Lab 0: RV64 内核调试

软工2002 庄毅非 3200105872

1 实验目的

安装虚拟机及Docker，通过在QEMU模拟器上运行Linux来熟悉如何从源代码开始将内核运行在QEMU模拟器上，学习使用GDB跟QEMU对代码进行联合调试，为后续实验打下基础。

编译内核并用 gdb + QEMU 调试，在内核初始化过程中（用户登录之前）设置断点，对内核的启 动过程进行跟踪，并尝试使用gdb的各项命令（如backtrace、nish、frame、info、break、 display、next等）。

在学在浙大中提交pdf格式的实验报告，记录实验过程并截图（4.1 - 4.4），对每一步的命令以及结 果进行必要的解释，记录遇到的问题和心得体会。

2 实验内容及要求

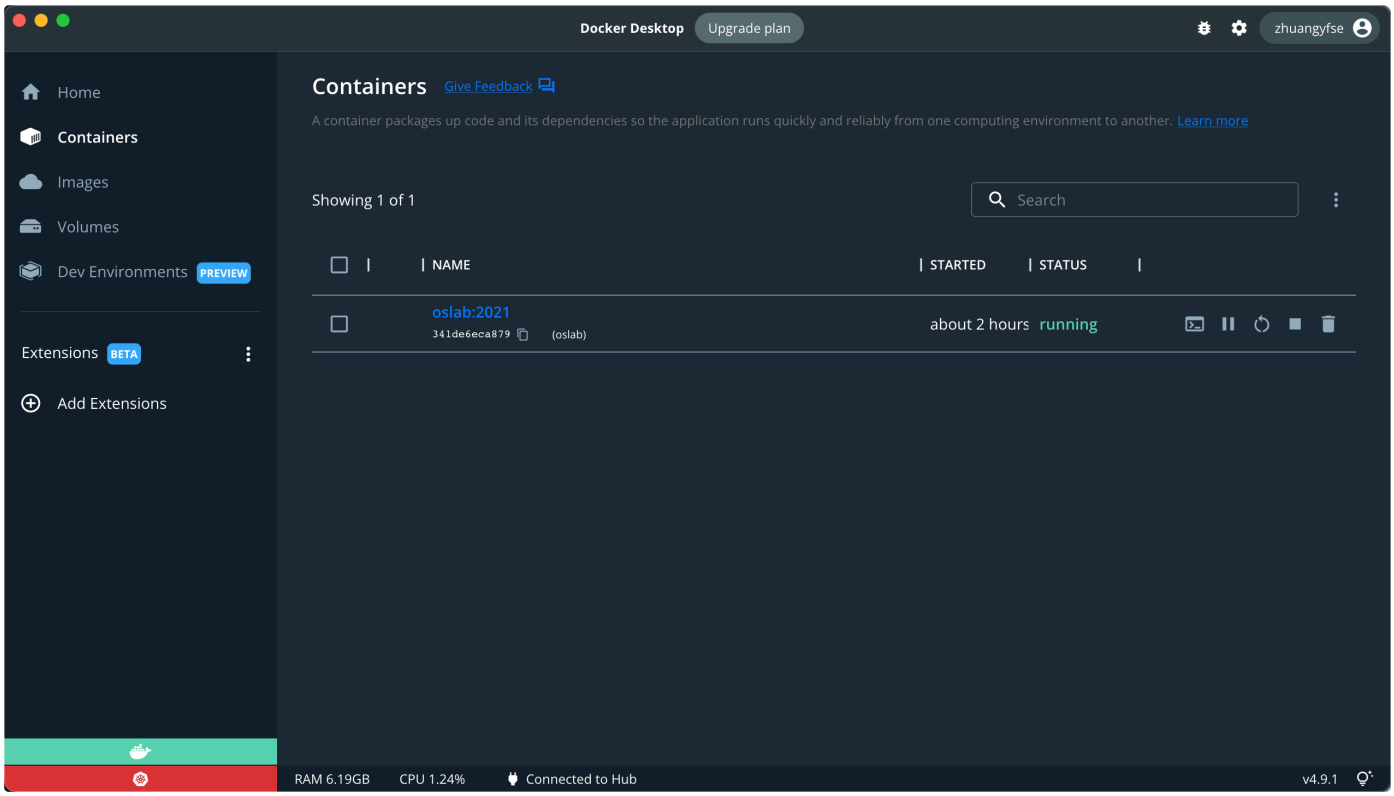
- 安装虚拟机软件、Ubuntu镜像，自行学习Linux基础命令。
- 安装Docker，下载并导入Docker镜像，创建并运行容器。
- 编译内核并用 gdb + QEMU 调试，在内核初始化过程中设置断点，对内核的启动过程进行跟踪，并尝试使用gdb的各项命令。

3 操作方法和实验步骤

3.1 安装docker，并创建容器

3.1.1 安装docker

由于我使用的系统为macos，所以这一步是直接使用 `brew install --cask docker` 命令进行docker图形界面的安装，安装完之后docker运行截图如下。



3.1.2 下载并导入docker镜像

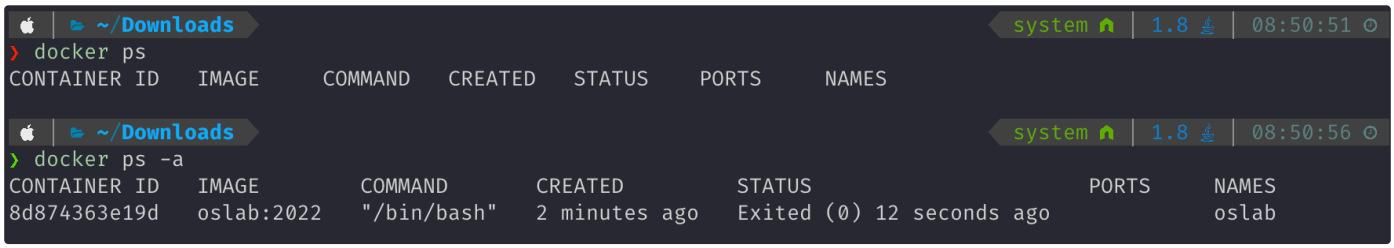
```
# 进入oslab.tar所在的文件夹，使用文档中给出的命令导入docker
$ cat oslab.tar | docker import - oslab:2022
$ docker image ls
```

```
~/Downloads                                                    system  1.8  08:44:37
❯ docker images
REPOSITORY                                TAG         IMAGE ID       CREATED          SIZE
oslab                                     2022        6fcb3d573eb3   8 seconds ago    2.89GB
k8s.gcr.io/kube-apiserver                 v1.21.2     106ff58d4308   12 months ago    126MB
k8s.gcr.io/kube-scheduler                 v1.21.2     f917b8c8f55b   12 months ago    50.6MB
k8s.gcr.io/kube-controller-manager        v1.21.2     ae24db9aa2cc   12 months ago    120MB
k8s.gcr.io/kube-proxy                     v1.21.2     a6ebd1c1ad98   12 months ago    131MB
docker/desktop-vpnkit-controller          v2.0        8c2c38aa676e   13 months ago    21MB
docker/desktop-storage-provisioner        v2.0        99f89471f470   14 months ago    41.9MB
k8s.gcr.io/pause                          3.4.1       0f8457a4c2ec   17 months ago    683kB
k8s.gcr.io/coredns/coredns                v1.8.0      296a6d5035e2   20 months ago    42.5MB
k8s.gcr.io/etcd                           3.4.13-0    0369cf4303ff   22 months ago    253MB
```

3.1.3 从镜像中创建一个容器，并进入该容器

```
# 创建docker容器，并将本地目录docker_volumn挂载到docker虚拟容器中的/have-fun-debugging目录下
$ docker run --name oslab -it -v "/Users/zhuangyifei/Desktop/tech_Learning/myhomework/third up/os/docker_volumn":/have-fun-debugging oslab:2022 /bin/bash
```


```
# 查看当前运行的容器
$ docker ps
# 查看当前存在的所有容器
$ docker ps -a
```




```
# 启动处于停止状态的容器
$ docker start 8d874363
```


```
# 进入已经运行的容器
$ docker exec -it 8d874363e19de3840a7a60d0a0d2c2cb75aa95fada6d1bfb8f3932316a075c08 /bin/bash
```

```
~/Downloads                                                    system  1.8  08:52:22
❯ docker exec -it 8d874363e19de3840a7a60d0a0d2c2cb75aa95fada6d1bfb8f3932316a075c08 /bin/bash
root@8d874363e19d:/# _
```

3.2 编译linux内核

```
$ pwd
/home/zhuangyifei/Desktop/os22fall-stu/src/lab0/linux
$ mkdir -p build/linux
# 设定内核携带debug info编译
$ make CROSS_COMPILE=riscv64-linux-gnu- ARCH=riscv  menuconfig
$ make CROSS_COMPILE=riscv64-linux-gnu- ARCH=riscv  -j4

```

```
  CC [M]  drivers/video/fbdev/core/sysfillrect.mod.o
  CC [M]  drivers/video/fbdev/core/sysimgblt.mod.o
  CC [M]  drivers/virtio/virtio_dma_buf.mod.o
  CC [M]  fs/efivarfs/efivarfs.mod.o
  CC [M]  fs/nls/nls_iso8859-1.mod.o
  LD [M]  arch/riscv/kvm/kvm.ko
  LD [M]  drivers/gpu/drm/dp/drm_dp_helper.ko
  LD [M]  drivers/gpu/drm/drm.ko
  LD [M]  drivers/gpu/drm/drm_kms_helper.ko
  LD [M]  drivers/gpu/drm/drm_panel_orientation_quirks.ko
  LD [M]  drivers/gpu/drm/drm_shmem_helper.ko
  LD [M]  drivers/gpu/drm/drm_ttm_helper.ko
  LD [M]  drivers/gpu/drm/i2c/ch7006.ko
  LD [M]  drivers/gpu/drm/i2c/sil164.ko
  LD [M]  drivers/gpu/drm/nouveau/nouveau.ko
  LD [M]  drivers/gpu/drm/radeon/radeon.ko
  LD [M]  drivers/gpu/drm/ttm/ttm.ko
  LD [M]  drivers/gpu/drm/virtio/virtio-gpu.ko
  LD [M]  drivers/i2c/algos/i2c-algo-bit.ko
  LD [M]  drivers/i2c/i2c-core.ko
  LD [M]  drivers/nvme/host/nvme-core.ko
  LD [M]  drivers/nvme/host/nvme.ko
  LD [M]  drivers/video/backlight/backlight.ko
  LD [M]  drivers/video/fbdev/core/cfbcopyarea.ko
  LD [M]  drivers/video/fbdev/core/cfbfillrect.ko
  LD [M]  drivers/video/fbdev/core/cfbimgblt.ko
  LD [M]  drivers/video/fbdev/core/fb_sys_fops.ko
  LD [M]  drivers/video/fbdev/core/syscopyarea.ko
  LD [M]  drivers/video/fbdev/core/sysfillrect.ko
  LD [M]  drivers/video/fbdev/core/sysimgblt.ko
  LD [M]  drivers/virtio/virtio_dma_buf.ko
  LD [M]  fs/efivarfs/efivarfs.ko
  LD [M]  fs/nls/nls_iso8859-1.ko
root@8d874363e19d:/have-fun-debugging/src/lab0/linux#
```

3.3 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=rootfs.img,format=raw,id=hd0
```

```
[    0.399235] ohci-platform: OHCI generic platform driver
[    0.400616] usbcore: registered new interface driver uas
[    0.401078] usbcore: registered new interface driver usb-storage
[    0.402191] mousedev: PS/2 mouse device common for all mice
[    0.404975] goldfish_rtc 101000.rtc: registered as rtc0
[    0.405538] goldfish_rtc 101000.rtc: setting system clock to 2022-06-28T02:04:10 UTC (1656381850)
[    0.408418] cpuidle-riscv-sbi: HSM suspend not available
[    0.409650] sdhci: Secure Digital Host Controller Interface driver
[    0.410367] sdhci: Copyright(c) Pierre Ossman
[    0.410967] sdhci-pltfm: SDHCI platform and OF driver helper
[    0.413150] usbcore: registered new interface driver usbhid
[    0.413468] usbhid: USB HID core driver
[    0.415780] NET: Registered PF_INET6 protocol family
[    0.424638] Segment Routing with IPv6
[    0.425099] In-situ OAM (IOAM) with IPv6
[    0.425854] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[    0.429357] NET: Registered PF_PACKET protocol family
[    0.430960] 9pnet: Installing 9P2000 support
[    0.431534] Key type dns_resolver registered
[    0.433475] debug_vm_pgtable: [debug_vm_pgtable         ]: Validating architecture page table helpers
[    0.444425] Legacy PMU implementation is available
[    0.479731] EXT4-fs (vda): mounted filesystem with ordered data mode. Quota mode: disabled.
[    0.480285] VFS: Mounted root (ext4 filesystem) readonly on device 254:0.
[    0.482932] devtmpfs: mounted
[    0.537187] Freeing unused kernel image (initmem) memory: 2164K
[    0.538099] Run /sbin/init as init process

Please press Enter to activate this console.
/ # ls
bin         etc         lost+found  sbin        usr
dev         linuxrc     proc        sys
/ # _
```

3.4 使用 GDB 对内核进行调试

这里原来的容器crash了，终端连接出现卡死，所以将原来的容器删除，创建了一个新的容器执行了上述步骤，以下步骤在新的容器中执行。

使用tmux多窗口终端实现。

```
# 在窗口1中运行，这里的-s是-gdb tcp:1234的缩写，-S表示在启动的时候停止
cpu，所以在窗口一中我们看不到任何输出
```

```
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=./rootfs.img,format=raw,id=hd0 -S -s
# 在窗口2中运行
gdb-multiarch ./vmlinux

# 连接到本地1234端口（默认用来调试链接的端口号）
(gdb) target remote :1234
# 设置gdb参数
(gdb) set riscv use-compressed-breakpoints on
# 在内核启动函数处建立断点
(gdb) b start_kernel
# 在0x80000000创建断点
(gdb) b *0x80000000
# 在0x80200000创建断点
(gdb) b *0x80200000
# 查看断点列表
(gdb) info br
```

```
root@6b1e31f98202:/oslab/os22fall-stu.nosync/src/lab0/linux# gdb-multiarch vmlinux
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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:
<http://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 vmlinux...
(gdb) target remote :1234
Remote debugging using :1234
0x0000000000001000 in ?? ()
(gdb) b start_kernel
Cannot access memory at address 0xffffffff808006c0
(gdb) set riscv use-compressed-breakpoints on
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff808006c0: file init/main.c, line 930.
(gdb) b *0x80000000
Breakpoint 2 at 0x80000000
(gdb) b *0x80200000
Breakpoint 3 at 0x80200000
(gdb) info br
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0xffffffff808006c0 in start_kernel at init/main.c:930
2       breakpoint     keep y   0x0000000080000000
3       breakpoint     keep y   0x0000000080200000
(gdb) _
```

```
# 删除断点2
(gdb) delete 2
# 查看断点列表
(gdb) info br
# 运行程序，遇到断点0x80000000
(gdb) continue
(gdb) delete 3
# 运行程序，遇到断点start_kernel
(gdb) continue
(gdb) delete 1
(gdb) s
(gdb) # 直接回车
(gdb) n
(gdb) # 直接回车
```

```
(gdb) delete 2
(gdb)  info br
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0xffffffff808006c0 in start_kernel at init/main.c:930
3       breakpoint     keep y   0x0000000080200000
(gdb) c
Continuing.

Breakpoint 3, 0x0000000080200000 in ?? ()
(gdb) delete 3
(gdb) c
Continuing.

Breakpoint 1, start_kernel () at init/main.c:930
930     {
(gdb) delete 1
(gdb) s
934             set_task_stack_end_magic(&init_task);
(gdb)
set_task_stack_end_magic (tsk=0xffffffff8120de40 <init_task>) at kernel/fork.c:958
958             stackend = end_of_stack(tsk);
(gdb) n
959             *stackend = STACK_END_MAGIC;    /* for overflow detection */
(gdb)
start_kernel () at init/main.c:935
935             smp_setup_processor_id();
```

```
# 查看调用栈
(gdb) backtrace
# 查看上一个调用栈
(gdb) up 1
# 查看栈信息
(gdb) i frame
```

```
(gdb) bt
#0  start_kernel () at init/main.c:935
#1  0xffffffff80001150 in _start_kernel () at arch/riscv/kernel/head.S:326
Backtrace stopped: frame did not save the PC
(gdb) up 1
#1  0xffffffff80001150 in _start_kernel () at arch/riscv/kernel/head.S:326
326             call soc_early_init
(gdb) i frame
Stack level 1, frame at 0xffffffff81204000:
 pc = 0xffffffff80001150 in _start_kernel (arch/riscv/kernel/head.S:326);
    saved pc = <not saved>
 Outermost frame: frame did not save the PC
 caller of frame at 0xffffffff81204000
 source language asm.
 Arglist at 0xffffffff81204000, args:
 Locals at 0xffffffff81204000, Previous frame's sp is 0xffffffff81204000
```

```
# 单步调试
(gdb) next
# 结束当前栈帧
(gdb) finish
# 退出gdb
(gdb) quit
```

```
(gdb) n
939             cgroup_init_early();
(gdb) finish
Run till exit from #0  start_kernel () at init/main.c:939
^C
Program received signal SIGINT, Interrupt.
arch_cpu_idle () at arch/riscv/kernel/process.c:42
42              raw_local_irq_enable();
(gdb) quit
A debugging session is active.

        Inferior 1 [process 1] will be detached.

Quit anyway? (y or n) y
Detaching from program: /oslab/os22fall-stu.nosync/src/lab0/linux/vmlinux, process 1
Ending remote debugging.
[Inferior 1 (process 1) detached]
```

4. 思考题

1. 使用 `riscv64-linux-gnu-gcc` 编译单个 `.c` 文件

   编译如下C文件

```c
int main() {
  int sum = 0;
  for (int i = 0; i < 100; i++) {
    sum += i;
  }
  return 0;
}
```

```
root@6b1e31f98202:/tmp# cat a.c
int main() {
  int sum = 0;
  for (int i = 0; i < 100; i++) {
    sum += i;
  }
  return 0;
}
root@6b1e31f98202:/tmp# riscv64-linux-gnu-gcc -g a.c
root@6b1e31f98202:/tmp# _
```

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

```
a.out:     file format elf64-littleriscv


Disassembly of section .plt:

00000000000004f0 <.plt>:
 4f0:   00002397                auipc   t2,0x2
 4f4:   41c30333                sub     t1,t1,t3
 4f8:   b183be03                ld      t3,-1256(t2) # 2008 <__TMC_END__>
 4fc:   fd430313                addi    t1,t1,-44
 500:   b1838293                addi    t0,t2,-1256
 504:   00135313                srli    t1,t1,0x1
 508:   0082b283                ld      t0,8(t0)
 50c:   000e0067                jr      t3

0000000000000510 <__libc_start_main@plt>:
 510:   00002e17                auipc   t3,0x2
 514:   b08e3e03                ld      t3,-1272(t3) # 2018 <__libc_start_main@GLIBC_2.27>
 518:   000e0367                jalr    t1,t3
 51c:   00000013                nop

Disassembly of section .text:

0000000000000520 <_start>:
 520:   02e000ef                jal     ra,54e <load_gp>
 524:   87aa                    mv      a5,a0
 526:   00002517                auipc   a0,0x2
 52a:   b0a53503                ld      a0,-1270(a0) # 2030 <_GLOBAL_OFFSET_TABLE_+0x10>
 52e:   6582                    ld      a1,0(sp)
 530:   0030                    addi    a2,sp,8
 532:   ff017113                andi    sp,sp,-16
 536:   00000697                auipc   a3,0x0
 53a:   0f668693                addi    a3,a3,246 # 62c <__libc_csu_init>
 53e:   00000717                auipc   a4,0x0
 542:   14670713                addi    a4,a4,326 # 684 <__libc_csu_fini>
 546:   880a                    mv      a6,sp
 548:   fc9ff0ef                jal     ra,510 <__libc_start_main@plt>
 54c:   9002                    ebreak
--More--
```

3. 使用 `make` 工具清除 Linux 的构建产物

```
root@6b1e31f98202:/oslab/os22fall-stu/src/lab0/linux# make clean
make[1]: *** Documentation/Kbuild: Is a directory.  Stop.
make: *** [Makefile:1859: _clean_Documentation] Error 2
```

运行之后无法正常完成中间产物的清理，上google查资料说是因为docker主系统为macos或者windows会导致这个问题。

之后在虚拟机中执行make clean成功。

```
zyf@zyf-virtual-machine:~/Desktop/linux$ make clean
  CLEAN   drivers/firmware/efi/libstub
  CLEAN   drivers/gpu/drm/radeon
  CLEAN   drivers/scsi
  CLEAN   drivers/tty/vt
  CLEAN   kernel
  CLEAN   lib
  CLEAN   usr
  CLEAN   vmlinux.symvers modules-only.symvers modules.builtin modules.builtin.m
odinfo
```

4. vmlinux和Image之间的区别

Image是linux内核镜像文件，没有经过压缩，可以直接引导linux启动；vmlinux是未压缩的，编译出来的原始内核文件，含有elf符号表，但是不能直接引导内核启动。

使用make --just-print可以发现，Image就是将vmlinux用objcopy去掉一些多余信息，比如符号表。

以下是在docker中执行file查看文件类型的输出。

```
root@6b1e31f98202:/oslab/os22fall-stu/src/lab0/linux/build/linux# file arch/riscv/boot/Image
arch/riscv/boot/Image: MS-DOS executable PE32+ executable (EFI application) RISC-V 64-bit (stripped to external PDB), for MS Windows
root@6b1e31f98202:/oslab/os22fall-stu/src/lab0/linux/build/linux# file vmlinux
vmlinux: ELF 64-bit LSB executable, UCB RISC-V, version 1 (SYSV), statically linked, BuildID[sha1]=13dcda92b5b733f4fc786c1878e638c7c08e3205, not stripped
```

5. 实验心得

- 通过这次lab,我初步了解了docker和ubuntu中使用qemu和gdb对于linux-kernel的调试方式，也了解了gdb中一些基础的命令。
- 在创建容器并挂载本地文件夹的时候，在创建成功之后总是无法在docker容器中找到对应的本地文件夹，后来发现需要在创建容器的时候加上-v参数
- 在gdb调试的时候总是提示无法连接到1234端口，后来发现是在另一个终端使用qemu运行linux的时候参数忘记加上-S -s了，修改之后gdb连接成功。