## ***Test Strategy***

Testing the mailbox complex synchronization is difficult. The nine tests below are focused on testing messaging for multiple threads and multiple processes in a heavily loaded system. The best method for catching corner case synchronization issues is to have as many thread and processes interacting with each other as possible. Test7 provides a command line interface for the full-blown multi-threaded, multi-process testing.

## ***Shared Memory PID List***

To facilitate testing I found it convenient to create a shared memory list of active PIDs participating in the test. As processes are spawned, they are added to the PID list. As they exit, they are removed. This provides ALL processes the ability to communicate with each other in a totally random fashion. The PID list utilizes synchronization of its own for adding and removing PIDs from the list. It also offers a function call

```
pid_t pidlist_get_random(void);
```

that enables the test application to retrieve a random PID from the list to use as a destination when sending a message. At times, it is possible for a PID to be present in the PID list even though the process may be in the midst of exiting. This is actually a useful test case since it exercises the code path for MAILBOX_INVALID within the SendMsg() routine.

There is a separate test program for the PID list called 'pidlist_test'. This is a multi-process test application used to verify the PID list synchronization.

## ***Test Programs***

1. **Basic Tests**
2. **Dual-Thread Test**
3. **Multi-threaded Test (10 threads)**
4. **Manage Mailbox Test**
5. **Dual-Process Test**
6. **Multi-Process Test (100 processes w/ PID list)**
7. **Multi-Process / Multi-Threaded Test (100 processes, Averaging 10 threads each)**
8. **Multi-Process / Multi-Thread with Random Stopping**
9. **Exit Deadlock Test**

```
/*
 * Test1
 *
 * Performs a series of basic tests, verifying proper return codes and mailbox message content.
 * Also performs negative testing for the message NULL and invalid length arguments.  All tests
 * are run from a single process and the test will exit() on any failure.
```

```
*
* Tests:
*   Basic Send / Receive Test
*                       Sends three messages one at a time of varying lengths and verifies each is received
*                       and the message content is correct.
*
*   Slow Send Test
*                       Sends a series of 10 messages paced one each second and verifies the messages are
*                       received and that the message content is correct.
*
*   Burst Send / Receive Test
*           Sends a burst of MAX_QUEUE_SIZE messages to test the depth of the mailbox. All messages
*           are then burst received and message contents verified.
*
*   Overflow Queue Test
*           The mailbox is filled by burst sending MAX_QUEUE_SIZE messages.  Another message send,
*           NON-BLOCKING then is attempted, with a failure of MAILBOX_FULL expected.
*
*             Underflow Queue Test
*                               The mailbox is filled by burst sending MAX_QUEUE_SIZE messages, followed by a burst
*                               read of MAX_QUEUE_SIZE.  Another NON-BLOCKING message read is attempted, with a failure
*                               of MAILBOX_EMPTY expected.
*
*   SendMsg Arguments Test
*                       Negative tests all arguments to the SendMsg routine.  Tests NULL messages, invalid message
*                       lengths (too long, 0 and negative).
*
*   RcvMsg Arguments Test
*                       Negative tests all arguments to the RcvMsg routine.  Tests handling of NULL pointers passed
*                       to RcvMsg() for *pid, *user_buf, and *len.
*
* Program will exit() on any failure.
*/

/*
* Test2
*
* Performs a series of two-thread tests as described below.
*
* All tests will exit() on any failure.
*
* Tests:
*   Test2a
*           Two thread test with thread T1 sending to thread T2.  T1 sends NUM_MESSAGES
*                       paced one each second.  NUM_MESSAGES is less than MAX_QUEUE_SIZE.  At the end
*                       of the test, the number of sent messages isverified against the number of
*                       received messages.  Both sending and receiving are NON-BLOCKING.
*
*   Test2b
*           Two thread test with thread T1 sending to thread T2.  T1 sends MAX_QUEUE_SIZE
*                       as a burst.  At the end of the test, the number of sent messages is
*                       verified against the number of received messages.  Sending is performed
*                       NON-BLOCKING, receive is BLOCKING.
*
*   Test2c
*           Two thread test with thread T1 sending to thread T2.  T1 sends 2*MAX_QUEUE_SIZE
*                       as a burst.  At the end of the test, the number of sent messages is
*                       verified against the number of received messages.  Both sending and receiving
*                       are BLOCKING.
*
*/

/*
* Test3
*           A multi-thread test where the main thread sources messages and
*           multiple threads receive.  This program creates NUM_THREADS (10)
*           threads to receive messages.  Each receiving thread will run for
*           TEST_TIMEOUT_SEC (6 seconds) in a NON-BLOCKING mode.
```

```
*               The main thread will run in BLOCKING mode and send a total of
*               NUM_MESSAGES messages (50,000).
*
* Program will exit() on any failure.
*/

/*
* Test4
*
* Performs testing of the ManageMailbox() function.
*
* Tests:
*               Test4a
*                               Verifies the count value of the ManageMailbox call.
*                               - Empty returns 0 count
*                               - Incremental send and verify counts
*                               - Filled returns MAX_QUEUE_SIZE count
*                               - Burst decrement and verify count
*
*               Test4b
*                               Verifies MAILBOX_STOPPED is reported correctly.
*                               - NON-BLOCKING send to stopped mailbox expects MAILBOX_STOPPED
*                               - BLOCKING SEND to stopped mailbox expects MAILBOX_STOPPED
*                               - BLOCKING READ to stopped mailbox with messages expects SUCCESS
*                               - NON-BLOCKING READs to stopped mailbox with messages expects SUCCESS
*                               - NON-BLOCKING READ to stopped and empty mailbox expects MAILBOX_STOPPED
*                               - BLOCKING READ to stopped and empty mailbox expects MAILBOX_STOPPED
*
*    Test4c
*                               Verifies mailbox restart handling.  Note - this is not a required
*                               feature and only has been implemented to work under strict constraints.
*                 1. All messages must be read out of the mailbox and the mailbox must
*                   report MAILBOX_EMPTY prior to the ManageMailbox() call to start
*                   the mailbox.
*                 2. All counts and semaphores are forced back to init values.  Full=0,
*                   Empty=MAX_QUEUE_SIZE, count=0, queue pointer are initialized to
*                   empty state, etc.  This may cause a memory leak if messages were
*                   actually left in the mailbox.
*
*
* Program will exit() on any failure.
*/

/*
* test5
*
* A two process test to send messages between two separate mailboxes.
*
* For a specified amount of time, the process will do the following:
*
*               - sleep randomly up to 0.25 seconds
*               - send a message to the peer process NON-BLOCKING
*               - sleep randomly up to 0.25 seconds
*               - receive a message NON-BLOCKING
*
*  If the send queue becomes full, the message will be dropped.
*  If the receive queue runs empty, the process will try again next loop.
*
*  The sender tracks the count of successful sends, the receiver tracks the count
*  of successful receives.  After each process completes, it reports it's
*  send/receive count.
*
*  Any unexpected mailbox error code will cause the program to exit().
*/

/*
* Test6
*
* A multi-process messaging test.  This test spawns 100 processes which
```

```
 *  utilize a shared memory PIDLIST library to track the process IDs of the
 *  active processes.  Every process has access to the PIDLIST, and therefore
 *  can send to every other process.  The test runs for 10 seconds and
 *  processes randomly choose another process to send a message to,
 *  then attempt to receive a message from their own mailbox.  Both
 *  sending and receiving are NON-BLOCKING.
 *
 *  Each process collects statistics about:
 *              - number of messages successfully sent
 *              - number of sends to a full mailbox (MAILBOX_FULL)
 *              - number of messages received
 *              - number of reads of an empty mailbox (MAILBOX_EMPTY)
 *
 *   Any unexpected mailbox error code will cause the program to exit().
 */

/*
 *  Test7
 *
 *  A multi-process / multi-threaded messaging test.  This program provides
 *  command line support for varying the parameters of the test.  By default,
 *  it will spawn 100 processes, each with an average of 10 threads.
 *
 *  Processes will run for a random amount of time and quit.  As they quit,
 *  new processes with new random threads are spawned to replace the old
 *  processes. The total test time could be up to 2 * TIMEOUT.
 *
 *  In the end, the total number of participating processes and threads are
 *  reported.
 *
 *  Each process registers its PID in a shared memory PIDLIST accessible to
 *  all processes.
 *
 *  Each thread chooses a random process and sends a message to the peer process.
 *  It then reads it own mailbox to receive a message.  Both sending and receiving
 *  are NON-BLOCKING.
 *
 * While sending and receiving, statistics are collected:
 *              - number of messages successfully sent
 *              - number of sends to a full mailbox (MAILBOX_FULL)
 *              - number of messages received
 *              - number of reads of an empty mailbox (MAILBOX_EMPTY)
 *
 *  Any unexpected return codes from send/receive will cause the program to exit().
 *
 *  NOTE: it is possible and likely that send will occur to invalid PIDs.  This occurs
 *  when the target process ends while the sending process is attempting post the message.
 *  In this case, the SendMsg lookup of the task_struct fails, and SendMsg reports
 *  MAILBOX_INVALID.
 */

Usage:   ./test7  [<max_processes> <mean_threads> <timeout>]
      max_processes : the number of processes to spawn
      mean_threads  : the average number of threads per process
      timeout       : the time in seconds the test will run

/*
 *  Test8
 *
 *  A multi-process / multi-threaded messaging test designed to exercies the stop-start
 *  functionality of the mailbox.
 *
 *  The test is basically the same as Test7, but in addition, will spawn a separate
 *  thread for each process that controls the mailbox state.  At random times during
 *  the test, it will call ManageMailbox() with stop = true, to stop the mailbox.  This
 *  does  not remove the PID from the PID list, so other processes may still continue to
 *  try to send to the mailbox.  Then, a short random amount of time later, the mailbox
 *  will be restarted by calling ManageMailbox() with stop = false;
 *
```

```
 *  Any unexpected return codes from send/receive will cause the program to exit().
 */

/*
 *  test9
 *
 *   A multi-process echo test for exit deadlock.
 *
 *   This test forks 50 child processes, sending a message to each as
 *   they block on receive.  The child then echoes the message back to the parent
 *   which ignores it letting the parent mailbox fill up.  Since the number
 *   of children is greater than the max queue size, some children will block
 *   on send.  The parent process then exits from main to test whether the
 *   blocked children will clean up and exit properly.
 *
 *   Any unexpected mailbox error code will cause the program to exit().
 */
```

***File List***

| | |
|---|---|
| patch-project4 | - the file for patching the mailbox implementation into the kernel source code. This will modify fork.c, exit.c and Makefile. It will also create cs502_mailbox.h and cs502_mailbox.c. |
| mailbox.h | - the user space header file for the mailbox API |
| mailbox.c | - the user space mailbox library implementation for the syscalls |
| pidlist.h | - provides the API for a shared memory library to maintain PIDs for test programs. |
| pidlist.c | - the shared memory PID list implementation for test support |
| test_utils.h | - the API for a set of helper utility functions to aid the test programs |
| test_utils.c | - the implementation of the helper test utility functions |
| Makefile | - the Makefile for all test programs and mailbox API |
| | |
| test1.c | - basic tests |
| test2.c | - dual-thread test |
| test3.c | - multi-threaded test |
| test4.c | - manage mailbox test |
| test5.c | - dual-process test |
| test6.c | - multi-process test |
| test7.c | - multi-process / multi-threaded test |
| test8.c | - stop / restart test |
| test9.c | - exit deadlock test |
| pidlist_test.c | - shared memory pid list test |

Example test outputs:

- test1_output.txt
- test2_output.txt
- test3_output.txt
- test4_output.txt
- test5_output.txt
- test6_output.txt
- test7_output.txt
- test8_output.txt
- test9_output.txt