kernel where task is currently waiting.
To synchronize sleeping and waiting calls so that no thread will sleep forever.

10.Why does the lock API in OS/161 provide lock_do_i_hold(), but not lock_get_holder()?

Because locks have to be released by the same thread that acquires them

11.Here are code samples for two threads that use binary semaphores. Give a sequence of execution and context switches in which these two threads can deadlock.

sequence:
1.me() :mutex lock
2.you() :data lock
Now me is trying to lock data and go to sleep. You is trying to lock mutex and go to sleep. So a dead lock occurred.

12.Propose a change to one or both of them that makes deadlock impossible. What general principle do the original threads violate that causes them to deadlock?

```
semaphore *mutex, *data;

void me() {
   P(mutex);
   /* do something */

   P(data);
   /* do something else */

   V(data);

   /* clean up */
   V(mutex);
}

void you() {
   P(mutex);
   P(data);

   /* do something */

   V(data);
   V(mutex);
```

cause: circular dependency.

13.Here are two more threads. Can they deadlock? If so, give a concurrent execution in which they do and propose a change to one or both that makes them deadlock free. sequence:
1.laurel() gets mutex, file1 , file2, release mutex, file1.

2.hardy()gets file1 and trying to get file2. And get paused.
3.laurel() trying to get file1 and get paused.
Deadlock.

```
lock *file1, *file2, *mutex;

void laurel() {
   lock_acquire(mutex);
   /* do something */
   lock_acquire(file1);
      /* write to file 1 */

   lock_acquire(file2);
   /* write to file 2 */

   lock_release(file2);
   lock_release(file1);
   lock_release(mutex);

   /* do something */

   lock_acquire(file1);
   lock_acquire(file2);
   /* read from file 1 */
   /* write to file 2 */

   lock_release(file2);
   lock_release(file1);
}

void hardy() {
      /* do stuff */

   lock_acquire(file1);
   /* read from file 1 */

   lock_acquire(file2);
   /* write to file 2 */

   lock_release(file1);
   lock_release(file2);

   lock_acquire(mutex);
   /* do something */
   lock_acquire(file1);
   /* write to file 1 */
   lock_release(file1);
   lock_release(mutex);
}
```