HOMEWORK 1

CS 444 Spring 2017 Team 11-06

Alex Wood, Miles Van de Wetering, Arman Hastings

Abstract

This is the report for group 11-06's homework 1. This document covers all required writings for homework one. In this assignment our team setup our shared workspace on the os-class sever, setup a git repository, and completed a concurrency assignment using the C language. This assignment was completed as a group effort working both in person and remotely.

CONTENTS

I. Assignment Description

The first homework is made up of two parts, the first build and run of the YOCTO kernel on a QEMU virtual machine, and the second a concurrency exercise. Theses tasks were to be completed by teams of three students working together.

II. Yocto Kernel build and QEMU VM

These commands were entered in the tcsh shell on the os-calss.engr.oregonstate.edu in order to pull the desired repository, create the appropriate directory structure, to build the kernel, and to launch the VM with our kernel build.

```
mkdir 11-06
chmod 777
git clone git://git.yoctoproject.org/linux-yocto-3.14 to get the yocto build
chmd 777
git checkout 'v3.14.26'
cp .config /scratch/spring2017/files/config3.14.26-yocto-qemu .config
make menuconfig
cp /scratch/opt/environment-setup-i586-poky-linux setup\_env
source setup\_env
make -j4 all
```

Once our Kernel was built, we needed to test it by running it on our virtual machine.The command for launching the VM and our kernel was supplied to us in the assignment. The following command was used:

```
qemu-system-i386 -gdb tcp::6606 -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".
```

- ”qemu-system-i386” - This is the QEMU invocation command. According to the documentation the expected format of the invocation is ”qemu-syste-i386 [options] [disk_image]” [1]
- Option 1 ”-gdb tcp::6606 -S” - This option is to enable gdb(gdb is the ”GNU Debugger”) on the the specified device. The device in this case is a TCP port. The port for our team was determined by using the base port umber of 5500, as instructed, plut the value of our team number. Our team is 1106, this gives us the port number 6606. Fianlly the -S switch tells QEMU to not start teh CPU until it is told to, by the gdb console. [1]
- Option 2 ”-nographic” - This option simply tells QEMU to run in command line mode(No Graphical User Interface).[1]
- Option 3 ”-kernel bzImage-qemux86.bin -drive file=core-image-lsb-sdk-qemux86.ext3” - Indicates to QEMU that a bzimage will be loaded and indicates an open file pointer for the file to load. [1]
- Option 4 ”if=virtio -enable-kvm” - This flag tells QEMU that if the virtio is set, to enable the KVM(Keyboard, Video and Mouse control) for the endpoint. [1]
- Option 5 ”-net none” - This flag indicates that the QEMU instance does not need to enable network support for the VM[1]
- Option 6 ”-usb” - This flag enables the VM’s USB driver.[1]
- Option 7 ”-localtime” - Sets the VM to use local system time[1]
- Option 8 ”–no-reboot” - Tells QEMU to exit instead of rebooting[1]

- Option 9 ”–append ”root=/dev/vda rw console=ttyS0 debug”.” - The This tells QEMU to redirect the serial port adn QEMU monitor to the console.[1]

Once the virtual machine was running, we needed to connect to it with GDB in order to continue the halted process. We opened a new shell, connected to os-class, and ran:

```
gdb
target remote localhost:6606
continue
```

then the qemu shell resumes its operations. We login as root, and use the uname command to display the version number.

III. Concurrency 1: The Producer-Consumer Problem

The producer-consumer problem was the focus of the first concurrency assignment. In the assignment teams were asked to create a program in which p-threads are used. One thread to produce random numbers in a shared memory space, then a second thread that reads memory space to consume that value. These threads needed to prevent the other thread from accessing the memory while one thread was reading (and consuming) or the other was writing to the shared memory space. The consumer would use the random number predetermined by the producer (depending on system the producer uses rdrand if present or an implementation of the Mersenne Twister discussed later) to determine the time it would take before it consumed a character from the shared memory space (between 3 and 7 seconds). The Producers job was to generate a random number using one of the two prescribed methods and to block the memory space so as to write the new value between 0 and 10. The function rdrand is widely supported, but not always, thus we were asked to implement both methods. This is accomplished by testing for a macro ”__i386__” to indicate weather or not the rdrand function would be present. When this macro is undefined, we implement random numbers via the Mersenne Twister algorithm. According to Wikipedia it is the most widely used general purpose pseudo random number generator.

Answers to questions on assignment.

1) The main point of the concurrency assignment was to get the students thinking about creating threads and managing the use of shared memory space. The assignment also aims to get the students to be ready to use unfamiliar tools and methods to account for differences between system architectures.
2) Our approach to the problem was first to take some time to understand what was being asked, and to fill in any gaps in our knowledge in how to accomplish the task. After spending some time independently working on the problem, we met as a team on Google Hangouts to hammer out a solution. The instructions limited us to the use of C and ASM to create a solution. We created a basic console application with one source file and one header file. The Mersenne Twister algorithm we used was from https://en.wikipedia.org/wiki/Mersenne_Twister. With a little modification this algorithm worked well within our program.The producer and consumer were both ”void *” types and took ”void *” as arguments. In order to fulfill the synchronization requirements of the exercise we used the C mutex to control access to shared resources.
3) The testing phase of this project was relatively limited due to the lack of interaction with the application. To figure out how to use the Mersenne Twister implementation, we wrote a small program to initialize, seed, and output values from the PRNG. Once this was completed, we used the modulo operation to restrict the output of the PRNG to the range we needed in each instance. During the implementation we used console output to indicate when each process

did its tasks, and the results of those tasks. The output of the program matched the expected output of the program, thus validating functionality.

4) We learned a lot about the QEMU invocation command, and about p-threads. This was new information for some of us, and familiar to others. With a little research, all team members had a good grasp of what we did and why we did it. We learned that we need to learn how to manage permissions better in Linux for our group folder.

IV. Worklog

Who	What	When
Alex, Arman, Miles	Worked on getting our group folder setup, pulling the YOCTO repo, creating our repo, building the kernel and running the QEMU VM to test.	Tuesday April 11, 11-1150am
Alex	Investigated Mersenne Twister and how to use it within a C program	Friday April 14, Off and on through-out the day
Miles	Alex wrote the concurrency functions and implemented the Mersenne Twister algorithm from Wikipedia	Sunday April 16, 10am-12pm
Alex	Modified the rdrand check to use the example from the website. This new method checks macros and the cpuid value to determin if rdrand can be used	Friday April 21, 7-8am

References

[1] "Qemu emulator user documentation." http://download.qemu-project.org/qemu-doc.html#QEMU-PC-System-emulator. Accessed: 4/13/2017.