Lab 0: Get Hand-on experience with Xv6

10/04/2023

TA Information

- Name: Lian Gao
- Email: lgao027@ucr.edu
- Lab & Office Hour:
  - When: Every Wednesday from 6 pm to 8:50 pm.
  - Where: Sproul Hall 22340
- Prefer Over-communication.
  - What is the issue/blocker?
  - The error message, logs.
  - What you have tried?
  - What is your plan to solve it? And why?
  - How to reproduce the problem?
- Attendance is not required but highly recommended for every lab.

Topics for today

- XV6 setup instruction
  - Sledge remote environment
  - Local environment: Docker or VM
  - General XV6 setup instructions
- Write a hello world program
- Debug XV6
  - working with GDB and qemu
  - working with vscode debugging GUI

What is xv6

- Xv6 is a **teaching operating system** developed in the summer of 2006 for MIT's operating systems course 6.828
- A toy OS with only 9299 lines of code but have everything you need to learn.
- Provides the basic interfaces of Unix-like systems.
- In this Course, we will make several improvements over XV6.
- We expect you to do all lab assignments alone but you are welcome to discuss with your classmates.

Getting Started - Setup Xv6 Environment

Setup a **remote** environment.

- Login to sledge.cs.ucr.edu using your UCR CS account.
- Add the following PATH to ~/.bashrc.
  - export PATH=/usr/local/pkgs/qemu-5.10/bin:/usr/local/pkgs/riscv-gnu-toolchain/bin:$PATH

Choose one of the following methods to setup a **local** environment (optional).

1. For Linux(Ubuntu, CentOS, Debian ...) machine, you can try directly build from https://github.com/hengyin/xv6-riscv.git
   a. General instructions: https://pdos.csail.mit.edu/6.S081/2020/tools.html
   b. You will need a RISC-V "newlib" tool chain from https://github.com/riscv/riscv-gnu-toolchain, and
   c. qemu compiled for riscv64-softmmu.
2. For other OS, you need to firstly setup a Linux environment via docker **or** VM.

Setup Xv6 Environment via Docker (Recommended)

- Install docker https://docs.docker.com/get-started/
- Get a xv6-riscv image from
  - https://drive.google.com/file/d/1zqVLUMyIcmE2cXa6GN_FF06yk9Xl_8Tm/view?usp=sharing
- Login to the container and build xv6 at /cs179f/xv6-riscv

Docker cheat sheet:

- Launch a docker image: docker run image_id
- Resume or pause a docker container: docker start/stop container_id
- Check all containers: docker ps -a
- Login container: docker exec -it container_id bash
- Save container into images: docker commit -p container_id image_id
- Pack the docker image: docker save image_id>./my_image.tar
- Unpack the docker image: docker load<my_image.tar

Setup Xv6 Environment via VM

- Download and install VB/Vagrant/SSH client:
  - Virtualbox: https://www.virtualbox.org/wiki/Downloads
  - Vagrant: http://www.vagrantup.com/downloads.html
  - SSH client PuTTy or Cygwin (Windows)
- Follow instructions in https://pdos.csail.mit.edu/6.S081/2020/tools.html

Setup xv6-riscv

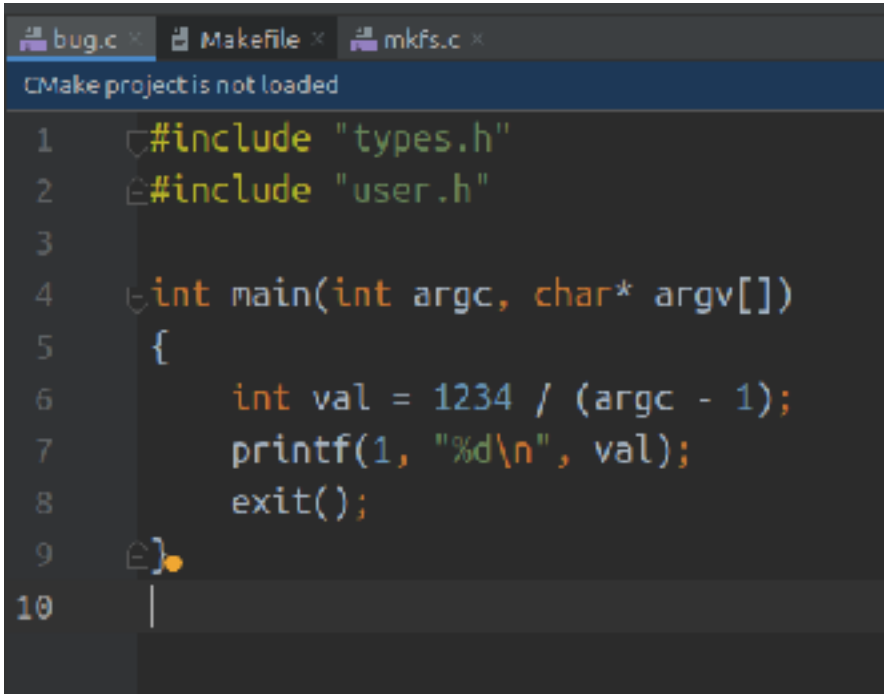
- The instructions in this page assumes you have a linux machine that can build xv6-riscv.
  a. Log into sledge, your docker container or your VM.
  b. Use this xv6-riscv version.
     - `git clone https://github.com/emidec/cs179f-fall23.git xv6-riscv`
  c. Follow instructions in your first Lab assignment

Common Issues

- cannot setup vscode debugging environment
  - comment out `@REM target remote 127.0.0.1:26000` inside auto generated `.gdbinit`
- make: qemu-system-riscv64: Command not found
  - export PATH=/usr/local/pkgs/qemu-5.10/bin:/usr/local/pkgs/riscv-gnu-toolchain/bin:$PATH
- Undefined item: "riscv:rv64"
  - use gdb-multiarch or riscv64-unknown-elf-gdb
- /usr/bin/env: 'python': No such file or directory
  - cp /usr/bin/python3 /usr/bin/python, otherwise install python.
- qemu-system-riscv64: Some ROM regions are overlapping. The following two regions overlap (in the memory address space):
  - qemu-system-riscv64 -machine virt **-bios none** -kernel kernel/kernel -m 128M -smp 3 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0
- user/sh.c:58:1: error: infinite recursion detected [-Werror=infinite-recursion]
  - https://github.com/mit-pdos/xv6-riscv/issues/130 append -Werror=infinite-recursion into Makefile

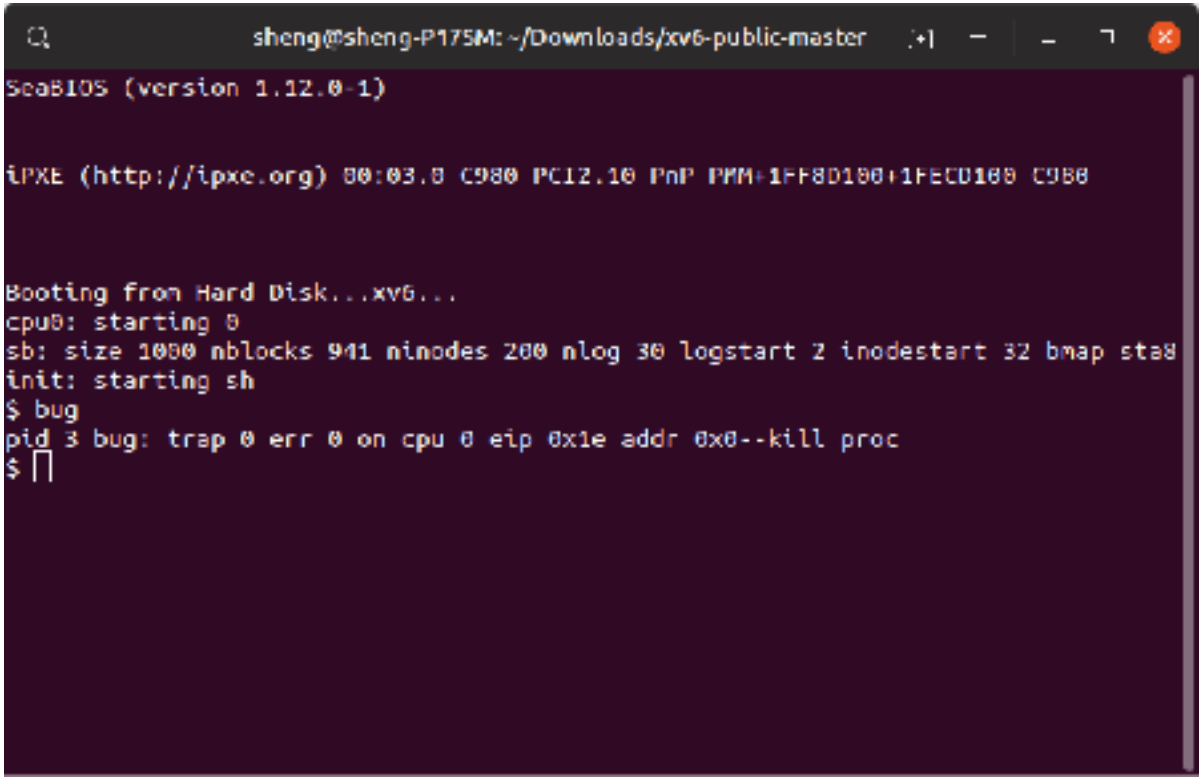
Debug XV6

Bug.c

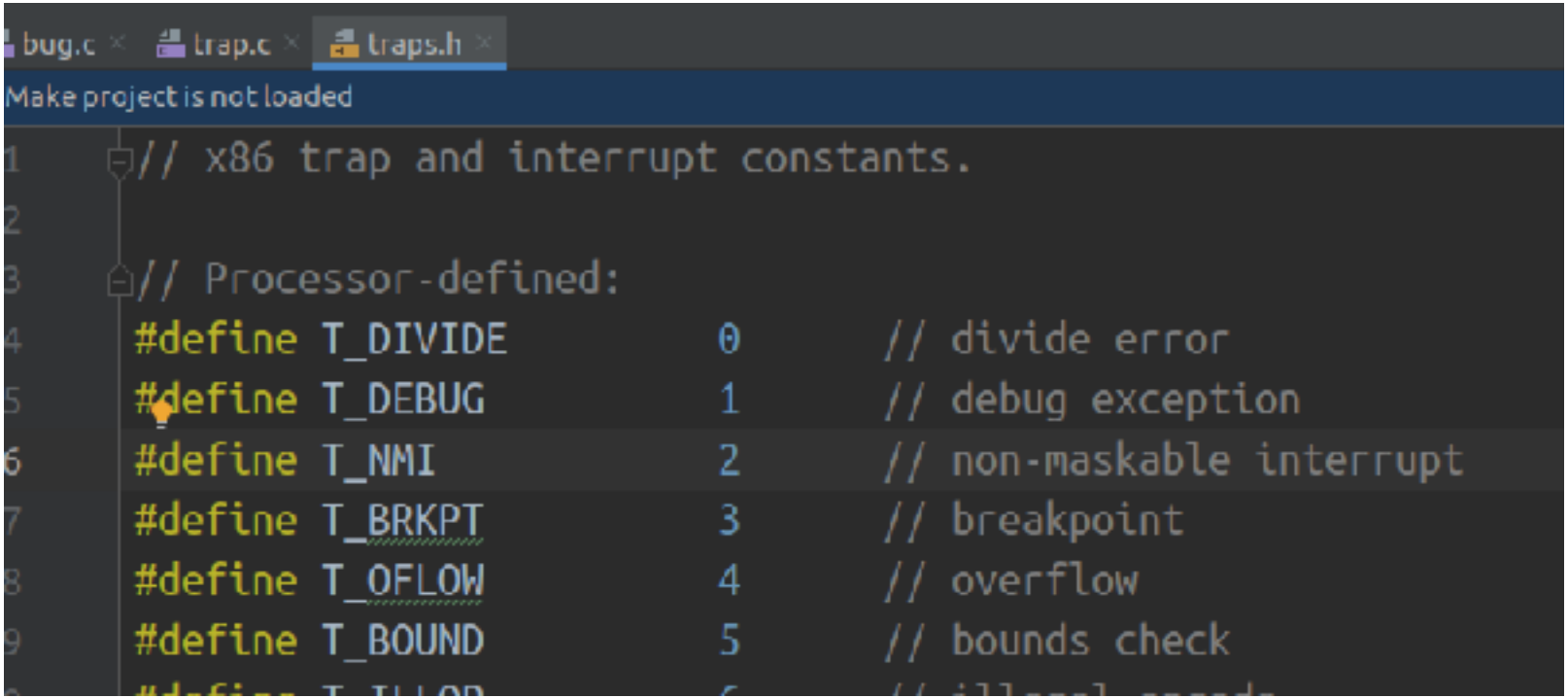


Program Output

keyword:
trap 0



Trace back

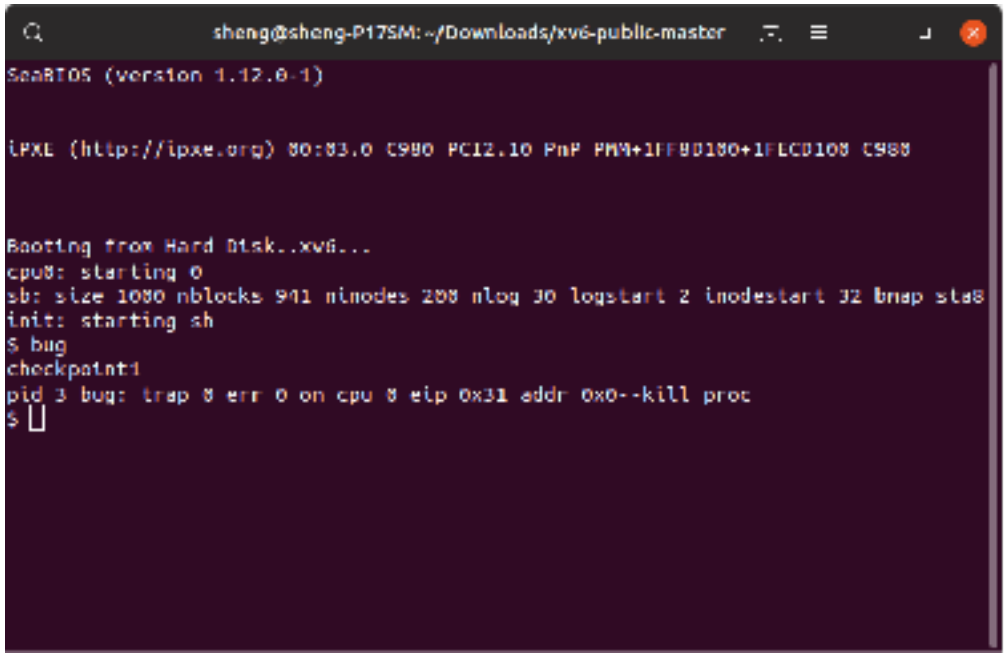

bug.c
trap.c
traps.h
Make project is not loaded
// x86 trap and interrupt constants.
// Processor-defined:
#define T_DIVIDE 0 // divide error
#define T_DEBUG 1 // debug exception
#define T_NMI 2 // non-maskable interrupt
#define T_BRKPT 3 // breakpoint
#define T_OFLOW 4 // overflow
#define T_BOUND 5 // bounds check

Use printf/cprintf

bug.c Makefile

CMake project is not loaded

```
1   #include "types.h"
2   #include "user.h"
3
4   int main(int argc, char* argv[])
5    {
6       printf(1, "checkpoint1\n");
7       int val = 1234 / (argc - 1);
8       printf(1, "checkpoint2\n");
9       printf(1, "%d\n", val);
10      printf(1, "checkpoint3\n");
11      exit();
12  }
13
```



Use GDB

Common GDB commands

- n or next: run till the next source line; step over subroutine calls
- s or step: run till it reaches a different source line; step into calls
- ni or nexti: step one instruction; step over calls
- si or stepi: step one instruction; step into calls
- print [varname]: print variable content
- layout asm: show disassembly code

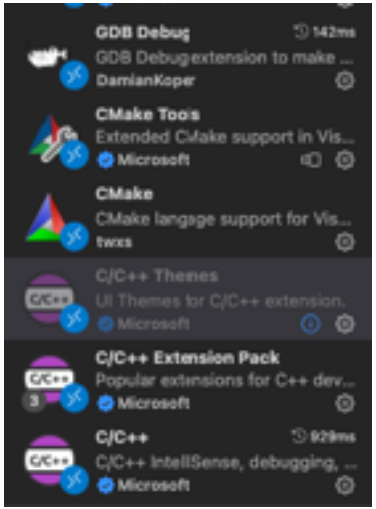
Cheat sheet at: https://gist.github.com/rkubik/b96c23bd8ed58333de37f2b8cd052c30

Vscode Debugging GUI

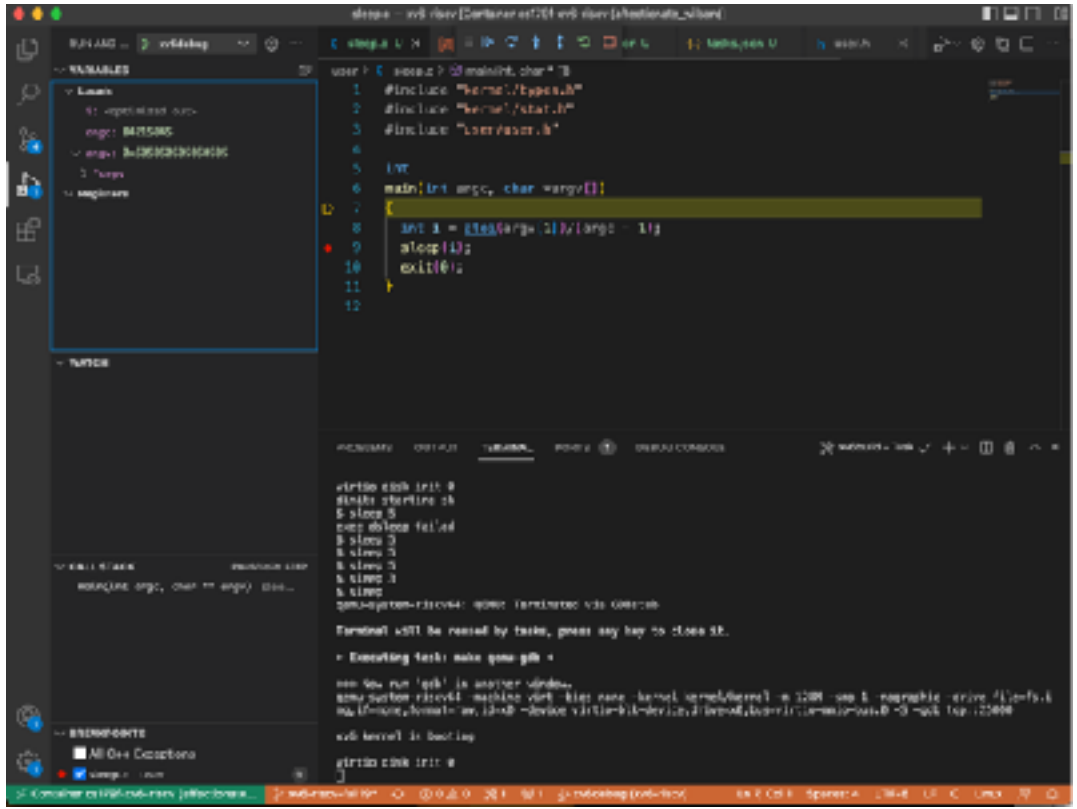
1. install these extensions.
2. create launch.json and tasks.json under .vscode

```
// xv6-riscv/.vscode/launch.json
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "xv6debug",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceFolder}/user/_${program}",
            "stopAtEntry": true,
            "cwd": "${workspaceFolder}",
            "miDebuggerServerAddress": "127.0.0.1:25000",
            "miDebuggerPath": "/usr/bin/gdb-multiarch",
            "MIMode": "gdb",
            "preLaunchTask": "xv6build"
        }
    ]
}
```

```
// xv6-riscv/.vscode/tasks.json
{
    "version": "2.0.0",
    "tasks": [
        {
            "label": "xv6build",
            "type": "shell",
            "isBackground": true,
            "command": "make qemu-gdb",
            "problemMatcher": [
                {
                    "pattern": [
                        {
                            "regexp": ".",
                            "file": 1,
                            "location": 2,
                            "message": 3
                        }
                    ],
                    "background": {
                        "beginsPattern": ".*Now run 'gdb' in another window.",
                        "endsPattern": "."
                    }
                }
            ]
        }
    ]
}
```



GUI (cont’d)



Lab 0 task

1. Setup the xv6 dev environment used in this course.
2. Understand the architecture of xv6 project (kernel, executables and Makefile)
3. Learn how to build and debug xv6
4. Create a new executable called helloworld for xv6 (like cat, ls etc.) (optional)
5. Run your helloworld program in xv6 in debugging mode, set a breakpoint in the OS kernel and play with it.(optional)
6. Before you start coding Lab 1 Assignment, read Chapter 1 of the xv6 book. (optional)

References

- XV6 Book: https://pdos.csail.mit.edu/6.828/2014/xv6/book-rev8.pdf
- https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/guides/files/DebuggingGuide.pdf
- mit 6.828 course page: https://pdos.csail.mit.edu/6.828/2019/tools.html
- https://www.cs.ucr.edu/~heng/teaching/cs179f-winter21/index.html