浙江大学实验报告

> 课程名称: 操作系统分析及实验
> 实验类型: 综合型/设计型
> 实验项目名称:docker+GDB+QEMU调试64位RISC-V LINUX
> 学生姓名: 李云帆
> 专业: 计算机科学与技术
> 学号:3200102555
> 电子邮件地址: **3200102555@zju.edu.cn**
> 手机: 17300988837

一 实验目的

搭建虚拟机、docker环境。通过在QEMU上运行Linux来熟悉如何从源代码开始将内核运行在QEMU模拟器上，并且掌握使用gdb协同QEMU进行联合调试，为后续实验打下基础。

二 实验内容

在虚拟机环境运行docker，并且安装：

· VMware Workstation 15 Player

· Ubuntu 18.04.5 LTS

· docker

· qemu 5.0.0

· riscv-gnu-toolchain

可能还需要安装：

· buildroot 2020.08-rc1

· linux 5.4.0-77-generic

三 主要仪器设备

windows11系统计算机

四 操作方法和实验步骤

1 搭建Docker环境

因为在ubuntu上的docker环境搭建并进入后, 已经加入系统变量的`riscv-glibc`无法正常识别导致`make`找不到`defconfig`, 因此我改用了在windows11上直接安装docker desktop进行实验

2 获取Linux源码和已经编译好的文件系统

从**链接**拷贝仓库

3 编译Linux内核

1. 将oslab.tar导入docker并查看

```
➜ Downloads  docker import '.\oslab (1).tar' oslab:2022
sha256:c48b0ce2d6e955258692b72c12dd49c08825fcf9f1a0c2ad9241f8ecc9efd280
➜ Downloads  docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
oslab        2022      c48b0ce2d6e9   3 minutes ago   2.89GB
ubuntu       18.04     35b3f4f76a24   2 weeks ago     63.1MB
ubuntu       22.04     2dc39ba059dc   3 weeks ago     77.8MB
ubuntu       latest    2dc39ba059dc   3 weeks ago     77.8MB
➜ Downloads
```

2. 从镜像创建一个容器并进入该容器, 退出后查看所有容器, 再次启动并进入

```
➜ Downloads  docker run --name oslab -it oslab:2022 /bin/bash
root@2fa48ba4b3f8:/# exit
exit
➜ Downloads  docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
➜ Downloads  docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS
   PORTS     NAMES
2fa48ba4b3f8   oslab:2022     "/bin/bash"   31 seconds ago   Exited (0) 9 seconds ago
            oslab
13e49bbd3310   ubuntu:22.04   "bash"        8 days ago       Exited (255) 7 minutes ago
            tender_easley
➜ Downloads  docker start oslab
oslab
➜ Downloads  docker ps
```

```
CONTAINER ID   IMAGE         COMMAND       CREATED              STATUS         PORTS
NAMES
2fa48ba4b3f8   oslab:2022    "/bin/bash"   About a minute ago   Up 5 seconds
 oslab
➜ Downloads  docker exec -it oslab bansh
OCI runtime exec failed: exec failed: unable to start container process: exec: "bansh":
executable file not found in $PATH: unknown
➜ Downloads  ✖ docker exec -it oslab bash
root@2fa48ba4b3f8:/# ls
bin   dev  home     lib    libexec  mnt  proc  run   share  sys  usr
boot  etc  include  lib64  media    opt  root  sbin  srv    tmp  var
root@2fa48ba4b3f8:/#
```

3. 编译Linux内核

```
root@2fa48ba4b3f8:/home/oslab/lab0# make -C linux O=/home/oslab/lab0/build/linux/
CROSS_COMPILE=riscv64-unknown-linux-gnu- ARCH=riscv CONFIG_DEBUG_INFO=y defconfig all -
j$(nproc)
```

编译完成后信息如下:

```
  ...
  LD      vmlinux
  SYSMAP  System.map
  MODPOST Module.symvers
  OBJCOPY arch/riscv/boot/Image
  CC [M]  fs/nfs/flexfilelayout/nfs_layout_flexfiles.mod.o
  GZIP    arch/riscv/boot/Image.gz
  LD [M]  fs/nfs/flexfilelayout/nfs_layout_flexfiles.ko
  Kernel: arch/riscv/boot/Image.gz is ready
make[1]: Leaving directory '/home/oslab/lab0/build/linux'
make: Leaving directory '/home/oslab/lab0/linux'
```

4 使用QEMU运行内核

```
/home/oslab/lab0# qemu-system-riscv64 -nographic -machine virt -kerne
l build/linux/arch/riscv/boot/Image -device virtio-blk-device,drive=hd0 -append "root=/
dev/vda ro console=ttyS0" -bios default -drive file=rootfs.ext4,format=raw,id=hd0 -netd
ev user,id=net0 -device virtio-net-device,netdev=net0
```

成功后输出如下:

```
...
Starting syslogd: OK
Starting klogd: OK
Running sysctl: OK
Starting mdev... OK
modprobe: can't change directory to '/lib/modules': No such file or directory
Initializing random number generator: OK
```

```
Saving random seed: [    4.538758] random: dd: uninitialized urandom read (512 bytes read)
OK
Starting network: udhcpc: started, v1.31.1
udhcpc: sending discover
udhcpc: sending select for 10.0.2.15
udhcpc: lease of 10.0.2.15 obtained, lease time 86400
deleting routers
adding dns 10.0.2.3
OK

Welcome to Buildroot
buildroot login:
```

运行`uname -a`, 输出如下, 可以确定运行的是`riscv64`架构

```
# uname -a
Linux buildroot 5.8.11 #1 SMP Sun Sep 25 20:44:02 UTC 2022 riscv64 GNU/Linux
```

5 使用GDB调试内核

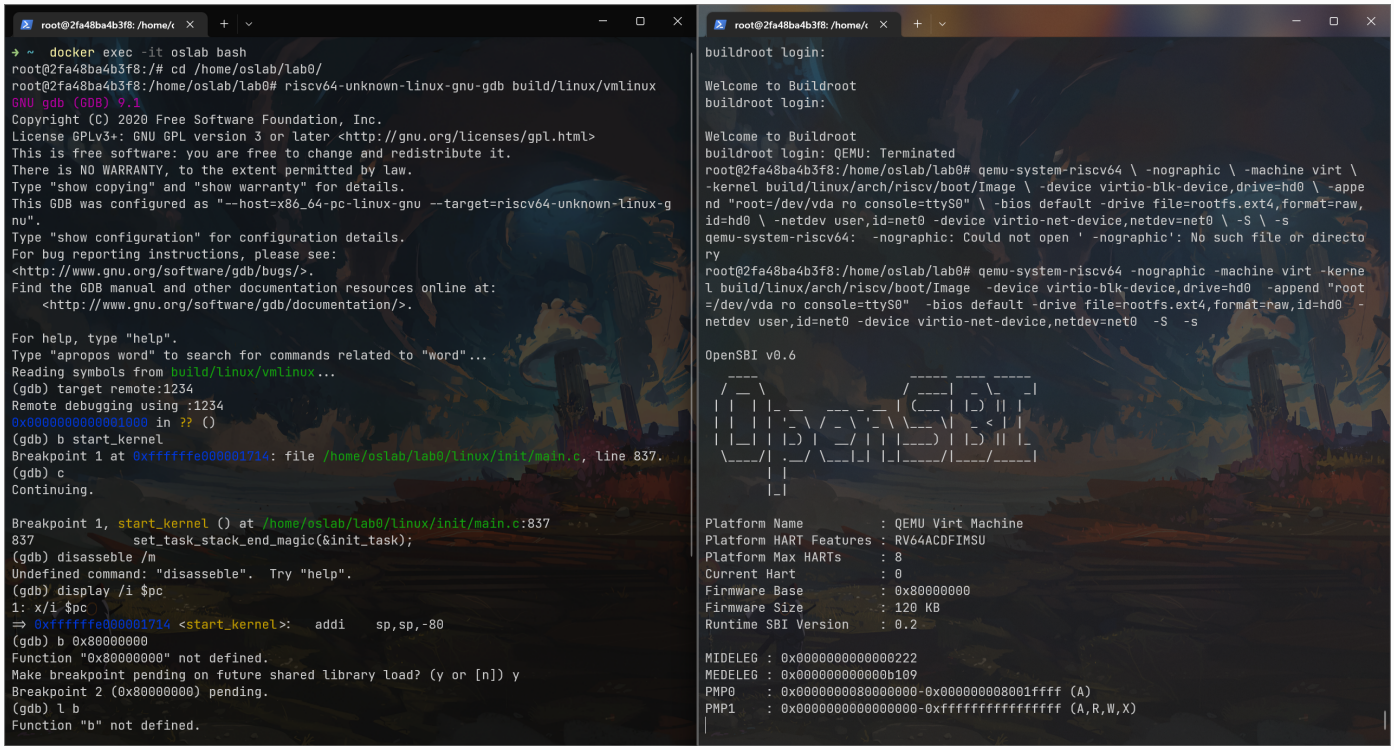
首先使用QEMU模拟运行内核

```
root@2fa48ba4b3f8:/home/oslab/lab0# qemu-system-riscv64 -nographic -machine virt -kernel
build/linux/arch/riscv/boot/Image  -device virtio-blk-device,drive=hd0  -append "root=/dev/vda
ro console=ttyS0"  -bios default -drive file=rootfs.ext4,format=raw,id=hd0  -netdev
user,id=net0 -device virtio-net-device,netdev=net0  -S  -s
# -S: 表示启动时暂停执行，这样我们可以在 GDB 连接后再开始执行
# -s: -gdb tcp::1234 的缩写，会开启一个 tcp 服务，端口为 1234，可以使用 GDB 连接并进行调 试
```

再打开一个新的terminal, 进入同一个容器,输入`riscv64-unknown-linux-gnu-gdb build/linux/vmlinux`使用GDB进行调试, 在`start_kernel`处打上断点进行调试, 之后continue, 发现能正常停在断点处

```
(gdb) target remote:1234
Remote debugging using :1234
0x0000000000001000 in ?? ()
(gdb) b start_kernel
Breakpoint 1 at 0xffffffe000001714: file /home/oslab/lab0/linux/init/main.c, line 837.
(gdb) c
Continuing.

Breakpoint 1, start_kernel () at /home/oslab/lab0/linux/init/main.c:837
837             set_task_stack_end_magic(&init_task);
(gdb)
```

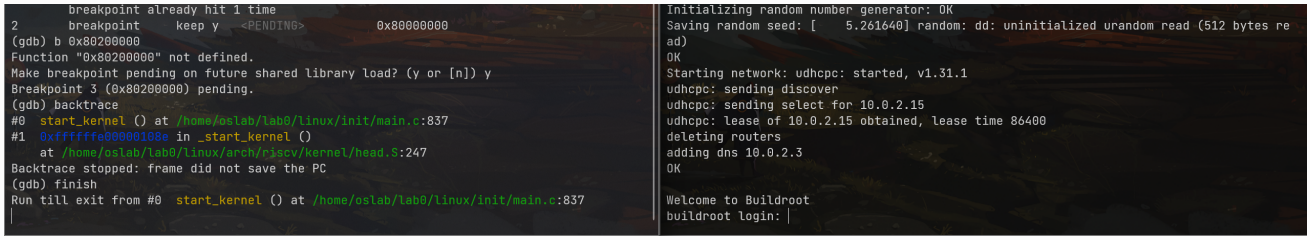


五 实验结果和分析

1. `backtrace`

```
(gdb) backtrace
#0  start_kernel () at /home/oslab/lab0/linux/init/main.c:837
#1  0xffffffe00000108e in _start_kernel ()
    at /home/oslab/lab0/linux/arch/riscv/kernel/head.S:247
Backtrace stopped: frame did not save the PC
```

2. `finish`



3. `frame`

```
(gdb) frame
#0  arch_cpu_idle () at /home/oslab/lab0/linux/arch/riscv/kernel/process.c:33
33              local_irq_enable();
```

4. `info`

```
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0xffffffe000001714 in start_kernel
                                                   at
/home/oslab/lab0/linux/init/main.c:837
        breakpoint already hit 1 time
2       breakpoint     keep y   <PENDING>          0x80000000
3       breakpoint     keep y   <PENDING>          0x80200000
```

5. `break`

```
(gdb) break
Breakpoint 4 at 0xffffffe000201480: file
/home/oslab/lab0/linux/arch/riscv/include/asm/irqflags.h, line 22.
```

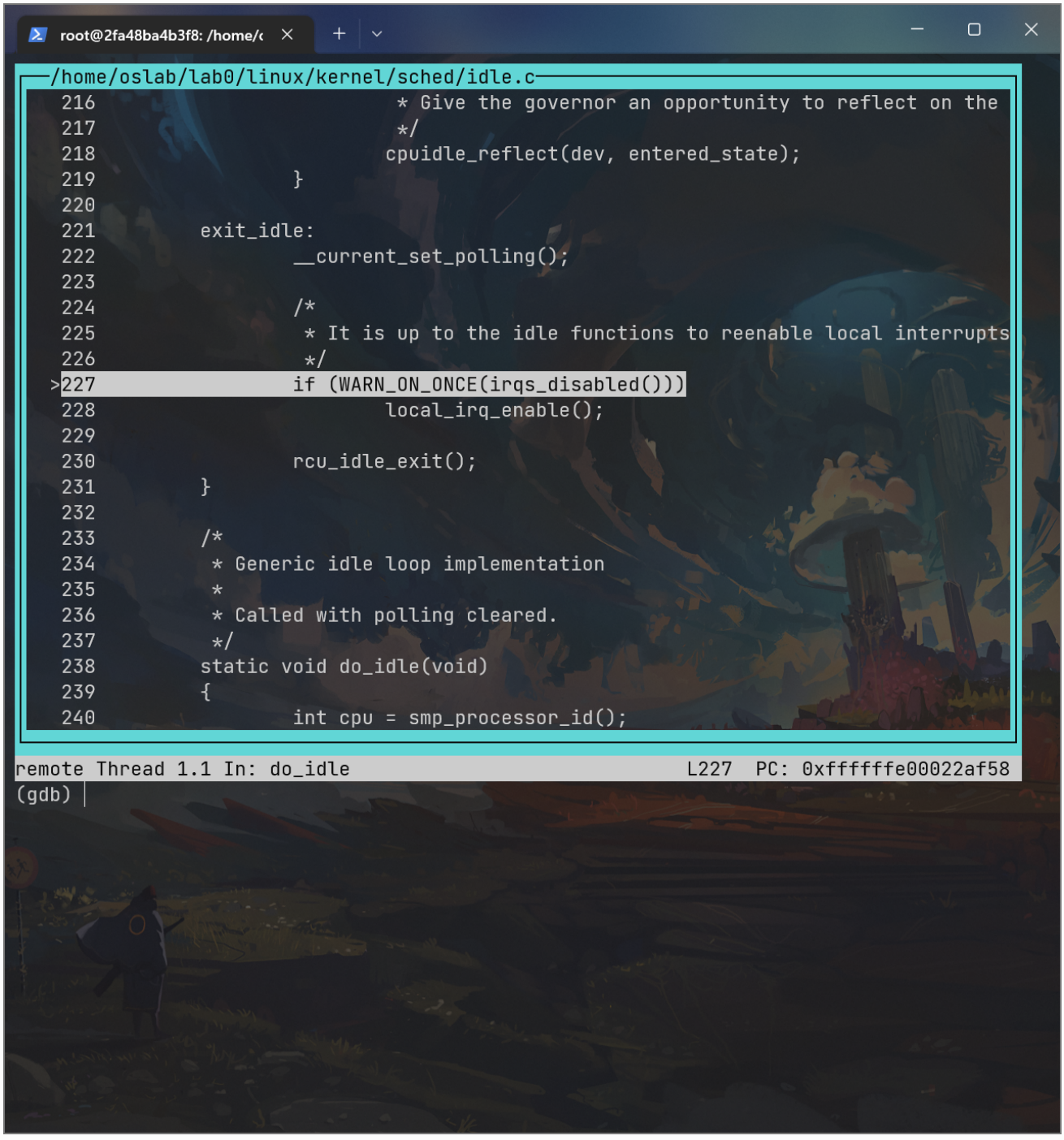
6. `display`

```
(gdb) display
1: x/i $pc
=> 0xffffffe000201480 <arch_cpu_idle+10>:       csrsi   sstatus,2
```

7. `next`

```
(gdb) next
cpuidle_idle_call () at /home/oslab/lab0/linux/kernel/sched/idle.c:227
227             if (WARN_ON_ONCE(irqs_disabled()))
1: x/i $pc
=> 0xffffffe00022af58 <do_idle+166>:    csrr    a5,sstatus
```

8. `layout`

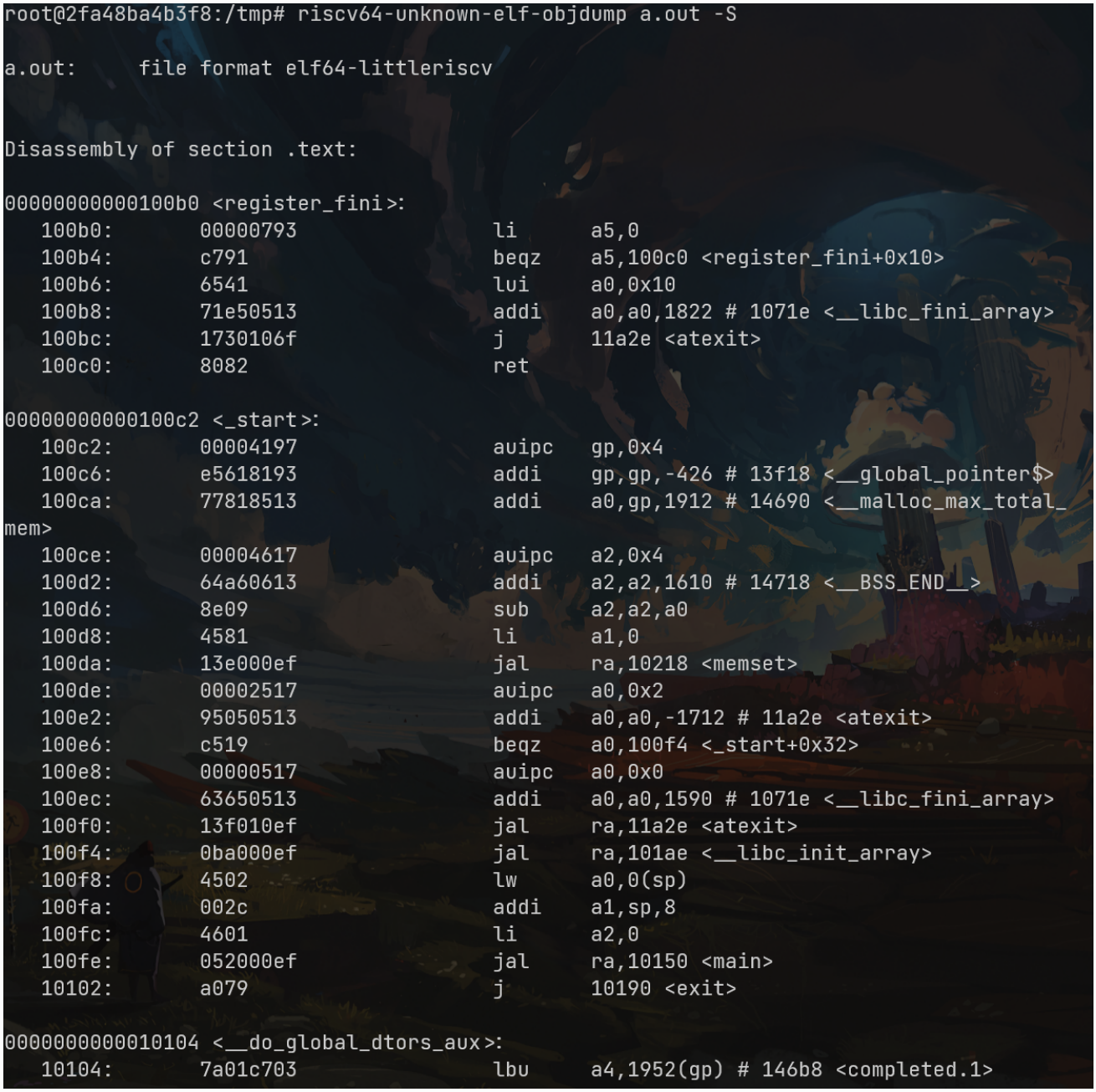


六 问题解答

1. 使用`riscv64-unknown-elf-gcc`编译单个.c文件

```
root@2fa48ba4b3f8:/tmp# riscv64-unknown-elf-gcc hello.c
root@2fa48ba4b3f8:/tmp# ls
a.out  hello.c
root@2fa48ba4b3f8:/tmp# ./a.out
hello!
hello!
hello!
hello!
hello!
hello!
hello!
hello!
hello!
```

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



```
root@2fa48ba4b3f8:/tmp# riscv64-unknown-elf-objdump a.out -S

a.out:     file format elf64-littleriscv


Disassembly of section .text:

00000000000100b0 <register_fini>:
   100b0:       00000793                li      a5,0
   100b4:       c791                    beqz    a5,100c0 <register_fini+0x10>
   100b6:       6541                    lui     a0,0x10
   100b8:       71e50513                addi    a0,a0,1822 # 1071e <__libc_fini_array>
   100bc:       1730106f                j       11a2e <atexit>
   100c0:       8082                    ret

00000000000100c2 <_start>:
   100c2:       00004197                auipc   gp,0x4
   100c6:       e5618193                addi    gp,gp,-426 # 13f18 <__global_pointer$>
   100ca:       77818513                addi    a0,gp,1912 # 14690 <__malloc_max_total_
mem>
   100ce:       00004617                auipc   a2,0x4
   100d2:       64a60613                addi    a2,a2,1610 # 14718 <__BSS_END__>
   100d6:       8e09                    sub     a2,a2,a0
   100d8:       4581                    li      a1,0
   100da:       13e000ef                jal     ra,10218 <memset>
   100de:       00002517                auipc   a0,0x2
   100e2:       95050513                addi    a0,a0,-1712 # 11a2e <atexit>
   100e6:       c519                    beqz    a0,100f4 <_start+0x32>
   100e8:       00000517                auipc   a0,0x0
   100ec:       63650513                addi    a0,a0,1590 # 1071e <__libc_fini_array>
   100f0:       13f010ef                jal     ra,11a2e <atexit>
   100f4:       0ba000ef                jal     ra,101ae <__libc_init_array>
   100f8:       4502                    lw      a0,0(sp)
   100fa:       002c                    addi    a1,sp,8
   100fc:       4601                    li      a2,0
   100fe:       052000ef                jal     ra,10150 <main>
   10102:       a079                    j       10190 <exit>

0000000000010104 <__do_global_dtors_aux>:
   10104:       7a01c703                lbu     a4,1952(gp) # 146b8 <completed.1>
```

3. 调试Linux时

    1. 在GDB中查看汇编代码

    ```
    (gdb) display /i $pc
    1: x/i $pc
    => 0xffffffe000001714 <start_kernel>:    addi    sp,sp,-80
    ```

    2. 在0x80000000处下断点

    ```
    (gdb) b 0x80000000
    Function "0x80000000" not defined.
    Make breakpoint pending on future shared library load? (y or [n]) y
    Breakpoint 2 (0x80000000) pending.
    ```

    3. 查看所有已下的断点

    ```
    (gdb) i b
    Num     Type           Disp Enb Address            What
    1       breakpoint     keep y   0xffffffe000001714 in start_kernel
                                                       at /home/oslab/lab0/linux/init/main.
    c:837
            breakpoint already hit 1 time
    2       breakpoint     keep y   <PENDING>          0x80000000
    ```

    4. 在0x80200000处下断点

    ```
    (gdb) b 0x80200000
    Function "0x80200000" not defined.
    Make breakpoint pending on future shared library load? (y or [n]) y
    Breakpoint 3 (0x80200000) pending.
    ```

    5. 单步调试一次

    ```
    (gdb) next
    cpuidle_idle_call () at /home/oslab/lab0/linux/kernel/sched/idle.c:227
    227             if (WARN_ON_ONCE(irqs_disabled()))
    1: x/i $pc
    => 0xffffffe00022af58 <do_idle+166>:    csrr    a5,sstatus
    ```

    6. 退出QEMU

       先后按下ctrl+A, X即可

4. 使用make工具清除Linux的构建产物

   在工作目录下输入`make clean`即可

5. vmlinux和image的关系和区别是什么

   Linux内核有多种格式的镜像, 包括vmlinux, Image, zImage等等,其中

    - 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.

七 讨论心得

1. 这次我首先在vmware中使用Ubuntu18.04LTS进行实验, 按照步骤进行换源等操作耗费了很久之后进行实验, 一开始发现系统中已经有了两个版本的内核， 但是后来在将windows上已经下载好的源码和oslab.tar文件与vmware进行共享的时候, vmware tools出现了问题, 导致实验无法正常进行, 我只能重新安装一遍
2. 第二次进行试验的时候, 在成功导入docker镜像之后, 由于在make编译的时候我直接复制了一长串指令, 导致出现错误, 为了解决这个错误我一开始以为是$PATH变量的问题, 后来以为是工具链的问题, 查找了将近3个小时的各种资料网站, 都没能解决, 这时我还是认为这是工具链出了问题, 于是我向同学借用了其他班记得oslab.tar包,并决定再换一个平台尝试
3. 这次我使用了docker desktop, 前面的步骤一切顺利, 然而依然在相同的地方卡住了, 最后经过多次尝试才在手动输入一遍指令之后得以顺利运行, 此时我意识到, 这么长的时间, 全都浪费在这个因为自己懒得手动输入一遍命令而导致的错误上, 我十分懊悔, 并且深刻地认识到了一个不好的习惯在工程中的破坏力是如此的巨大, 如果一个细节处理不好, 就有可能导致巨大的错误, 以后一定要细心细心再细心!