Lab Assignment 3 : Working with XV6

August 19, 2020

Objective :

- This assignment is the introduction to XV6, an x86-based re-implementation of Unix v6. The goal of this lab is to give you a basic introduction to the xv6 OS, its shell, and processes in general. You will run a few simple programs and monitor their behavior using the *proc* file system and other related tools.

Instructions

- In this lab we focus on getting *xv6* up and running under **QEMU** and debugging it under **GDB** at the *C* source code level.
- Read through Chapter 0 in the xv6 textbook (available in the web-link provided in the course website), specially the description about how the *xv6 shell* works.

Installing QEMU : Login to your Linux systems, open a terminal and type the following at the linux prompt :
$*sudo apt-get install qemu*

QEMU is an emulator that provides the virtual platform to run xv6 kernel image on the host linux machine

Booting XV6 :

- Fetch and unzip the xv6 source (provided to you through *Google Classroom* or download from the local link given in the web page)
- Build xv6 - On a terminal type the following :
  $ cd xv6
  $ make
- Run xv6 under QEMU - Type the following at the terminal :

  $ make qemu

The following will happen :

- A separate window will appear containing the display of the virtual machine
- After a few seconds, **QEMU**'s virtual *BIOS* will load *xv6*'s **boot loader** from a virtual hard drive image contained in the file *xv6.img* and the **boot loader** will run in turn load and run the *xv6* kernel
- After everything is loaded, you should get a '$' prompt in the xv6 display window and be able to enter commands into the rudimentary but functional xv6 shell.
- The small file system you're examining and modifying here resides on a second virtual disk, whose initial contents **QEMU** initialises from the file *fs.img*. Later in the course we will examine how *xv6* accesses and modifies this file system.
- Try the following at the xv6 prompt :
  $ ls
  $ echo Hello!
  $ cat README
  $ grep run README
  $ cat README | grep run | wc
  $ echo MY NEW FILE ¿ newfile
  $ cat newfile

Remote Debugging xv6 under QEMU :

- The easiest way to debug xv6 under QEMU is to use GDB's remote debugging feature and QEMU's remote GDB debugging stub. Remote debugging is a very important technique for kernel development
- The basic idea is that the main debugger (*GDB* in this case) runs separately from the program being debugged (the *xv6* kernel atop *QEMU*) - they could be on completely separate machines.
- The debugger and the target environment communicate over some simple communication medium, such as a *network socket* or a *serial cable*, and a small remote debugging stub handles the "immediate supervision" of the program being debugged in the target environment. This way, the main debugger can be a large, full-featured program running in a convenient environment for the developer atop a stable existing operating system, even if the kernel to be debugged is running directly on the bare hardware of some other physical machine and may not be capable of running a full-featured debugger itself.
- A small remote debugging stub is typically embedded into the kernel being debugged; the remote debugging stub implements a simple command language that the main debugger uses to inspect and modify the target program's memory, set breakpoints, start and stop execution, etc.
- Compared with the size of the main debugger, the remote debugging stub is typically minuscule, since it doesn't need to understand any details of the program being debugged such as high-level language source files, line numbers, or C types, variables, and expressions:

it merely executes very low-level operations on behalf of the much smarter main debugger.

– When we are doing kernel development using a virtual machine such as QEMU, remote debugging may not be quite as critical: for example, xv6 can also be run under the Bochs emulator, which is much slower than QEMU but has a debugger built-in and thus does not require the use of GDB remote debugging. On the other hand, while usable, the Bochs debugger is still not as complete as GDB, so we will primarily use GDB with QEMU's remote debugging stub in this course.

To run xv6 under QEMU and enable remote debugging:

– Open new terminal(T1) and enter the 1st command
$ nm kernel | grep_start

– note_the_break_point

– Open a new terminal(T2) and enter the 2nd command
$ make qemu-nox-gdb

– Open a new terminal(T3) and enter the following command
$ gdb
$ target remote: port_no obatined from terminal T2
$ br * 0X break_point
$ continue or c

Closing QEMU session :

Close the QEMU session, destroying the state of the xv6 virtual machine by entering *quit* at the *QEMU* promt in the original window from which you started QEMU or t by pressing **CTRL-C** in that window.