

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：2024年9月12日

一、实验内容

1. 搭建实验环境

- 使用 WSL 2 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

SHELL

```
$ 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`

SHELL

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

- `gdb` 来对在 `qemu` 上运行的 Linux 内核进行调试

SHELL

```
$ 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 6.11-rc2，并拷贝至wsl用户目录中，并解压

SHELL

```
$ tar -zxvf linux-6.11-rc7.tar.gz -C ~
```

```
frey@DESKTOP-5BJAETF:~$ ls
c linux-6.11-rc7 linux-6.11-rc7.tar.gz
```

- 使用 git 工具 clone仓库：<https://github.com/ZJU-SEC/os24fall-stu>。其中已经准备好了根文件系统的镜像

SHELL

```
$ git clone https://github.com/ZJU-SEC/os24fall-stu.git
$ cd os24fall-stu/src/lab0
$ ls
rootfs.img # 已经构建完成的根文件系统的镜像
```

```
frey@DESKTOP-5BJAETF:~$ git clone https://github.com/ZJU-SEC/os24fall-stu.git
Cloning into 'os24fall-stu'...
remote: Enumerating objects: 78, done.
remote: Counting objects: 100% (78/78), done.
remote: Compressing objects: 100% (46/46), done.
remote: Total 78 (delta 33), reused 66 (delta 21), pack-reused 0 (from 0)
Receiving objects: 100% (78/78), 1.97 MiB | 675.00 KiB/s, done.
Resolving deltas: 100% (33/33), done.
frey@DESKTOP-5BJAETF:~$ cd os24fall-stu/src/lab0
frey@DESKTOP-5BJAETF:~/os24fall-stu/src/lab0$ ls
rootfs.img
```

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

SHELL

```
$ cd linux-6.11-rc7
$ 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-6.11-rc7$ 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=../os24fall-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：

#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=../os24fall-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-6.11-rc7/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-6.11-rc7$ qemu-system-riscv64 -nographic -machine virt -kerne
l ./arch/riscv/boot/Image -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro co
nsole=ttyS0" -bios default -drive file=../os24fall-stu/src/lab0/rootfs.img,format=raw,id=h
d0 -S -s
```

运行到断点时：

```
Boot HART ID           : 0
Boot HART Domain       : root
Boot HART ISA          : rv64imafdcu
Boot HART Features     : scounteren,mcounteren,time
Boot HART PMP Count    : 16
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 filesystem)
[ 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
0x00000000000001000 in ?? ()
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff80a00734
(gdb) continue
Continuing.

Breakpoint 1, 0xffffffff80a00734 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-6.11-rc7/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

```
1  #include <stdio.h>
2
3  int main(){
4      int a = 1;
5      int b = 2;
6      int c = a + b;
7      printf("%d", c);
8  }
```

- 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  c  linux-6.11-rc7  linux-6.11-rc7.tar.gz  os24fall-stu  test.c
```

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

SHELL

```
$ 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：

```

000000000000668 <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

```

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
(gdb)

```

6. 继续运行直到触发 0x80200000 处的断点

```

B+> 0x80200000    li    s4, -13
      0x80200002    j      0x802010d0
      0x80200006    nop
      0x80200008    unimp
      0x8020000a    addi    s0, sp, 8
      0x8020000c    unimp
      0x8020000e    unimp
      0x80200010    sw      s0, 0(s0)
      0x80200012    .4byte 0x157
      0x80200016    unimp
      0x80200018    unimp
      0x8020001a    unimp
      0x8020001c    unimp
      0x8020001e    unimp
      0x80200020    c.slli64      zero
      0x80200022    unimp
      0x80200024    unimp

```

```

emote Thread 1.1 In:
breakpoint 2 at 0x80200000
(gdb) clear * 0x80000000
deleted breakpoint 1
(gdb) countinue
undefined command: "countinue". Try "help".
(gdb) continue
continuing.

breakpoint 2, 0x0000000080200000 in ?? ()
(gdb) _

```

7. 单步调试一次

```

Breakpoint 2, 0x0000000080200000 in ?? ()
(gdb) next
Cannot find bounds of current function
(gdb) si
0x0000000080200002 in ?? ()
(gdb) _

```

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，该映像包含整个内核和 debug 标识。该映像可用于定位内核问题，但体积太大缺乏压缩不能直接引导 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。它是压缩过的系统映像，可直接引导 Linux 系统启动，在启动过程中解压释放到内存中。

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