COMP3500 – Frequently Asked Questions
Project 3 – Synchronization Mechanisms

REVISED: Oct. 1, 2020
VERSION 3.0

1. **Project 3 Requirements.** Do we need to implement new functions in addition to the ten required functions in `synch.c` (e.g., lock_release()lock_acquire())? (Contributed by Jacob Justice, Fall'18)

   Answer: You only need to fill out the ten functions in `synch.c`. Please be mindful that you must implement a total of five functions to support the lock mechanism; you should implement the other five functions for the CV mechanism. No new function prototypes should be created for project 3, implying that you don't have to update the header file.

2. **Project 3 Options.** Can we still SSH into a remote tux machine for project 3? (Fall'18)

   Answer: Yes, you can choose to carry out project 3 on any remote Tux machine through SSH.

3. **Test Lock/CV Implementations**. After we finish all the method and code, how can we know if we wrote them all right? How can we test them?

   Answer: Please follow the following instructions to rebuild kernel for project 3.
   Please following the commands below to test your implementation:

   ```
   cd ~/cs161/root
   ./sys161 kernel
   ```

   On os161's commandline:

   ```
   type: ?
   type: ?t
   type: sy1 /* test semaphore */
   type: sy2 /* test lock implementation */
   type: sy3 /* test CV implementation */
   ```

4. 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) */
```

5. When we were working on our project today, my group and 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.

6. **“`./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? (Updated in Fall’19)

   Answer: If you don’t implement `lock` or `cv` properly, you will be likely to encounter this problem. Before implementing the `cv` mechanism, please try to implement the basic `lock` functions first. Next, your should start the implementation of the `cv` mechanism.

7. **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).

8. **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;
}
```

9. **What parameter should we pass to `thread_sleep()`?** For function `thread_sleep()` in `thread.c` it shows that the parameter is an address. In the function "P" in the semaphore struct it calls thread_sleep with the parameter "sem" which is a semaphore. This seems to imply that in the lock, we should call thread_sleep(lock), but Doesn't thread_sleep need to put the current thread to sleep? Shouldn't we call `thread_sleep(curthread)`?

   Answer: No, you can't call `thread_sleep(curthread)`. When you call thread_sleep(), you don't pass "`curthread`" as an argument. This is because (1) `curthread` is a global variable, meaning that any function can access "`curthread`" without passing it as an argument, (2) you only pass "semaphore pointer" or "cv pointer" as an argument to `thread_sleep()`.

10. **Lock holder in CV functions.** For all the CV functions, the lock holder has to be `curthread` so we check this by using `assert` in the 3 cv functions?

    Answer: You don't use assert in this case. This is because in your `cv_signal` and `cv_wait` functions, your functions should call `lock_do_i_hold()` first. If the function returns "`false`" (meaning that "`curthread`" isn't a lock holder), then issue "`panic`". Note that "`curthread`" must be the lock holder in `cv_signal` and `cv_wait` functions.

11. **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).

12. **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.

13. **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.

14. **Enabling Debug Messages.** In the last project we were told to sprinkle debug messages throughout the code. It says that "If the debug DB_VM flag is set, the debug message will be printed on the console" but does not mention how to set this flag. (Contributed by Matthew Cather, Fall'18)

Answer: If you want to enable debug messages related to the VM module, you can set the flag as follows:

```
u_int32_t dbflags = DB_VM;
```

In case you must enable debug messages related to two modules (e.g., VM and System call), you may configure the flag using the following statement:

```
u_int32_t dbflags = DB_VM | DB_SYSCALL;
```

15. **cv_signal vs. cv_broadcast.** If I understand the synch.h correctly, cv_broadcast is a function that calls cv_signal multiple times until all threads sleeping on cv are woken up. I remember you mentioning in class there was no need to add extra variables to the cv struct, but if I understand

everything correctly, wouldn't we want a "count" variable on the struct? (Contributed by Jacob Justice, Fall'18)

Answer: Your proposed solution (e.g., call cv_signal multiple times) is feasible and highly appreciated. My proposal is to make cv_broadcast independent of cv_signal. In my design, I created a wakeup_all() function that is an extension of the existing wakeup() function. I will cover my design in today's lecture. Importantly, I firmly believe the merit of your implementation lies in its simplicity. The challenge of your design is how to determine the number of times cv_signal() should be repeated invoked. If we take your approach, we have to introduce a counter in the cv structure. Let's discuss this intriguing design idea in our class.

16. **Implementing locks.** To implement the lock functions in project 3, I presumably need a lock that is declared and exists somewhere that is persistent over multiple processes. Where do I declare this lock for it to exist in such a way? (Fall'19)

    Answer: (1) Let's assume your question is about the implementation of the lock mechanism. Answer: You should NOT use a lock to implement the lock functions. Please read the following lecture notes in Canvas for details:

    14-Project 3-2 Locks and Condition Variables.pptx
    15-Project 3-3 Thread Sleep and Wakeup.pptx
    17-Project 3-4 Testing Lock and CV Implementation.pptx

    (1) If your question is related to the test driver, the answer is given below.
    Answer: You don't need to declare any lock variable to test your lock implementation, because the test driver has been implemented in os161. Please read section 3.1 in the project specification (long version) below:

    3.1 Built-in Thread Tests
    Important! When you booted OS/161 in project 2 (a.k.a., ASST0), you may have seen the options to run the thread tests. The thread test code makes use of the semaphore synchronization primitive. You should be able to trace the execution of one of these thread tests in GDB to see how the scheduler acts, how threads are created, and what exactly happens in a context switch. You should step through a call to mi_switch() and see exactly where the current thread changes.

    Thread test 1 (" tt1 " at the prompt or tt1 on the kernel command line) prints the numbers 0 through 7 each time each thread loops. Thread test 2 (" tt2 ") prints only when each thread starts and exits. The latter is intended to show that the scheduler doesn't cause any starvation (e.g., the threads should all start together, run for a while, and then end together).

17. **Implementing `cv_wait()`.** Is there supposed to be an if statement in `cv_wait`()? (Fall'19)

    Answer: No, there is no if statement in the implementation of cv_wait(). The boolean experssion checking a condition is placed in the 'while' statement outside t
    he cv_waitl() function.

18. **git diff.** I committed the updated os161 source code multiple times. How can I use 'git diff' to compare and show all the changes made by me in this project? (Contributed by Brandon Molyneaux, Fall'19)

    Answer: In project 3, we made an initial commit in step 2 and then the final commit in step 6. If you type `$git log`, you will see the commits made by you in the past. See Brandon's example below:

```
commit 8de55f19754756d96a0442e81797b51b28d44fd9
Author: Brandon Molyneaux <bdm0041@tux058.eng.auburn.edu>
Date:   Thu Sep 26 22:52:29 2019 -0500

     ASST1a final commit

commit 76474051e67306f22de1ce7ca856335a8a922fac
Author: Brandon Molyneaux <bdm0041@tux051.eng.auburn.edu>
Date:   Sun Sep 15 18:45:35 2019 -0500

    ASST1a initial commit
```

    From here, you follow the exact same steps you would as mentioned in the assignment but you would reference the commit:

```
$git
diff 8de55f19754756d96a0442e81797b51b28d44fd9..76474051e67306f22de1ce7c
a856335a8a922fac > ../project3/asst1a.diff
```

    Likewise with branches, you would do the name of the branches:

```
$git diff branch1..branch2 > ../project3/asst1a.diff
```

19. **git diff commit error.** I did not realize that my initial commit at the begging of the project failed and I completed the project and am trying to run the git diff command now and it won't work. I used git log to check and that is when I noticed that my initial commit was not there, but my final commit was. Is there any way I can fix this? (Fall'20)

    Answer: If you can't create a git diff file for any reason, you will have to submit the two source code files: (1) synch.c and (2) thread.c on Canvas. Please schedule a demonstrate section with the GTA after your submission.

20. **Testing Errors.** I went to cs161/root and typed in ./sys161 kernel and then ran sy2. The only error I get is

```
thread 11: Mismatch on testval3/testval1
Test failed
Lock test done.
```

    I'm really not sure how to read the error. I assume that the two variables don't match, but I'm not sure how to go about fixing it. (Fall'20. Contributed by David Joy)

    Answer: The answers to this question are found in the source file synchtest.c (located in src/kern/test/). This result comes from the print statement on line #114 (part of the fail function), which is caused by the condition on line 144 failing (evaluating to true, which it shouldn't). This is caused by a race condition, which is almost certainly a result of an incorrect implementation of the

lock mechanism, specifically the lock_acquire and lock_release functions.

21. **Testing Errors.** When I run sy3, I received the error cv_wait: must be busy waiting. I'm also not sure how to fix it or what it means. (Fall'20. Contributed by David Joy)

    Answer: This error message is telling you that your function is busy waiting instead of using the thread sleep and wakeup functions, as was specified. This could be a result of the faulty lock implementation identified in the lock test.

22. **Ramsize Problem.** So I'm using Lab's Tux machine, on sys161.conf file it shows the ramsize is 2097152, but on kernel it shows there are "1872k physical memory available". Do i have to increase the ramsize on sys161.conf file? (Fall'20. Contributed by David Joy)

    Answer: The purpose of increasing the RAM was to provide enough memory for the (many) threads used by the sy2 and sy3 tests. Unless your tests are failing from not being able to initialize threads, os161 has enough memory.