

Lab 0: GDB & QEMU 调试 64 位 RISC-V LINUX

姓名：姜雨童
学号：3220103450
日期：2024.09.18

1 实验内容及原理

1.1 实验目的

- 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

1.2 实验环境

Ubuntu 22.04

```
jyt@fine:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 22.04.2 LTS
Release:        22.04
Codename:       jammy
jyt@fine:~$
```

2 实验过程与代码实现

2.1 搭建实验环境

更新软件包列表并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 update
2 $ sudo apt install gcc-riscv64-linux-gnu
3 $ sudo apt install autoconf automake autotools-dev curl libmpc-dev libmpfr-dev libgmp-
  dev \
4     gawk build-essential bison flex texinfo gperf libtool patchutils bc \
5     zlib1g-dev libexpat-dev git
```

```
jyt@fine: ~
jyt@fine:~$ sudo apt update
[sudo] password for jyt:
Hit:1 http://mirrors.ustc.edu.cn/ubuntu jammy-security InRelease
Hit:2 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
71 packages can be upgraded. Run 'apt list --upgradable' to see them.
jyt@fine:~$
```

```
jyt@fine: ~/os24fall-stu/src/le X + v
jyt@fine:~$ sudo apt install gcc-riscv64-linux-gnu
[sudo] password for jyt:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  binutils-riscv64-linux-gnu cpp-11-riscv64-linux-gnu cpp
  gcc-11-riscv64-linux-gnu-base gcc-12-cross-base-ports l
  libc6-dev-riscv64-cross libc6-riscv64-cross libgcc-11-d
  linux-libc-dev-riscv64-cross
```

```
jyt@fine:~$ 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'
gawk is already the newest version (1:5.1.0-1ubuntu0.1).
gawk set to manually installed.
git is already the newest version (1:2.34.1-1ubuntu1.11).
git set to manually installed.
The following additional packages will be installed:
  dpkg-dev fakeroot libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libcurl4 lib
```

安装模拟器qemu，用以启动 riscv64 平台上的内核：

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

```
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for install-info (6.8-4build1) ...
jyt@fine:~$ sudo apt install qemu-system-misc
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  acl alsa-topology-conf alsa-ucm-conf glib-networking glib
  gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstre
  libasound2-data libasyns0 libavc1394-0 libboost-iostrea
```

安装gdb，用以调试qemu上运行的Linux内核：

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

```
Processing triggers for libc-bin (2.35-0ubuntu3.8) ...
jyt@fine:~$ sudo apt install gdb-multiarch
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 98 not upgraded.
Need to get 4591 kB of archives.
```

2.2 获取 Linux 源码和已经编译好的文件系统

利用wget从<https://www.kernel.org>下载最新的Linux源码（截至实验时为 `linux-6.11-rc7`），并解压：

```
1 $ wget https://git.kernel.org/torvalds/t/linux-6.11-rc7.tar.gz
2 $ tar -zxvf linux-6.11-rc7.tar.gz
```

```
jyt@fine:~$ wget
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try 'wget --help' for more options.
jyt@fine:~$ wget https://git.kernel.org/torvalds/t/linux-6.11-rc7.tar.gz
--2024-09-13 09:18:40-- https://git.kernel.org/torvalds/t/linux-6.11-rc7.tar.gz
Resolving git.kernel.org (git.kernel.org)... 145.40.73.55, 2604:1380:40e1:4800::1
Connecting to git.kernel.org (git.kernel.org)|145.40.73.55|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/snapshot/linux-6.11-rc7.tar.gz [following]
--2024-09-13 09:18:42-- https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/snapshot/linux-6.11-rc7.tar.gz
Reusing existing connection to git.kernel.org:443.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/x-gzip]
Saving to: 'linux-6.11-rc7.tar.gz'

linux-6.11-rc7.tar.gz      [          ] 7.31M 1.88MB/s
```

使用git工具clone实验提供的仓库：

```
1 $ git clone https://github.com/ZJU-SEC/os24fall-stu.git
2 $ cd os24fall-stu/src/lab0
3 $ ls
4 rootfs.img # 已经构建完成的根文件系统的镜像
```

```
jyt@fine:~$ git clone https://github.com/ZJU-SEC/os24fall-stu.git
Cloning into 'os24fall-stu'...
remote: Enumerating objects: 82, done.
remote: Counting objects: 100% (82/82), done.
remote: Compressing objects: 100% (53/53), done.
remote: Total 82 (delta 36), reused 60 (delta 18), pack-reused 0 (from 0)
Receiving objects: 100% (82/82), 1.97 MiB | 2.50 MiB/s, done.
Resolving deltas: 100% (36/36), done.
jyt@fine:~$ cd os24fall-stu/src/lab0
-bash: cd: os24fall-stu/src/lab0: No such file or directory
jyt@fine:~$ cd os24fall-stu/src/lab0
jyt@fine:~/os24fall-stu/src/lab0$ ls
rootfs.img
jyt@fine:~/os24fall-stu/src/lab0$
```

2.3 编译Linux内核

```
1 $ cd linux-6.11-rc7 # 修改为实际路径
2 $ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- defconfig # 使用默认配置
3 $ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- -j$(nproc) # 编译
```

```

jyt@fine: ~/linux-6.11-rc7
linux-6.11-rc7/virt/lib/Kconfig
linux-6.11-rc7/virt/lib/Makefile
linux-6.11-rc7/virt/lib/irqbypass.c
jyt@fine:~$ cd linux-6.11-rc7
jyt@fine:~/linux-6.11-rc7$ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- defconfig
HOSTCC scripts/basic/fixdep
HOSTCC scripts/kconfig/conf.o
HOSTCC scripts/kconfig/confdata.o
HOSTCC scripts/kconfig/expr.o
LEX scripts/kconfig/lexer.lex.c
YACC scripts/kconfig/parser.tab.[ch]
HOSTCC scripts/kconfig/lexer.lex.o
HOSTCC scripts/kconfig/menu.o
HOSTCC scripts/kconfig/parser.tab.o
HOSTCC scripts/kconfig/preprocess.o
HOSTCC scripts/kconfig/symbol.o
HOSTCC scripts/kconfig/util.o
HOSTLD scripts/kconfig/conf
*** Default configuration is based on 'defconfig'
#
# configuration written to .config
#
jyt@fine:~/linux-6.11-rc7$ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- -j$(nproc)
WRAP arch/riscv/include/generated/uapi/asm/errno.h
WRAP arch/riscv/include/generated/uapi/asm/fcntl.h
WRAP arch/riscv/include/generated/uapi/asm/ioctl.h
WRAP arch/riscv/include/generated/uapi/asm/ioctls.h
UPD include/generated/uapi/linux/version.h
WRAP arch/riscv/include/generated/uapi/asm/ipcbuf.h
WRAP arch/riscv/include/generated/uapi/asm/mman.h
WRAP arch/riscv/include/generated/uapi/asm/msgbuf.h
WRAP arch/riscv/include/generated/uapi/asm/param.h

```

编译完毕:

```

LD [M] net/bridge/br_netfilter.ko
LD [M] net/can/can.ko
LD [M] net/can/can-raw.ko
LD [M] net/can/can-bcm.ko
LD [M] net/can/can-gw.ko
NM System.map
SORTTAB vmlinux
OBJCOPY arch/riscv/boot/Image
Kernel: arch/riscv/boot/Image is ready
GZIP arch/riscv/boot/Image.gz
Kernel: arch/riscv/boot/Image.gz is ready
jyt@fine:~/linux-6.11-rc7$

```

2.4 使用 QEMU 运行内核

```

1 $ qemu-system-riscv64 -nographic -machine virt -kernel .arch/riscv/boot/Image \
2   -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0" \
3   -bios default -drive file=../os24fall-stu/src/lab0/rootfs.img,format=raw,id=hd0

```

```
jyt@fine:~/linux-6.11-rc7$ 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=../os24fall-stu/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      : 0x80000000
Firmware Size      : 100 KB
Runtime SBI Version : 0.2

Domain0 Name       : root
Domain0 Boot HART   : 0
Domain0 HARTs       : 0*
Domain0 Region00    : 0x0000000008000000-0x0000000008001fff ( )
Domain0 Region01    : 0x0000000000000000-0xffffffffffff (R,W,X)
Domain0 Next Address : 0x0000000080200000
Domain0 Next Arg1    : 0x0000000087000000
```

使用 `Ctrl+A`，松开后按下 `X` 退出QEMU:

```
[ 0.445859] devtmpfs: mounted
[ 0.473934] Freeing unused kernel image (initmem) memory: 2256K
[ 0.474586] Run /sbin/init as init process

Please press Enter to activate this console.
/ # QEMU: Terminated
```

2.5 使用 GDB 对内核进行调试

这一步开启两个终端，一个启动linux（图左），另一个使用gdb与QEMU远程通信（图右），进行调试，代码和结果如下：

```
1 # Terminal 1
2 $ qemu-system-riscv64 -nographic -machine virt -kernel ./arch/riscv/boot/Image \
3 -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0" \
4 -bios default -drive file=../os24fall-stu/src/lab0/rootfs.img,format=raw,id=hd0 -S -s
5
6 # Terminal 2
7 $ gdb-multiarch ./linux-6.11-rc7/vmlinux
8 (gdb) target remote :1234 # 连接 qemu
9 (gdb) b start_kernel      # 设置断点
10 (gdb) continue           # 继续执行
11 (gdb) quit               # 退出 gdb
```

```
jyt@fine:~/linux-6.11-rc7$ qemu-system-riscv64 -nographic -m
achine virt -kernel ./arch/riscv/boot/Image -device virt
io-blk-device,drive=hd0 -append "root=/dev/vda ro console=tt
yS0" -bios default -drive file=./os24fall-stu/src/lab0/
rootfs.img,format=raw,id=hd0 -S -s

OpenSBI v0.9

Platform Name      : riscv-virtio,qemu
Platform Features  : timer,mfdeleg
Platform HART Count : 1
Firmware Base      : 0x80000000
Firmware Size      : 100 KB
Runtime SBI Version : 0.2

Domain0 Name       : root
Domain0 Boot HART  : 0
Domain0 HARTs      : 0*
Domain0 Region00   : 0x0000000080000000-0x00000000800
1ffff ()
Domain0 Region01   : 0x0000000000000000-0xffffffffffff
ffff (R,W,X)
Domain0 Next Address : 0x0000000080200000
Domain0 Next Arg1   : 0x00000000087000000
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

e
/home/jyt/.hushlogin file.
jyt@fine:~$ gdb-multiarch ./linux-6.11-rc7/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/licen
ses/gpl.html>
This is free software: you are free to change and redistribute i
t.
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-6.11-rc7/vmlinux...
(gdb) target remote :1234
Remote debugging using :1234
0x0000000000001000 in ?? ()
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff80a00734
(gdb) continue
Continuing.

Breakpoint 1, 0xffffffff80a00734 in start_kernel ()
(gdb)
```

后续尝试了一些gdb命令：

- **b** (**break** 的简写)：设置断点（见上图）
- **continue**：从断点后继续执行（见上图）
- **info**：提供各种信息，这里查看了断点信息（中间重跑了一遍上述代码，因此两个断点都在start_kernel处）

```
Command name abbreviations are allowed if unambiguous.
(gdb) info breakpoints
Num      Type             Disp Enb Address                  What
1        breakpoint      keep y   0xffffffff80a00734 <start_kernel>
          breakpoint already hit 2 times
2        breakpoint      keep y   0xffffffff80a00734 <start_kernel>
          breakpoint already hit 1 time

(gdb) bt
#0  0xffffffff80a00734 in start_kernel ()
#1  0xffffffff80001164 in _start_kernel ()
Backtrace stopped: frame did not save the PC
(gdb) f
#0  0xffffffff80a00734 in start_kernel ()
(gdb) f 1
#1  0xffffffff80001164 in _start_kernel ()
(gdb)
```

- **bt** (**backtrace** 的简写)：查看函数的调用的栈帧和层级关系（见上图）
- **f** (**frame** 的简写)：切换函数的栈帧，这里尝试切换了f0和f1（见上图）
- **layout asm**：显示汇编代码

```

io-blk-device,drive=hd0 -app
ys0" -bios default -driv
rootfs.img,format=raw,id=hd0

OpenSBI v0.9

Platform Name      :
Platform Features  :
Platform HART Count :
Firmware Base      :
Firmware Size      :
Runtime SBI Version :

Domain0 Name       :
Domain0 Boot HART  :
Domain0 HARTs      :
Domain0 Region00   :
1ffff (C)
Domain0 Region01   :
fffff (R,W,X)
Domain0 Next Address :
Domain0 Next Arg1   :
Domain0 Next Mode   :
Domain0 SysReset    :

Boot HART ID       :
Boot HART Domain   :
Boot HART ISA       :
Boot HART Features  :
Boot HART PMP Count :
Boot HART PMP Granularity :
Boot HART PMP Address Bits :
Boot HART MHPM Count :
Boot HART MHPM Count :
Boot HART MIDELEG   :
Boot HART MEDELEG   :

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,sp,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,0xff610
0xffffffff80a00758 <start_kernel+36>     jalr    168(ra)
0xffffffff80a0075c <start_kernel+40>     auipc   ra,0x4
0xffffffff80a00760 <start_kernel+44>     jalr    1136(ra)
0xffffffff80a00764 <start_kernel+48>     auipc   ra,0xe
0xffffffff80a00768 <start_kernel+52>     jalr    1768(ra)
0xffffffff80a0076c <start_kernel+56>     csrwi   sstatus,2
0xffffffff80a00770 <start_kernel+60>     li      a5,1
0xffffffff80a00772 <start_kernel+62>     auipc   a4,0xb15
0xffffffff80a00776 <start_kernel+66>     sb      a5,-1838(a4)
0xffffffff80a0077a <start_kernel+70>     auipc   ra,0x8

remote Thread 1.1 In: start_kernel L?? PC: 0xffffffff80a00734
The "remote" target does not support "run". Try "help target" or "continue".
(gdb) continue
Continuing.
[Inferior 1 (process 1) exited normally]
(gdb) target remote :1234
Remote debugging using :1234
0x00000000000001000 in ?? ()
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff80a00734
(gdb) continue
Continuing.
Breakpoint 1, 0xffffffff80a00734 in start_kernel ()
(gdb)

```

- `next`：单步执行程序，逐行执行，不会进入函数内部

```

(gdb) next
Single stepping until exit from function start_kernel,
which has no line number information.

```

- `stepi`：执行单条指令

```

(gdb) stepi
0xffffffff80a0073c in start_kernel ()
(gdb) finish
Run till exit from #0  0xffffffff80a0073c in start_kernel ()

```

- `finish`：结束当前函数，返回到函数调用点（见上图）

3 实验中遇到的问题及解决方法

Q1：直接下载解压linux源码拖入相应文件夹，编译时报错 `Permission denied`。

A1：利用 `wget` 下载源码，`tar` 相关指令解压（代码见2.2）。


```
jyt@fine:~/os24fall-stu/src/lab0$ cd ../../../../linux-6.10.10
jyt@fine:~/linux-6.10.10$ make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- defconfig
HOSTCC scripts/basic/fixdep
HOSTCC scripts/kconfig/conf.o
HOSTCC scripts/kconfig/confdata.o
HOSTCC scripts/kconfig/expr.o
LEX scripts/kconfig/lexer.lex.c
YACC scripts/kconfig/parser.tab.[ch]
HOSTCC scripts/kconfig/lexer.lex.o
HOSTCC scripts/kconfig/menu.o
HOSTCC scripts/kconfig/parser.tab.o
HOSTCC scripts/kconfig/preprocess.o
HOSTCC scripts/kconfig/symbol.o
HOSTCC scripts/kconfig/util.o
HOSTLD scripts/kconfig/conf
*** Default configuration is based on 'defconfig'
sh: 1: ./scripts/cc-version.sh: Permission denied
scripts/Kconfig.include:45: Sorry, this C compiler is not supported.
make[2]: *** [scripts/kconfig/Makefile:95: defconfig] Error 1
make[1]: *** [/home/jyt/linux-6.10.10/Makefile:695: defconfig] Error 2
make: *** [Makefile:240: __sub-make] Error 2
jyt@fine:~/linux-6.10.10$
```

Q2: 使用QEMU运行内核时, 报错 `No such file or directory`。

A2: 路径错误, 将路径中 `image` 改成 `Image`。

```
jyt@fine:~$ cd linux-6.11-rc7
jyt@fine:~/linux-6.11-rc7$ 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=../os24fall-stu/src/lab0/rootfs.img,format=raw,id=hd0
./arch/riscv/boot/image: No such file or directory
qemu-system-riscv64: could not load kernel './arch/riscv/boot/image'
jyt@fine:~/linux-6.11-rc7$ 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=../os24fall-stu/src/lab0/rootfs.img,format=raw,id=hd0
OpenSBI v0.9
```

4 思考题

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

```
Please press Enter to activate this console. QEMU: Terminated
jyt@fine:~/linux-6.11-rc7$ cd ../
jyt@fine:~$ ls *.c
os_lab0_test.c
jyt@fine:~$ riscv64-linux-gnu-gcc -o os_lab0_test.exe os_lab0_test.c
jyt@fine:~$ ls *.exe
os_lab0_test.exe
jyt@fine:~$
```

```
C os_lab0_test.c
1  #include <stdio.h>
2  int main(void)
3  {
4      int a, b;
5      a = 10;
6      b = 2 * a;
7  }
```

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

```

jyt@fine: ~
jyt@fine:~$ riscv64-linux-gnu-objdump -d os_lab0_test.exe
os_lab0_test.exe:      file format elf64-littleriscv

Disassembly of section .plt:

0000000000000540 <.plt>:
540:  00002397      auipc    t2,0x2
544:  41c30333      sub      t1,t1,t3
548:  ac83be03      ld       t3,-1336(t2) # 2008 <__TMC_END__>
54c:  fd430313      addi     t1,t1,-44
550:  ac838293      addi     t0,t2,-1336
554:  00135313      srli     t1,t1,0x1
558:  0082b283      ld       t0,8(t0)
55c:  000e0067      jr       t3

0000000000000560 <__libc_start_main@plt>:
560:  00002e17      auipc    t3,0x2
564:  ab8e3e03      ld       t3,-1352(t3) # 2018 <__libc_start_main@GLIBC_2.34>
568:  000e0367      jalr     t1,t3
56c:  00000013      nop

Disassembly of section .text:

0000000000000570 <_start>:
570:  022000ef      jal      ra,592 <load_gp>
574:  87aa         mv       a5,a0
576:  00002517      auipc    a0,0x2
57a:  aba53503      ld       a0,-1350(a0) # 2030 <_GLOBAL_OFFSET_TABLE_+0x10>
57e:  6582         ld       a1,0(sp)
580:  0030         addi     a2,sp,8

```

主函数如下:

```

0000000000000628 <main>:
628:  1101         addi     sp,sp,-32
62a:  ec22         sd       s0,24(sp)
62c:  1000         addi     s0,sp,32
62e:  47a9         li       a5,10          a=10;
630:  fef42423     sw       a5,-24(s0)
634:  fe842783     lw       a5,-24(s0)
638:  0017979b     slliw    a5,a5,0x1      b=2*a;
63c:  fef42623     sw       a5,-20(s0)
640:  4781         li       a5,0
642:  853e         mv       a0,a5
644:  6462         ld       s0,24(sp)
646:  6105         addi     sp,sp,32
648:  8082         ret
jyt@fine:~$

```

3. 调试Linux时:

(a) 在 GDB 中查看汇编代码 (不使用任何插件的情况下)

查阅资料后得知使用 `disassemble` 指令时, 加入 `/m` 可使源码和汇编码一起排列, 加入 `/r` 可以看到16进制代码。

`layout asm` 指令显示汇编代码窗口。

- 1 `$ gdb os_lab0_test.exe # 查看os_lab0_test对应汇编代码`
- 2 `(gdb) disassemble /m main`

```
(gdb) disassemble /m main
Dump of assembler code for function main:
0x00000628 <+0>:      add    %edx, (%ecx)
0x0000062a <+2>:      and    %ah, %ch
0x0000062c <+4>:      add    %dl, (%eax)
0x0000062e <+6>:      test   $0xf4242347, %eax
0x00000633 <+11>:     incb   -0x64017bd9(%ebx)
0x00000639 <+17>:     xchg   %eax, %edi
0x0000063a <+18>:     pop    %ss
0x0000063b <+19>:     add    %ah, (%ebx)
0x0000063d <+21>:     es hlt
0x0000063f <+23>:     incb   0x62853e47(%ecx)
0x00000645 <+29>:     fs
0x00000646 <+30>:     .byte 0x5
0x00000647 <+31>:     popa
0x00000648 <+32>:     .byte 0x82
0x00000649 <+33>:     .byte 0x80
End of assembler dump.
(gdb)
```

1 | (gdb) layout asm

```
jyt@fine: ~
0x628 <main>      add    %edx, (%ecx)
0x62a <main+2>    and    %ah, %ch
0x62c <main+4>    add    %dl, (%eax)
0x62e <main+6>    test   $0xf4242347, %eax
0x633 <main+11>   incb   -0x64017bd9(%ebx)
0x639 <main+17>   xchg   %eax, %edi
0x63a <main+18>   pop    %ss
0x63b <main+19>   add    %ah, (%ebx)
0x63d <main+21>   es hlt
0x63f <main+23>   incb   0x62853e47(%ecx)
0x645 <main+29>   fs
0x646 <main+30>   .byte 0x5
0x647 <main+31>   popa
0x648 <main+32>   .byte 0x82
0x649 <main+33>   .byte 0x80
exec No process In:
(gdb)
```

- 1 | \$ gdb-multiarch ./linux-6.11-rc7/vmlinux
- 2 | (gdb) target remote :1234 # 连接 qemu
- 3 | (gdb) layout asm # 查看linux内核的汇编代码

```
> 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 fnmadd.s    ft6, ft4, fs4, fs1, unknown
0x102c unimp
0x102e unimp
remote Thread 1.1 In:
(gdb)
```

(b) 在 0x80000000 处下断点

- 1 | (gdb) b *0x80000000

```
exec No process In:
(gdb) b *0x80000000
Breakpoint 1 at 0x80000000
(gdb)
```

(c) 查看所有已下的断点

1 | (gdb) i breakpoints

```
(gdb) i breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y    0x80000000
(gdb)
```

(d) 在 `0x80200000` 处下断点

1 | (gdb) b *0x80200000

```
(gdb) b *0x80200000
Breakpoint 2 at 0x80200000
(gdb)
```

(e) 清除 `0x80000000` 处的断点

1 | (gdb) delete 1

```
(gdb) delete 1
(gdb) i breakpoint
Num      Type      Disp Enb Address      What
2        breakpoint keep y    0x80200000
(gdb)
```

(f) 继续运行直到触发 `0x80200000` 处的断点

1 | (gdb) continue

```
(gdb) continue
Continuing.

Breakpoint 2, 0x0000000080200000 in ?? ()
(gdb)
```

(g) 单步调试一次

1 | (gdb) stepi

```
(gdb) stepi
0x0000000080200002 in ?? ()
(gdb)
```

(h) 退出 QEMU

使用 `Ctrl+A`，松开后按下 `X` 退出 QEMU:

```

Boot HART MHPM Count      : 0
Boot HART MIDELEG         : 0x00000000000000222
Boot HART MEDELEG         : 0x0000000000000b109
QEMU: Terminated
jyt@fine:~$

```

4. 使用 `make` 工具清除 Linux 的构建产物

```
1 | $ make clean
```

```

QEMU: Terminated
jyt@fine:~$ cd ./linux-6.11-rc7
jyt@fine:~/linux-6.11-rc7$ make clean
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
jyt@fine:~/linux-6.11-rc7$

```

5. `vmlinux` 和 `Image` 的关系和区别是什么?

区别: `vmlinux` 是直接编译出来的原生内核镜像文件, 未经压缩且包含调试信息, 可以直接用于调试。可以用来开发/调试内核, 但是不太适合将其部署到生产系统中。`Image` 同样是未压缩的, 但是优化掉了一些信息(经过objcopy转换, 去掉了一些符号表等信息)。它包含了实际运行在生产系统上的内核代码, 但是一般不含调试信息, 因此文件更小, 适合部署到生产系统中。

关系: 二者都是未压缩的Linux内核镜像文件, 构建Linux内核时会先生成 `vmlinux` 文件, 然后将其转化成 `Image` 文件, 以供实际使用。

5 心得体会

Lab0更多侧重于配置环境和熟悉相关操作, 总体而言难度较小, 踩坑较少。