HW1/Concurrency 1 Write Up
CS 444 Spring 2017

Brandon Dring, William Buffum, Samuel Jacobs

April 21, 2017

Abstract

A simple C based solution to the producer and consumer problem. This program requires that you pass in a command line argument number. Whatever number that is passed in at the command line, is the number of respective consumer and producer threads. So I.E (hw1 5) will produce 5 producer threads and 5 consumer threads. Each producer that is generated will sleep for a random number generated by rdrand, or the mersenne twister depending on the user computer. Then it will create some more random numbers to be added to the struct, with an arbitrary value, and the sleep number that the element in the struct array posesses. So when a consumer, consumes that element in the struct array, they sleep for the number found in the struct element. And they also print out the other arbitrary random number generated. Also, with a buffer size of 32, if there are 32 elements to the buffer, the producer will wait until the consumer takes a number out. And conversely, if there are 0 elements in the buffer, a consumer will wait til the producers produce something to take out.

1 COMMANDS

A log of commands used to perform the requested actions, as well as a write-up of your concurrency solution.:

- mkdir /scratch/spring2017/11-08; cd /scratch/spring2017/11-08
- git clone git://git.yoctoproject.org/linux-yocto-3.14
- cp /scratch/opt/environment-setup-i586-poky-linux .
- cd linux-yocto-3.14; source ../environment-setup-i586-poky-linux
- cp /scratch/spring2017/files/config-3.14.26-yocto-qemu ./.config
- make -j4
- qemu-system-i386 -gdb tcp::5608 -S -nographic -kernel arch/i386/boot/bzImage -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".
- (NEWSHELL) $GDB
- target remote :5608

2 FLAGS

An explanation of each and every flag in the listed qemu command-line

- *-gdb tcp:5608*
  The -gdb option with tcp:5608 allows us to target the process in gdb and manually continue execution.
- *-S*
  The -S flag stops the cpu from from starting when we boot up the VM. This ensures that we have to manually continue the process from gdb.
- *-nographic*
  -nographic allows user to disable graphical output of qemu so that we only work from the commandline.
- *-kernel*
  This flag specifies where the bzImage to boot resides. The bzImage is a compressed kernel image.
- *-drive*
  This flag defines a new drive to use. We are using file=core-image-lsb-sdk-qemux86.ext3 to specify the disk image for this drive. We also use the if=virtio to define the type of interface to which the drive connects.
- *-enable-kvm*
  This flag signifies that we have KVM full virtualization support. It allows us to have private virtualized hardware.
- *-net*
  This flag makes sure the no network devices are configured by overriding the default configuration.
- *-usb*
  This flag enables the USB drivers.
- *-localtime*
  This flag is necessary to get the correct data and time.
- *–no-reboot*
  Instead of rebooting the VM when you run the reboot command, it shuts down the VM.
- *–append*
  Uses root=/dev/vda rw console=ttyS) debug as the command line for the kernel.

3 Concurrency Questions

What do you think the main point of this assignment is?

The point of the assignment I think is just a warm up on how to think and implement something concurrently in C. Specifically using the pthreads library, on a *nix based system. Then thinking about how a program can run in parallel, with one thread producing a job to do, and another thread consuming the job that was posted. Along with synchronizing program resources between two threads, so that only one thread can access the shared resource at a time. There is also a touch at writing some assembly code to use the rdrand command, and storing values in registers. The assignment is really a warm up into getting into a parallel thinking mindset. Apart from just the concurrency exercise, its also gives experience into writing a Latex document when most people just use word.

How did you personally approach the problem? Design decisions, algorithm, etc.

I personally started with just trying to create the producer, and the resource they share (the struct in this case). Then create the producer function, so that when I spawn off a thread, it can be pointed to a function that uses a mutex and adds to the shared struct value. From there my side was done, my teammate then took over and created the consumer function. So that when a consumer thread is created they can be pointed to a function that sets mutexes consumes the resource. Then my last teammate took over in doing the rdrand implementation and writing the assembly code, and taking in command line arguments that spawns off a passed in value of threads.

How did you ensure your solution was correct? Testing details, for instance.

To test there was multiple test cases that had to be covered. To start, there had to be the base case of just 2 threads running, one a producer and one a consumer. Both accessing the same shared resource, and synchronizing them with mutexes. Breaking it down further, there was testing solely to make sure that the consumer thread generates the random number efficiently, and places them into the struct, all the way up to the limit, in this case it was 32 entries. Then vice versa, there was testing to make sure that the consumer thread can pull things out of the shared resource. After that there was testing to make sure that a producer thread doesnt add to the buffer if it has 32 entries, and that the consumer thread doesnt pull from a buffer with 0 entities. After that was taking in command line arguments for an arbitrary number of threads to use. We had to make sure that creating more than one producer and one consumer wouldnt throw the system into deadlock as well. All testing was done via printf, every time a producer/consumer touches the shared struct it logs what its values are and what the thread id is that is touching the resource at the time.

What did you learn?

We as a group learned how to write assembly code in a C file. Along with making a Latex file, and how to compile a Latex file with a Makefile. We also brushed up on our skills of synchronizing multiple threads we create.

4 Version Control History

Detail	Author	Description
1040abe	El-Dringo-Brannde	Initial commit
4bbf811	El-Dringo-Brannde	Started on Producer
840842e	El-Dringo-Brannde	Test
dfd3384	El-Dringo-Brannde	Producer is now.. Producing
0c3e4d1	El-Dringo-Brannde	fixed a bug
63a39d9	El-Dringo-Brannde	Cleaned up code
6e508ca	El-Dringo-Brannde	Init
6d03080	samtjacobs	Added rdrand, consumre
2ed6259	samtjacobs	Style changes
6cb351f	El-Dringo-Brannde	Turn in ready
14b1d30	El-Dringo-Brannde	Merge branch 'master' of https://github.com/El-Dringo-Brannde/CS444
b156e61	El-Dringo-Brannde	Real turn in ready
ea2bf27	WilliamBuffum	added commandline arguments for user to specify number of each thread
b34f487	El-Dringo-Brannde	Fixed negative index issue added some debug stuff
b6f0da0	samtjacobs	Added makefile
e8b356d	El-Dringo-Brannde	Fixed multithreading issue and code style changes

Work Log

- *4-13-2017*
  Brandon Dring started on instantiating producer

- *4-13-2017*
  Brandon Dring Producer thread now adds to the buffer struct and increments index

- *4-13-2017*
  Brandon Dring refactored the code to be more readable

- *4-13-2017*
  Brandon Dring Changed from sleeping if the buffer is full to setting a mutex condition variable

- *4-19-2017*
  Samuel Jacobs Implemented rdrand and the consumer threads to work appropriately

- *4-19-2017*
  Samuel Jacobs Refactored code to be more readable and changed the names of some variables, along with altering control flow

- *4-20-2017*
  William Buffum Implemented command line arguments to be able to dictate number of threads spawned

- *4-21-2017*
  Brandon Dring Fixed bug where consumers would pull items at a negative index.

- *4-21-2017*
  Brandon Dring Fixed multithreading bug where both consumer and producer would add and pull when they aren't supposed too.

- *4-21-2017*
  Brandon Dring Added a script that produces a git log of the VCS repo