ECE 354: Part 4

Group: ECE.354.S11-G031
Members: Ben Ridder (brridder), Casey Banner (cccbanne), David Janssen (dajjanss)

This document is concerned with the high level design of our six testing processes. These processes are used by three test cases to check the validity of different parts of the operating system. The first test case is dedicated to testing the functionality of the delayed send primitive. Next, the memory management system is tested with regards to the blocked queue and allocation. Last, the third test case examines multiple situations where the operating system is expected to return an error. The final process is used as a test management system that tracks the success and failures of the prior test cases which inherently tests message passing.

Delayed Send Tests

This test case makes use of two processes: `test_delay_sender()` and `test_delay_receiver()`. The first process sends six delayed messages with unique delays to the receiving process. These messages are not sent in the order they are expected to be received in. Next, the receiver process runs and attempts to receive these messages in the correct order. If all the messages are received in the correct order a `TEST_SUCCESS` message is sent to the test management process, otherwise a `TEST_FAILURE` message is sent. If there is a problem with the message sending of the `delayed_send()` primitive the testing harness will hang.

Memory Management Tests

This test uses two processes, `test_memory_watchdog()` and `test_memory_allocator()`, to verify the functionality of blocked queues and memory allocation. First, the watchdog process allocates a block and then sends a message to the allocator process. Upon receiving this message the allocator requests as many memory block as are available. Once it is blocked the watchdog process will continue to run. The next step of the watchdog process is to release the memory block that it requested, and then releases processor. At this point the allocator process becomes unblocked and is now able to release all of its memory blocks. If no deallocations fail then `TEST_SUCCESS` is sent to the the test management system, else `TEST_FAILURE` is sent.

Error Checking Tests

Error checking tests deal with the edge cases that should return error values from the kernel primitives. Like all the other tests, this is pseudo-scheduled from the managing process using differing priority levels and blocking on a message. The first few checks are against the memory system. It checks that releasing memory that is in invalid locations: both over and under where the memory blocks are located. Next, it tries to deallocate the same memory block twice. For each primitive that takes a process id as a parameter, an id that is out of range is sent in to ensure that the ranges are checked by the primitive. The last error check performed is for the delayed send to ensure that a negative delay time is handled. If at any point in the test an `RTX_ERROR` is returned from the process, then `TEST_FAILURE` is sent to the test manager and the test is halted at this point. If all primitives return the expected value, then `TEST_SUCCESS` is sent to the test manager.

Test Management System

The three tests are ultimately controlled by the last of the six processes, which is run right after the first test is completed. Once each set of tests is completed, the manager process receives a message from the testing processes. The next test set is selected by lowering the finished processes' priorities and raising the priorities of the next test processes. The next test case is started by sending a message to the relevant process. Results of each test are recorded for

the final output of results. Once all tests are completed, the results are output in the required format. This management process should not be preempted by the other test processes due to it having a higher priority level then the others. Test processes are blocked waiting for a message from the test manager. This message block is passed back and forth from the test processes and the manager so it is ultimately released by the manager. Listing 1 provides an outline of this process.

Listing 1: Test Management System Pseudo Code

```
void test_management():
    while(1):
        receive message
        if message is not from the last test case:
            lower priority of previous test processes
            raise priority of next test processes
            send message to start of next test process
        if message is from last test case:
            release message block
        check message for test result:
            if successful:
                increment success counter
            if failure:
                increment failure counter
            output result

        if all tests are done:
            output total results
```