浙江大学实验报告

课程名称：操作系统
实验项目名称：GDB & QEMU 调试 64 位 RISC-V LINUX
学生姓名：展翼飞 学号：3190102196
电子邮件地址：1007921963@qq.com
实验日期： 2023年9月22日

一、实验内容

1.搭建实验环境

- 使用VMware构建Ubuntu 22.04.3 LTS虚拟机，并安装编译内核所需要的交叉编译工具链和用于构建程序的软件包

```
$ sudo apt install gcc-riscv64-linux-gnu
$ sudo apt install autoconf automake autotools-dev curl libmpc-dev libmpfr-dev
libgmp-dev \ gawk build-essential bison flex texinfo gperf libtool patchutils bc \
zlib1g-dev libexpat-dev git
#sudo 使用管理员权限 apt 包管理工具
```

在使用apt安装包时，遇到如下问题：

```
frey@DESKTOP-5BJAETF:~$ sudo apt install gcc-riscv64-linux-gnu
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package gcc-riscv64-linux-gnu
frey@DESKTOP-5BJAETF:~$
```

在使用sudo apt-get update与sudo apt-get upgrade更新包列表与apt软件后安装命令正常运行

- 安装用于启动 riscv64 平台上的内核的模拟器 `qemu`

```
$ sudo apt install qemu-system-misc
```

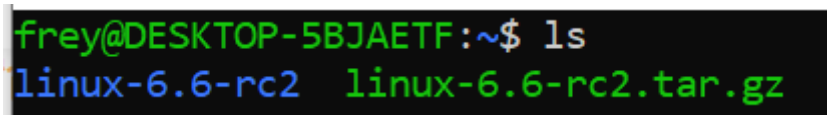
- `gdb` 来对在 `qemu` 上运行的 Linux 内核进行调试

```
$ sudo apt install gdb-multiarch
```

2. 获取 Linux 源码和已经编译好的文件系统

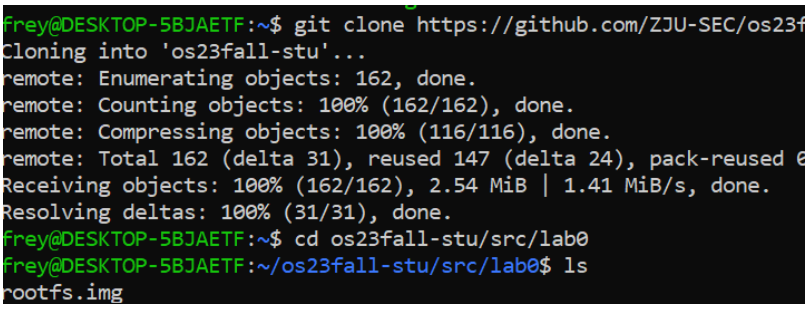
- 从 https://www.kernel.org 下载最新的 Linux 源码 6.6-rc2，并拷贝至wsl用户目录中，并解压

```
$ tar zxvf linux-6.6-rc2.tar.gz -C ~
```



- 使用 git 工具 clone仓库：https://github.com/ZJU-SEC/os23fall-stu。其中已经准备好了根文件系统的镜像

```
$ git clone https://github.com/ZJU-SEC/os23fall-stu.git
$ cd os23fall-stu/src/lab0
$ ls
rootfs.img # 已经构建完成的根文件系统的镜像
```



3.编译 Linux 内核

- 进入解压后的linux内核源码文件夹，编译linux内核

```
$ cd linux-6.6-rc2
$ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- defconfig
#在内核根目录下根据RISC-V默认配置生成一个名为 `.config` 的文件，包含了内核完整的配置，内核在编译时会根据 `.config` 进行编译
$ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- -j4
#使用4线程编译内核
```

- 编译后结果如下图所示

```
frey@DESKTOP-5BJAETF:~/linux-6.6-rc2$ ls
COPYING         Makefile        certs     ipc                    net       usr
CREDITS         Module.symvers  crypto    kernel                 rust      virt
Documentation   README          drivers   lib                    samples   vmlinux
Kbuild          System.map      fs        mm                     scripts   vmlinux.a
Kconfig         arch            include   modules.builtin        security  vmlinux.o
LICENSES        block           init      modules.builtin.modinfo  sound
MAINTAINERS     built-in.a      io_uring  modules.order          tools
```

4.使用 QEMU 运行内核

编译内核后，在内核源码文件夹中使用QEMU运行内核：

```
$ qemu-system-riscv64 -nographic -machine virt -kernel ./arch/riscv/boot/Image \
-device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0" \
-bios default -drive file=../os23fall-stu/src/lab0/rootfs.img,format=raw,id=hd0
#`-nographic`: 不使用图形窗口，使用命令行
#`-machine`: 指定要 emulate 的机器为RISC-V VirtIO board
#`-kernel`: 指定内核 image为该路径下的linux内核
#`-device`: 指定要模拟的设备为virtio-blk-device，并指定硬盘设备hd0作为后端
#`-append cmdline`: 使用 cmdline 作为内核的命令行
#`-bios default`: 使用默认的 OpenSBI firmware 作为 bootloader
#`-drive, file=<file_name>`: 使用rootfs.img作为文件系统
```

结果如下图：

```
frey@DESKTOP-5BJAETF:~/linux-6.6-rc2$ qemu-system-riscv64 -nographic -machine virt -kern
 ./arch/riscv/boot/Image -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro c
sole=ttyS0" -bios default -drive file=../os23fall-stu/src/lab0/rootfs.img,format=raw,id=
0

OpenSBI v0.9
   ____                    ____  ____  _____
  / __ \                  / ___|| __ \|_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name             : riscv-virtio,qemu
Platform Features         : timer,mfdeleg
Platform HART Count       : 1
Firmware Base             : 0x80000000
Firmware Size             : 100 KB
Runtime SBI Version       : 0.2
```

```
[    0.912695] EXT4-fs (vda): mounted filesystem c3e9bbca-ec22-47f9-a368-187b21172fc1 ro w
ith ordered data mode. Quota mode: disabled.
[    0.918887] VFS: Mounted root (ext4 filesystem) readonly on device 254:0.
[    0.924568] devtmpfs: mounted
[    0.957512] Freeing unused kernel image (initmem) memory: 2204K
[    0.961993] Run /sbin/init as init process

Please press Enter to activate this console.
/ # ls
bin         etc         lost+found  sbin        usr
dev         linuxrc     proc        sys
/ #
```

5.使用 GDB 对内核进行调试

开启两个 Terminal Session，一个 Terminal 使用 QEMU 启动 Linux，另一个 Terminal 使用 GDB 与 QEMU 远程通信（使用 tcp::1234 端口）进行调试：

```
#Terminal 1
$ qemu-system-riscv64 -nographic -machine virt -kernel ./arch/riscv/boot/Image \
-device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0" \
-bios default -drive file=../os23fall-stu/src/lab0/rootfs.img,format=raw,id=hd0 -S
-s
#- `-S`: 启动时暂停 CPU 执行
#- `-s`: `-gdb tcp::1234`的简写

#Terminal 2
$ gdb-multiarch ~/linux-6.6-rc2/vmlinux
(gdb) target remote :1234 # 连接 qemu
(gdb) b start_kernel # 设置断点
(gdb) continue # 继续执行
(gdb) quit # 退出 gdb
```

结果：

Terminal 1启动后直接停止执行，直至Terminal 2连结qemu键入gdb continue指令后继续执行

```
frey@DESKTOP-5BJAETF:~/linux-6.6-rc2$ qemu-system-riscv64 -nographic -machine virt -kernel
 ./arch/riscv/boot/Image -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro con
sole=ttyS0" -bios default -drive file=../os23fall-stu/src/lab0/rootfs.img,format=raw,id=hd
0 -S -s
```

运行到断点时：

```
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 54
Boot HART MHPM Count      : 0
Boot HART MHPM Count      : 0
Boot HART MIDELEG         : 0x0000000000000222
Boot HART MEDELEG         : 0x000000000000b109
```

gdb调试结束后：

```
[    1.008967] VFS: Mounted root (ext4 filesyste
[    1.014474] devtmpfs: mounted
[    1.050468] Freeing unused kernel image (init
[    1.054925] Run /sbin/init as init process

Please press Enter to activate this console.
/ # _
```

Terminal 2通过gdb远程连接Terminal 1中的qemu进行调试：

```
(gdb) target remote :1234
Remote debugging using :1234
0x0000000000001000 in ?? ()
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff80a006ac
(gdb) continue
Continuing.

Breakpoint 1, 0xffffffff80a006ac in start_kernel ()
(gdb) quit
A debugging session is active.

        Inferior 1 [process 1] will be detached.

Quit anyway? (y or n) y
Detaching from program: /home/frey/linux-6.6-rc2/vmlinux, process 1
Ending remote debugging.
[Inferior 1 (process 1) detached]
```

二、思考题

1. 使用 `riscv64-linux-gnu-gcc` 编译单个 `.c` 文件

- 编写一个简单的.c文件，并拷贝至wsl

```c
#include <stdio.h>

int main(){
    int a = 1;
    int b = 2;
    int c = a + b;
    printf("%d", c);
}
```

- 使用 `riscv64-linux-gnu-gcc` 编译得到编译产物a.out：

```
frey@DESKTOP-5BJAETF:~$ riscv64-linux-gnu-gcc test.c
frey@DESKTOP-5BJAETF:~$ ls
a.out  linux-6.6-rc2  linux-6.6-rc2.tar.gz  os23fall-stu  test.c
```

2. 使用 `riscv64-linux-gnu-objdump` 反汇编 1 中得到的编译产物

```
$ riscv64-linux-gnu-objdump -d a.out  #编译指令-d显示程序可执行部分反汇编结果
```

其中main函数反汇编结果如下：

```
0000000000000668 <main>:
668:   1101                    addi    sp,sp,-32
66a:   ec06                    sd      ra,24(sp)
66c:   e822                    sd      s0,16(sp)
66e:   1000                    addi    s0,sp,32
670:   4785                    li      a5,1
672:   fef42223                sw      a5,-28(s0)
676:   4789                    li      a5,2
678:   fef42423                sw      a5,-24(s0)
67c:   fe442783                lw      a5,-28(s0)
680:   873e                    mv      a4,a5
682:   fe842783                lw      a5,-24(s0)
686:   9fb9                    addw    a5,a5,a4
688:   fef42623                sw      a5,-20(s0)
68c:   fec42783                lw      a5,-20(s0)
690:   85be                    mv      a1,a5
692:   00000517                auipc   a0,0x0
696:   02650513                addi    a0,a0,38 # 6b8 <_IO_stdin_used+0x8>
69a:   f07ff0ef                jal     ra,5a0 <printf@plt>
69e:   4781                    li      a5,0
6a0:   853e                    mv      a0,a5
6a2:   60e2                    ld      ra,24(sp)
6a4:   6442                    ld      s0,16(sp)
6a6:   6105                    addi    sp,sp,32
6a8:   8082                    ret
```

3. 调试 Linux 时：

1. 在 GDB 中查看汇编代码
   使用gdb连接qemu后，使用指令 `layout asm` 查看汇编代码

```
   0x1030  c.slli64        zero
   0x1032  unimp
   0x1034  unimp
   0x1036  unimp
   0x1038  unimp
   0x103a  .2byte  0x8020
   0x103c  unimp
   0x103e  unimp
   0x1040  nop
   0x1042  unimp
   0x1044  unimp
   0x1046  unimp
   0x1048  unimp
   0x104a  unimp
   0x104c  unimp
   0x104e  unimp
   0x1050  unimp

emote Thread 1.1 In:
```

2. 在 0x80000000 处下断点

```
(gdb) b * 0x80000000
Breakpoint 1 at 0x80000000
```

3. 查看所有已下的断点

```
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000080000000
(gdb)
```

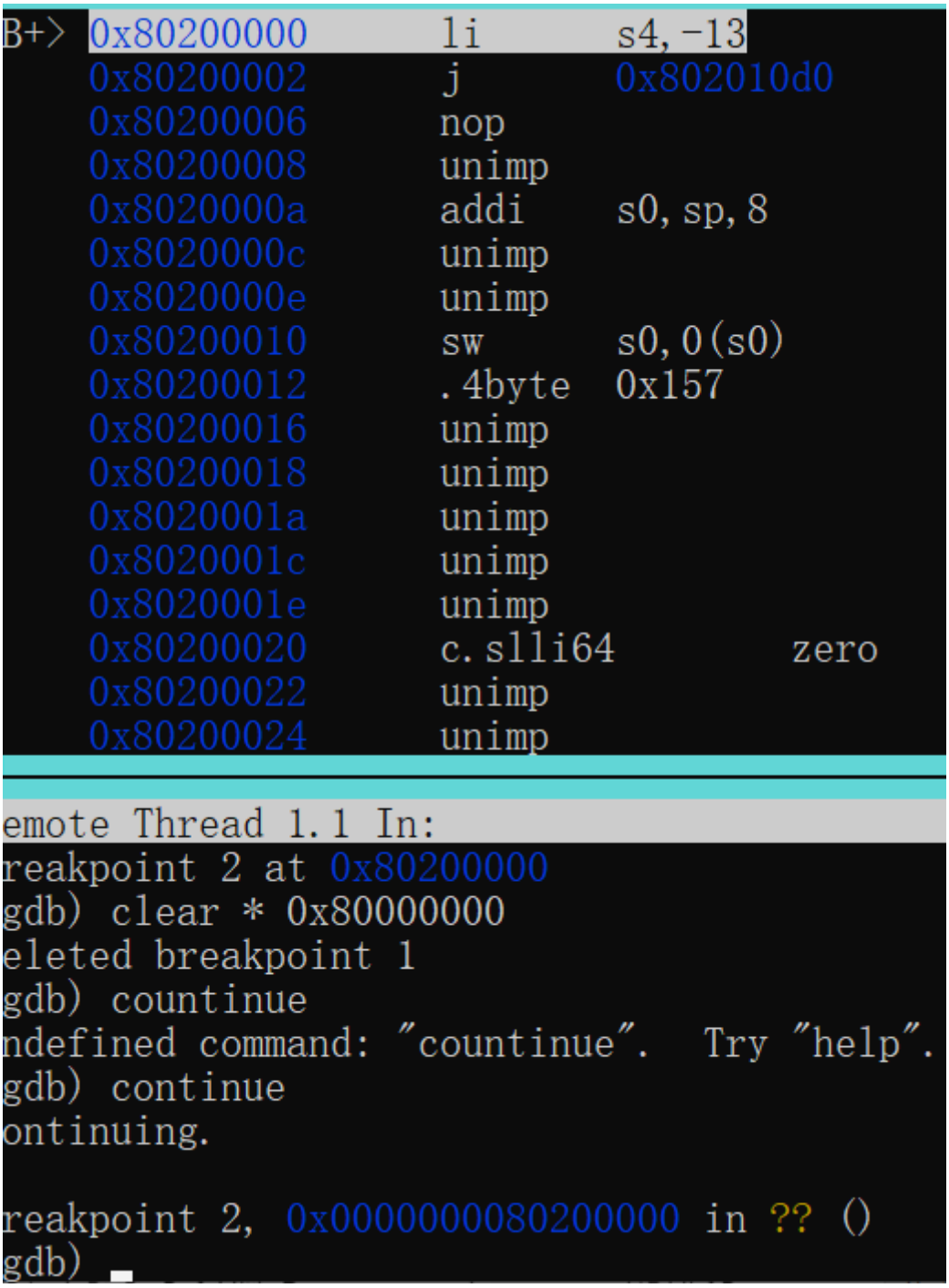
4. 在 0x80200000 处下断点

```
(gdb) b * 0x80200000
Breakpoint 2 at 0x80200000
```

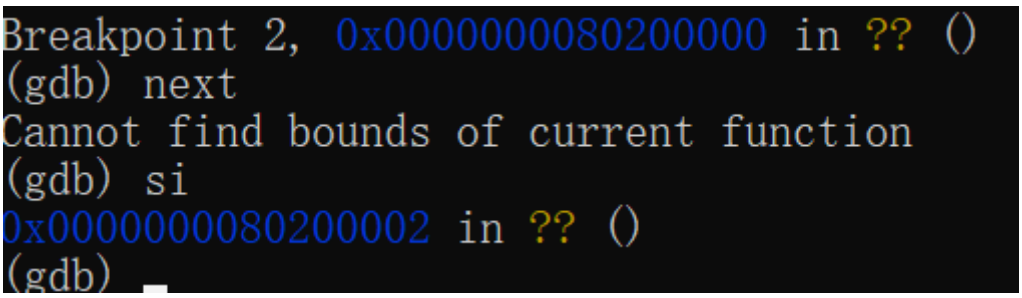
5. 清除 0x80000000 处的断点

```
(gdb) clear * 0x80000000
Deleted breakpoint 1
(gdb) info break
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x0000000080200000
```

6. 继续运行直到触发 0x80200000 处的断点



7. 单步调试一次



8. 退出 QEMU

4. 使用 `make` 工具清除 Linux 的构建产物

使用指令`$ make clean`即可清除当前文件夹下的构建产物，同时可以编写makefile中`clean:`的部分使用命令行命令指定需要清除的构建产物

5. `vmlinux` 和 `Image` 的关系和区别是什么?

- vmlinux是Linux内核编译出来的原始的内核文件，elf格式，未做压缩处理。该映像可用于定位内核问题，但不能直接引导Linux系统启动。
- Image是Linux内核编译时，使用objcopy处理vmlinux后生成的二进制内核映像。该映像未压缩，可直接引导Linux系统启动。

三、讨论心得

1. 在使用wsl，docker之类工具新建linux操作系统环境时，使用apt-get获取软件前要注意使用apt-get update更新软件列表，避免unable to locate的错误
2. 在cpu核心数量与内存大小允许时，编译时可以增加多线程编译的指令来缩短编译时间
3. 操作系统内核源码在不同的硬件体系结构下既有跨架构的相同部分，也有依赖于体系结构的代码部分，根据不同环境要选择相应的编译工具链与配置