# 实验 3 — GDB+QEMU 调试 64 位 RISC-V LINUX

姓名:	汤尧	学号:	3200106252	学院:	计算机学院	
课程名称:	计算机系统II		同组学生姓名:	无		
实验时间:	周四 3, 4, 5 节设	果 实验地点:	紫金港机房	指导老师:	卢立、申文博	

## 一、 实验目的和要求

- 1. 了解容器的使用
- 2. 使用交叉编译工具,完成 Linux 内核代码编译
- 3. 使用 QEMU 运行内核
- 4. 熟悉 GDB 和 QEMU 联合调试

## 二、 实验内容和原理

### 2.1 实验内容

搭建 Docker 环境,编译 linux 内核,使用 QEMU 运行内核,使用 GDB 对内核进行调试。

### 2.2 设计模块

### 2.2.1 linux 使用基础

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(Terminal)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。

### 2.2.2 Docker 使用基础

Docker 是一种利用容器(container)来进行创建、部署和运行应用的工具。Docker 把一个应用程序运行需要的二进制文件、运行需要的库以及其他依赖文件打包为一个包(package),然后通过该包创建容器并运行,由此被打包的应用便成功运行在了 Docker 容器中。之所以要把应用程序打包,并以容器的方式运行,主要是因为在生产开发环境中,常常会遇到应用程序和系统环境变量以及一些依赖的库文件不匹配,导致应用无法正常运行的问题。Docker 带来的好处是只要我们将应用程序打包完成(组装成为 Docker imgae),在任意安装了 Docker 的机器上,都可以通过运行容器的方式来运行该应用程序,因而将依赖、环境变量等带来的应用部署问题解决了。

### 2.2.3 linux 内核编译基础

### 交叉编译

交叉编译指的是在一个平台上编译可以在另一个架构运行的程序。例如在 x86 机器上编译可以在 RISC-V 架构运行的程序,交叉编译需要交叉编译工具链的 支持,在我们的实验中所用的交叉编译工具链就是 riscv-gnu-toolchain。

#### 内核配置

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。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。

### 常见参数

- ARCH 指定架构,可选的值包括 arch 目录下的文件夹名,如 x86、arm、arm64 等,不同于 arm 和 arm64,32 位和 64 位的 RISC-V 共用 arch/riscv 目录,通过使用不同的 config 可以编译 32 位或 64 位的内核。
- CROSS\_COMPILE 指定使用的交叉编译工具链,例如指定 CROSS\_COMPILE=riscv64-unknown-linux-gnu-,则编译时会采用 riscv64-unknown-linux-gnu-gcc 作为编译器,编译可以在 RISC-V 64 位平台上运行的 kernel。

### 2.2.4 GDB 使用基础

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。 被调试的程序可以是和 gdb 在同一台机器上(本地调试,or native debug),也可以是不同机器上(远程调试, or remote debug)。

总的来说, qdb 可以有以下 4 个功能:

- 启动程序,并指定可能影响其行为的所有内容
- 使程序在指定条件下停止
- 检查程序停止时发生了什么
- 更改程序中的内容,以便纠正一个 bug 的影响

## 三、 主要仪器设备

• Ubuntu 虚拟机

## 四、 操作方法与实验步骤

按实验指导,同实验结果的步骤。

## 五、 实验结果与分析

#### 4.1 搭建 Docker 环境

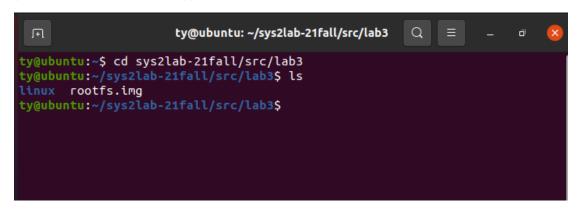
创建新的 docker, 把用户的 home 目录映射到 docker 镜像内的 have-fundebugging 目录。经实验,发现 docker 镜像中的文件和本地完全同步,在本地对这个目录进行改动 docker 中也会相应变动。

遇到问题: docker 不能重名,经过几次试着输入,现在有几个 docker 有叫 oslab, oslab0, oslab1.

```
riter |buntu:~/linux-5.15$ docker run --name oslab1 -it -v ${
HOME}:/have-fun-debugging oslab:2021 bash
root@f7bf9d5e9abb:/# ls
      have-fun-debugging
                            lib64
                                    opt
bin
                                                  root
                            libx32
boot
                                                  run
                                                         tmp
                                    ргос
dev
      lib
                            media
                                                  sbin
      lib32
etc
                                                         var
root@f7bf9d5e9abb:/#
```

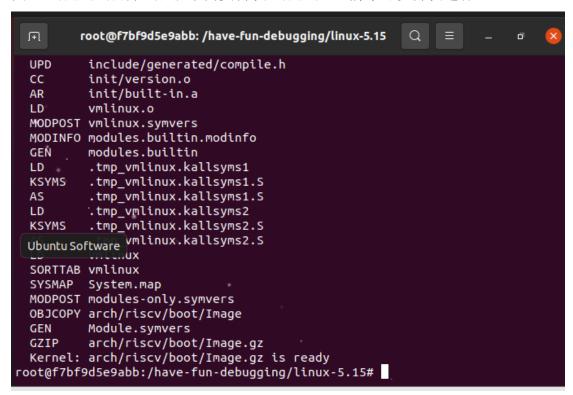
#### 4.2 获取 Linux 源码和已经编译好的文件系统

遇到问题:从官网下载 linux 源码是压缩了两层的版本,需要进行两次解压才能得到可用的 linux-5.15 文件夹。



#### 4.3 编译 linux 内核

在 docker 镜像的路径下编译,因为第一遍生成配置然后编译成功了但是忘记截图,生成配置成功之后就不需要再次生成配置。编译可以再次进行。

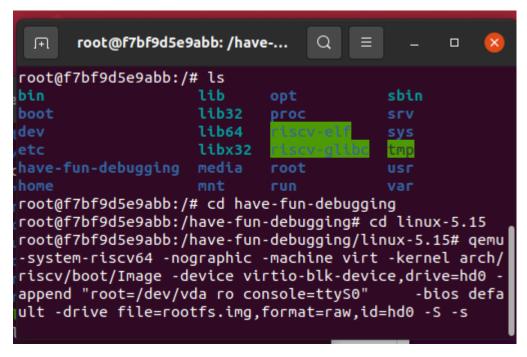


#### 4.4 使用 QEMU 运行内核

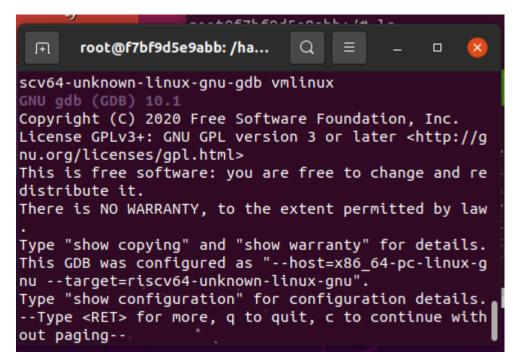
遇到问题: path/to/······为相对路径非绝对路径。在已经进入该文件夹时就不需要再重复该文件夹的路径。

```
root@f7bf9d5e9abb: /have-fun-debugging/linux-5.15
     2.067642] sdhci-pltfm: SDHCI platform and OF driver helper
     2.075023] usbcore: registered new interface driver usbhid
     2.080684] usbhid: USB HID core driver
     2.090492] NET: Registered PF_INET6 protocol family
    2.138413] Segment Routing with IPv6
    2.141403] In-situ OAM (IOAM) with IPv6
     2.154038] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
     2.187532] NET: Registered PF_PACKET protocol family
     2.206766] 9pnet: Installing 9P2000 support
     2.221248] Key type dns_resolver registered
 Rhythmbox 15] debug_vm_pgtable: [debug_vm_pgtable
                                                       ]: Validating a
renttecture page table helpers
     2.515167] EXT4-fs (vda): mounted filesystem with ordered data mode. O
pts: (null). Quota mode: disabled.
     2.519749] VFS: Mounted root (ext4 filesystem) readonly on device 254:
0.
    2.570822] devtmpfs: mounted
     2.665570] Freeing unused kernel image (initmem) memory: 2144K
     2.689819] Run /sbin/init as init process
Please press Enter to activate this console.
```

使用 QEMU 启动 linux

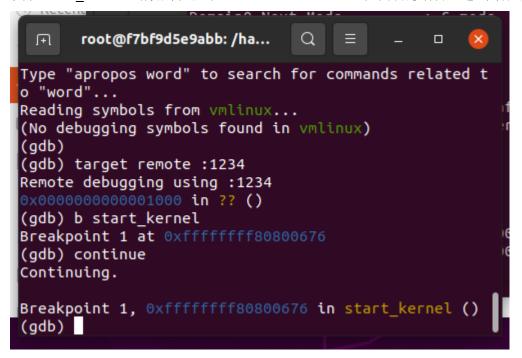


使用 GDB 和 QEMU 远程通信



连接 qemu,设置断点,执行,退出 gdb。

发现 start kernel 函数的位置在 0xfffffff80800676,在开始执行后遇到断点。



## 六、 思考题

一、 编译单个 c 文件



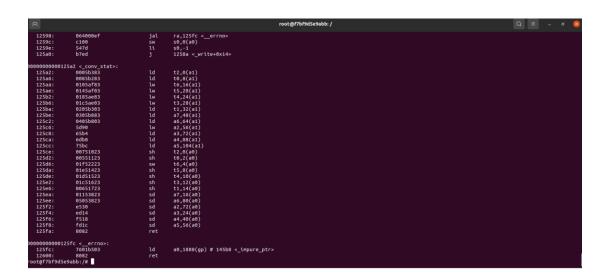
touch test.c //新建文件

vi test.c //编辑代码

Esc+Shift:+输入 qw //退出编辑

riscv64-unknown-elf-gcc -o test test.c //编译 test.c 文件

二、 使用 riscv64-unknown-elf-objdump 反汇编 1 中得到的编译产物



输入 riscv64-unknown-elf-gcc -d objdump test 查看汇编。

- 三、 调试 linux
- 1. 查看汇编代码

```
root@f7bf9d5e9abb: /ha...
                                 Q
  FI.
                                                0xffffffff80800672 <trap_init+8>
                                              addi
    0xffffffff80800674 <trap_init+10>
                                              ret
 B+>0xffffffff80800676 <start_kernel>
                                              addi
    0xffffffff80800678 <start_kernel+2>
                                              sd
    0xffffffff8080067a <start_kernel+4>
                                              sd
    0xffffffff8080067c <start_kernel+6>
                                              sd
    0xffffffff8080067e <start_kernel+8>
                                              addi
    start kernel
                              PC: 0xffffffff80800676
                        L??
Help
```

2. 设置断点并查看

```
Q = - #
                                             t0,0x0
     >0x1000
                         auipc
                                             a2,t0,40
a0,mhartid
                         addi
      0x1004
0x1008
                         csrr ad
ld a1,32(t0)
ld t0,24(t0)
jr t0
unimp
0x8000
                          unimp
                          unimp
                          0x8700
remote Thread 1.1 In:
(gdb) b * 0x80200000
Breakpoint 2 at 0x80200000
(gdb) info break
                                                                                                                                                    L?? PC: 0x1000
           Type
breakpoint
                                     Disp Enb Address
                                                                                  What
Num
                                     keep y 0x000000080000000
keep y 0x0000000080200000
            breakpoint
(gdb)
```

3. 单步执行

```
root@f7bf9d5e9abb: /have-fun-debugging/linux-5.15
                                      t0,0x0
a2,t0,40
                      auipc
                      addi
                                       a0,mhartid
                     ld a1,32(t0)
ld t0,24(t0)
jr t0
unimp
0x8000
                      unimp
                      unimp
                      unimp
0x8700
     te Thread 1.1 In:
                                                                                                                              L?? PC: 0x1008
          breakpoint
                               keep y
          breakpoint
(gdb) si
            0000001004 in <mark>?? ()</mark>
(gdb) si
          90000001008 in ?? ()
(gdb)
```

#### 4. 清除断点

```
auipc
addi
                                         t0,0x0
a2,t0,40
a0,mhartid
     >0x1008
                       CSTT
                       ld a1,32(t0)
ld t0,24(t0)
jr t0
unimp
0x8000
                        unimp
                       unimp
                       unimp
                       0x8700
remote Thread 1.1 In:
                                                                                                                                      L?? PC: 0x1008
(gdb) start
The "remote" target does not support "run". Try "help target" or "continue".
(gdb) delete 1
(gdb) info break
                                 Disp Enb Address
keep y 0x0000000080200000
           Type
breakpoint
                                                                           What
(gdb)
```

5. 继续运行到断点停止,在断点停止时不能单步执行。

#### 6. 退出 QEMU

### 四、 使用 make 工具清除 Linux 的构建产物

```
CLEAN drivers/firms/e/fillstub
```

### 五、 vmlinux 和 lmage 的关系和区别是什么?

vmlinuz 是可引导的、可压缩的内核镜像, vm 代表 Virtual Memory.Linux 支持虚拟内存,因此得名 vm.它是由用户对内核源码编译得到,实质是 elf 格式的文件.也就是说, vmlinux 是编译出来的最原始的内核文件,未压缩.这种格式的镜像文件多存放在 PC 机上.

Image 是经过 objcopy 处理的只包含二进制数据的内核代码,它已经不是 elf 格式了,但这种格式的内核镜像还没有经过压缩.