Homework 1 Report

CS444 Fall17

Zach Lerew, Rohan Barve
Group 26

October 9, 2017

Abstract

This is an abstract

Command line qemu parameters

- -h Display help and exit
- -version Display version information and exit
- -machine [type=]name[,prop=value[,...]] Select the emulated machine by name. Use "-machine help" to list available machines.
- -cpu model Select CPU model ("-cpu help" for list and additional feature selection)
- -accel name[,prop=value[,...]] This is used to enable an accelerator. Depending on the target architecture, kvm, xen, hax or tcg can be available. By default, tcg is used. If there is more than one accelerator specified, the next one is used if the previous one fails to initialize.
- –smp [cpus=]n[,cores=cores][,threads=threads][,sockets=sockets][,maxcpus=maxcpus] Simulate an SMP system with n CPUs. On the PC target, up to 255 CPUs are supported. On Sparc32 target, Linux limits the number of usable CPUs to 4. For the PC target, the number of cores per socket, the number of threads per cores and the total number of sockets can be specified. Missing values will be computed. If any on the three values is given, the total number of CPUs n can be omitted. maxcpus specifies the maximum number of hotpluggable CPUs.
- -numa node[,mem=size][,cpus=firstcpu[-lastcpu]][,nodeid=node]
- -numa node[,memdev=id][,cpus=firstcpu[-lastcpu]][,nodeid=node]
- -numa dist,src=source,dst=destination,val=distance
- -numa cpu,node-id=node[,socket-id=x][,core-id=y][,thread-id=z] Define a NUMA node and assign RAM and VCPUs to it. Set the NUMA distance from a source node to a destination node
- -add-fd fd=fd,set=set[,opaque=opaque] Add a file descriptor to an fd set.
- -set group.id.arg=value Set parameter arg for item id of type group
- -global driver.prop=value
- -mem-path path Allocate guest RAM from a temporarily created file in path
- -mem-prealloc Preallocate memory when using -mem-path
- -k language Use keyboard layout language (for example "fr" for French). This option is only needed where it is not easy to get raw PC keycodes (e.g. on Macs, with some X11 servers or with a VNC or curses display). You don't normally need to use it on PC/Linux or PC/Windows hosts
- -audio-help Will show the audio subsystem help: list of drivers, tunable parameters
- -soundhw card1[,card2,...] or -soundhw all Enable audio and selected sound hardware. Use 'help' to print all available sound hardware
- -balloon none Disable balloon device
- -balloon virtio[,addr=addr] Enable virtio balloon device (default), optionally with PCI address addr
- -name name Sets the name of the guest. This name will be displayed in the SDL window caption. The name will also be used for the VNC server. Also optionally set the top visible process name in Linux. Naming of individual threads can also be enabled on Linux to aid debugging
- -uuid uuid Set system UUID
- -fda file
- -hdd file Use file as hard disk 0, 1, 2 or 3 image

- -cdrom file Use file as CD-ROM image (you cannot use -hdc and -cdrom at the same time). You can use the host CD-ROM by using /dev/cdrom as filename
- -blockdev option[,option[,option[,...]]] Define a new block driver node. Some of the options apply to all block drivers, other options are only accepted for a specific block driver. See below for a list of generic options and options for the most common block drivers
- -drive option[,option[,option[,...]]] Define a new drive. This includes creating a block driver node (the backend) as well as a guest device, and is mostly a shortcut for defining the corresponding -blockdev and -device options
- -full-screen Start in full screen
- -nographic Normally, if QEMU is compiled with graphical window support, it displays output such as guest graphics, guest console, and the QEMU monitor in a window. With this option, you can totally disable graphical output so that QEMU is a simple command line application. The emulated serial port is redirected on the console and muxed with the monitor (unless redirected elsewhere explicitly). Therefore, you can still use QEMU to debug a Linux kernel with a serial console. Use C-a h for help on switching between the console and monitor
- -display type Select type of display to use. This option is a replacement for the old style -sdl/-curses/... options
- -usb Enable the USB driver (if it is not used by default yet)
- -usbdevice devname Add the USB device devname. Note that this option is deprecated, please use "-device usb-..." instead
- -full-screen Start in full screen
- -vnc display[,option[,option[,...]]] Normally, if QEMU is compiled with graphical window support, it displays output such as guest graphics, guest console, and the QEMU monitor in a window. With this option, you can have QEMU listen on VNC display display and redirect the VGA display over the VNC session. It is very useful to enable the usb tablet device when using this option (option -device usb-tablet). When using the VNC display, you must use the -k parameter to set the keyboard layout if you are not using en-us

Kernel build,test,VM Qemu setup

- mkdir /scratch/fall2017/26.
- cd /scratch/fall2017/26.
- git clone git://git.yoctoproject.org/linux-yocto-3.19.
- cd linux-yocto-3.19.
- git checkout 'v3.19.2', cd ../
- cp /scratch/files/environment-setup-i586-poky-linux* ./
- cp /scratch/files/bzImage-qemux86.bin ./
- cp /scratch/files/core-image-lsb-sdk-qemux86.ext4 ./
- source environment-setup-i586-poky-linux
- cd linux-yocto-3.19.
- make clean.
- cp /scratch/files/config-3.19.2-yocto-standard ./.config.

- makemenuconfig, adjust general setting Local version to needed group no.
- Save the file as .config and exited menuconfig.
- Compiled the kernel using: make -j4 bzImage. Used 4 threads.
- qemu-system-i386 -gdb tcp::5526 -S -nographic -kernel linux-yocto-3.19/arch/x86/boot/bzImage -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”
- Open new terminal session and launch gdb.
- Within gdb connect to specified port: (gdb) target remote:5526.
- This step should result in the kernel booting and a login screen qemux86 appearing.
- The login is root and there is no password so I simply hit Enter.

Concurrency assignment

Purpose

The assignment states that the purpose of concurrency assignments is to hone your skills in thinking in parallel. Parallel programming is a large independent topic worthy of study. When it comes to operating systems, synchronous work is less common and less useful than parallel async work. This assignment helps us practice those skills.

Approach

The problem was fairly easy to solve when you think in terms of objects and analogies.
There are items in a buffer, which must be created by producers, and consumed by consumers.
The buffer can be thought of as a cookie jar with a small opening, only one hand can get in at a time.
Once an item is taken out of the jar, another hand can get into the jar to do its job.
Items can be created and consumed outside of the jar (in parallel), and then fight for a chance to access it when they need to.
Out of this analogy easily comes the pieces needed to solve this problem:

- An item with some data.
- A buffer to store those items.
- A consumer who locks the mutex, checks the buffer for an item, takes one if it can, and then unlocks the mutex.
- A producer who locks the mutex, checks if the buffer is full, adds a new item if it can, and then unlocks the mutex.
- And lastly a main function to spawn threads and initialize data.

Testing

With the above approach on paper, a loosely TDD approach was taken.
This problem was small enough however, that a manual test could be used for each step, rather than a traditonal failing unit test.
Final results were independently verified by both team members.

Learned

Every time I work with C, I am reminded of its power, as well as its drawbacks. This is a fairly simple problem to solve, which makes C an overly complicated tool to solve it with.
In a professional work environment, the tools that can be used are typically limited to the code base owned by the company.
Regardless of the problem and the tools that exist, if the code base is in C#, the solution should be too (with occasional exceptions).
I understand that the purpose of the class and assignment is low level kernel programming in C (the language of this 'code base'). However, why use a 700 bhp v8 in a 4 door sedan that you drive to the grocery store in bumper-to-bumper traffic?

Version control log

Git repo log

Work log

This assignment has three distinct sections:

- Kernel building and qemu emulation
- Consumer & Producer concurrency problem
- This document

The work was split between us as such -
Zach: Concurrency problem, document base / makefile
Rohan: Kernel and VM Qemu setup, configuration and test , concurrency problem review, document body