Project 4

Susan Paradiso

… [Project write-up omitted — HCL]

Test Cases:

Project4_SampleTests/testmailboxX.c

The assignment included a suite of tests, testmailboX.c, where X is 1-7. These tests pass and the output is included (output is testX.txt).

Note that these are basic tests. Among other issues, they are not self-checking, they do not check error returns for SendMsg()/RcvMsg(), they are self-timed (sleep) and they do not wait for children to finish. {This is not a complaint, it was very nice to have them}. Also note that the output frequently says things like "Child send failed" when in fact the child send was expected to fail (because the dest-pid was stopped or had exited, for example).

test/p4test.c

I created one test program that implements multiple tests. The files (in the test subdir) p4test.c/p4test.h implement the bulk of the test code. The file p4utils.c/p4utils.c implements some utility functions such as print/error routines. The user specifies a test number to execute on the command line.

Note that because of rand() calls, specific results are seed dependant.

Some p4tests use the main (only) process, some use to threads or children, and some have threads and children. Barrier synchronization and locking is used to coordinate threads. 'Start' messages are used to coordinate children. Wait/join is used to ensure all children/threads completed cleanly.

p4test is self-checking. It checks thread and child results also. If it prints TEST PASSED, then the test passed. p4test checks for correct return values on send/rcv/manage operations, it checks fork/pthread_create return values and wait/join results.

Because the tests are self-checking and generate many threads/children, the test code is non-trivial on initial inspection.

Kernel Messages:

My mailbox implementation does not print many messages to the kernel log. Error conditions print an error message to the log. The other log messages are: send/rcv down() interrupted by a signal; ManageMailbox-with-stop-set reports how many send/rcv waiters it has to up(); MailboxDestroy() reports how many messages it frees when the mailbox is being destroyed.

Usage:

make clean: cleans up previous builds

make: makes p4test

./p4test XX: executes test 'XX' in p4test.

Tests:

p4test 1,2 and 3: These tests verify that too long messages are handled correctly. Test 1 also enables slow_mode, test 2 disables slow_mode, and test 3 doesn't change slow_mode state, otherwise the three tests are identical. The (main) test sends good/too-long messages interleaved. It sends a negative length too-long message and a 129 byte too-long message. It also sends (full+10) messages, all too-long, to make sure no resources are reserved and not freed when a too long message is attempted. ManageMailbox is used to verify that too-long messages don't change the receiver's num_msgs.

> [Note: Ignore "slow mode." This is something that Susan inserted into her kernel to increase the probability of a race condition. HCL]

p4test 4: This verifies all (legal) message lengths including a zero-length message.

p4test 5: This verifies good messages. The parent and one child each send/receive 25 messages from each other.

p4test 6: This verifies that a read-block from a stopped and empty mailbox does not block. The mailbox is empty, and it is also stopped, so no new messages can be sent to it. RcvMsg()-blocking would hang forever waiting for a message if it were treated the same as a RcvMsg() to a running mailbox. Instead, the mailbox kernel code returns MAILBOX_EMPTY, despite the RcvMsg() hainvg block set.

> [Note: The return code on this test is a matter of interpretation. I believe that it should return MAILBOX_STOPPED if, in fact, the mailbox is stopped and empty. HCL]

p4test 7: This verifies full/empty mailboxes. Verify the mailbox is empty, fill it, and send a non-blocking message to the full mailbox (verifying full occurs only after max/32 messages). Read the messages, they should all be good. Then do a RcvMsg() from the empty mailbox, checking return value. Finally, fill the mailbox again then exit with a full/non-stopped mailbox.

p4test 8: This verifies full using a child. The child fill's the parent's mailbox and blocks on the next SendMsg(). The parent polls for a full mailbox using ManageMailbox(!stop), then the parent reads one message. This unblocks the child. The child then does a SendMsg()-no-block, and it should get MAILBOX_FULL back. The mailbox is exited with a full/non-stopped mailbox.

p4test 9: This verifies that we can send and receive a zero length message.

p4test 10: This verifies bad SendMsg() arguments: a bad message pointer. Good messages are sent/received after the bad messages to verify that everything still operates correctly after the error.

p4test 11: This verifies bad RcvMsg() arguments: bad message pointer, bad length pointer and bad destination-pid pointer. Good messages are sent/ received after the bad messages to verify that everything still operates correctly after the error.

p4test 12: This verifies bad ManageMailbox() arguments: bad num_msgs pointer. A good ManageMailbox is executed after the bad one to verify that everything still operates correctly after the error.

p4test 13: This verifies sending a message to an invalid (non-mailbox) pid correctly reports an error. Pid 1 is used as the invalid pid.

p4test 14: This verifies that threads share mailboxes with the parent. The thread sends a message to the parent and itself. The parent's and thread's num_msgs are verified to be two, then the messages are read/verified by the thread.

p4test 15: This verifies mailbox deletion, while (possibly) in-use. This starts a large number (100 each) of sending and receiving children. The receivers will read some random number of messages then exit (without stopping their mailbox). The senders will send until all receive mailboxes are gone (stopped or invalid). The test does allow/expect SendMsg() to get MAILBOX_STOPPED and MAILBOX_INVALID.

p4test 16: This is the same as test 15, except the receivers call ManageMailbox()-with-stop-set before exiting.

p4test 17: This verifies that threads and children can send to each other. The children and the tx-threads exit after they send some specified number of messages. When all children/tx-threads exit, the parent issues a ManageMailbox()-with-stop-set, then the receive (rx) threads end. This test generates large numbers of messages.

p4test 18: This verifies that multiple calls to ManageMailbox()-with-stop-set works cleanly. Multiple (100 each) children and threads are started. The children call SendMsg()/block, the threads call SendMsg()/non-block, and fill the parent's mailbox. Once the mailbox is full, the threads issue ManageMailbox()-with-stop-set. The first should stop the mailbox, the other should not hang. The children exit once the mailbox is stopped. The threads read the mailbox via RcvMsg(), with blocking randomly set or cleared. This RcvMsg() can return good/stopping/empty depending on when the ManageMailbox()-stop and the RcvMsg() execute with respect to each other.

p4test 19: This is an 'exer' (exerciser) to address a test the professor described in the assignment. This starts multiple children, grandchildren and threads and they send/receive to each other. I added this test on the last day the assignment was due, because it is not listed in the grading rubric (but it is listed in the assignment itself). I emailed professor Lauer and he said this specific test was not required, as long as other testing covered was thorough. This test passes with many threads/msgs. I added in children and hit code bugs in the test, and ran out of time to complete debugging it. I do feel that tests 1-18 above are very thorough and covers all cases.