CS444 Assignment 1

Kernel, Concurrency and Latex

Alec Zitzelberger and Chase Coltman

Date October 9, 2017

Assignment 1

iqemu Steps Taken:

1. Open up two shells, call them Shell A and Shell B.
2. Git clone git://git.yoctoproject.org/linux-yocto-3.19
3. Change extension to 3.19.2
4. Be inside the linux-yocto-3.19.2 directory
5. Mv ../../../files/bzImage-qemux86.bin .
6. Mv ../../../files/config-3.19.2-yocto-standard .
7. Mv ../../../files/environment-setup-i586-poky-linux .
8. Enter bash for both Shell A and B
9. Shell A: source environment-setup-i586-poky-linux
10. Shell A: qemu-system-i386 -gdb tcp::5514 -S -nographic -kernel bzImage-qemux86.bin -drive file=core-image-lsb-sdk-qemux86.ext4,if=virtio -enable-kvm -net none -usb -localtime –no-reboot –append ”root=/dev/vda rw console=ttyS0 debug”.
11. Shell B: source environment-setup-i586-poky-linux
12. Shell B: $GDB
13. Shell B: Target remote :5514, then enter continue and hit enter
14. Shell A: enter root and hit enter. Enter reboot and hit enter when ready to quit
15. Hey it worked! Now to make our own
16. Mv config-3.19.2-yocto-standard .config
17. Make -j4 all
18. Enter in make menuconfig
19. Go to general and change the kernel info to group 14 homework 1
20. Find bzImage in the folder arch/x86/boot/bzImage
21. Cp arch/x86/boot/bzImage /scratch/fall2017/14/linux-yocto-3.19.2
22. Shell A: source environment-setup-i586-poky-linux
23. Shell A: qemu-system-i386 -gdb tcp::5514 -S -nographic -kernel bzImage -drive file=core-image-lsb-sdk-qemux86.ext4,if=virtio -enable-kvm -net none -usb -localtime –no-reboot –append ”root=/dev/vda rw console=ttyS0 debug”.
24. Shell B: source environment-setup-i586-poky-linux
25. Shell B: $GDB

26. Shell B: Target remote :5514
27. Shell B: Continue
28. Shell A: Root
29. Logged into VM and ready to go
30. Uname -a prints out our group number 14, homework 1

iqemu flags

1. -gdb tcp::5514
   (a) This opens a gdbserver on TCP port 5514.
2. -S
   (a) This means Do not start CPU at startup.
3. -nographic
   (a) Disables graphical output so that Qemu is a simple command line application.
4. -kernel bzImage
   (a) This command says to use the bzImage as the kernel image, which can either be a linux kernel or in multiboot format.
5. -drive
   (a) file=core-image-lsb-sdk-qemux86.ext
      i. This option defines which disk image to use with this drive. For this situation, the disk image would be core-image-lsb-sdk-qemux86.ext
   (b) if=virtio
      i. This option defines on which type of interface is connected.
6. -enable-kvm
   (a) Enable KVM full virtualization support. This option is only available if KVM support is enabled when compiling
7. -net-none
   (a) Indicate that no network devices should be configured. It is used to override the default configuration which is activated if no -net options are provided
8. -usb
   (a) Simply enables the USB driver
9. -locatime
   (a) This is used to let the RTC start at the current local time
10. –no-reboot

(a) Exit instead of rebooting

11. –append root=/dev/vda rw console=ttyS0 debug

    (a) This uses root=/dev/vda rw console=ttyS0 debug as the kernel command line

Concurrency Questions:

1. The point of this assignment has been to implement a multithreaded solution in order to better understand parallel programming. In addition, it has helped to familiarize the class with using the rdrand functionality on intel CPUs to produce true random numbers.

2. I approached the problem by dividing up the functionality of the two thread types into two different functions. I implemented new code in waves. First was the threading, then mutex locking, semaphore blocking, etc. At each step, I tested to make sure that the layers were correctly coded. The algorithm was simple. Each function that the threads worked contained an infinite while loop that checked for the lock availability and did work based on the values detected in the global variables. If the buffer was empty or full, the while loop contained code to do the appropriate blocking. When running the code, users enter the number of consumers and producers. This makes our solution a multiple consumer and producer solution.

3. To ensure the correctness, both myself and my partner tested the code at each stage of development. To be able to see the correctness, the program is littered with print statements that report what is going on for each thread. These print statements report the currently handled item number, the time spent working on it, and the current buffer contents.

4. Most of what we learned involved just relearning things from OS1 like how to multi-thread, mutex lock, and using semaphores. The new content was primarily how to use the Twister number generator and the rdrand assembly code. The assembly code proved to be somewhat of a challenge when implementing in C, but it was good to learn.

Work.log for Chase Coltman

1. Began working on installing Kernel and attempting to get it run. Worked through instructions until I got to

    (a) Steps Taken:

        i. Created the repo and gave permissions for Alec to be able to enter and make changes

        ii. Attempted to move files to the correct location

    (b) Date - 9/30/2017

    (c) Time - About 2 hours

    (d) End-point - Frustrated and not sure what I am doing wrong

    (e) Finished - No, the Kernel had a bunch of error so I waited until next class to ask a peer

2. Attempted to work some more on getting Kernel to work

    (a) Steps Taken:

        i. Moved all the files into the correct directory

ii. Seems to be working but not sure as it freezes when I run the code

(b) Date - 10/3/2017

(c) Time - About 2 hours

(d) End-point - Compiles and runs but freezes at the end

(e) Finished - No, Kernel is compiling but not sure on next step

3. Completed all the requirements for the Kernel to run

   (a) Steps Taken:

      i. Finally got it working. Everything was actually working on the 3rd when I attempted to do it before, just had to open another shell client.

      ii. Finished the implementation and did the write-up with all the steps that I took to figure it out

   (b) Date - 10/5/2017

   (c) Time - About 3-4 hours

   (d) End-point - Runs and I am able to enter the kernel and VM with both the bzImage file and the one we make when we run make -j4 all

   (e) Finished - Yes, It worked and I followed all the steps to make sure

4. Finished writing up what all the flags do

   (a) Steps Taken:

      i. Googled qemu i386 flags to give me a detailed readme

      ii. Found this link:http://www.tin.org/bin/man.cgisection=1&topic=qemu-system-i386

      iii. Wrote in what each flag did and learned what exactly we do when we run that long command

   (b) Date - 10/7/2017

   (c) Time - About 1 hour

   (d) End-point - Finished

   (e) Finished - Yes, Reason for it being approximately a time is because I was at work and actually have no idea how long it took

5. Learning how to LaTeX

   (a) Steps Taken:

      i. Watched the online tutorial

      ii. Copy-pasted write up from Google Doc and formatted for LaTeX

   (b) Date - 10/9/2017

   (c) Time - About 2 hours

   (d) End-point - Finished on my end, sending to Alec for his log and final product

   (e) Finished - No

Work.log for Alec Zitzelberger

1. Began working on the Concurrency Solution

(a) Steps Taken:
    i. Researched how to use pthreading and mutexes in C
    ii. Used the hellopthread.c code as a baseline for development as it contained a lot of syntax info
(b) Date - 10/5/2017
(c) Time - About 2 hours
(d) End-point - Had a decent understanding of the code libraries I would need for pthreading
(e) Finished - No, This would take more time to create the first alpha

2. Finished working on the first version of the Concurrency Solution code
    (a) Steps Taken:
        i. Continued to implement what I had learned during my research
        ii. Finished writing the code for simple pthreading behavior and tested it locally
        iii. Pushed the V1 to the MAIN repo
    (b) Date - 10/6/2017
    (c) Time - About 1 hours
    (d) End-point - Had a basic version that chaotically did work in multiple threads
    (e) Finished - Yes, completed the work phase I had begun the day before
3. Worked on the second version of the Concurrency Solution code
    (a) Steps Taken:
        i. Implemented mutex locking
        ii. Tested the code that now locked the threads correctly
        iii. Pushed V2 to the MAIN repo
    (b) Date - 10/6/2017
    (c) Time - About 1 hours
    (d) End-point - Had code that followed the locking criteria of the solution
    (e) Finished - Yes, completed the phase of coding mutexes
4. Worked on the third version of the Concurrency Solution code
    (a) Steps Taken:
        i. Researched semaphore signalling for C code
        ii. Implemented semaphore blocking
        iii. Tested the code that now blocked the threads correctly given the state of the buffer
        iv. Pushed V3 to the MAIN repo
    (b) Date - 10/6/2017
    (c) Time - About 1 hours
    (d) End-point - Had code that followed the blocking criteria of the solution
    (e) Finished - Yes, completed the phase of coding conditional thread blocking
5. Worked on the fourth version of the Concurrency Solution code
    (a) Steps Taken:

Table 1: Version Control

Date	Action	Version
9:00_Oct 5, 2017	Created Project with basic threading in local repo.	V1
2:33_Oct 6, 2017	Pushed Project to MAIN repo.	V1
3:11_Oct 6, 2017	Added mutex locking in local repo.	V2
3:16_Oct 6, 2017	Pushed Project to MAIN repo.	V2
4:35_Oct 6, 2017	Added semaphore blocking in local repo.	V3
4:42_Oct 6, 2017	Pushed Project to MAIN repo.	V3
2:03_Oct 8, 2017	Added rdrand and Mersenne in local repo.	V4
5:24_Oct 8, 2017	Pushed Project to MAIN repo.	V4
10:02_Oct 9, 2017	Replaced semaphore with pthread_wait conditionals in local repo.	V5
10:10_Oct 9, 2017	Pushed Project to MAIN repo.	V5

i. Researched rdrand and Mersenne Twister random number generation for C code
ii. Implemented random number generation
iii. Tested the code that now generated random numbers locally
iv. Pushed V4 to the MAIN repo

(b) Date - 10/8/2017

(c) Time - About 2 hours

(d) End-point - Had code that followed the random number criteria of the solution

(e) Finished - Yes, completed the phase of coding random number generation

6. Worked on the fifth version of the Concurrency Solution code

(a) Steps Taken:

i. Researched pthread wait blocking for C code
ii. Implemented pthread wait blocking
iii. Tested the code that now blocked the threads correctly given the state of the buffer
iv. Pushed V5 to the MAIN repo

(b) Date - 10/9/2017

(c) Time - About 1 hours

(d) End-point - Had code that followed the blocking criteria of the solution

(e) Finished - Yes, completed the phase of coding conditional thread blocking

textbfVersion Control Table