COMP3500 – Frequently Asked Questions

Project 4 – Concurrent Computing: Cats and Mice

1. While trying project 4, we boot up os161, and go to the test menu, but there isn't a 1a option test for the cat and mouse.

   Answer: Please follow the following instructions to rebuild kernel for project 3.

   ```
   %cd ~/cs161/src
   %./configure
   %cd ~/cs161/src/kern/conf
   %./config ASST1

   % cd ../compile/ASST1
   % make depend
   % make
    % make install
   ```

   Now you can reboot your OS161 and you will find the following new menu.

   ```
   OS/161 kernel [? for menu]: ?

   OS/161 kernel menu
       [?o] Operations menu                [1c] Stoplight
       [?t] Tests menu                     [kh] Kernel heap stats
       [1a] Cat/mouse with semaphores      [q] Quit and shut down
       [1b] Cat/mouse with locks and CVs

   Operation took 0.061087400 seconds
   OS/161 kernel [? for menu]:
   ```

2. How to use semaphores in OS/161?

   Answer: Please follow the sample code below to crease and use semaphores to solve the cats-mice synchronization problem.

   ```
   struct semaphore *testsem; /* Declare a semaphore */

   testsem = sem_create("testsem", 2); /* Create and init */
   if (testsem == NULL) {
         panic("synchtest: sem_create failed\n");
   }

   P(testsem); /* Wait(testsem) */

   kprintf("ok\n");

   V(testsem); /* Signal(testsem) */
   ```

3. How to implement “play around” and “eat food” in the cats-mice algorithms?

Answer:

```
#define MAXTIME 3 /* You may change 3 to any number */

clocksleep(random() % MAXTIME);
```

4. I am having issues with implementing the catlock portion of the assignment. I tried using a setup similar to the semaphore implementation but this hasn't worked out.

   Answer: catlock.c is use the lock and condition variable mechanisms to solve the cats-mice problem. You must implement the lock and condition variable mechanisms first before working on catlock.c

5. Does it matter where we put the play around portion of cat and mouse? Is it supposed to be immediately after they finish eating or just at the end to make it wait before it tries to eat again?

   Answer: Use the following state to make cats or mice to wait before eat again.

   ```
   clocksleep(random() % MAXTIME);
   ```

6. When we were working on our project today, I ran into an error when running my implementations for the cat and mouse problem. The error was "Fatal Exception 2 (TLB miss on load) in kernel mode". Do you have any advice on what could cause this error?

   Answer: "TLB miss bug" can be detected using gdb. In the attached file (GDB and OS161-TLB Miss.html), you will find instructions how to use gdb to keep track of "TLB miss" related bugs. You may search "TLB miss" in the attached html file for important information.

7. In the catsem.c and catlock.c files, are we allowed to modify the parameters of functions like catsem() and mousesem(), or are we supposed to just modify the code inside and use the current parameters?

   Answer: There is no need to modify the parameters of the functions like `catsem()` and `mousesem()`. If you choose to modify the prototypes (e.g., parameter lists) of any function, you have to modify the callers of the function accordingly.

8. **Early exit to menu/debugging issue.**
   We were debugging catsem, and after one cat printed its "in the kitchen" message the "operation took x seconds" and the main menu came back up in the run terminal. However, we were able to continue to step through catsem in the GDB terminal. Is this behavior normal for GDB and os161 when a bug is encountered?

   Answer: To avoid "early exit to menu", your test driver code (see, for example, `catmouselock()`) must wait until all the cat and mouse threads are done. The sample code is given below:

   ```
   lock_acquire(mutex);
   while (num_cats_done < 6 || num_mice_done < 2)
       cv_wait(donecv, mutex);
   lock_release(mutex);
   ```

9. **"`./sys161 kernel`" not working all of a sudden**

Running the kernel has suddenly began spitting out the following lines and then quitting...

```
sys161: System/161 release 1.14, compiled Aug 31 2016 14:16:22
sys161: 246000    cycles (119995k, 0u, 126005i)
sys161: 0 irqs 0 exns 0r/0w disk 0r/0w console 0r/0w/1m emufs 0r/0w
net
sys161: Elapsed real time: 0.010549 seconds (23.3197 mhz)
sys161: Elapsed virtual time: 0.009840000 seconds (25 mhz)
```

Why did we encounter this problem?

Answer: If you don't implement `lock` properly, you are likely to encounter this problem. Before implementing the `lock` mechanism, please try to implement the `cats_and_mice` concurrent program using the existing semaphore mechanism. If you encounter any problem with the `cats_and_mice` program, you may need to address the synchronization issues. Next, your team should start the implementation of the lock mechanism.

10. **Part 3 Testing**. In this step it discusses testing built-in threads such as tt1 and tt2 in the GDB debugger. Is part 3 only information for a later part of the project or is are we actually supposed to be running and testing these threads in step 3? (In short: Does part 3 have any actual steps to do or should we read it and move on to part 4?) And if we do have actual steps in part 3 how do we go about running and checking the threads.

Answer: Part 3 shows you an easy way to test your concurrent threads. You will follow the instructions to run and check your threads (e.g., cat threads and mouse threads).

11. **Meaning of "when cats aren't eating, they will not see mice eating."** In the project specifications it is stated that "when cats aren't eating, they will not see mice eating." (1) Does this mean that a cat can walk in at any time on a mouse that is eating? Or (2) does it just mean that a cat will not eat a mouse when that cat is not currently eating?

Answer: (1) "Does this mean that a cat can walk in at any time on a mouse that is eating?"
In this case, the cat can't walk into the kitchen; the cat has to wait outside the kitchen.

(2) "Does it just mean that a cat will not eat a mouse when that cat is not currently eating?"
Yes, you are right. The cat will not eat a mouse who is enjoying a dish, if the cat is waiting outside.

12. **thread_sleep() and thread_wakeup().** Can you explain the `thread_sleep()` and `thread_wakeup()` functions to us?

Answer: In order to understand the implementation of `thread_sleep()` and `thread_wakeup()`, you have to study the `thread/thread.c` source code file. The global queue of sleepers is declared on lines 28-29 (see below).

```
28 /* Table of sleeping threads. */
29 static struct array *sleepers;
```

This queue is initialized in the `thread_bootstrap()` function as follows:

```
/*
 * Thread initialization.
 */
struct thread *
thread_bootstrap(void)
{
        struct thread *me;

        /* Create the data structures we need. */
        sleepers = array_create();
        if (sleepers==NULL) {
                panic("Cannot create sleepers array\n");
        }
```

The `thread_sleep()` function (see line 496) calls the `mi_switch()` function (see line 337) to place the current thread (i.e., cur_thread) into the sleepers list (i.e., the list of sleepers). `mi_switch()` in turn invokes the `array_add(sleepers, cur)` (see line 383) to insert `cur_thread` into `sleepers`.

The `thread_wakeup()` function (see line 511) wakes up one or more threads who are sleeping on `"sleep address" ADDR`. If you only want to wake up a single sleeper, you will have to modify this function by adding a `break` statement at the end of the if statement. For example,

```
If (t->t_sleepaddr == addr) {
    …
    assert(result == 0);
    break;
}
```

13. **Initialization.** Do we need to initialize values for

```
turn_type
cats_in_this_turn
cats_eat_count
dish1_busy
dish2_busy
mydish?
```

If we are supposed to initialize these should they be global variables to local to catlock, mouselock, etc?

Answer: All the global variables must be initialized by the coordinator function (i.e., the parent thread of cats/mice threads).

14. **OS161 Panicking on startup after implementing lock.** I just finished implementing the lock, and everything compiles fine. However, after it starts, os161 fails with this error:

```
panic: Assertion failed: SAME_STACK(curkstack-1,
(vaddr_t)tf), at ../../arch/mips/mips/trap.c:220
(mips_trap)
```

How to solve this problem?

Answer: There's probably a bug in your lock implementation. For example, if you have a print statement at the top of one of the lock methods, you may have the above error.

15. **Waking a single thread.** I have completed my synch.c implementation and it seems to work fine when I test it. However, I implemented cv_signal() and cv_broadcast() with the same code which leads me to believe that my solution is incorrect. My problem is that I could not find any way of waking a single thread in thread.c. thread.c only provides thread_wakeup() which will wake all threads sleeping on the same flag. Is there something I am missing?

Answer: You'll need to make a new function in thread.c that only wakes up one thread. It is very similar to thread_wakeup() since you only need to add one line of code to the body of the for loop. Be sure to put its prototype in thread.h. The lecture notes “08c1-Project 3-7 Thread Sleep and Wakeup.pptx” shows you how to implement a new thread_wakeup() function that only wakes up a single thread. Please download the lecture notes on Canvas.