浙江大学

本科实验报告

课程名称：操作系统

姓　　名：夏尤楷

学　　院：计算机科学与技术学院

系：计算机科学与技术系

专　　业：计算机科学与技术

学　　号：3210104331

指导教师：夏莹杰

2023 年 10 月 6 日

浙江大学操作系统实验报告

实验名称：GDB & QEMU 调试 64 位 RISC-V LINUX

电子邮件地址：3210104331@zju.edu.cn 手机：15058004449

实验地点：玉泉曹光彪西 503 实验日期：2023 年 10 月 6 日

一、实验目的和要求

1. 使用交叉编译工具，完成 Linux 内核代码编译；
2. 使用 QEMU 运行内核；
3. 熟悉 GDB 和 QEMU 联合调试。

二、实验过程

本实验使用 Ubuntu 22.04.2 LTS Windows Subsystem for Linux 2 进行。

1. 搭建实验环境

首先输入以下命令，输入后如下图：

```
1. $ sudo apt install  gcc-riscv64-linux-gnu
2. $ sudo apt install  autoconf automake autotools-dev curl libmpc-dev libm
   pfr-dev libgmp-dev \
  gawk build-essential bison flex texinfo gperf libtool patchutils bc \
  zlib1g-dev libexpat-dev git
```

```
tangkeke@DESKTOP-IOOKJCA:~$ sudo apt install  gcc-riscv64-linux-gnu
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  binutils binutils-common binutils-riscv64-linux-gnu binutils-x86-64-linux-gnu cpp-11-riscv64-linux-gnu
  cpp-riscv64-linux-gnu gcc-11-cross-base-ports gcc-11-riscv64-linux-gnu gcc-11-riscv64-linux-gnu-base
  gcc-12-cross-base-ports libasan6-riscv64-cross libatomic1-riscv64-cross libbinutils libc6-dev-riscv64-cross
  libc6-riscv64-cross libctf0 libgcc-11-dev-riscv64-cross libgcc-s1-riscv64-cross libgomp1-riscv64-cross
  linux-libc-dev-riscv64-cross
Suggested packages:
  binutils-doc gcc-11-locales cpp-doc gcc-11-doc gdb-riscv64-linux-gnu gcc-doc
The following NEW packages will be installed:
  binutils-riscv64-linux-gnu cpp-11-riscv64-linux-gnu cpp-riscv64-linux-gnu gcc-11-cross-base-ports
  gcc-11-riscv64-linux-gnu gcc-11-riscv64-linux-gnu-base gcc-12-cross-base-ports gcc-riscv64-linux-gnu
  libasan6-riscv64-cross libatomic1-riscv64-cross libc6-dev-riscv64-cross libc6-riscv64-cross
  libgcc-11-dev-riscv64-cross libgcc-s1-riscv64-cross libgomp1-riscv64-cross linux-libc-dev-riscv64-cross
The following packages will be upgraded:
  binutils binutils-common binutils-x86-64-linux-gnu libbinutils libctf0
5 upgraded, 16 newly installed, 0 to remove and 103 not upgraded.
Need to get 32.8 MB/38.2 MB of archives.
After this operation, 112 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

（第二个命令输入后的情况忘了截图了）

再输入如下命令，安装用于启动 riscv64 平台上的内核的模拟器 qemu。

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

输入后需要等待片刻才能安装完成。

最后输入如下命令，安装 gdb 来对在 qemu 上运行的 Linux 内核进行调试，如图：

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

```
tangkeke@DESKTOP-IOOKJCA:~$  sudo apt install gdb-multiarch
[sudo] password for tangkeke:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  gdb-multiarch
0 upgraded, 1 newly installed, 0 to remove and 103 not upgraded.
Need to get 4589 kB of archives.
After this operation, 18.2 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 gdb-multiarch amd64 12.1-0ubuntu1~22.04 [4589 kB]
Ign:1 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 gdb-multiarch amd64 12.1-0ubuntu1~22.04
Get:1 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 gdb-multiarch amd64 12.1-0ubuntu1~22.04 [4589 kB]
Fetched 58.5 kB in 5min 28s (178 B/s)
Selecting previously unselected package gdb-multiarch.
(Reading database ... 38636 files and directories currently installed.)
Preparing to unpack .../gdb-multiarch_12.1-0ubuntu1~22.04_amd64.deb ...
Unpacking gdb-multiarch (12.1-0ubuntu1~22.04) ...
Setting up gdb-multiarch (12.1-0ubuntu1~22.04) ...
```

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（实验期间的最新源码为 6.6-rc4 版本）的压缩包并解压，如图：

```
tangkeke@DESKTOP-IOOKJCA:~/os$ ls
linux-6.6-rc4  linux-6.6-rc4.tar.gz
```

并输入以下命令，使用 git 工具 clone 实验仓库，以获得准备好的根文件系统的镜像，如图：

```
$ git clone https://gitee.com/zju_xiayingjie/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
```

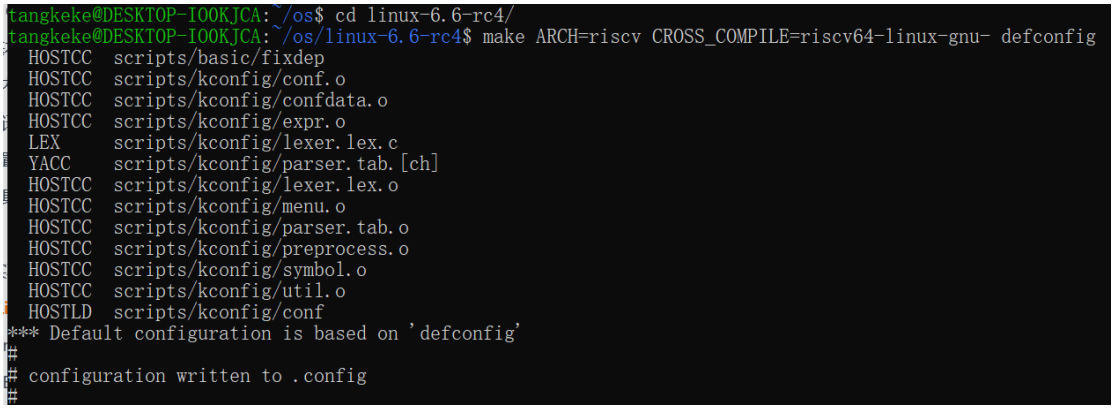
```
tangkeke@DESKTOP-IOOKJCA:~/os$ git clone https://gitee.com/zju_xiayingjie/os23fall-stu.git
Cloning into 'os23fall-stu'...
remote: Enumerating objects: 147, done.
remote: Counting objects: 100% (147/147), done.
remote: Compressing objects: 100% (121/121), done.
remote: Total 147 (delta 33), reused 97 (delta 6), pack-reused 0
Receiving objects: 100% (147/147), 1.94 MiB | 1.18 MiB/s, done.
Resolving deltas: 100% (33/33), done.
tangkeke@DESKTOP-IOOKJCA:~/os$ cd os23fall-stu/src/lab0/
tangkeke@DESKTOP-IOOKJCA:~/os/os23fall-stu/src/lab0$ ls
rootfs.img
```

3. 编译 Linux 内核

输入以下命令：

```
$ cd path/to/linux
$ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- defconfig
# 使用默认配置
$ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- -j$(nproc) # 编译
```

前两条命令执行后效果如下图：



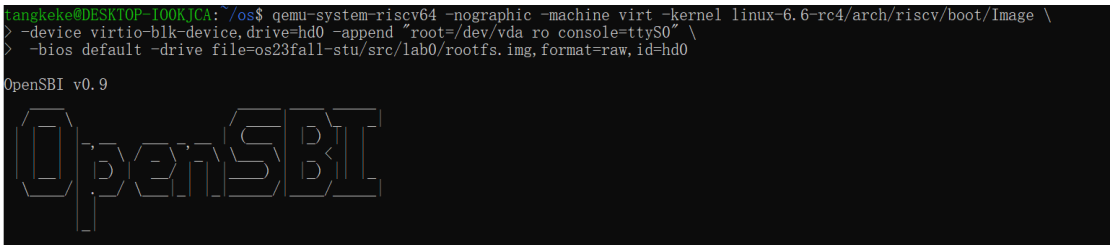
第三条命令执行需要等待较长时间。

4. 使用 QEMU 运行内核

输入以下命令：

```
1. qemu-system-riscv64 -nographic -machine virt -kernel linux-6.6-rc4/arch/riscv/boot/Image \
2.     -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0" \
3.     -bios default -drive file=os23fall-stu/src/lab0/rootfs.img,format=raw,id=hd0
```

执行后，运行内核效果如下图：



使用 Ctrl+A，松开后再按下 X 键即可退出 QEMU，如图：

5. 使用 GDB 对内核进行调试

开启两个终端，在终端一中输入以下命令，使用 QEMU 启动 Linux：

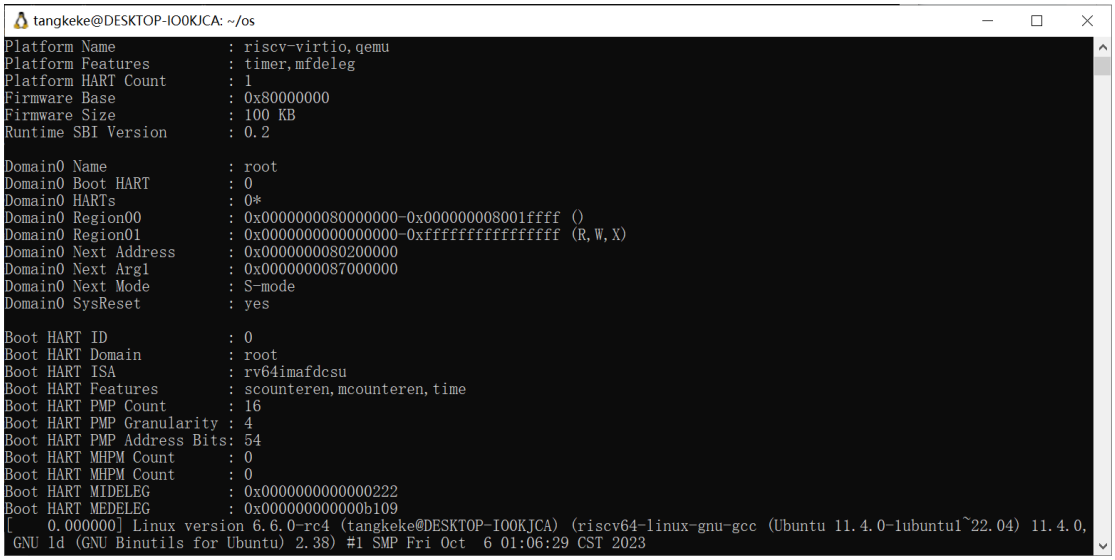
```
1. qemu-system-riscv64 -nographic -machine virt -kernel linux-6.6-rc4/arch/riscv/boot/Image \
2.     -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0" \
```

```
3.     -bios default -drive file=os23fall-stu/src/lab0/rootfs.img,format
=raw,id=hd0 -S -s
```

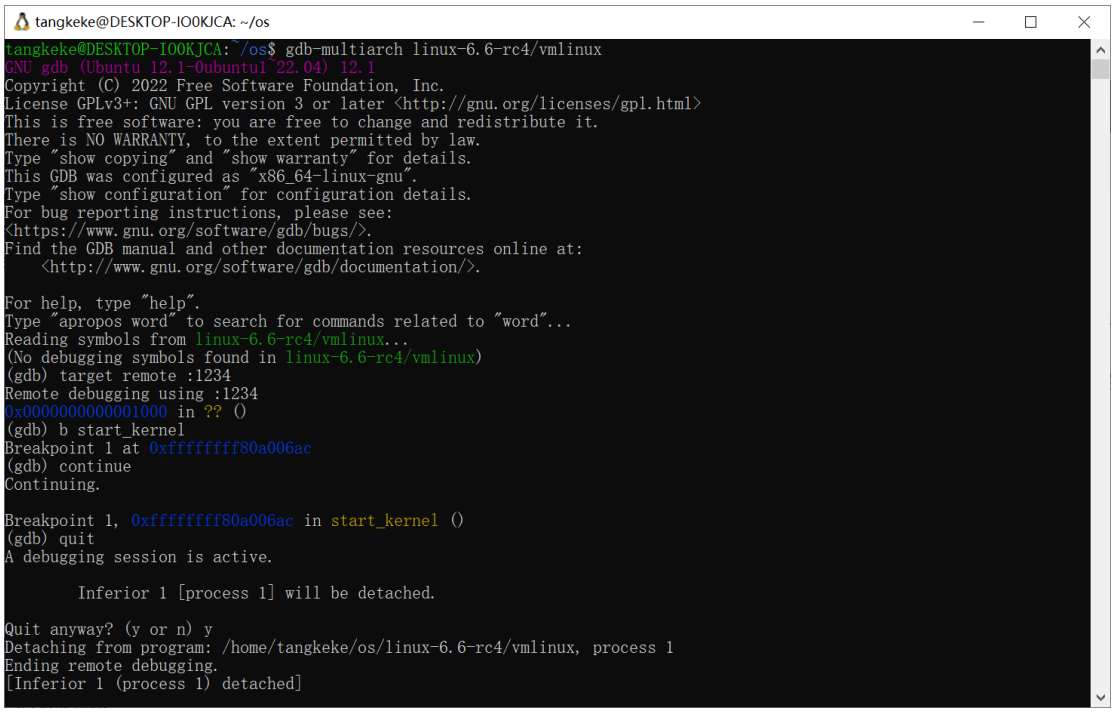
在终端二中依次输入如下命令，使用 GDB 与 QEMU 远程通信（使用 tcp::1234 端口）进行调试：

```
1.  gdb-multiarch linux-6.6-rc4/vmlinux
2.  (gdb) target remote :1234    # 连接 qemu
3.  (gdb) b start_kernel         # 设置断点
4.  (gdb) continue               # 继续执行
5.  (gdb) quit                   # 退出 gdb
```

终端一在调试期间显示如下：

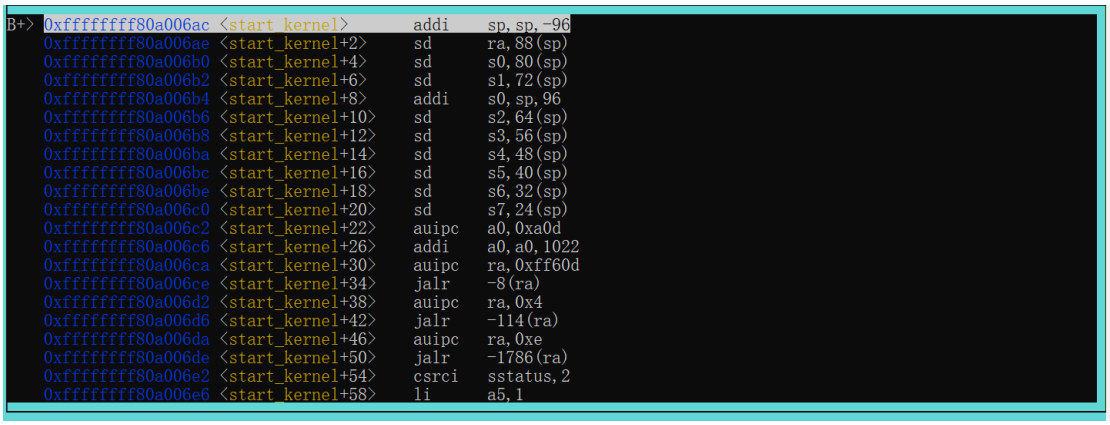


终端二在依次输入命令后显示如下：



尝试使用 GDB 的各项命令，观察 GDB 所在终端窗口变化：

使用“layout asm”，窗口上部显示出内核的汇编代码：



输入“i r ra”，显示寄存器 ra 内存储的值：

输入“backtrace”，查看函数的调用的栈帧和层级关系：

三、讨论和心得

这次实验让我对 QEMU、GDB 等工具有了初步的了解。以往对代码的调试更多地是在 IDE 这样高集成的、拥有较友好的图形界面的环境下进行，而这次使用 GDB 的调试是通过在只有纯文字的终端输入命令来进行，这样子可能麻烦了些，操作起来没那么自然了，但操作更触及程序根本，占用内存更小了。这样的调试是值得学习的。
同时，这次实验使我对 Linux 命令的使用更加熟练了。

四、思考题

1.输入 riscv64-linux-gnu-gcc <source.c> -o <file>。
2.输入 riscv64-linux-gnu-objdump <file> -d。
3.在 GDB 所在的窗口终端
（1）输入 layout asm。
（2）输入 b * 0x80000000。
（3）输入 i b。
（4）输入 b * 0x80200000。
（5）输入 clear * 0x80000000。
（6）输入 continue。
（7）输入 next。

（8）输入 quit 后，在 QEMU 所在终端依次按下 ctrl+A 和 x。

4.输入 make clean。

5.Image 是 vmlinux 被 objcopy 处理后的产物。vmlinux 是 elf 格式的，可用于定位内核问题，但不能直接引导 Linux 系统启动；Image 是二进制的，可直接引导 Linux 系统启动。

五、附录

无。