Assignment 1

October 15, 2018

Prepared by

Group 5

Alexandar Hull *(hullale)*
Allison Sladek *(sladeka)*
Anish Asrani *(asrania)*
Mazen Alotaibi *(alotaima)*

Abstract

This assignment served as an introduction to using the Linux kernel and writing code with parallelism. We explored using the emulator and the toolkit with the kernel, paving the way for more exciting kernel applications in the future. With concurrency, we wrote a basic program to demonstrate scheduling and managing threads. Grasping the inner-workings of this is vital to understand how an operating system functions and showcases things that are not directly visible to the average user.

CONTENTS

1 LOG OF COMMANDS

```
-bash-4.2$ ssh os2
-bash-4.2$ mkdir /scratch/fall2018/group5
-bash-4.2$ cd /scratch/fall2018/group5
-bash-4.2$ git clone git://git.yoctoproject.org/linux-yocto-3.19
-bash-4.2$ git checkout tags/v3.19.2
-bash-4.2$ cp /scratch/files/core-image-lsb-sdk-qemux86.ext4 .
-bash-4.2$ cp /scratch/files/bzImage-qemux86.bin .
-bash-4.2$ source /scratch/files/environment-setup-i586-poky-linux
-bash-4.2$ qemu-system-i386 -gdb tcp::5505 -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"
```

In another terminal window: (Alternatively could use screen)

```
-bash-4.2$ gbd
(gdb) target remote tcp:localhost:5505
(gdb) c
qemux86 login: root
(gdb) quit
 -bash-4.2$ fuser -k 5505/tcp
```

2 QEMU COMMAND-LINE

- **-gdb tcp::5505**= Wait for a gdb connection on TCP port 5505.
- **-S**= Do not start CPU on startup (user will have to manually type 'c').
- **-nographic**= Causes qemu to be only a command line to increase speed and avoid unnecessary graphics.
- **-kernel bzImage-qemux86.bin**= Specify the kernel image path.
- **-drive file=core-image-lsb-sdk-qemux86.ext4,if=virtio**= Define a new drive with the path to the image. Connect the drive with virtio interface.
- **-enable-kvm**= Enable KVM full virtualization support. KVM=Kernel-based Virtual Machine.
- **-net none**= Avoid creating a network interface card.
- **-usb**= Enable the USB driver.
- **-localtime**= Sets the time to the current UTC local time.
- **–no-reboot**= Exit instead of rebooting.
- **–append "root=/dev/vda rw console=ttyS0 debug"**= Use "root=/dev/vda rw console=ttyS0 debug" as kernel command line.

3 CONCURRENCY

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

The main point of this assignment was to introduce parallelism and gain some experience in writing threaded programs. Concurrency is a good way to demonstrate some of the functions a kernel regularly performs, like scheduling and running processes. It gives us a better understanding about how resources are managed by a computer. Knowing how to write this kind of program will likely become important to understanding some of the topics we'll be covering in class.

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

We chose to logically split the tasks into individual c and header files. For example, we have a file for the producer, consumer, random, and one for the tasks. Each of these files has only 1-3 functions. We felt that splitting them up into individual files would be easier to understand and would make the program feel more organized.

We decided to store the tasks in an array of structs. This array is statically allocated to a size of BUFFER_SIZE which is globally defined. This array is essentially a queue, except it wraps around. There are two integers, bufferHead and bufferTail. bufferHead indicates the next available spot in the array. bufferTail is the index of oldest task, which should be removed next. We used the modulo operator to deal with the wrapping. This form was chosen because it was easy to implement, and didn't require any dynamic memory allocation or raw pointers. Alternatively, we could have kept the array sorted after every update instead of wrapping around, but this would slow down the algorithm.

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

We started to test smaller sections of our code to ensure they were spewing values within the range that we expected and then expanded from there. There were multiple runs to verify that we weren't just getting the results by chance.

To test behavior when the buffer is full or empty, we added an extra thread for 2 producers or consumers. This way the producers would always be faster than the consumer so the buffer would eventually fill up. We did the same by adding 2 consumers to make sure the buffer would be empty.

3.4 What did you learn?

This exercise gave us more practice with parallel programming. It also gave us a better understanding of how tasks may be scheduled and managed.

4 Version Control Log

Description	Date	Name	Commit
Added readme	Oct 3, 2018	Anish	5c1ccfe5d9dfbf11d9aab95deeb933a385fa8dda
Added kernel source files	Oct 3, 2018	Allison	b62e2eb6820bdaedb8d686509ad6c77ba8e2ec7c
Save readme to local branch	Oct 3, 2018	Allison	7a72a85dbc5390cb96970549ba07c183e62c68f3
Merge pull request #1 from ASladek/master	Oct 3, 2018	Alexander	9e40db94a87c1ebb79d01ed62aa2ef9d553fc525
Added skeleton code	Oct 9, 2018	Alexander	8b148072d2d470e582f7addf05da7ceceb3c3287
Added some task code	Oct 10, 2018	Alexander	eff657f6d5a91b8acfa01d9517920f4502210bbc
Added Empty/Full checks. Now call produce/consume only	Oct 10, 2018	Alexander	de28c3b10e5de0339023b78b9c11d517524b5b6d
Merge pull request #2 from hullale/master	Oct 13, 2018	Anish	e481223aa1106e40422e12bb6bff3b7813efa213
Added threading, rand, and sleeps	Oct 14, 2018	Alexander	e2fde6fbbbd353f3bea548f602549f94ce658338
Merge pull request #3 from hullale/master	Oct 14, 2018	Mazen	bd6031d2cfac945683828fa08bc2ac5b6c99ed8d
support for rdrand	Oct 14, 2018	Alexander	728f484dd413b72aa14832ea85e7441db31b1c44
Merge pull request #4 from hullale/master	Oct 14, 2018	Alexander	30e04fd582af5c69611901ad3095ff4d7bcb0d0e
”Added 32 buffer size, rdrand, print formatting”	Oct 14, 2018	Mazen	fd8ac0af00f28a2917ce1081abfa86fb184ee842
Remove comment	Oct 14, 2018	Mazen	6f7eb6fe3877ee906dcee45c8b86376e213fde24
assignment folder	Oct 14, 2018	Alexander	6e8966180be02ec614b7fa526ca6fce6e82e1e8c

5 Work Log

5.1 October 3rd, 2018

The group convened to read the assignment description, set up a repository, and coordinate tasks. We chose Github because we’re all familiar with it and can set our repository to private. The initial clone of the kernel took longer than expected as we ran into trouble with git modules. All group members were then granted access to the group5 scratch folder.

5.2 October 7th, 2018

We walked through the process of running the kernel together, making sure each group member was able to run the commands to test the emulator and the toolchain, completing the ”Getting Acquainted” portion of the assignment. The files for the write-up were initialized and we began filling in sections.

5.3 October 8th, 2018

Today there was some minor report reformatting.

5.4 October 9th, 2018

Alex added skeleton code for the concurrency section of the assignment.

5.5 October 10th, 2018

Alex updated the concurrency files, adding tasks and calling produce() and consume() without any concurrency yet.

5.6 October 14th, 2018

The group convened to pair-program for the concurrency part where we collectively worked on a shared screen. Most of the testing was done on the engineering machines.

5.7 October 15th, 2018

The write-up was completed.