Writeup 1

CS 444

October 29, 2018

Anton Synytsia, Eytan Brodsky, David Jansen

Contents

1 Command Log

Logging into OS2

1) ssh os2.engr.oregonstate.edu

Creating Group Folder

1) cd /scratch/fall/2018
2) mkdir group1

Cloning Repository

1) cd /scratch/fall2018/group1/
2) git clone git://git.yoctoproject.org/linux-yocto-3.19
3) git checkout tags/v3.19.2

Copying Files

1) cp -R /scratch/files. /scratch/fall2018/group1

Setting Up Environment

1) cd /scratch/fall2018/group1/
2) source environment-setup-i586-poky-linux
3) qemu-system-i386 -gdb tcp::5501 -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”

This will hang, which it should; to unhang the process, follow the steps below.

Debugging

1) Start another terminal in OS2 and run the following commands:
2) gdb
3) target remote tcp::5501
4) continue

Part 2: Testing Toolchain

1) cd /scratch/fall2018/group1/linux-yocto-3.19
2) cp /scratch/files/config-3.19.2-yocto-standard .config
3) make -j4 all

Compiling Writeup

1) cd /scratch/fall2018/group1/Writeup1
2) make

2 Qemu Explanation

Qemu Flags

1) **-S:** Do not start the CPU at startup.
2) **-nographic:** Disable graphical output so that QEMU is a command line only application.
3) **-kernel:** Use a bzImage as the kernel, in our case it is using the Intel x86 architecture.
4) **-drive file=core-image-lsb-sdk-qemux86.ext4,if=virtio:** this is used to open an image used file descriptors
5) **-enable-kvm:** Allows full virtualization support.
6) **-net none:** There is no on-board NIC.
7) **-usb:** Enables USB driver.
8) **-localtime:** Legacy option that's currently undocumented. Replaced by -rtc "localtime", which lets the TRC start at the current UTC time.
9) –**no-reboot:** exit instead of rebooting
10) –**append "root=/dev/vda rw console=ttyS0 debug":** Enables debug text to display on the user's terminal

3 Concurrency

1) The objective of concurrency is to understand how to synchronize operations between two tasks.
2) We use pthreads for multi-threading and mutexes for critical sections. One buffer, two threads (plus the main thread), and one mutex are involved in producing and consuming jobs. Details regarding the role of each are described below:

   Buffer The buffer is a circular FIFO queue, containing jobs awaiting to be consumed. The buffer's starting and ending points are marked by `jhead` and `jtail` variables. Produced jobs are appended to the circular queue, incrementing and wrapping `jtail`. Consumed jobs are removed out of the circular queue, incrementing and wrapping `jhead`.
   The buffer has a limit defined by `MAX_JOBS` preprocessor. Once `jtail` reaches the limit, that is one less than `jhead`, the producer stops creating jobs until space becomes available again.

   Producer The producer thread is a continuous while loop, which does the following:

   a) Acquire mutex via `pthread_mutex_lock`.
   b) Check if the jobs queue is not full. If the queue is full, simply unlock the mutex and continue the loop.
   c) Create a job with a random wait time (2 to 9 seconds) and a random number, between 0 and 1000. The randomization is achieved by `_rdrand32_step` intrinsic if supported or `mt19937ar.c` if not.
   d) Append job to the queue, incrementing and wrapping `jtail`.
   e) Release the mutex via `pthread_mutex_unlock`.
   f) Sleep 3 to 7 seconds.

   Consumer The consumer thread is also a continuous while loop, which does the following:

   a) Acquire mutex.
   b) Check if queue is not empty. If empty, release the mutex and continue the loop.
   c) Get job at `jhead`.
   d) Print out the number value of job.
   e) Copy job wait time to a separate variable, `dt`.

f) Increment and wrap `jhead`.

g) Release the mutex.

h) Sleep for a wait time indicated at `dt`.

3) In order to validate concurrency, we printed out every time a job was produced and every time a job was about to be consumed. A 3-7 delay added after producing a job, to ensure that a job can be consumed while jobs are being produced. Once the buffer fills up, the producer waits for consumer to consume a job.

4) Although I, Anton, had previous experience with mutexes and multi-threading, I did learn that concurrency can as well be achieved with semaphores.

4 Version Control

acronym	meaning
V	`version`
tag	`git tag`
MF	Number of `modified files`.
AL	Number of `added lines`.
DL	Number of `deleted lines`.

V	tag	date	commit message	MF	AL	DL
1		2018-10-08	Initial commit	1	1	0
2		2018-10-08	Create .gitignore	1	6	0
3		2018-10-08	Update state	1	0	1
4		2018-10-08	Setup readme	1	5	0
5		2018-10-08	Update README.md	1	2	0
6		2018-10-08	Uploaded concurrency	2	225	0
7		2018-10-08	changed permissions	4	0	0
8		2018-10-09	Setup writeup TeX	2	126	0
9		2018-10-09	Added pygments	1	98	0
10		2018-10-09	included pygements to preamble	1	8	6
11		2018-10-09	First sucessful compile of writeup1	6	2008	2
12		2018-10-09	Updated gitignore	5	3	2005
13		2018-10-09	Setup Writeup	1	10	20
14		2018-10-09	Added git attributes to enforce EOL	1	7	0
15		2018-10-09	EOL deal	2	1	2
16		2018-10-09	Trying to fix TEX compiling	2	79	1
17		2018-10-09	Added sections to writeup	3	8	81
18		2018-10-09	Added the writeup for Command Logs and for the Qemu flags	11	97	10

Please continue at the next page

V	tag	date	commit message	MF	AL	DL
19		2018-10-10	Work on concurrency writeup	8	30	56
20		2018-10-10	More work on Concurrency writeup	2	12	7
21		2018-10-10	Made Concurrency use circular queue	1	41	34
22		2018-10-10	Fixed concurrency print type	1	9	6
23		2018-10-10	Renamed Assignment1 folder to Concurency and finished my writeup	7	251	244
24		2018-10-11	Disabled BIB command in makefile	2	3	3
25		2018-10-11	Added compile instructions	1	26	3
26		2018-10-11	README adjustments	1	3	3
27		2018-10-13	corrected name	1	1	1
28		2018-10-13	explain -localtime and -append root=/dev/vda rw console=ttyS0 debug	1	2	2
29		2018-10-13	add work log.	1	6	0
30		2018-10-13	Added the version control table	1	31	1
31		2018-10-13	Changed up how the lists were made	1	48	31
32		2018-10-15	Refactored rdrand to use X86 intrinsic and created a general rand function	1	88	69
33		2018-10-15	Updated concurrency writeup to reflect the refactored code.	1	7	10

5 Work Log

Getting Acquainted The ”Getting Acquainted” section was broken up between the group members to more evenly balance the work. In week two we started and completed the core of the assignment, which was to set up a working version of the yocto kernel where we tested both the emulator and the toolchain. This process was faster than expected since everything surprisingly just worked. At the beginning of week three we started working on the write up, where each person was assigned a specific section to complete sometime before the due date.

Concurrency We completed the ”Concurrency” section on week two while at the library. This took a couple of hours to write a working prototype, debug it, and add specific features such as buffer limits, addition speed, etc... mainly for verifying functionality. We followed a loose group coding technique where one person would write the code and the other would make suggestions, corrections, and look up different documentations as needed. This made the coding process go very smoothly and we were able to finish relatively quickly.