Dan Cassidy
CSCI-C 435
Homework 1

Problem

Add code to the MINIX 3 kernel to keep track of the number of messages sent from process (or task) *i* to process (or task) *j*. Print this matrix when the F4 key is hit.

What I Did

In summary, I expanded the `proc` structure, captured messages that were successfully sent, and allowed the user to display a matrix of these messages by hitting the F4 key.

Here is what I did in simple list form in chronological order, basically akin to a rolling changelog. Explanations and the process will come later in the "Approach" section.

- Expanded `proc` struct with an unsigned long array `p_mess_sent` of size `NR_TASKS + NR_PROCS` in order to have a place to record the number of messages being sent. [/usr/src/kernel/proc.h]
- <Test point 1>
- Added prototype for `message_dmp()` function. [/usr/src/servers/is/proto.h]
- Hijacked F4 key for `message_dmp()` function. [/usr/src/servers/is/dmp.c]
- Added `message_dmp()` function stub to print off dummy text when F4 is pressed. [/usr/src/servers/is/dmp_kernel.c]
- <Test point 2>
- Added test initialization code for `p_mess_sent` array to initialize index `0` to the offset process number `p_nr + NR_TASKS`. [/usr/src/kernel/main.c]
- Fleshed out `message_dump()` function to display the test initialization value. [/usr/src/servers/is/dmp_kernel.c]
- <Test point 3>
- Modified `do_fork()` function to clear `p_mess_sent` when called, and to also zero out the number of messages sent by all other procs to any previous process that had the same pid. [/usr/src/kernel/system/do_fork.c]
- <Test point 4>
- Removed test initialization code for `p_mess_sent` array and replaced it with the formal initialization code. [/usr/src/kernel/main.c]
- <Test point 5>
- Added message sending capture. [/usr/src/kernel/proc.c]
- Fleshed out `messaging_dmp()` function to facilitate testing of message sending capture. [/usr/src/services/is/dmp_kernel.c]
- <Test point 6>
- Added paging and finalized `messaging_dmp()` function. [/usr/src/servers/is/dmp_kernel.c]
- <Test point 7>
- Recompiled commands. [/usr/src/commands/]
- <Test point 8>

How To Run It

- Install virtual appliance.
- Start virtual appliance.
- Allow Minix to load on its own by NOT using any of the 3 options that appear during boot.
- Log in. (Optional)
- Hit 'F4' and observe the messaging matrix display.
- Continue to hit 'F4' and observe the matrix paging in left-to-right, top-to-bottom order.

Approach

Based on what we discussed in class, I decided to make my first goal be to expand the `proc` structure so I had a place to record the number of messages being sent by adding an array named `p_mess_sent` with `NR_TASKS + NR_PROCS` elements. In the process of making that change, a major question that I came up against was how overflow should be handled. In exploring the Minix kernel, it appeared to me that there were no checks on whether the `p_user_time` or `p_sys_time` members of the `proc` structure overflowed so I therefore decided to apply that methodology to my code, resulting in the deliberate absence of overflow checks and handling procedures.

In creating the function to actually display the message dump, `messaging_dmp()`, I went through several stages in getting it working. The first was to alter proto.h and dmp.c so I could call my function when F4 was pressed, and the second was creating a stub function to call. After verifying that this was working, I then went back and added a test initialization value (the process ID, or pid) into element 0 in the `p_mess_sent` array in the `proc` structure and further modified `messaging_dmp()` to display those values.

This led me to an interesting discovery that outside of the initial system processes, the rest of the processes didn't have their proper test initialization values. After much thinking, it came to me that it was because those processes were simply being forked from the base system processes, and the process table entries copied. This meant that the child's message count needed to be reset after the process gets forked. In this case, my code will first reset the entire `p_mess_sent` array for the new child process, then it will iterate through the entirety of the process table and zero the specific element of all `p_mess_sent` arrays that corresponds to the offset pid of the child process on the chance that the child pid was used in the past by some other process.

Once tested successfully, I replaced the test initialization code in the kernel's `main()` function with the final form of that code.

It was at this point that I decided to add the code to actually record a message being sent. Given that the `mini_send()` function in proc.c is the clearinghouse for all messages being sent, this was the logical place to put the code. After some initial indecision, I decided to add the code to record that a message was successfully sent right before `mini_send()` returned `OK` to indicate success. This means that I would be certain to count the message.

After this, the main thing that was left was to build out the `messaging_dmp()` function and made sure it printed things correctly. All told, the process to create the matrix printout of the number of messages sent was comparatively simple, in that it did not require mucking about with the internals of the kernel itself. I created two different ways to page through the data, left-to-right/top-to-bottom and top-to-bottom/left-to-right, based on conditional compilation. The left-to-right/top-to-bottom method is what I ultimately chose.

Testing

One of the key aspects of this homework was doing both incremental changes, and incremental testing. This helped to ensure that if I did break something, I knew what I broke since I only changed one thing (or one set of tightly coupled things) since the last compilation and testing cycle. As noted in the **What I Did** section, there were 8 main testing points.

- Test point 1: Recompiled Minix kernel and rebooted into new image to ensure that OS booted and the different …`_dmp()` functions still worked and I hadn't nuked the process table.
- Test point 2: Recompiled services and ensured that the dummy message printed out correctly when 'F4' was pressed.
- Test point 3: Recompiled kernel and services and rebooted into new image. This made sure that the OS booted. After that, 'F4' was pressed to ensure the test initialization values were being displayed. (This is where I ran into the funny business regarding forked processes.)
- Test point 4: Recompiled kernel and verified that the test initialization values were what was expected. Test point 3 verified that test values were being transferred, so this test is simply making sure that for the base system processes the test value still applies, and for the other processes that they get zeroed out.
- Test point 5: Recompiled kernel and rebooted into new image to ensure it still booted. Also tested 'F4' function to ensure that all 0's printed, indicating successful initialization of the array.
- Test point 6: Recompiled kernel and services and rebooted into new image. Nothing blew up, which was nice. Hit 'F4' several times in a row and ensured that the number of messages sent was increasing.
- Test point 7: Recompiled services and rebooted, then pressed 'F4' to ensure that paging was working properly.
- Test point 8: It was around this time that I realized I needed to make sure that other commands that touched the process table, such as `ps` and `top` still worked. Initial testing revealed that `ps` was simply broken, and `top` was displaying bad data and calculations. After recompiling all the commands, `top` worked correctly again, but `ps` was still broken. As it turned out, it needed to have its stack+malloc space increased by at least 30k, by using the command "`chmem +30000 /usr/bin/ps`". I also gave it another 20k just for padding. After this, `ps` worked just fine.

What Was Difficult

There were two difficult parts for me during this assignment. The first was figuring out that doing a fast boot by hitting '1' ended up corrupting … SOMETHING in the operating system resulting in totally bogus numbers. The second was actually figuring out why my test values for the `p_mess_sent` array were getting overwritten. This ended up confusing me quite thoroughly for a little while, but ultimately was completely logical.

What Was Easy

Ironically, once I figured out where, adding the code to capture the number of messages being sent back and forth was actually pretty easy.