Project report on

Advanced Operating Systems Project

Ganesh Ramkrushna Zilpe
1207578251
Master of Science in Computer Science
Arizona State University
Ganesh.zilpe@asu.edu

I. INTRODUCTION

Advanced operating system course deals with multiple concepts of operating system: Protection and file systems, synchronization, communication, security, etc. Practical approach is very important to understand these concepts. Therefore, studying and experimenting with operating system source code is important to do that. This project report explains character device driver development for communicating between two machine using logical clock for synchronization and using IPv4 for communication. During the development of the project, I have learned to develop character device driver for Linux. I have learned to create virtual machine (e.g. VMware) on host and installing and operating the guest operating system (e.g. Ubuntu) on it. During this project, I have built the kernel module, for that I have downloaded complete source code Linux. The character device driver is built as kernel module. Thus, the project is mainly involved

The project is divided in three phases based according to the development of project.

Phase 1: Installation of Ubuntu along with downloading and compiling the Linux Kernel and build, debug a .ko.
Phase 2: Build a communication channel that uses normal read/write system calls to communicate over IP.
Phase 3: Use a logical clock to impose an ordering on Event and Acknowledgement messages.

II. PHASE 1

This phase deals with to make ready Ubuntu for the phase two. This phase is divided into three parts based on functionality.

A. *Creating virtual machine on host machine and installing Ubuntu on it*

Ubuntu is free of charge Linux-based operating system. For this project, I have used VMware Player 3.0.0 build-203739 for creating virtual machine. I have installed Ubuntu .04.1 LTS (long-term support) as a guest operating system on the virtual machine. Host operating system is Windows 7(32-bit). We can do the project on machine where Ubuntu is directlt installed on hardware. But, here we are taking help of virtual machine because we are going to change the soirce code of kernel. If something goes wrong, then it is very easy to delete one virtual machine and create other than installing new OS on hardware actually.

B. *Download and compile Linux Kernel*

In preparation for compiling applications and kernel objects to run in the Ubuntu development environment, the kernel source code and associated utilities should be available on machine.

Download and install the utilities which are needed to perform the compile and packaging of the kernel. Start by first updating the package list. After that, download all of the needed packages. Some will have already been installed so this is a shotgun approach but should capture any missing utilities. Download and decompresses the source code for the latest kernel from the Ubuntu source tree. Also, now do the clean build.

Important Commands:

- *sudo apt-get update*
- *sudo apt-get install dpkg-dev kernel-package gcc libc6-dev binutils make bin86 module-init-tools gawk gzipgrep libncurses5-dev*
- *apt-get source linux-image-$(uname -r)*
- *make-kpkg clean*
- *fakeroot make-kpkg --initrd --revision=1.0.cse536 kernel_image*

C. *Build and Debug a .ko and an application program*

In this part of project, I build and install a simple .ko and an application program that accesses the .ko as a device driver. Here, the simplest of the driver types, a character I/O driver is used. Create application program and compile, execute it. Also, create C file of module and create Makefile for the same with necessary configuration setting. Now rebuild the kernel including the new module.

Important Commands:

- *make menuconfig*
- *fakeroot make-kpkg --initrd --revision=1.0.cse536 kernel_image*
- *make M=drivers/char/cse536 modules*

III. PHASE 2

This phase deals with the communication between two machines through module. In this, the module created in Phase II is modified to communicate with module on other computers by interfacing with standard IPv4. Internet Protocol (IP) normally operates in the standard internet network stack and is able to interface with a variety of transport layer modules as well as link layer modules. Typical transport layer modules include TCP and UDP. A typical link layer module would be Ethernet. In the current world of network technology IPv4 is the internet glue and the lower layer modules interface IPv4 with local hardware while the upper layer modules provide application programs access to IPv4. We are not using the standard socket interface used by most network programs. We allow application program to communicate in full duplex with application program running on any arbitrary network end location via ipv4 using standard read/write system calls. We are not changing any functionality of Ipv4 layer. While performing write on the machine A, a read buffer at machine B is filled with data we have written at machine A and vice-versa.

While communicating with other machine, the other machine is agreed to communicate on specific protocol number. We are using 234 as protocol number in this project. Other students of the class are also using 234 as protocol number, so that we can communicate with each other at the end of phase 3.

This phase of project is using different methods of module. I have explained these methods along with the functionalities of them for this project.

A. *Register protocol:*

As mentioned in the project scenario, standard protocol is not going to be used. Hence, we need to register new protocol number to networking model. Be careful, choosing proper protocol number is very important, because we cannot use existing protocol number. All protocol numbers are given in in.h file.

```
static int cse536_add_protocol(void)
```

B. *Write the message:*

As we are using character device for communication between two machines, while sending the message, we need to write the message to the character device buffer first, so that character driver can send that message over the network.

This write() method is used for two different reasons: 1. for setting the destination machine IP address and 2. for writing the message data. In this code, when buf[0] = 1 means the data written on command prompt is destination address. If buf[0] = 2 or more then the data is actual message string need to be sent over the network. Before sending the data, the destination address needs to be entered. Otherwise, there will be no destination address, so message won't be sent.

```
static ssize_t cse536_write(struct file *file,
const char *buf, size_t count, loff_t * ppos)
```

C. *Send the message:*

As we have destination address and also data to be sent, we can send the message over network. Here, we create the IPv4 packet with stuffing the message data into sk_buff. All network-related queues and buffers in the kernel use a common data structure, struct sk_buff. This is a large struct containing all the control information required for the packet (datagram, cell, whatever). The sk_buff elements are organized as a doubly linked list, in such a way that it is very efficient to move a sk_buff element from the beginning/end of a list to the beginning/end of another list. A queue is defined by struct sk_buff_head, which includes a head and a tail pointer to sk_buff elements.

```
static int cse536_sendmsg(char *data, size_t
len)
```

D. *Receive the message:*

As the message is sent over the network with the destination address, it may arrive at the destination. If the message is arrived at the destination, then the message data needs to be stored at the buffer of character driver.

```
int cse536_rcv(struct sk_buff *skb)
```

E. *Read the message:*

If the message is arrived at the destination, then the message data gets stored at the buffer of character driver. We can read the data from the character driver.

```
static ssize_t cse536_read(struct file *file,
char *buf, size_t count,loff_t *ptr)
```

IV. PHASE 3

The goal of this phase is to build a communication channel that uses normal read/write system calls to communicate over IP for sending event and acknowledgement.

This phase of project uses development of phase 2 to communicate between machines to implement a logical clock. When sending a message from machine M1 to machine M2, we want to have an ordering to determine among several messages which came first. We can maintain a common clock for each host that will place an ordering on the messages that are communicated between the hosts.

Important points:

- use clock for ordering and store it in the .ko
- wait for 5 sec for receiving acknowledgement before retransmitting the event, if does not receive for the second time, then consider acknowledgement is unobtainable
- Packet Format: For event Record ID = 1and for acknowledgement Record ID = 0
  - 4 Record ID – ack=0 or event =1
  - 4 final clock
  - 4 original clock
  - 4 source IP
  - 4 destination IP
  - 236 string

Some highlight changes for the methods mentioned in Phase 2:

A. *Write the message:*

For this phase, we need to write the message along with assigning written packet structure to linked list of cse536 buffer, so that we can read this filled packet structure in the application program and can be able to send to monitor. Also, after sending the written message, increment the logical clock of the module.

B. Receive the message:

If received message is event, then we need to send back the acknowledgement to the sender. For checking whether received message is event or not, we can check the record_id of message. For event, record_id is 1. Also, we need to synchronize the logical clock of module with the final clock set in the received message. If received message is event and module clock is less than the final clock of received event, then assign final clock of received event + 1 to the module logical clock.

C. Receive the message:

If the received message is acknowledgement, then with reading the acknowledgement, we need to send the acknowledgement to the Monitor from application program.

D. Sending event and acknowledgement to the Monitor

If we sent event to other machine/user, the we need to send that event to Monitor also. Similarly, if we get the acknowledgement for the event we have sent in the past, then also we need to send the acknowledgement to the Monitor.

V. RESULTS

In the first phase of project, I have leaned to create virtual machine and installing Ubuntu on it. Along with tha, downloading the Linux kernel source code and creating the character driver along with creating, building and attaching kernel module with devices. For the second phase, I have created character device drive and successfully communicated between two virtual machines like full duplex. During the final third phase, I have successfully communicated with other computer using logical clock for synchronization i.e. my module is able to communicate with different machine at the same time without changing in any code and also able to show/send successfully Event and Acknowledgement on/to Monitor software.

I have come across some difficulties and following are my findings:

A. Problem with Ubuntu on 32-bit hardware without graphics accelerator/graphics card

There is one problem when we install Ubuntu 14.04.1 on underlying hardware which supports only 32-bit operating system. After installation of ubuntu on VMware player, when we login to the OS, we get flickered desktop with no visible icon or taskbar. I found out the solution for this problem. When you create virtual machine in VMware player, you can change the virtual machine settings. In the setting, you can see Display setting. Uncheck the checkbox of Accelerate 3D graphics.

B. Problem registering protocol:

All by-default protocols are registered in the namespace. But, as we are added new protocol and we are not going to create separate file for its functionality because we are not following network layer of TCP/IP model.

Therefore, when I compile kernel module, I got following error:

Error: Could not insert module driver/char/cse536/cse5361.ko : resource temporarilly unavailable

Hence, we need to add protocol as non-namespace aware. For this, I have found a patch to add non-namespace aware protocol. This modification is done in /net/ipv4/protocol.c.

```
int inet_add_protocol(const struct net_protocol *prot, unsigned char
protocol)
{
        +#ifdef CONFIG_NAMESPACES
                if (!prot->netns_ok) {
                        pr_err("Protocol %u is not
                        namespace aware, cannot
                        register.\n",protocol);
                        return -EINVAL;
                }
        +#endif
        return !cmpxchg((const struct net_protocol
**)&inet_protos[protocol], NULL, prot) ? 0 : -1;
```

After adding this change in above method, I compiled complete kernel. But, this did not work also. I got the same error also. Then after carefully studying what happens actually so that this error message is coming, it is coming because of **.netns_ok**. I have also posted the query in class discussion and with the help of answer of this post, I figured out the solution. So I modified my protocol structure as follows:

```
   static const struct net_protocol cse536_protocol = {
        .handler     = cse536_rcv,
        .err_handler = cse536_err,
        .no_policy   = 1,
       >> .netns_ok = 1,
      };
```

VI. PROJECT CONTRIBUTION & KNOWLEDGE ACQUIRED

This project is an individual activity and complete project is done by me with the help of code snippets provided by professor. During this project, I have learned important topics of project like creating kernel module, character device, use of semaphore, communication over IPv4 along with synchronization and ordering of messages in the form of event and acknowledgement.

VII. REFERENCES

http://www.ubuntu.com/download/desktop/contribute/?version=14.04.1&architecture=i386
https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/6_0
http://askubuntu.com/questions/218296/problems-running-ubuntu-12-10-in-vmware-player-5
http://linux.die.net/man/1/minicom
http://www.gnu.org/software/ddd/
https://docs.oracle.com/cd/E23823_01/html/816-4554/ipov-6.html
http://www.linuxfoundation.org/collaborate/workgroups/networking/sk_buff
http://www.spinics.net/lists/netdev/msg248984.html