CmpE 142

Section 1

Remote Memory Access Kernel

Nicholas Foulk, #17

Thinh Phan,

Gary Lai, #25

Alejandro Reyna, #37

Jun Ming Liu, #28

Table of Contents

Assumptions Page 2

Division of Work Page 3

System Architecture, Modules, Pseudo Code Page 4

Testing Page 6

Results Page 6

Conclusion Page 7

Work Cited Page 8

**Assumptions**

Before doing any coding, it was important to identify the major components required for the project. It was clear that there were no exact Linux modules that would complete the whole project. It was assumed that this project required a large amount of researching before any coding could have been done. An important note to remember was tinkering with the kernel is dangerous because it ran the risk of destroying the system. It was important to research and learn about Linux kernel modules and modifications before inserting or removing any new kernel modules. Google was the best help available. Most of the concepts required in the project required specific researches. Many sources of information were required in this project. One major aspect of the project was the connection between two virtual machines. The original idea was to connect two machines to connect to two laptops via the ethernet ports, an ethernet cable, and a switch but it proved to be difficult. Sharing memory between two process through a physical networking ethernet cable proved to be difficult. As a result, the group decided to connect two virtual machines through the virtualbox software.

Inside the virtualbox, there was option to a virtual LAN of multiple virtual machines. A DHCP server with one static IP was created as a virtual server and a subnet mask was added on a single machine. Sockets were created for the server and client, then the sockets were bound to their respective side. The server assumed that it would “be listening” for the client. After creating a virtual LAN for the machines, both virtual machines pinged each other to make sure the connection was established. If the pinging worked, it meant the virtual server worked. It was assumed that getting the connection to work between the two virtual machines was the most important part in the project. After getting the network code to work, the rest of the project code would follow through. Once the connection between the server and client was functional, then the memory map module was created.

By testing the memory map module in the userspace first, the integration was then implemented into the TCP server that was created. With the memory mapping module created, it was assumed that the client could create a file that would have been sent to the server. To create a stable connection, both client and server had to be modified so the server would not close the connection when the client was sending a data over. In kernel space, a file, “debugfs”. was created, This was further modified in the userspace with the memory mapping function. In the kernel space, a fault handler was created which was linked to the memory mapping function in the user space. It could had the ability to map the pages. The netlink library was incorporated into the kernel and user space so the kernel would be able to communicate with the user space processes. It was assumed that most of the code would be running on kernel space except for netlink and memory mapping.

**Division of Work**

The progress report was written to divide the work between each group mates, but it was different when it came to executing the project. Although the progress report show that each group mate should have understood an aspect of the coding, it became clear that the coding research and implementation had to be a collaborative effort. The project required simultaneously understanding of the topics required in the project at the same time. As a result, it was decided that three team members would focus on the programming while the other two team members documented the progress, actions, and researched anything that required for the project. Nicholas and Thinh were the major programmers of the project, while Gary was a support programmer and researcher. Alejandro and Jun, the documenters, researched some necessary topics for the project and documented the programmers’ actions and progress. The documenters condensed the programmer’s code into pseudo codes. This meant that the documenters needed to have an understanding of the code even though they did not physically write any of it. While the programmers were still coding and researching, the documenters recorded the initial or final intentions and what were some of the mistakes that occurred.

**System Architecture, Modules, Pseudo Code**

* System Architecture:

Intel i5 3210M

4 GB RAM

Oracle VM VirtualBox 4.3.26, running Ubuntu 14.04

* Pseudo Code

**Self-Made Server Module with mmap capabilities**

Initialization

mmap\_open();

mmap\_close():

mmap\_fault();

vm\_operations mappings();

file\_operations open();

file\_operation close();

file\_operation mmap();

file\_operation mappings();

Create\_SocketNetlink();

Create\_NetworkSocket();

Bind\_NetworkSocket();

NetworkSocket\_Listen();

Accept\_NetworkSocket();

Receive\_Message();

Create\_DebugfsFile()

Create debugfs file named “mmap example1”;

Create netlink between server and user process;

MMAP debugfs using argv[1]

Send argv[1] from user process to Server

Send argv[1] to client from server;

Exit\_Commands();

Release\_ServerSocket();

ReceivingBuffer\_SocketRelease();

Remove\_DebugfsFile();

Release\_NetlinkKernelSocket();

**Self-Made Client Module**

Initialize\_MessageBuffer();

Create\_ClientSocket();

Connect\_ClientSocket();

DeclareBuffer\_Lengh();

Send\_ClientMessage();

Receive\_Buffer();

Receive\_Message();

Allocate\_KernelMemory()

Exit\_Commands();

Release\_Socket();

**Modified mmap User Space Module**

Open\_Debugfs(“mmap\_example12”);

mmap(“mmap\_example12”);

Print\_Message();

Create\_NetlinkSocket();

Bind\_NetlinkSocket();

Initialize\_Socket();

Allocate\_MessageHeaderMemory();

Copy\_Data(Input data)

Insert Data into message;

Send\_MessageThroughKernel();

Receive\_MessageThroughKernel();

Close\_NetlinkSocket();

**Testing**

There were multiple steps involved in testing if the project worked correctly. First thing that had to be tested was if the the virtual machines. one with the server module and the other with the client, were able to communicate with each other through the network. Pings were sent the the ip addresses of each virtual machine and checked if data was being sent. The test showed that the virtual machines were able to successfully ping each other. After the network worked correctly, the server and client modules were checked to see if data was able to be passed between the two. The server and client ran on the same virtual machine, and used inaddr\_any to check if there were able to communicate with each other. The server and client through the network was tested, by first inserting server module, then inserting the client module to see if it would unlock the server module. The dmesg of each module was checked to see if the sockets,messages,and send/receive were working correctly. Finally, the mmap was tested, but by first inserting the server module and then the client module. When the server module unlocked, data was able to be sent from the server to the client. The command line was then used to send five messages to the client, and used dmesg to check if the client was receiving the correct amount of bytes as well as the correct text.

**Results**

After facing difficulties with pinging two virtual machines through ethernet cord and a switch, a virtual lan was created between two virtual machines. The virtual lan enabled the virtual machines to ping each other through the use of terminal. Afterwards, a demo code was used to test the connection between the two virtual machines with one being a client and one being a server. The code demonstrated a call and response between the two servers. After the connection between the server and client sockets, the demo code simulated a conversation between both sides. When testing the server/client module, the correct data was being send from the server and the client was receiving the correct amount of bytes and data and vice versa. Overall, the machines were able to ping between each other, communicate with each other either through the same virtual machines or through the network, and were able to send and receive data. A successful netlink connection was created between the mmap user process which updated the debugfs file and then sent the data, sent the argv[1] back the server, and the server relayed that message back to the client. A custom mm\_fault handler handler was used to map the new pages onto shared physical memory.

**Conclusion**

After finishing this programming project, it was learned that most of the project required a large amount of knowledge in linux kernel modification. There were many bugs and statements that were not checked. Writing more error checking functions was very preferable but the large amount of different errors made it difficult. There were many bugs that caused core and stack dump. The only way to fix these errors was to restart the virtual machines. If the client was ran first before the server was up, then there was a core and stack dump errors shown on the client virtual machine dmessage. Another recommendation would have been to code the server and client in user space because any bugs or bad programming caused the server to shut down and disable module unloading.

**References**

Corbert, J., Rubni, A., & Kroah-Hartman, G. (2005). Memory Mapping and DMA. In *Linux Device*

*Drivers* (3rd ed., p. 640). Sebastopol, CA: O'Reilly Media.

Cox, A. (n.d.). Network Buffers And Memory Management. Retrieved April 26, 2015, from

<http://www.tldp.org/LDP/khg/HyperNews/get/net/net-intro.html>

Day, R. (2010, June 10). An introduction to Linux kernel programming - An introduction to Linux

kernel programming. Retrieved April 23, 2015, from

[http://crashcourse.ca](http://crashcourse.ca/introduction-linux-kernel-programming/introduction-linux-kernel-programming)

Johnson, D. (2014, January 20). Linux Kernel Reading Guide. Retrieved April 23, 2015, from

<http://www.linux.org/threads/linux-kernel-reading-guide.5384/>

Francis, S. (2014, July 17). The right way to monitor virtual memory on Linux | The LogicMonitor

Blog. Retrieved April 24, 2015, from

<http://blog.logicmonitor.com/2014/07/17/the-right-way-to-monitor-virtual-memory-on-linux/>

GANAPATHY, L. (2013, June 13). How to Compile Linux Kernel from Source to Build

Custom Kernel. Retrieved April 24, 2015, from

<http://www.thegeekstuff.com/2013/06/compile-linux-kernel/>

Gorman, M. (2004). *Understanding the Linux Virtual Memory Manager*. Upper Saddle River,

NJ:Prentice Hall

Gordon, S. (2013, November 9). Creating a Virtual Network of Linux Guests using VirtualBox.

Retrieved April 25, 2015, from

[https://sandilands.info/sgordon](https://sandilands.info/sgordon/creating-a-virtual-network-of-linux-guests-using-virtualbox).

Keller, A. (2008, July 1). Kernel Space - User Space Interfaces. Retrieved April 26, 2015, from

<http://people.ee.ethz.ch/~arkeller/linux/kernel_user_space_howto.html#ss8.1>