Project 1: Getting Acquainted

Christian Armatas and Mihai Dan

Abstract

This document contains a log of commands used to perform the requested actions, an explanation of every flag listed in the qemu command-line, answers to the concurrency problem, a version control log, and a work log. The solution to Project 1 can be depicted by consulting the formally mentioned sections. The purpose of this project was to get acquainted with the tools necessary, such as the kernel and virtual machine, as well as to familiarize us with multithreading and the associated challenges.

COMMAND LOG

The command line arguments used to complete Homework 1. Actions performed include setting up the virtual machine, building the kernel, and starting the VM using the new kernel.

- ssh armatasc@os-class.engr.oregonstate.edu
- cd /scratch/fall2016
- mkdir cs444-017
- cd cs444-017
- git clone git://git.yoctoproject.org/linux-yocto-3.14
- cp environment-setup-i586-poky-linux /scratch/fall2016/cs444-017
- cp core-image-lsb-sdk-qemux86.ext3 config-3.14.26-yocto-qemu /scratch/fall2016/cs444-017
- cp bzImage-qemux86.bin config-3.14.26-yocto-qemu /scratch/fall2016/cs444-017
- source /scratch/opt/environment-setup-i586-poky-linux
- qemu-system-i386 -gdb tcp::5517 -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”
- gdb
  - target remote :5517
  - continue
- cp /scratch/fall2016/files/config-3.14.26-yocto-qemu /scratch/fall2016/cs444-017/yocto/.config
- make -j4 all
- source ./environment-setup-i586-poky-linux
- qemu-system-i386 -gdb tcp::5517 -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”
- gdb
  - file vmlinux
  - target remote :5517
  - continue

Command used to give Mihai permissions to the folder.

- ./`acl_open` -r /scratch/fall2016/cs444-017 danm

QEMU COMMAND-LINE FLAGS

The command line flags for the QEMU command are listed below, along with their functionality.

- -gdb
  - launches program in debugging mode
- tcp::5517
  - specifies the TCP port number to the one assigned in class
- -S
  - does not launch CPU on start-up, allows for manual launch
- -nographic
  - disables graphical output
- -kernel bzImage-qemux86.bin
  - uses specified file as the kernel image
- -drive file=core-image-lsb-sdk-qemux86.ext3,if=virtio
  - specifies desired drive file and decides which type of interface the drive is connected on
- -enable-kvm
  - enables full KVM Virtualization support
- -net none
  - override default configuration and indicate that no network devices should be configured
- -usb

  - enables USB driver
- -localtime
  - sets time to local time
- -no-reboot
  - exits instead of rebooting
- –append ”root=/dev/vda rw console=ttyS0 debug”
  - command line parameters given to the kernel

VERSION CONTROL LOG

Created the main source control directory on ”os-class /scratch/fall2016”	October 3, 2016 14:37:02
Added the yocto envirnment dir, config file, env-setup file, and kernel to main dir	October 5, 2016 16:21:54
Created the ”vmlinux” kernel in our main dir and confirmed it ran	October 10, 2016 10:30:15
Completed the ”rdrand” vs ”mtrand” code in C and ASM; Implemented the functions	October 10, 2016 22:10:28
Created the Makefile	October 11, 2016 18:12:36
Finalized comments in the ”concurrencyProb.c” file	October 11, 2016 19:42:06
Completed the final copy of ”concurrencyProb.c”	October 11, 2016 21:02:17

WORK LOG

- Sun, 10/2
  - Took a look at the assignment and read through the requirements
  - Broke the assignment into a few main sections: Setup, C/ASM code, and write-up
- Mon, 10/3
  - Christian re-downloaded and -re-configured his putty terminal + file transfer software on his new PC
  - Re-familiarized ourselves with the terminal, os-class server, and bash
  - Created our ”/scratch/fall2016/cs444-017” as our source control repo on os-class...
  - Didn’t quite understand how to get the yoctoproject file, switch the tag, or get the config/kernel/drive files
- Wed, 10/5
  - Used the command ”git clone git://git.yoctoproject.org/linux-yocto-3.14” to get a copy of the environment
  - Navigated to the location of the environment configuration script: ”cp environment-setup-i586-poky-linux /scratch/fall2016/cs444-017”
  - Navigated to the location of the starting kernel and drive file: ”cp bzImage-qemux86.bin core-image-lsb-sdk-qemux86.ext3 config-3.14.26-yocto-qemu /scratch/fall2016/cs444-017”
  - Command to open QeMu:
  - This launches qemu in debug mode w/ a halted CPU
  - To run the VM, connect gbd (from yocto: GDB) to remote target port 5517; This causes the OS to boot
  - Still need to buid instance of kernel and ensure it boots in the VM
- Sat, 10/8
  - Mihai looked into the concurrency exercise and did some seperate research regarding the solution
  - Christian tried to run the ”qemu-system-i386” command but ran into complications
- Sun, 10/9
  - Looked into the ”qemu-system-i386” command and realized we were only able to call ”qemu-img” and ”qemu-io”
  - Reading over the ”You must source the appropriate file...” line, we relalized that meant the literal command, SOURCE
  - ’source /scratch/opt/environment-setup-i586-poky-linux’ -¿ This gave us the ”Qemu” commands
  - ”qemu-system-i386 ...” now gives us a frozen terminal (it appeared). Though this is the ”halted CPU” mentioned in the assignment; Now we need to connect the GDB
  - Inside the yocto environment, ”make menuconfig” will bring up the kernel configuration menu
  - Mihai ran ’gdb’ inside the yocto environment (while the VM was in debug mode)...
  - How to target the same port via yocto gdb: ’target remote :5517’
  - ”... and continue the process” meant the literal command ”continue” after ’target remote’, inside gdb
  - AND WE ARE IN! To login, we used ”root” with no password

  - Can't do much inside here yet - Need to figure out how to copy "config-..." script to our "SRC ROOT"
  - Shutdown QEMU: 'shutdown -H now
- Mon, 10,10
  - Christian figured out how to copy the config file: 'cp /scratch/fall2016/files/config-3.14.26-yocto-qemu /scratch/fall2016/cs444-017/linux-yocto-3.14/.config' This prompted "Overwrite .config?" - Y - Enter
  - After about 5 minutes (i literally got coffee, as recommended), the kernel "vmlinux" was created! 'cp vmlinux /scratch/fall2016/cs444-017' - Enter
  - NOW, from the gdb, 'file vmlinux' will read from the built kernel: 'target remote :5517' will connect to our port - 'continue' - This runs our kernel in the VM on port 5517!
  - Mihai began the "rand" and ASM research (what we needed to do in ASM) and Christian began coding the concurrency
  - Using ASM, we need to check if "rdrand" or "mtrand" is acceptable for use... Mihai completed the rand functions
  - The minimal concurrency model (w/ no wait times on "work") compiled
  - Adding producer/consumer conditions and wait times caused only the production, until the buffer was full
  - After the buffer fills, the consumer will begin consuming all 32 buffers...
  - Will not have time to complete the mess of issues within the concurrency problem tonight...
- Tues, 10/11
  - We implemented the rand switching and ASM code into the "concurrencyProb.c"
  - Mihai completed the TeX document as Christian debugged the concurrency code...
  - We went back to a previous version of our code because it was more simple
  - After walking through, checking over, and changing conditions for sleep/wait/etc., the code worked! Now, the producer will produce structs into the buffer as the consumption occurs (including random wait times)
  - Created a simple make file, to where "make all" creates the executable for "concurrencyProb.c"
  - Completed some finalizing and touching up work (removing debugging printf's, adding helpful comments, etc.)
  - Completed the assignment!

Questions

1) What do you think the main point of this assignment is?
   - The purpose of this assignment is to familiarize us with multithreading and the associated challenges. Designing and creating a solution for this problem requires parallel thinking and research on synchronization. The problem presented consisted of producer and consumer threads which added and removed items to a limited size buffer. To a void conflict, the buffer had to remain locked whenever a thread was altering it.
2) How did you personally approach the problem? Design decisions, algorithm, etc.
   - In class and from the assignment, we learned this problem would require the use of multiple threads, structs, and random numbers (generated in ASM). First, we broke this down into a few portions, the C code and the ASM code. The ASM will simply check/calculate a random value from "rdrand" or "mtrand", depending on which function can run. The C code will need to have simultaneous interactions between the producers, consumers, and the buffer containing all of this data. We designed a sudo-code outline displaying where we need to lock/unlock the mutex, signal the producer/consumer conditions, wait on producer/consumer conditions, and sleep in the main, producer, and consumer functions. To check how the buffer was filling up, when the producer created a struct, and when the consumer consumed a struct, we used print statements.
3) How did you ensure your solution was correct? Testing details, for instance.
   - In order to check the correctness of our algorithm, we checked the output from the program. Print statements were placed before and every after production or consumption, depicting the buffer size, sleep time, and randomly generated number.
4) What did you learn?
   - One of the biggest take-aways from this project is learning about the importance of locking the buffer when it is in use. This allows for the buffer to only be modified by one thread at a time, avoiding any conflicts. The project also allowed for a reminder on how to use threads in C, as well as introducing how to lock using Mutex. Another aspect we found interesting is being able to use assembly code inside of C code.