

浙江大学

操作系统实验零

课程名称:	操作系统
作业名称:	GDB & QEMU 调试 64 位 RISC-V LINUX
姓 名:	仇国智
学 号:	3220102181
电子邮箱:	3220102181@zju.edu.cn
联系电话:	13714176104
指导教师:	申文博

2024 年 9 月 10 日

一. 实验内容及简要原理

1. 实验内容

- 使用交叉编译工具, 完成 Linux 内核代码编译
- 使用 QEMU 运行内核
- 熟悉 GDB 和 QEMU 联合调试

2. 实验原理

在 Linux 环境下, 人们通常使用命令行接口来完成与计算机的交互。终端是用于处理该过程的一个应用程序, 通过终端你可以运行各种程序以及在自己的计算机上处理文件。在类 Unix 的操作系统上, 终端可以为你完成一切你所需要的操作。

QEMU 是一个功能强大的模拟器, 可以在 x86 平台上执行不同架构下的程序。我们实验中采用 QEMU 来完成 RISC-V 架构的程序的模拟。

GNU 调试器 (英语: GNU Debugger, 缩写: gdb) 是一个由 GNU 开源组织发布的、UNIX/LINUX 操作系统下的、基于命令行的、功能强大的程序调试工具。借助调试器, 我们能够查看另一个程序在执行时实际在做什么 (比如访问哪些内存、寄存器), 在其他程序崩溃的时候可以比较快速地了解导致程序崩溃的原因。被调试的程序可以和 GDB 运行在同一台机器上, 并由 GDB 控制 (本地调试 native debug)。也可以只和 gdb-server 运行在同一台机器上, 由连接着 gdb-server 的 GDB 进行控制 (远程调试 remote debug)。

GDB 基本命令介绍

- layout asm: 显示汇编代码
- start: 单步执行, 运行程序, 停在第一执行语句
- continue: 从断点后继续执行, 简写 c
- next: 单步调试 (逐过程, 函数直接执行), 简写 n
- step instruction: 执行单条指令, 简写 si
- run: 重新开始运行文件 (run-text: 加载文本文件, run-bin: 加载二进制文件), 简写 r
- backtrace: 查看函数的调用的栈帧和层级关系, 简写 bt

- break 设置断点，简写 b
- 断在 foo 函数：b foo
- 断在某地址：b * 0x80200000
- finish：结束当前函数，返回到函数调用点
- frame：切换函数的栈帧，简写 f
- print：打印值及地址，简写 p
- info：查看函数内部局部变量的数值，简写 i；查看寄存器 ra 的值：i r ra
- display：追踪查看具体变量值
- x/4x <addr>：以 16 进制打印 <addr> 处开始的 16 Bytes 内容

内核配置是用于配置是否启用内核的各项特性，内核会提供一个名为 defconfig（即 default configuration）的默认配置，该配置文件位于各个架构目录的 configs 文件夹下，例如对于 RISC-V 而言，其默认配置文件为 arch/riscv/configs/defconfig。使用 make ARCH=riscv defconfig 命令可以在内核根目录下生成一个名为.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。配置之间存在相互的依赖关系，直接修改 defconfig 文件或者.config 有时候并不能达到想要的效果，或是给进一步内核配置带来同步问题。因此如果需要修改配置一般采用 make ARCH=riscv menuconfig 的方式对内核进行配置。

make 是用于程序构建的重要工具，它的行为由当前目录或 make -C 指定目录下的 Makefile 来决定。更多有关 Makefile 的内容可以参考 Learn Makefiles With the tastiest examples。下面用本次实验中可能用到的用于编译 Linux 内核的编译命令作为示例：

```

1    $ make help # 查看make命令的各种参数解释
2
3    $ make <target-name> # 编译名为 <target-name> 的目标文件或目标任务
4    $ make defconfig #
      使用当前平台的默认配置，在x86机器上会使用x86的默认配置
5    $ make clean # 清除所有编译好的 object 文件
6    $ make mrproper # 删除所有编译产物和配置文件
7
8    $ make -j<thread-count> # 使用 <thread-count>
      个物理线程来进行多线程编译

```

```

9      $ make -j4 # 编译当前平台的内核, -j4 为使用 4 线程进行多线程编译
10     $ make -j$(nproc) # 编译当前平台的内核, -j$(nproc)
        为以全部机器硬件线程数进行多线程编译
11
12     $ make <var-name>=<var-value> # 在编译过程中将 <var-name>
        变量的值手动设置为 <val-value>
13     $ make ARCH=riscv defconfig # 使用 RISC-V 平台的默认配置
14     $ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- # 编译 RISC-V
        平台内核

```

二. 实验具体过程与代码实现

1. 搭建实验环境

首先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:

```

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

```

接着是用于启动 riscv64 平台上的内核的模拟器 qemu:

```

1      $ sudo apt install qemu

```

还需要用 gdb 来对在 qemu 上运行的 Linux 内核进行调试:

```

1      $ sudo apt install gdb-multiarch

```

```

/mnt/d/course/24_1/OS/lab0/template Py base 17:31:36
> sudo apt install gcc-riscv64-linux-gnu
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
gcc-riscv64-linux-gnu is already the newest version (4:11.2.0-1ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 169 not upgraded.

/mnt/d/course/24_1/OS/lab0/template Py base 17:31:48
> 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
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'libexpat1-dev' instead of 'libexpat-dev'
autoconf is already the newest version (2.71-2).
automake is already the newest version (1:1.16.5-1.3).
autotools-dev is already the newest version (20220109.1).
bc is already the newest version (1.07.1-3build1).
bison is already the newest version (2:3.8.2+dfsg-1build1).
build-essential is already the newest version (12.9ubuntu3).
flex is already the newest version (2.6.4-8build2).
libgmp-dev is already the newest version (2:6.2.1+dfsg-3ubuntu1).
libmpc-dev is already the newest version (1.2.1-2build1).
libmpfr-dev is already the newest version (4.1.0-3build3).
libtool is already the newest version (2.4.6-15build2).
patchutils is already the newest version (0.4.2-1build2).
gperf is already the newest version (3.1-1build1).
texinfo is already the newest version (6.8-4build1).
curl is already the newest version (7.81.0-1ubuntu1.17).
gawk is already the newest version (1:5.1.0-1ubuntu0.1).
git is already the newest version (1:2.34.1-1ubuntu1.11).
libexpat1-dev is already the newest version (2.4.7-1ubuntu0.3).
zlib1g-dev is already the newest version (1:1.2.11.dfsg-2ubuntu9.2).
0 upgraded, 0 newly installed, 0 to remove and 169 not upgraded.

/mnt/d/course/24_1/OS/lab0/template Py base 17:31:52
> sudo apt install qemu-system-misc
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
qemu-system-misc is already the newest version (1:6.2+dfsg-2ubuntu6.22).
0 upgraded, 0 newly installed, 0 to remove and 169 not upgraded.

/mnt/d/course/24_1/OS/lab0/template Py base 17:32:03
> sudo apt install gdb-multiarch
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
gdb-multiarch is already the newest version (12.1-0ubuntu1~22.04.2).
0 upgraded, 0 newly installed, 0 to remove and 169 not upgraded.

```

图 1: 安装交叉编译工具链和软件包

2. 获取 Linux 内核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。

```

1    $ wget
      https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.10.9.tar.xz
2    $ tar -xf linux-6.10.9.tar.xz

```

```
~/course/OSZJU
➤ wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.10.9.tar.xz
~2024-09-10 18:39:18~ https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.10.9.tar.xz
Resolving cdn.kernel.org (cdn.kernel.org)... 151.101.77.176, 23.235.32.32, 104.156.80.32, ...
Connecting to cdn.kernel.org (cdn.kernel.org)|151.101.77.176|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 145156944 (138M) [application/x-xz]
Saving to: 'linux-6.10.9.tar.xz'

linux-6.10.9.tar.xz      100%[=====] 138.43M  8.77MB/s   in 16s

2024-09-10 18:39:45 (8.89 MB/s) - 'linux-6.10.9.tar.xz' saved [145156944/145156944]

~/course/OSZJU
➤ tar -xvf linux-6.10.9.tar.xz

~/course/OSZJU
~/course/OSZJU
~/course/OSZJU
➤ ls
linux-6.10.9  linux-6.10.9.tar.xz  os24fall-stu
```

图 2: 下载 Linux 内核源码

使用 git 工具 clone 实验仓库。其中已经准备好了根文件系统的镜像。

```
1 $ git clone https://github.com/ZJU-SEC/os24fall-stu.git
2 $ cd os24fall-stu/src/lab0
3 $ ls
```

```
~/course/OSZJU
➤ cd os24fall-stu/src/lab0

~/course/OSZJU/os24fall-stu/src/lab0 main
➤ ls
rootfs.img
```

图 3: 下载实验仓库

3. 编译 Linux 内核

先配置内核，然后编译内核。

```
1 $ cd linux-6.10.9
2 $ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- defconfig
3 $ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- -j$(nproc)
```

```
~/course/OSZJU/linux-6.10.9
➤ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- defconfig
*** Default configuration is based on 'defconfig'
#
# No change to .config
#

~/course/OSZJU/linux-6.10.9
➤ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- -j$(nproc)
CALL scripts/checksyscalls.sh
Kernel: arch/riscv/boot/Image is ready
Kernel: arch/riscv/boot/Image.gz is ready
```

图 4: 编译 Linux 内核

4. 使用 QEMU 运行内核

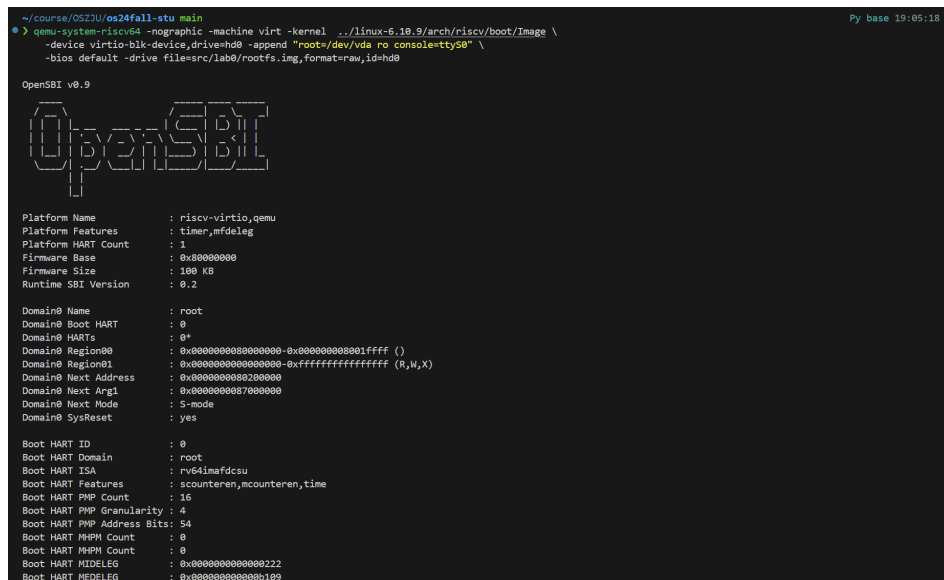
```

1  $ qemu-system-riscv64 -nographic -machine virt -kernel
    ../linux-6.10.9/arch/riscv/boot/Image \
2  -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro
    console=ttyS0" \
3  -bios default -drive file=src/lab0/rootfs.img,format=raw,id=hd0

```

参数解释：

- -nographic: 不使用图形界面
- -machine virt: 使用 virt 机器
- -kernel: 指定内核镜像
- -device virtio-blk-device,drive=hd0: 使用 virtio 块设备
- -append: 内核启动参数
- -bios default: 使用默认 BIOS
- -drive file=src/lab0/rootfs.img,format=raw,id=hd0: 使用 rootfs.img 作为硬盘



```

~/course/OS2U/os24fall-stu main
$ qemu-system-riscv64 -nographic -machine virt -kernel ../linux-6.10.9/arch/riscv/boot/Image \
-device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0" \
-bios default -drive file=src/lab0/rootfs.img,format=raw,id=hd0

OpenSBI v0.9

Platform Name      : riscv-virtio,qemu
Platform Features  : timer,mfdeleg
Platform HART Count : 1
Firmware Base      : 0x00000000
Firmware Size      : 100 KB
Runtime SBI Version : 0.2

Domain0 Name       : root
Domain0 Boot HART  : 0
Domain0 HARTs      : 0*
Domain0 Region00    : 0x0000000000000000-0x0000000000000000 ()
Domain0 Region01    : 0x0000000000000000-0xffffffffffffffff (R,W,X)
Domain0 Next Address : 0x0000000000000000
Domain0 Next Arg1    : 0x0000000000000000
Domain0 Next Mode    : S-mode
Domain0 SysReset     : yes

Boot HART ID       : 0
Boot HART Domain    : root
Boot HART ISA       : rv64imafdcsv
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    : 0x0000000000000109

```

图 5: 使用 QEMU 运行内核

5. 使用 GDB 对内核进行调试

在一个终端中运行 QEMU，另一个终端中运行 gdb-multiarch。

terminal 1:

```
1 $ qemu-system-riscv64 -nographic -machine virt -kernel
  ../linux-6.10.9/arch/riscv/boot/Image \
2 -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro
  console=ttyS0" \
3 -bios default -drive file=src/lab0/rootfs.img,format=raw,id=hd0
  -S -s
```

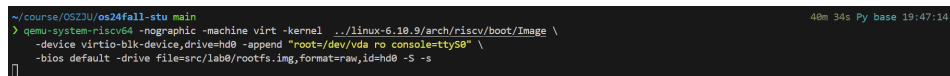


图 6: 在 QEMU 中启用 GDB 调试启动 linux 内核

terminal 2:

```
1 $ gdb-multiarch ../linux-6.10.9/vmlinux
2 (gdb) target remote :1234 # 连接到 QEMU
3 (gdb) b start_kernel # 设置断点
4 (gdb) continue # 继续执行
5 (gdb) quit # 退出 gdb
```



```
~/course/OSZJU/os24fall-stu main
gdb-multiarch ./linux-6.10.9/vmlinux
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04~3) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ../linux-6.10.9/vmlinux...
(No debugging symbols found in ../linux-6.10.9/vmlinux)
(gdb) target remote :1234
Remote debugging using :1234
0xffffffff80a00734 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)
Please answer y or n.
A debugging session is active.

Inferior 1 [process 1] will be detached.

Quit anyway? (y or n) y
Detaching from program: /home/morretti/course/OSZJU/linux-6.10.9/vmlinux, process 1
Ending remote debugging.
[Inferior 1 (process 1) detached]

~/course/OSZJU/os24fall-stu main
Py base 19:52:05
22s Py base 19:52:30
```

图 7: 使用 GDB 对内核进行调试

三. 实验结果与分析

实验结果:

```
Register group: general
r0 0x00000000 r1 0xffffffff80a00011 sp 0xffffffff81403e gp 0xffffffff8150d1 tp 0xffffffff8140cc
r2 0x00000000 r3 0xffffffff80a00012 t2 0x1000 4096 fp 0x00017f20 p1 0x1 1
r4 0xffffffff80a00013 r5 0x0 0 a2 0xffffffff80a0479 a3 0xffffffff80a067 a4 0xffffffff80a055
r6 0xffffffff80a00014 r7 0xffffffff80a00015 a6 0xffffffff80a068 a7 0xffffffff80a055 a2 0x02000000 s3 0x87000000
r8 0xffffffff80a00016 r9 0x0 0 s4 0x00000000 s5 0x0 0 s6 0x80000000 s7 0x800120e8 s8 0x80013100
r10 0x7f 127 s10 0x0 0 s11 0x0 0 t3 0x80808080 s9 0x0 0 t4 0x2 2
r11 0x27 39 t6 0x0 0 pc 0xffffffff80a00734

B> 0xffffffff80a00734 <start_kernel> addi sp,sp,-96
0xffffffff80a00736 <start_kernel+2> sd ra,88(sp)
0xffffffff80a00738 <start_kernel+4> sd s0,80(sp)
0xffffffff80a0073a <start_kernel+6> sd s1,72(sp)
0xffffffff80a0073c <start_kernel+8> addi s0,s0,96
0xffffffff80a0073e <start_kernel+10> sd s2,64(sp)
0xffffffff80a00740 <start_kernel+12> sd s3,56(sp)
0xffffffff80a00742 <start_kernel+14> sd s4,48(sp)
0xffffffff80a00744 <start_kernel+16> sd s5,40(sp)
0xffffffff80a00746 <start_kernel+18> sd s6,32(sp)
0xffffffff80a00748 <start_kernel+20> sd s7,24(sp)
0xffffffff80a0074a <start_kernel+22> sd s8,16(sp)
0xffffffff80a0074c <start_kernel+24> auipc a0,0xa0c
0xffffffff80a00750 <start_kernel+28> addi a0,a0,1332
0xffffffff80a00754 <start_kernel+32> auipc ra,0xfff610
0xffffffff80a00758 <start_kernel+36> jalr -634(ra)

Remote Thread 1:1 In: start_kernel
(gdb) frame
#0 0x00000000000001004 in ?? ()
(gdb) c\
c
Undefined command: "cc". Try "help".
(gdb) b start_kernel
Note: breakpoint 1 also set at pc 0xffffffff80a00734.
Breakpoint 2 at 0xffffffff80a00734
(gdb) c
Continuing.

Breakpoint 1, 0xffffffff80a00734 in start_kernel ()
(gdb) frame
#0 0xffffffff80a00734 in start_kernel ()
(gdb) n
Single stepping until exit from function start_kernel,
which has no line number information.
```

图 8: 使用 GDB 对内核进行调试

使用了 GDB 对内核进行调试，尝试了一些 GDB 的基本命令，如设置断点、查看寄存器等。

四. 遇到的问题及解决方法

问题：在使用 apt 安装软件包时报错解决方法：使用 apt update 更新软件源

五. 总结与心得

通过本次实验，我学会了如何使用交叉编译工具链和 QEMU 运行内核，熟悉了 GDB 的基本命令。在执行 QEMU 命令时，了解了如何指定内核的启动参数，如何指定内核的启动地址，明白了内核的镜像文件和虚拟硬盘文件的作用。在使用 GDB 调试内核时，学会了如何设置断点、查看寄存器、查看内存等操作。。通过本次实验，我对操作系统的底层编译和调试有了更深入的了解，对操作系统的运行机制有了更深刻的认识。

六. 思考题

1. 使用 riscv64-linux-gnu-gcc 编译单个.c 文件

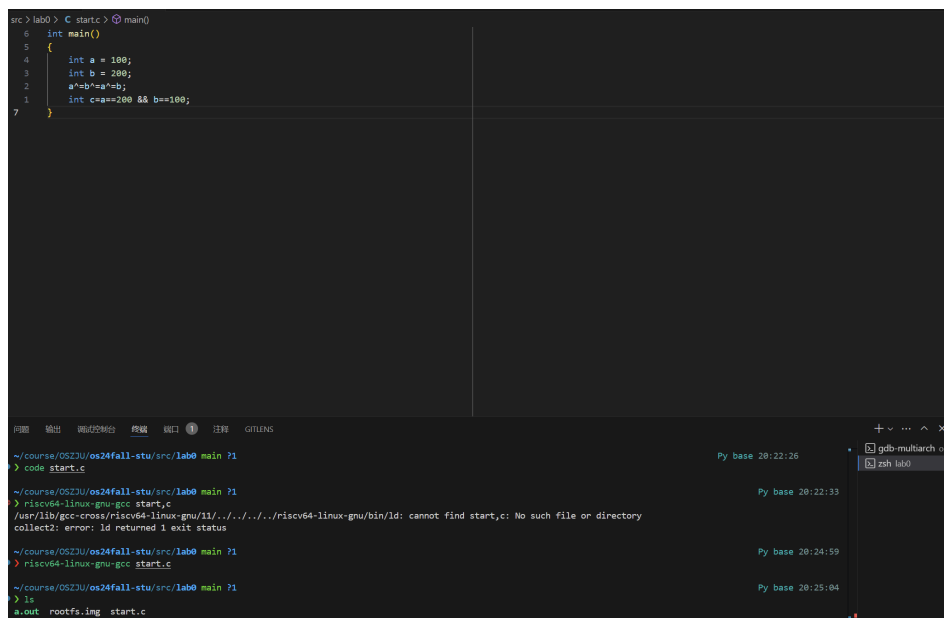


图 9: 使用 riscv64-linux-gnu-gcc 编译单个.c 文件

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

使用命令:

```
1 $ riscv64-linux-gnu-objdump -d a.out
```

```

39 00000000000005ec <__do_global_ctors_aux>:
21 620: 6402          ld s0,0(sp)
20 622: 0141          addi sp,sp,16
19 624: 8082          ret
18
17 0000000000000626 <frame_dummy>:
16 626: bf69          j 5c0 <register_tm_clones>
15
14 0000000000000628 <main>:
13 628: 1101          addi sp,sp,-32
12 62a: ec22          sd s0,24(sp)
11 62c: 1000          addi s0,sp,32
10 62e: 06400793      li a5,100
9 632: fef42223      sw a5,-28(s0)
8 636: 0c800793      li a5,200
7 63a: fef42423      sw a5,-24(s0)
6 63e: fe442783      lw a5,-28(s0)
5 642: 873e          mv a4,a5
4 644: fe842783      lw a5,-24(s0)
3 648: 8fb9          xor a5,a5,a4
2 64a: fef42223      sw a5,-28(s0)
1 64e: fe442783      lw a5,-28(s0)
113 652: fe842703      lw a4,-24(s0)
1 656: 8fb9          xor a5,a5,a4
2 658: fef42423      sw a5,-24(s0)
3 65c: fe842783      lw a5,-24(s0)
4 660: fe442703      lw a4,-28(s0)
5 664: 8fb9          xor a5,a5,a4
6 666: fef42223      sw a5,-28(s0)
7 66a: fe442783      lw a5,-28(s0)
8 66e: 0007871b      sext.w a4,a5
9 672: 0c800793      li a5,200
10 676: 00f71c63      bne a4,a5,68e <main+0x66>
11 67a: fe842783      lw a5,-24(s0)
12 67e: 0007871b      sext.w a4,a5
13 682: 06400793      li a5,100
14 686: 00f71463      bne a4,a5,68e <main+0x66>
15 68a: 4785          li a5,1
16 68c: a011          j 690 <main+0x68>
17 68e: 4781          li a5,0
18 690: fef42623      sw a5,-20(s0)
19 694: 4781          li a5,0
20 696: 853e          mv a0,a5
21 698: 6462          ld s0,24(sp)
22 69a: 6105          addi sp,sp,32
23 69c: 8082          ret
24

```

图 10: 使用 riscv64-linux-gnu-objdump 反汇编编译产物，略去前面的启动代码

3. 按如下要求调试 Linux

- 在 GDB 中查看汇编代码（不使用任何插件的情况下）
- 在 0x80000000 处下断点
- 查看所有已下的断点
- 在 0x80200000 处下断点

- 清除 0x80000000 处的断点
- 继续运行直到触发 0x80200000 处的断点
- 单步调试一次
- 退出 QEMU
- 使用 make 工具清除 Linux 的构建产物

回答：使用如下指令进行调试：

```

1  $ gdb-multiarch ../linux-6.10.9/vmlinux
2  (gdb) target remote :1234
3  (gdb) layout asm
4  (gdb) b *0x80000000
5  (gdb) info breakpoints
6  (gdb) b *0x80200000
7  (gdb) clear *0x80000000
8  (gdb) continue
9  (gdb) stepi
10 (gdb) quit

```

The screenshot shows a GDB terminal window with the following content:

```

> 0x1000 auipc    t0,0x0
0x1004 addi      a2,t0,40
0x1008 csrr      a0,mhartid
0x100c ld        a1,32(t0)
0x1010 ld        t0,24(t0)
0x1014 jr        t0
0x1018 unimp
0x101a .2byte    0x8000
0x101c unimp
0x101e unimp
0x1020 unimp
0x1022 .2byte    0x8700
0x1024 unimp
0x1026 unimp
0x1028 fmaddd.s    ft6,ft4,fs4,fs1,unknown
0x102c unimp
0x102e unimp
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

```

Below the assembly code, the terminal shows the following GDB commands and output:

```

remote Thread 1.1 In:
(gdb) b *0x80000000
Breakpoint 1 at 0x80000000
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint       keep y   0x0000000000000000
(gdb) b *0x80200000
Breakpoint 2 at 0x80200000
(gdb) clear *0x80000000
Deleted breakpoint 1
(gdb)

```

图 11: 在 GDB 中查看汇编代码、设置断点、查看断点、清除断点

```

0x80200000  li    s4,-13
> 0x80200002  j      0x80201000
0x80200006  nop
0x80200008  unimp
0x8020000a  addi   s0,sp,8
0x8020000c  unimp
0x8020000e  unimp
0x80200010  sw     s0,0(s0)
0x80200012  addi   a4,sp,132
0x80200014  unimp
0x80200016  unimp
0x80200018  unimp
0x8020001a  unimp
0x8020001c  unimp
0x8020001e  unimp
0x80200020  c.slli64      zero
0x80200022  unimp
0x80200024  unimp
0x80200026  unimp
0x80200028  unimp
0x8020002a  unimp
0x8020002c  unimp
0x8020002e  unimp
0x80200030  lw     s2,20(sp)
0x80200032  fadd.s  ft6,ft2,ft5,rmm
0x80200036  unimp
0x80200038  lw     t1,52(sp)
0x8020003a  fmadd.s fa0,ft0,ft4,ft0,rne
0x8020003e  unimp
0x80200040  lw     a2,12(a0)
0x80200042  unimp
0x80200044  lw     s1,100(s0)

Remote Thread 1.1 In:
(gdb) b *0x80000000
Breakpoint 1 at 0x80000000
(gdb) info breakpoints
Num   Type      Disp Enb Address      What
1     breakpoint keep y   0x0000000000000000
(gdb) b *0x80200000
Breakpoint 2 at 0x80200000
(gdb) clear *0x80000000
Deleted breakpoint 1
(gdb) continue
Continuing.

Breakpoint 2, 0x0000000000200000 in ?? ()
(gdb) stepi
0x0000000000200002 in ?? ()
(gdb)

```

图 12: 在 GDB 中继续运行、单步调试

```

~/course/OS2JU/os24fall-stu main ??
gdb-multiarch ../linux-6.10.9/vmlinux
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ../linux-6.10.9/vmlinux...
(No debugging symbols found in ../linux-6.10.9/vmlinux)
(gdb) target remote :1234
Remote debugging using :1234
0x0000000000010000 in ?? ()
(gdb) layout asm
Detaching from program: /home/morretti/course/OS2JU/linux-6.10.9/vmlinux, process 1
Ending remote debugging.
[Inferior 1 (process 1) detached]

~/course/OS2JU/os24fall-stu main ??

```

图 13: 退出 QEMU

4. 使用 make 工具清除 Linux 的构建产物

使用命令:

```
1 $ make mrproper
```

```
~/course/OS2JU/os24fall-stu main ?2 2m 59s Py base 20:49:21
> cd ../linux-6.10.9

~/course/OS2JU/linux-6.10.9 Py base 20:51:03
> make mrproper
CLEAN drivers/firmware/efi/libstub
CLEAN drivers/gpu/drm/radeon
CLEAN drivers/scsi
CLEAN drivers/tty/vt
CLEAN init
CLEAN kernel
CLEAN lib/raid6
CLEAN lib
CLEAN security/apparmor
CLEAN security/selinux
CLEAN usr
CLEAN .
CLEAN modules.builtin modules.builtin.modinfo .vmlinux.export.c
CLEAN scripts/basic
CLEAN scripts/dtc
CLEAN scripts/kconfig
CLEAN scripts/mod
CLEAN scripts/selinux/genheaders
CLEAN scripts/selinux/mdp
CLEAN scripts
CLEAN include/config include/generated .config .version Module.symvers
```

图 14: 使用 make 工具清除 Linux 的构建产物

5. vmlinux 和 Image 的关系和区别是什么？

vmlinux 是包含完整调试符号信息的未压缩内核映像，主要用于开发和调试。Image 是经过处理的内核映像，去除了 vmlinux 调试符号信息，用于实际的内核运行，体积更小。