CS4411 Project 2. Implement a system call

Due Feb. 6 11:59 PM

Purpose

The purpose of this project is to setup experimental environments for xv6 OS and to understand system call.

Project summary

1) Install QEMU required for the xv6
2) Download, compile, and boot xv6
3) Add a new system call to xv6 and test it
4) Submit your files to Canvas

Install QEMU

1) Create QEMU installation dir
   mkdir $HOME/opt
2) Download the QEMU4.2
   wget https://download.qemu.org/qemu-4.2.0.tar.xz
   tar xvJf qemu-4.2.0.tar.xz
   cd qemu-4.2.0
   ./configure --disable-kvm --disable-werror --prefix=$HOME/opt/bin --target-list="i386-softmmu x86_64-softmmu"
   make
   make install
3) Add QEMU installation dir to the PATH variable
   Add " export PATH=$PATH:$HOME/opt/bin" to the $HOME/.bashrc file.

Install XV6

1) Download the XV6 source code
   git clone https://github.com/mit-pdos/xv6-public
2) Compile and run XV6
   cd xv6
   make qemu-nox
3) Quit from xv6
   To exit xv6, type ctrl-a x.

GDB debug XV6

1) Run xv6 with GDB port open
   cd xv6
   make qemu-nox-gdb
   You will see the tcp port xxxx
2) Run GDB with the kernel binary
   gdb
   (gdb) target remote :xxxx
   (gdb) file kernel
   (gdb) continue

Adding the system call

Add a new system call that takes no arguments and returns the number of runble processes. See below for hints about how to do this.

For this task, you might want to read Chapter 3 of the xv6 book describing how exceptions, traps, interrupts, etc. How system call dispatching is implemented in syscall.c and syscall.h, and how the handlers for individual system calls are implemented in sysproc.c and sysfile.c.

Basic steps for adding system calls:

1) Create a sys_getrunble function based on an existing simple system call function like sys_uptime. (For this assignment, you do not need to (but are allowed to) use a spinlock and acquire or release like uptime does since we do not care how your code works with or multiple processors.)
2) Add a system call number for your new system call to syscall.h.
3) Add your sys_getrunable to the table in syscall.c.
4) Read procdump( ) in proc.c.
   Find out how to iterate through the process queue in XV6.
5) Edit usys.S and user.h to create a system call wrapper function that invokes your system call from a normal user program.

Testing the system call

1) Using echo.c as a template, create a new program to run your getrunble system call and print the results.
2) Edit Makefile by adding your program to UPROGS, similar to how echo is included on this list.
3) Run make and then make qemu to boot the OS with your new program included.
4) If your test program crashes after finishing, you may have forgotten to exit() at the end. (Returning from main() will not work.)

5) You could run other programs that call (e.g. by outputting anything to the console) and/or have your test program make writes (using the system call wrapping function directly or by taking advantage of printf() calling write) to verify that the count make sense.

Submission

1) Remove the object files, binary files and the FS image file

   make clean

   Issue the above command when you are under the xv6-public folder.

2) Compress the xv6 folder

   tar cvfz xv6-your-last-name.tar.gz xv6-public/*

   Issue the above command when you are outside of the xv6-public folder. Note that only files under the xv6-public are include the archive and do not put the sub-folder to it. Otherwise the size of archive is 18MB.

3) Upload the xv6-your-last-name.tar.gz to Canvas

Credit

This assignment is based on Charles Reiss's assignment.