

Writeup 1

Alex Hoffer

CS 444
Spring 2017

Abstract

This first write up describes important details from the completion of Project 1 for D. Kevin McGrath's Operating Systems II class. Topics from Project 1 that are to be covered in this work include the building of the Linux Yocto kernel on Oregon State's engineering server, usage of the qemu virtual machine, and a solution of the Producer-Consumer concurrency problem using the C programming language's POSIX threads execution model.

Contents

1	Log of Commands to Build Yocto Kernel	2
2	Log of Commands to Load Qemu	2
3	Flags in the listed Qemu command line	2
4	Concurrency	2
4.1	Main point of assignment	2
4.2	Personal approach to problem	3
4.3	Ensuring solution was correct	3
4.4	What I learned	3
5	Version control log	3
6	Work log	3

1 Log of Commands to Build Yocto Kernel

put code here

2 Log of Commands to Load Qemu

3 Flags in the listed Qemu command line

The listed Qemu command line is:

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

The following list describes each flag:

- *-gdb* tells the system to wait to connect to gdb on the device tcp::???, this device being passed in as an argument.
- *-S* is a flag that tells the system to wait for a gdb connection.
- *-nographic* is a visualization option that turns off graphical output.
- *-kernel* is a flag that saves time and resources by granting the system a Linux kernel image, rather than requiring a full boot.
- *-drive* handles disk options and takes 1 to n number of arguments to specify these options, in the case of this command the file we passed to this flag is parsed according to the virtio interface.
- *-enable-kvm* is a flag that enables KVM virtualization support without any setting turned off.
- *-net* is a flag that sets network options.
- *-usb* is a flag that turns on the USB driver.
- *-localtime* is a flag that tells the system to use local time.
- *-no-reboot* tells the system to exit, not reboot, when rebooting is an option.
- *-append* is a flag that represents the command line necessary to interact with the kernel.

4 Concurrency

The following subsections answer the four questions as outlined on the Project 1 page on Kevin McGrath's course website.

4.1 Main point of assignment

There were several reasons for the assignment. One, I think, was to re-familiarize us with POSIX Threads, which we learned about in CS344 but likely forgot about before we got to this class. Second, I think that it was to teach us about the Producer-Consumer problem, which is a canonical example of concurrency. Third, it was to force us to use inline assembly language in our programs to help us learn how to implement ASM in C.

4.2 Personal approach to problem

My approach for the use of threads came from guidelines to the canonical Producer-Consumer problem that I found in the *Linux Programming Interface* text. For example, statements like `pthread_create`, `pthread_join`, `pthread_t`, `pthread_mutex_t`, and `pthread_cond_t` were all found in that book and that book instructed me on how to use them. I used these static pthread statements rather than dynamic ones because frankly, I could not think of a reason why dynamic ones would be necessary for the purposes of this assignment. My approach for the buffer and the item to be held in the buffer was just to use a struct to represent the buffer and a struct to represent the item and have the buffer hold a statically allocated array of pointers to items and for each item to have two data fields: a random number and a random waiting period, both that had to be implemented in ASM. Given the statements necessary to control threads and mutexes and the structs necessary to store data, I solved the problem by creating a thread to correspond to a Producer, having that Producer thread lock all of the shared data using a mutex and then produce a random item. After unlocking the shared data within the Producer thread, I created a Consumer thread that locked the data again, consumed the first non-null index in the buffer, and then unlocked the data again. I contained these Producer and Consumer thread creation statements in a forever loop (`while(1)`) so that when the program is run it is eternally creating producer and consumer threads and producing and consuming random data.

4.3 Ensuring solution was correct

4.4 What I learned

5 Version control log

6 Work log