CS 444 - Spring 2016 - Assignment 2

Cody Malick
malickc@oregonstate.edu

April 26, 2016

Abstract

Concurrency is an important issue when working with shared resources, especially resources being actively used by the kernel. The problem with sharing resources across threads is safely accessing the resources without causing a race condition. This problem is solved by using a mutex in order to lock down the resource until the thread is done with it's task, then releasing it for other threads. By using this process, we can create multithreaded programs that can safely access shared resources. Following are the details on how I approached this problem.

Contents

1 Concurrency Assignment 1

1.1 Assignment Purpose

I think this assignment was used a primer for working with the kernel, where multithreading and shared resource management are important. Being able to write programs that don't deadlock or cause race conditions is important when working with a kernel.

1.2 Problem Approach

The over assignment wasn't that difficult to conceptualize. A producer and a consumer were needed, and the synchronization construct could be a simple data structure. I used a queue structure for my buffer. The main reason was simplicity, first-in first-out made sense in this situation. I thought about using a stack, but that created a problem where the first message may never be reached once burried under new messages. I used a very simple layout for the buffer: an array of the message struct, and an int to track the current size of the buffer.

```
struct message {
  int number;
  int wait_period;
};

/*buffer is a basic queue, first in, first out implementation*/
struct buffer {
  struct message b[32];
  int size;
};
```

The queue has two mutator functions: push and pop. Push would put a new message on the queue, while pop would take the first message out, and move all following messages forward.
To handle the producer function, I spawn my threads using `pthread_create()` and handing the threads a producer function and consumer function. Once the threads enter these functions, they enter the infinite loops that are the assigned program. Here are the producer and consumer functions I used:

```
void *producer_function(void *ptr)
{
  /*cast void pointer to buffer pointer*/
  struct buffer *buff = (buffer *)ptr;
  struct message item;

  char prod[ ] = "Producer";

  while(1) {
      /*check if buffer is full*/
      if (buff->size < (sizeof(buff->b)/sizeof(struct message))) {
          pthread_mutex_lock(&mutex);
      /*UP, LO, CUP, CLO, PUP, and PLO, are constants declared to avoid magic numbers*/
          item.number = rand_gen(UP, LO);
          item.wait_period = rand_gen(CUP, CLO);

          buff_push(buff, item);

          print_column(prod, item.number, item.wait_period, buff->size);
          pthread_mutex_unlock(&mutex);

          sleep(rand_gen(PUP, PLO));
          }
      }
}
```

```
void *consumer_function(void *ptr)
{
  /*cast void pointer to buffer pointer*/
  struct buffer *buff = (buffer*)ptr;
  struct message item;

  char cons[ ] = "Consumer";

  int wait_time = 0;
  while(1) {

      if (buff->size > 0) {

          pthread_mutex_lock(&mutex);

          item = buff->b[0];
          buff_pop(buff);

          print_column(cons, item.number, item.wait_period, buff->size);
          pthread_mutex_unlock(&mutex);

          sleep(item.wait_period);
      }
  }
}
```

The producer and consumer functions each lock the shared buffer with `pthread_mutex_lock(&mutex)` and unlock with `pthread_mutex_unlock(&mutex)` when they're done accessing it. I decided to include the `printf()` statement within the mutex locks so that I could actively print out the status of the queue's size.
To handle the rdrand requirement, I took code from Intel's online guide to rdrand [1]. Using that code, I had to do a check in the main function and set a global flag, `int rdrand`, so that when my random function `rand_gen()` is called it knows which generator to use. If rdrand is not supported, it calls Mersenne Twister with the `genrand_int32()` function call. Here is my random generator function:

```
int rand_gen(int upr_bound, int lwr_bound)
{
  if (rdrand) {
      /*rdrand required unsigned 32 bit int*/
      uint32_t num = 0;
      if (rdrand32_step(&num))
          return abs((int)num) % upr_bound + lwr_bound;
  }
  /*abs to solve negative numbers from generator*/
  return abs((int)genrand_int32()) % upr_bound + lwr_bound;
}
```

Lastly, to show that program is operating properly, my output does the following: lists if it's the producer or consumer, what the randomly generated number is, what the wait time of that message is, and how many items are in the buffer after that production or consumption is complete.
Example:

```
Producer | Number:587 | Wait:  6 | buffer:  1 |
Consumer | Number:587 | Wait:  6 | buffer:  0 |
```

1.3 Testing

I did some manual testing to ensure that my pthread implementation was working. The first was to run the program multiple times and see that different printf statements printed at different times, a good indicator that the threads were in fact spawning and running properly.

The other manual test I did was to move my mutex locks outside my producer and consumer function infinite loops. When I did that, in the producer's case, the producer would never release the mutex on the buffer, and the consumer would never be able to access the buffer. In the opposite case, the producer would produce zero or one messages, put them in the buffer, and the consumer would then lock the buffer for the rest of the program runtime. This resulted in nothing happening as the producer couldn't put anything in the buffer, and the consumer would never consume.

1.4 What did I learn?

I learned quite a bit in this assignment! I learned how to use pthreads (as this was not covered in my OS I class), how to use mutex to control which thread has access to a resource, how to implement some basic inline assembly, and gave myself a basic refresher on the C language and its caveats.
I also learned the basics needed to build and implement a linux kernel, how to connect to and debug a Qemu VM, and how to use LaTeX!

2 Qemu Flags

The command we are told to use is the following:
`qemu-system-i386 -gdb tcp::5500086 -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"`
Here's the breakdown of the flags [2]:

`-gdb tcp::xxx`: Wait for gdb connection on device, in this case, tcp port xxx

`-S`: Do not start CPU at startup

`-nographic`: Disables graphical output, and redirects it to a virtual port

`-kernel bzImage-qemux96.bin`: Use specified file as the kernel image

`-drive`: Define a new drive

- `file=` Defines which disk image to use with the drive
- `if=` Defines which type of interface the drive is connected through. Types are ide, scsi, sd, mtd, floppy, pflash, virtio

`-enable-kvm` Enable KVM full virtualization support. KVM == Kernel-based Virtual Machine

`-net`: Creates a new network interface card and connects it to VLAN n

`-usb`: Enables USB driver

`-localtime`: Sets current machine time to current local time

`--no-reboot`: Exit instead of rebooting

`--append` Gives the kernel command line arguments

- `root=/dev/vda rw` Gives read/write access to the vda (Virtualization-Aware Disk)
- `console=ttyS0 debug` Directs the kernel to use serial port ttyS0 in debug mode

3 Work Log

Work log of time spent and on what:

`qemu`: April 3 - 4 hours

`concurrency, pthreads`: April 5 - 2 hours

`concurrency`: April 6 - 1 hours

`Latex`: April 7 - 2 hours

`pthreads termination problem`: April 8 - 2 hours

`mutex setup`: April 8 - 1 hour

`Linux Kernel Coding Standars, fixed code`: April 8 - 1 hour

`Twister setup and debugging`: April 8 - 3 hours

`mutex setup`: April 8 - 1 hour

`RdRand research and debugging`: April 9 2 hours

`Latex & Report`: April 10 - 8 hours – yes really –

Total time: 27 hours

4 Command Log

Here is my history for the Qemu part of the assignment:

```
455  git clone git://git.yoctoproject.org/linux-yocto-3.14
456  ls
457  cp linux-yocto-3.14/ /scratch/spring2016/cs444-086/
458  cp -r linux-yocto-3.14/ /scratch/spring2016/cs444-086/
459  cd /scratch/spring2016/cs444-086
460  ls
461  git remote
462  git origin
463  git --help
464  git config remote
465  git config origin
466  git config
467  git config --get-all
468  git config --local --get-all
469  git config --global --get-all
470  git status
471  git remote -v
472  git remote rm origin
473  git remote
474  git remote -v
475  git status
476  git pull
477  git push
478  git log
479  git log --color
480  source /scratch/opt/environment-setup-i586-poky-linux
481  cp /scratch/opt/environment-setup-i586-poky-linux .
482  ls
483  git add .
484  git commit -m "Copied env script to local repo"
485  git log
486  ls
487  cp /scratch/spring2016/files/bzImage-qemux86.bin .
488  cp /scratch/spring2016/files/core-image-lsb-sdk-qemux86.ext3 .
489  cp /scratch/spring2016/files/config-3.14.26-yocto-qemu .
490  ls
491  git add .
492  git status
```

```
493  git add bzImage-qemux86.bin
494  cat .gitignore
495  cat .gitignore | grep .bin
496  ls
497  mkdir vmfiles
498  mv bzImage-qemux86.bin vmfiles/
499  mv -t vmfiles/ core-image-lsb-sdk-qemux86.ext3
500  mv config-3.14.26-yocto-qemu vmfiles/
501  git add .
502  git status
503  git commit -m "added vm files"
504  printenv
505  cd vmfiles/
506  git checkout v3.14.26
507  ls
508  cd ..
509  ls
510  git branch
511  git checkout master
512  cat README
513  ls
514  rm .
515  rm ./*
516  rm -rf ./*
517  ls
518  cp ~/git/linux-yocto-3.14/ .
519  cp ~/git/linux-yocto-3.14/ . -r
520  ls
521  mv linux-yocto-3.14/* .
522  ls
523  rm linux-yocto-3.14/
524  cd linux-yocto-3.14/
525  ls
526  cd ..
527  rm -r linux-yocto-3.14/
528  cd linux-yocto-3.14/
529  ls -a
530  mv .git ../
531  mv .git ../ -rf
532  mv .git ../
533  cd ..
534  ls
535  rm -rf ./*
536  ls -a
537  rm .git -rf
538  rm .gitignore -rf
539  rm .mailmap -rf
540  ls
541  ls -a
542  cp ~/git/linux-yocto-3.14/* .
543  cp ~/git/linux-yocto-3.14/* . -r
544  ls -la
545  cd .git
546  ls ~/git/linux-yocto-3.14/
547  ls ~/git/linux-yocto-3.14/ -la
548  ls -la
549  cp ~/git/linux-yocto-3.14/.git . -r
```

```
550  cp ~/git/linux-yocto-3.14/.gitignore . -r
551  ls
552  git status
553  cp ~/git/linux-yocto-3.14/.mailmap .
554  git status
555  git checkout v3.14.26
556  git checkout master
557  git branch
558  cp /scratch/spring2016/files/* ./vmfiles/
559  mkdir vmfiles
560  cp /scratch/spring2016/files/* ./vmfiles/
561  ls vmfiles/
562  git branch -a
563  git ls-remote
564  git reset --hard
565  git status
566  git checkout tags/v3.14.26
567  git checkout master
568  git checkout refs/tags/v3.14.26
569  git checkout master
570  git tag -l
571  git checkout -b v3.14.26 v3.14.26
572  git status
573  git describe --tags
574  ls vmfiles/
575  gitgit remote -l
576  git remote -v
577  git remote rm origin
578  git remote -v
579  git status
580  git add . -f
581  git status
582  git add --help
583  git add vmfiles/ -f
584  git checkout -b assignment0
585  git branch
586  git checkout master
587  git branch -D assignment0
588  git checkout v3.14.26
589  git checkout -b a0 v3.14.26
590  git checkout a0
591  git branch
592  git branch a0 v3.14.26
593  git status
594  git commit -m "Added vm files"
595  git status
596  git branch a0 v3.14.26
597  git checkout master
598  ls -a
599  git branch -D v3.14.26
600  git branch
601  git branch a0 v3.14.26
602  git branch
603  git checkout a0
604  git status
605  git log
606  mkdir vmfiles
```

```
607  cp /scratch/spring2016/files/* vmfiles/
608  cp /scratch/opt/environment-setup-i586-poky-linux vmfiles/
609  git add vmfiles/ -f
610  git log
611  git status
612  git commit -m "added vm files"
613  git remote -v
614  git log
615  ls
616  cd vmfiles/
617  ls
618  source environment-setup-i586-poky-linux
619  qemu-system-i386 -gdb tcp::5586 -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"
620  qemu-system-i386 -gdb tcp::5500086 -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"
621  top
622  qemu-system-i386 -gdb tcp::5500086 -S -nographic -kernel ../arch/x86/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"
623  history
624  vim ~/.bashrc
625  source ~/.bashrc
626  cat ~/.bashrc
627  startq
628  cd
629  ls
630  cd cs
631  ls
632  mkdir 444
633  cd 444/
634  history > commands.log
```

5 Version Control Log

Created with gitlog2latex

Author	Date	Message
codymalick	2016-04-05	threads working
codymalick	2016-04-06	removed duplicate directories
codymalick	2016-04-06	added data structure
codymalick	2016-04-07	latex hello world
codymalick	2016-04-07	working make file and example.tex
Cody Malick	2016-04-07	Merge pull request #6 from codymalick/latex
codymalick	2016-04-08	Working weekly template
codymalick	2016-04-08	working on debug
codymalick	2016-04-08	resolved barrier issue
codymalick	2016-04-08	mutex lock working, consumer and producer working
codymalick	2016-04-08	fixed some styling
codymalick	2016-04-08	random function with twister working
codymalick	2016-04-08	prep for rdrand
codymalick	2016-04-08	summary 1 done
Cody Malick	2016-04-08	Merge pull request #7 from codymalick/summary_1

codymalick	2016-04-08	fixed typo
codymalick	2016-04-08	fixed stuff
codymalick	2016-04-09	added new directory for summary 2
codymalick	2016-04-09	completed summary 2
codymalick	2016-04-09	fixed summary 1
codymalick	2016-04-09	typos fixed
Cody Malick	2016-04-09	Merge pull request #8 from codymalick/summary_2
codymalick	2016-04-09	fixed spaces -> tabs, debug message added
codymalick	2016-04-09	Twister and rdrand working
codymalick	2016-04-09	rdrand correction, confirmed working on surfacebook
codymalick	2016-04-09	removed binary
codymalick	2016-04-09	removed binary
codymalick	2016-04-09	removed duplicate file
codymalick	2016-04-09	removed rdseed as it isn't supported on OS class
codymalick	2016-04-10	removed external readme
codymalick	2016-04-10	fixed output formatting, cleaned code, updated makefile
codymalick	2016-04-10	combined makefile working
codymalick	2016-04-10	layout done, toc working
codymalick	2016-04-10	IEEtran working, added reference
codymalick	2016-04-10	Added reference to bibtex, updated .tex, added listings.dtx
codymalick	2016-04-10	qemu flag descriptions done
codymalick	2016-04-10	done with report
Cody Malick	2016-04-10	Merge pull request #10 from codymalick/con1_writeup

os-class linux repo, only one commit on this repo:

Author	**Date**	**Message**
codymalick	2016-04-03	added vm files

6 Bibliography

References

[1] J. M. (2014, may) Intel digital random number generator (drng) software implementation guide. [Online]. Available: https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide

[2] A. Liguori. (2010, jan) Qemu emulator user documentation. [Online]. Available: http://wiki.qemu.org/download/qemu-doc.html