Mailbox Algorithm

The kernel module is split into four parts. One, contained in mailbox.c, manages the individual mailboxes and their functions. Another, contained in mailbox\_manager.c, manages the hashtable containing mailboxes. The third, contained in message.c, handles creating and deleting messages. Lastly, the code in module.c is the LKM interface, containing the system call overrides and the code to insert the overrides.

Module:

**Send Message:**

* Get mailbox (and “claim” a copy of it)
* Create message
* Add message to destination mailbox
* Unclaim mailbox

**Get message:**

* Get mailbox (and “claim” a copy of it)
* Get message
* Copy message information to userspace
* Unclaim mailbox

**Manage mailbox:**

* Get mailbox (and “claim” a copy of it)
* Copy number of messages to user
* If user wants to stop mailbox, stop mailbox
* Unclaim mailbox

**Exit thread:**

* If number of live threads is 1, remove mailbox
* Continue exit

**Exit group:**

* Remove mailbox
* Continue exit

Mailbox:

**Add message to mailbox:**

* Lock mailbox
* If mailbox is stopped, unlock mailbox, return
* If the mailbox is full,
  + If block is false, unlock and return with an error
  + Otherwise:
    - Increment number of waiting calls
    - Wait until mailbox is stopped or no longer full. If that fails unlock, decrement number of waiting calls and return with an error
    - Wake up all locked modify calls so one can get lock
    - If mailbox is stopped, unlock and return with error
* Add message to head or tail depending on specified location
* Increment number of messages
* Notify other locked threads so one other will have the chance to gain lock
* Unlock and return

**Remove message from mailbox:**

* Lock mailbox
* If mailbox is stopped, unlock mailbox, return
* If the mailbox is empty,
  + If block is false, unlock and return with an error
  + Otherwise:
    - Increment number of waiting calls
    - Wait until mailbox is stopped or no longer empty. If that fails unlock, decrement number of waiting calls and return with an error
    - Wake up all locked modify calls so one can get lock
    - If mailbox is stopped, unlock and return with error
* Get next message
* Decrement number of messages
* Notify other locked threads so one other will have the chance to gain lock
* Unlock and return

**Stop mailbox:**

* Lock mailbox
* Set mailbox stopped to stopped
* Wake up all locked threads and let them exit
* Unlock mailbox

**Exit mailbox:**

* Same as stop mailbox, but set mailbox stopped to stopped *and* exiting

**Destroy mailbox:**

* Stop mailbox
* Lock mailbox
* Wait until all threads have stopped using the mailbox
* Remove all messages from message list and destroy them
* Unlock mailbox
* Wait until no more functions have claimed the mailbox
* Free memory cache for messages in the mailbox

Mailbox manager:

**Init:**

* For all the elements in hash table, init the hash list

**Get mailbox:**

* Try to find mailbox in hash table
* If mailbox not in hashtable, create mailbox, put mailbox in hash table
* Claim mailbox
* Return mailbox

**Remove mailbox:**

* Try to find mailbox in hash table
* If mailbox in hashtable, stop and exit the mailbox. In a separate kernel thread, run the mailbox deletion
* Otherwise, return mailbox invalid

**Mailbox deletion thread:**

* Wait until task exits
* Remove mailbox from hash table
* Destroy mailbox
* Finish exit

Tests

**mailbox\_test: cleaned up versions of basic functionality and bug tests**

* test\_send\_message - Test if messages can be sent and received
* test\_message\_overflow\_wait - Tests if programs that chose to wait until able to send a message behave properly
* test\_send\_stopped\_mailbox - Tests sending message to stopped mailbox
* test\_recieve\_empty\_mailbox - Tests getting message from empty mailbox
* crash\_test - Crash test

**mailbox\_errror\_test: Try to invoke all the possible ways to get an error**

* c\_supports\_bitwise\_and\_right – Test if c supports bitwise left and right operations
* bad\_process\_id – Try sending message to invalid process id
* mailbox\_full – Fill up mailbox, then send another message without blocking, fails if MAILBOX\_FULL error not thrown
* mailbox\_empty – Receive all messages in mailbox, then receive another without blocking, fails if MAILBOX\_EMPTY error not thrown
* mailbox\_exited – Try sending a message to a child process that has already exited, fails if MAILBOX\_INVALID error not thrown
* mailbox\_stopped – Try sending to and receiving from a stopped mailbox. Fails if MAILBOX\_STOPPED errors are not thrown
* blocked\_wait\_then\_stopped – Tests stopping in the middle of a blocked message send
* blocked\_wait\_rcv\_then\_stopped – Tests stopping in the middle of a blocked message recieve
* msg\_arg\_error\_invoke – *IDK, khazy???*
* msg\_len\_errors – *IDK, khazy???*
* fifo\_even\_if\_errors - Tests that we are receiving messages in first-in-first-out, even if there's an error while reading once.
* recieve\_messages\_even\_after\_stopped – Test receiving messages after a mailbox has been stopped
* closing\_thread\_does\_not\_stop\_or\_destroy\_mailbox – Test to make sure that exiting a thread does not stop or destroy the mailbox
* rapid\_fire\_send\_and\_throw\_an\_exit\_in\_there - Hopefully this can invoke the Mailbox dereference race condition. This invokes the pointer dereference race condition every once in a while, not the best test...a better test would fork and rerun this several times. If this test fails, you get a kernel oops.
* rapid\_fire\_send\_recieve\_track\_how\_many\_messages\_we\_get\_eventaully - Stress test sending and receiving messages, one way. Hypothetically, two way is the same thing, just need to create two threads in each process
* the\_crazy\_test\_that\_is\_suggested\_in\_the\_pdf\_handout – stress test in the pdf. Made in like 20 minutes, doesn’t clean up threads properly, need to sigint once cpu usage drops to 0

Output:

./mailbox\_test

./mailbox\_error\_test