Writeup 1

CS 444
Spring 2017

Abstract

This writeup is the writeup for Project 1 for CS444 Operating Systems 2. The write-up includes the Linux Yocto kernal on os-class.engr.oregonstate.edu and the solution of Producer-Consumer in C.

Contents

1 Log of Commands to Build Yocto Kernel

1. Open two terminal windows. In terminal #1, do steps 2-12. In terminal #2 do step
2. Log on to os-class : pengc@os-class.oregonstate.edu(Please use your own user nam
3. Call cd scratch/spring2017/10-01 . If you don't have this folder(10-01), call m
4. Call git clone git://git.yoctoproject.org/linux-yocto-3.14 to download the proje
5. To switch to the correct tag, call cd linux-yocto-3.14, and then git checkout v3
6. Before we build our kernel or run qemu, we should configure the environment, so
7. Follow 8-12 steps, make a kernel instance for your group.
8. Run cp /scratch/spring2017/files/config-3.14.26-yocto-qemu .config
9. Run make menuconfig and you will get a window
10. In the widow do the following: press / and type in LOCALVERSION, press enter.
11. Hit 1, press enter and then edit the value to be -10-01-hw1 (-10-01-hw1 for grou
12. Run make -j4 all, your kernel instance will be built with 4 threads
13. Run cd .. and then run gdb. Stop here for now.
14. In terminal #2, do step 6.
15. To make a copy for the starting kernel and the drive file located in /scratch/sp
16. Call cp /scratch/spring2017/files/bzImage-qemux86.bin . (This . is an operand)
17. Call /scratch/spring2017/files/core-image-lsb-sdk-qemux86.ext3 . (This . is an
18. Try run the starting kernel : Call qemu-system-i386 -gdb tcp::5601 -S -nographic
19. Since in step 18, we run qemu in debug mode with the CPU halted. We need to use
20. In terminal #1, it is now in gdb. Run target remote :5601 to connect the qemu.
21. Run continue. Then you will see the change in terminal #2.
22. If you succeed in running qemu, you will be asked to login. Type root and enter
23. Use reboot to reboot the VM.
24. Try run the kernel instance we created in steps 8-12. The kernel instance we bu
25. Run qemu-system-i386 -gdb tcp::5601 -S -nographic -kernel linux-yocto-3.14/arch/
-drive file=core-image-lsb-sdk-qemux86.ext3,if=virtio -enable-kvm -net none -usb -localt
26. Do steps 20-22 again. You should find the difference in kernel names. The name s
27. reboot the vm and use q to quit gdb. You are done with the kernel portion of HW

2 Log of Commands to Load Qemu

3 Flags in the listed Qemu command line

The listed Qemu command line is:

```
qemu-system-i386 -gdb tcp::???? -S -nographic -kernel bzImage-qemux86.bin

-drive file=core-image-lsb-sdk-qemux86.ext3,if=virtio -enable-kvm

-net none -usb -localtime --no-reboot --append

"root=/dev/vda rw console=ttyS0 debug".
```

The following list describes each flag:

- *-gdb* instructs qemu to wait for input from the subsequent flag.
- *tcp::5628* gives qemu the device to listen on for gdb input.
- *-S* tells qemu to not start the CPU at startup.

- *-nographic* disables graphical output so qemu appears as a command line application only.
- *-kernel* followed by a file specifies the kernel to be used, in this case the bzImage we created.
- *-drive* defines a new drive; the following arguemnts specify the disk image file (file=core-image-lsb-sdk-qemux86.ext3) and the interface option (if=virtio).
- *-enable-kvm* enables KVM virtualization, a very complicated function that assists in emulating hardware.
- *-net* interfaces with network options; assumedly as it is followed by "none" the VM does not have networking support.
- *-usb* enables the USB driver.
- *-localtime* enables the machine to start at the current local time.
- *–no-reboot* configures the system to shutoff instead of reboot.
- *–append* tells qemu to use the following argument ("root=dev/vdarw console=tty50 debug") as the kernel command line.

4 Concurrency

The following subsections answer the four questions as outlined on the Project 1 page on Kevin McGrath's course website.

4.1 Main point of assignment

There were several reasons for the assignment. One, I think, was to re-familiarize us with POSIX Threads, which we learned about in CS344 but likely forgot about before we got to this class. Second, I think that it was to teach us about the Producer-Consumer problem, which is a canonical example of concurrency. Third, it was to force us to use inline assembly language in our programs to help us learn how to implement ASM in C.

4.2 Personal approach to problem

My approach for the use of threads came from guidelines to the canonical Producer-Consumer problem that I found in the *Linux Programming Interface* text. For example, statements like pthread_create, pthread_join, pthread_t, pthread_mutex_t, and pthread_cond_t were all found in that book and that book instructed me on how to use them. I used these static pthread statements rather than dynamic ones because frankly, I could not think of a reason why dynamic ones would be necessary for the purposes of this assignment. My approach for the buffer and the item to be held in the buffer was just to use a struct to represent the buffer and a struct to represent the item and have the buffer hold a statically allocated array of pointers to items and for each item to have two data fields: a random number and a random waiting period, both that had to be implemented in ASM. Given the statements necessary to control threads and mutexes and the structs necessary to store data, I solved the problem by creating a thread to correspond to a Producer, having that Producer thread lock all of the shared data using a mutex and then produce a random item. After unlocking the shared data within the Producer thread, I created a Consumer thread that locked the data again, consumed the first non-null index in the buffer, and then unlocked the data again. I contained these Producer and Consumer thread creation statements in a for loop that runs the number of times the user indicates in a command line argument so that when the program is run it creates the corresponding producer and consumer threads and produces and consumes random data.

4.3 Ensuring solution was correct

To ensure that the buffer was properly being added/removed to and to ensure that both threads were properly locking and unlocking, I added print statements in important functions such as when the global mutex was locked and unlocked in order to know whether these calls were actually being made. It is easy to verify that the threads are switching back and forth because both threads have different print statements that are only accessed within their specific thread is running. It is also easy to realize that the buffer is only sequentially being accessed by the two different threads in accordance with the mutex locking and unlocking because otherwise both threads would be accessing the same resource and either a memory read/write issue would occur or both threads would be depending on each other for an inordinate amount of time, resulting in *deadlock*. Obviously, my use of the pthread condition functionality allowed both threads to wait until resources were available, thus preventing deadlock. Finally, to ensure that the entire integrated software worked properly, I used a makefile to generate an executable of the code that accepts 1 argument that corresponds to the number of times a loop that creates the threads shall occur. To ensure the executable achieved what I wanted it to, I ran the software with command line arguments of 1, 2, 3, 4, and 5. The corresponding output for these arguments properly demonstrated that the different threads were called 1, 2, 3, 4, and 5 times, respectively, which is correct behavior.

4.4 What I learned

Austin reviewed knowledge of concurrency and with the synchro2 problem discussed in the recitation, is starting to get into the parallel mindset. Using mutexes again helped this process to remember to not allow multiple threads to access the same data. The assignment was a good warmup for any future problems in parallel.

5 Version control log

History of versions

5.1 git logs

597d814 was Austin, 43 minutes ago, message: Tex edit
304e7e8 was IStallcup, 4 hours ago, message: small changes to concurrency
5336067 was IStallcup, 5 hours ago, message: Merge branch 'master' of https://github.com/Boeinco/CS444
6 fa3edf was IStallcup, 5 hours ago, message: added qemu flag explanation
5ffd876 was Isaac Stallcup, 5 hours ago, message: Create .gitignore
66d601f was Austin, 12 hours ago, message: Writeup work
53de57c was Austin, 12 hours ago, message: fixed command arg c fb1942 was Austin, 32 hours ago, message: writeup updated
401f05e was Austin, 32 hours ago, message: writeup updated
8425628 was IStallcup, 33 hours ago, message: Added some more stuff
cf27ae5 was IStallcup, 33 hours ago, message: added some writeup formatting
6e073c6 was Austin, 34 hours ago, message: Template tex file done
dae7e2e was Austin, 35 hours ago, message: concurrency mostly done
4d7f1cc was Austin, 2 days ago, message: temp files
3e7be70 was Austin, 2 days ago, message: some files in
599f66e was Austin, 2 days ago, message: started concurrency c 6a4f0c was Austin, 2 days ago, message: Testing directories
8e404a9 was Austin Nguyen, 3 days ago, message: first commit

Version	Date	Author(s)	Changes

6 Work log

6.1 April 18-17

Austin created git project and started working on concurrency. Used latex template to set up the latex document and git commits under username Boeinco.

6.2 April 19-17

Austin started working on the new project requirement of the command line argument after creating the main project frame. After creating mutexes and having them sleep for the specified random amount of time, work was done on translating it to the producer/consumer problem.

6.3 April 20-17

Austin finished up the producer/consumer problem and added the qemu instructions. Isaac did research on the flags during the linux commands and also helped format the latex file. Some latex work was done.

6.4 April 21-17

Austin did the section of version control and other parts of the document.