Project 1: Getting Acquainted

By David Baugh and Justin Sherburne
Group 21

Log of commands to build the Kernal:

1. `git clone git://git.yoctoproject.org/linux-yocto-3.19`

   This command clones the yocto client to your current directory

2. `git checkout 'v3.19.2'`

   Checkout the 'v3.19.2' tag and then source the instructor's files as follows.

3. `source /scratch/files/environment-setup-i586-poky-linux`

4. `cp /scratch/files/core-image-lsb-sdk-qemux86.ext4 core-image-lsb-sdk-qemux86.ext4`

5. `cp /scratch/files/config-3.19.2-yocto-standard .config`

6. `make menuconfig`

   Here we make a couple of changes before building. First we verify that it is in 32-bit mode, then we rename the hostname to whatever we would like. In this case we named it Group 21 HW1.

7. `make -j4 all`

   Make the kernal. This finalizes our build before starting

8. `qemu-system-i386 -gdb tcp::5521 -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"`

   Now the shell will hang waiting for the CPU to start. In another shell:

   (a) `gdb`

   (b) `target remote localhost:5521`

   (c) `continue`

   Finally, to exit the VM:

9. `poweroff`

Concurrency writeup

For our concurrency problem we first implemented the random number generators for later use. Them we broke down our program into the functions that we needed. These functions included a consumer and producer functions to be initiated later. Then we created buffer modifying functions to add and remove items as needed. The final touches, and the hardest part, was getting all of the individual functions to work together.

Breakdown of qemu command

```
qemu-system-i386 -gdb tcp::5521 -S -nographic -kernel bzImage-qemux86.bin -drive file=core-image-lsb
if=virtio -enable-kvm -net none -usb -localtime --no-reboot --append "root=/dev/vda
rw console=ttyS0 debug"
```

Source: *https://qemu.weilnetz.de/doc/qemu-doc.html*

- `qemu-system-i386`

  This portion of the command tells the system that we are going to be running qemu-i386. Qemu is just a program on the OS server that allows us to emulate our linux core.

- `-gdb tcp::5521 -S`

  This enables GDB support for our core. It also opens a port so you can connect using GDB, in our case 5521. The -S suspends the processor until a GDB connection is made to allow for debugging during startup. If you remove the -S the kernal will start rather than waiting for GDB to connect.

- `-nographic`

  Disables the graphic UI that qemu usually opens. Instead you only have the CLI.

- `-kernel bzImage-qemux86.bin`

  This command specifies bzImage as the kernal image.

- `-drive file=core-image-lsb-sdk-qemux86.ext4,if=virtio`

  The drive portion of this command defines a new drive for the kernal to use. The `file=core-image-lsb-sdk-qemux8` *specifies which disk image to use, in this case we copied the disk image from the instructors files. The virtio specifies what type of interface it is using.*

- `-enable-kvm`

  Enables full KVM virtualization support

- `-net none`

  Disables the use of any network devices.

- `-usb`

  Enables the USB driver if it is not enabled by default.

- `-localtime`

  This is a shortcut for the -rtc tag that sets up the clock for the kernal. Specifically this enables the time to be set to the local server time.

- `--no-reboot`

  Allows you to exit rather than rebooting.

- `--append "root=/dev/vda rw console=ttyS0 debug"`

  Allows you to direct linux boot without needing a full bootable image.

Questions

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

   I think the main point of the concurrency assignment was to teach us how to manage many threads running at the same time. Creating processes is the easy part of this assignment, getting those to work together successfully is the difficult part.

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

   Initially we adopted the random number generators from the provided resources on the course website. After that, we knew we needed a producer, a consumer, and a buffer in-between. After those were implemented we divided them into their own threads with additional functions to add and remove from the buffer as needed. We wanted the code to be as modular as possible, so most aspects of this program are divided into their own functions.

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

   When we ran into some issues with our number generators we used some print statements to view the values of the generators as they were running. This led us to realize that the numbers weren't being correctly seeded. Additionally, we used print statements to track when items went in and out of the buffer.

4. What did you learn?

   We learned that paired programming leads to less profanity. Bouncing ideas off of another person can reduce frustration immensely. Also, creating the threads were fairly simple, it was communicating without blocking that was the difficult part.

Version Control Log

We are just beginning our version control for this class. We have two ways that we are tracking versions with. One being a backup of the directory once we are using a stable version of our core. The other is Github where we will make periodic updates to our yocto core repository. We are currently at version 1, because what we have is our base kernal image.

Work Log

- Tuesday October 3rd, 2017:

  This was the day we got started on the project. Here we successfully cloned the yocto core and got through the menuconfig command during the setup. We could not figure out why the qemu command wasn't running, so we stopped for the evening.

- Thursday October 5th, 2017:

  On Thursday we hit the ground running, and built the kernal. We were incorrectly sourcing the files needed for qemu. After getting that we were able to connect via GDB and start the kernal process. We stopped shortly after that.

- Saturday October 7th, 2017:

  On Saturday we started on the writeup and the concurrency assignment separately. We coordinated as needed between the two projects. We utilized some of the number generator files supplied on the course website for our concurrency program, as well as latex resources.

- Sunday October 8th, 2017:

  On Sunday we worked through most of the programming and latex document leaving only a couple of final bugs to sort out. This was the threads, and the skeleton of the latex document.

- Monday October 9th, 2017:

  On Monday we finished up programming and the document and submitted them for grading.