Project 1

Ty Skelton

Abstract

The first project of Operating Systems 2. This assignment is meant to introduce us to working with the the linux kernel and writing concurrent programs in C. The process of building the kernel to loading it on the VM connected to the debugger will be explained, along with a write-up of a concurrency programming assignment. The concurrency programming assignment solves the producer-consumer problem in C. This assignment will lay the ground work for the rest of the class.

I. Booting the Kernel on the VM

A. Log of Commands

1) : Acquiring a local copy of the Kernel by running

```
$ git clone git://git.yoctoproject.org/linux-yocto-3.14
```

2) : Copying over all the necessary files into the root of my linux tree:

```
$ cp /scratch/spring2015/files/config-3.14.26-yocto-qemu ./.config
$ cp /scratch/spring2015/files/bzImage-qemux86.bin ./
$ cp /scratch/spring2015/files/core-image-lsb-sdk-qemux86.ext3 ./
```

3) : Building the kernel:

```
$ make -j4 all
```

4) : Writing a run script:

```
#!/bin/bash

source /scratch/opt/environment-setup-i586-poky-linux

qemu-system-i386 -gdb tcp::5618 -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"
```

5) : Running the script for the first time:

```
$ chmod u+x run
$ ./run
```

6) : Creating the gdb initializer script:

```
target remote :5618
symbol-file linux-yocto-3.14/vmlinux
```

7) : Connecting gdb from another shell:

```
$ gdb
```

8) : After typing continue in the gdb instance, I was able to successfully login with the credentials of root.

B. Qemu CLI Flags

-gdb tcp::5618 This flag will tell Qemu to open a gdb server on the following device. We specify to a reserved tcp port.
-S This flag instructs Qemu to not start the CPU at start up and to wait for a continue from the device monitor.
-nographic Normally Qemu displays output to VGA. With this flag it will bypass that entirely and spin up a headless command line application.
-kernel bzImage-qemux86.bin Specifies the particular kernel to use.
-drive file=core-image-lsb-sdk-qemux86.ext3,if=virtio This flag specifies the drive to use, with some following options. The file option defines a disk image and the if option defines the type of interface the device is connected to.
-enable-kvm This flag enables full KVM (Kernel-based Virtual Machine) support.
-net none Instructs the VM that no network devices should be configured.
-usb Enables the USB drivers.
-localtime Sets the time to the localtime of the calling machine.
–no-reboot Exits rather than rebooting.
–append "root=/dev/vda rw console=ttyS0 debug" Sends command line arguments to the kernel.

C. Concurrency

1) Main Point of Assignment: The main point of this assignment was to refresh our skills in C and introduce basic concepts of concurrent programming. Having both threads running with opposite purposes operating on a shared resource forces the programmer to start thinking with such a mindset. As Kevin hinted at in lecture, down the line we'll have more challenging assignments both in and out of class that are focused on concurrency in design.

2) Approach: My approach for this assignment was fairly simple. As a general practice for approaching programming assignments I like to lay the foundation/pseudo code out and then develop discrete chunks. For a clean start, I initially set up pthreads and got both my 'consumer' and 'producer' functions to run simultaneously and print to stdout. After making sure my threads were operating correctly, I then introduced the shared buffer and the mutex lock stored in a struct.

It was semi-challenging to make sure the producer wouldn't go over the size limit and the consumer wouldn't try to pop until it was time. After fixing a few bugs, I ironed out the shared resource for the two threads and turned my sights onto the in-line assembly code. The in-line assembly was for checking the cpuid output for the 30th bit set in the ecx register, which told the program whether or not it could run the rdrand instruction. If it could, then it would execute more assembly, which made use of the instruction and then generated a random number within the confines of the modulus operation.

3) Ensuring solution was correct: I was able to make sure my solution was correct through a couple different methods. Firstly, I tested the sleep times supplied to both the producer and consumer threads. This meant I would crank up the speed on each one individually to see how my program handled the heightened activity. When the consumer slept for 0 seconds it would quickly pop off whatever the producer had just supplied to the shared buffer. Conversely, when the producer slept for 0 seconds it would fill the buffer to the max of 32 items and then patiently block until the Consumer was able to pop the top item.

Outside of increasing the stress levels of the program I did a few other minor tests. Since I prefer to develop locally, I had to test my code on os-class. This ended up being semi necessary, because I needed to check for whether or not my cpu could use RDRAND should pass on my workstation and fail on os-class. I also kept tabs on my program by using extensive print statements to stdout that would let me make sure the program acted as expected.

4) What was learned: Coming into this class I didn't have a very strong understanding of concurrency or the tools that made it possible (e.g. pthreads). This assignment helped me refresh some of my lost knowledge from operating systems 1, taught me about concurrency, and helped me gain better insight on how the class will go as a whole. Outside of the concurrency assignment, building the Linux kernel was also very helpful. At first the lengthy commands and complex set up served to do little more than intimidate me, but after being required to describe the flags and the steps it ended being demystified greatly.

D. Version Control Log

Commit	Author	Description
d871807	Ty Skelton	initial commit
58fcc8c	Ty Skelton	clean master branch to build from for different devices
fd268d2	Ty Skelton	cleans up repository structure and files
4d9d85e	Ty Skelton	adds config
670cd9a	Ty Skelton	reorganizes file structure:
f45c406	Ty Skelton	Merge pull request 1 from skeltont/os-class
54d3081	Ty Skelton	adds first half of assignment to write-up
c6f7bee	Ty Skelton	adds report, concurrency assignment1
a15697e	Ty Skelton	fixes rdrand check failing on os-class and adds makefile for concurrency
9cd4722	Ty Skelton	adds output style to concurrency, fixes make file for writeup
10ecc5f	Ty Skelton	fixes rdrand capability checking

E. Work Log

Date	**Description**
March 29	Created private repository on GitHub and began setting up environment for development.
March 31	Successfully built kernel and attached it to GDB after creating some cool config files.
April 4	Created LaTex document and wrote first half of write-up regarding the kernel build (A & B).
April 5	Started working on concurrency assignment and was able to achieve base functionality.
April 7	Finished concurrency assignment, remaining work was to do in-line assembly. Added color flair.
April 8	Finished writing up concurrency assignment and verified all make files worked on os-class.