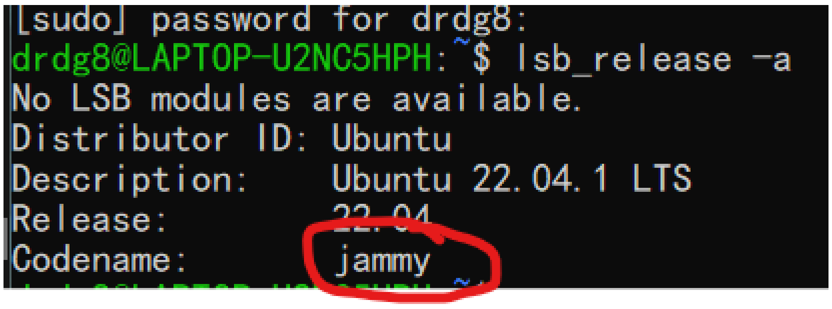
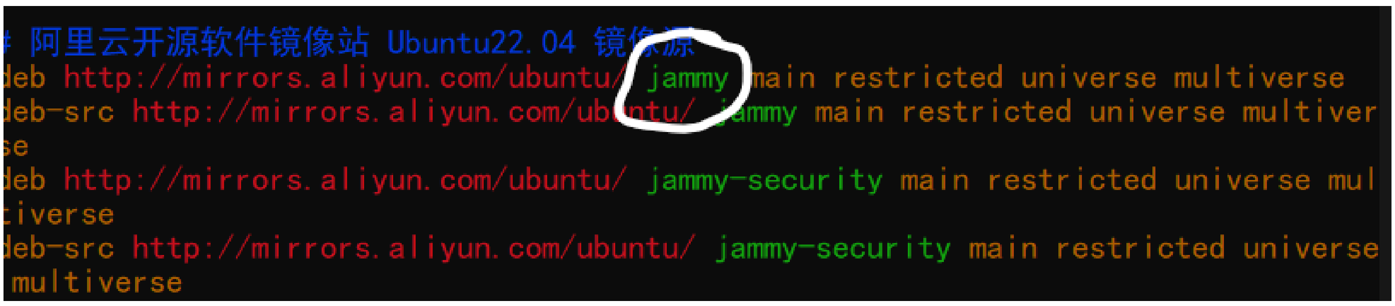
lllab0

安装必要的软件包

> 首先升级wsl到2，并升级Ubuntu到22.04，并安装必要的软件包。由于忘记截图，我简略描述问题。一个是国外源不稳定，一个是源版本没有对应，导致一堆错误。过程繁复，不再赘述。总之最终是安装完了。



```
[sudo] password for drdg8:
drdg8@LAPTOP-U2NC5HPH:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 22.04.1 LTS
Release:        22.04
Codename:       jammy
```

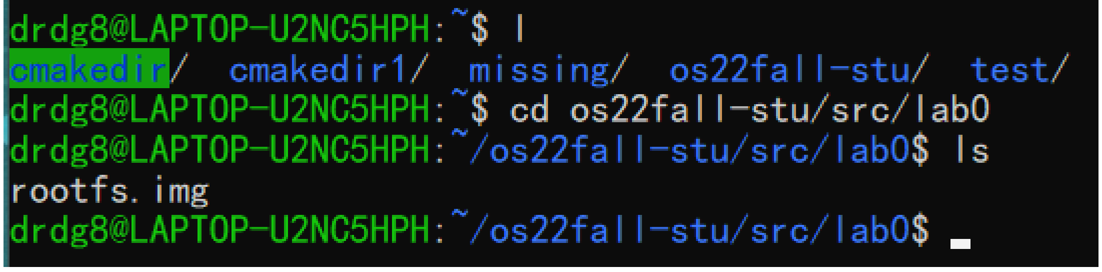


```
# 阿里云开源软件镜像站 Ubuntu22.04 镜像源
deb http://mirrors.aliyun.com/ubuntu/ jammy main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ jammy main restricted universe multiver
se
deb http://mirrors.aliyun.com/ubuntu/ jammy-security main restricted universe mul
tiverse
deb-src http://mirrors.aliyun.com/ubuntu/ jammy-security main restricted universe
 multiverse
```

之后将仓库克隆到本地，并下载linux内核

> 我将linux内核直接放在os22fall-stu目录下了，路径为：

> /home/drdg8/os22fall-stu/linux-6.0-rc5



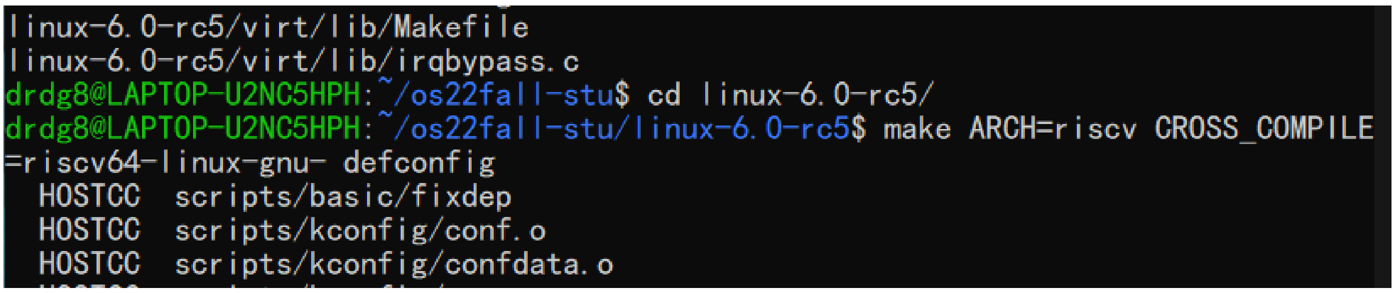
```
drdg8@LAPTOP-U2NC5HPH:~$ l
cmakedir/  cmakedir1/  missing/  os22fall-stu/  test/
drdg8@LAPTOP-U2NC5HPH:~$ cd os22fall-stu/src/lab0
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/src/lab0$ ls
rootfs.img
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/src/lab0$
```

```
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu$
```

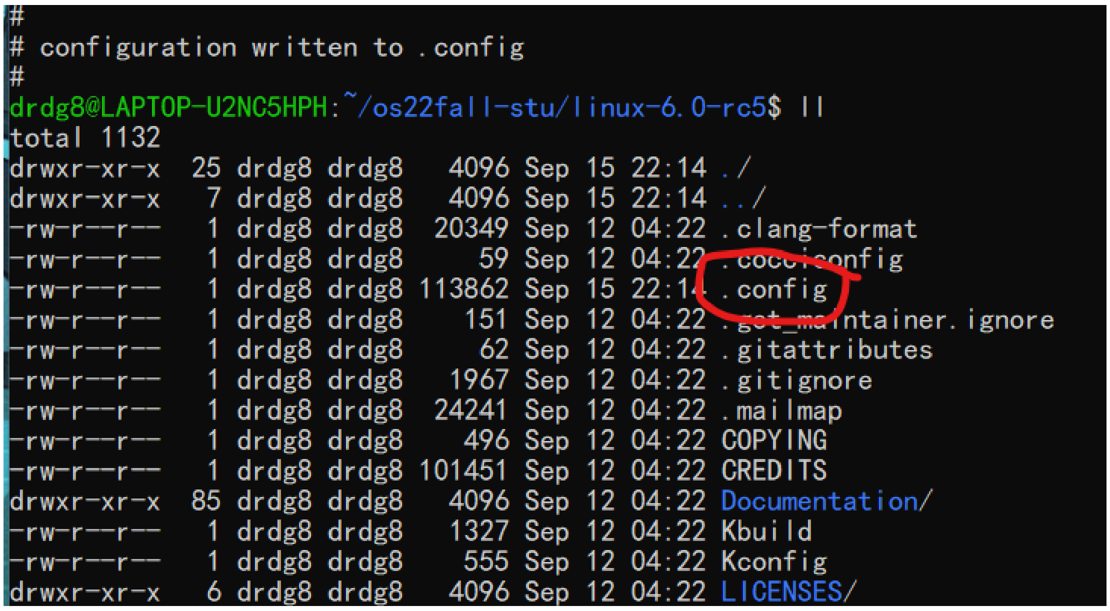
可以看到根文件系统的镜像已经在目录里了。

将linux内核解压并编译

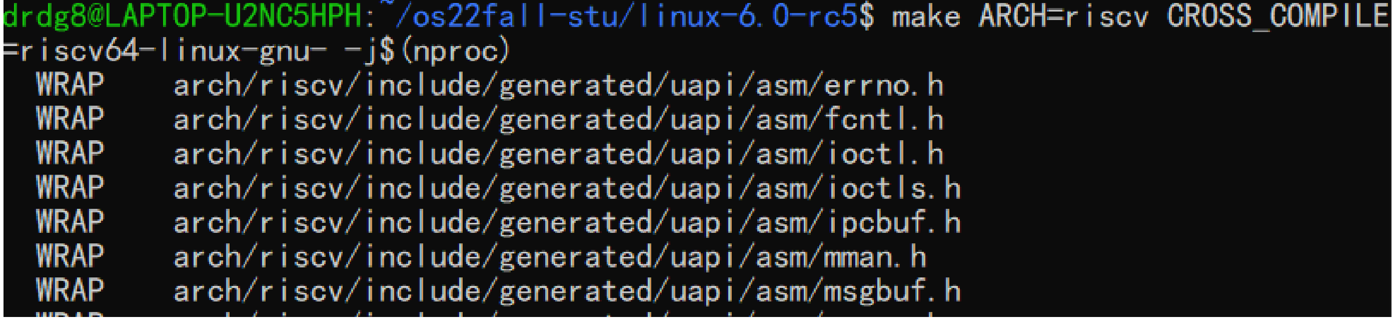
在第一次构建时，我直接用了``defconfig``，默认是没有调试信息的，所以在后面做gdb调试时，``n/p/display``等命令都不能实现。因此在最后重新构建linux kernel时，加上调试信息选项后就能正常使用了。



```
linux-6.0-rc5/virt/lib/Makefile
linux-6.0-rc5/virt/lib/irqbypass.c
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu$ cd linux-6.0-rc5/
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/linux-6.0-rc5$ make ARCH=riscv CROSS_COMPILE
=riscv64-linux-gnu- defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/confdata.o
```



```
#
# configuration written to .config
#
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/linux-6.0-rc5$ ll
total 1132
drwxr-xr-x 25 drdg8 drdg8   4096 Sep 15 22:14 ./
drwxr-xr-x  7 drdg8 drdg8   4096 Sep 15 22:14 ../
-rw-r--r--  1 drdg8 drdg8  20349 Sep 12 04:22 .clang-format
-rw-r--r--  1 drdg8 drdg8     59 Sep 12 04:22 .cocciconfig
-rw-r--r--  1 drdg8 drdg8 113862 Sep 15 22:14 .config
-rw-r--r--  1 drdg8 drdg8    151 Sep 12 04:22 .get_maintainer.ignore
-rw-r--r--  1 drdg8 drdg8     62 Sep 12 04:22 .gitattributes
-rw-r--r--  1 drdg8 drdg8   1967 Sep 12 04:22 .gitignore
-rw-r--r--  1 drdg8 drdg8  24241 Sep 12 04:22 .mailmap
-rw-r--r--  1 drdg8 drdg8    496 Sep 12 04:22 COPYING
-rw-r--r--  1 drdg8 drdg8 101451 Sep 12 04:22 CREDITS
drwxr-xr-x 85 drdg8 drdg8   4096 Sep 12 04:22 Documentation/
-rw-r--r--  1 drdg8 drdg8   1327 Sep 12 04:22 Kbuild
-rw-r--r--  1 drdg8 drdg8    555 Sep 12 04:22 Kconfig
drwxr-xr-x  6 drdg8 drdg8   4096 Sep 12 04:22 LICENSES/
```



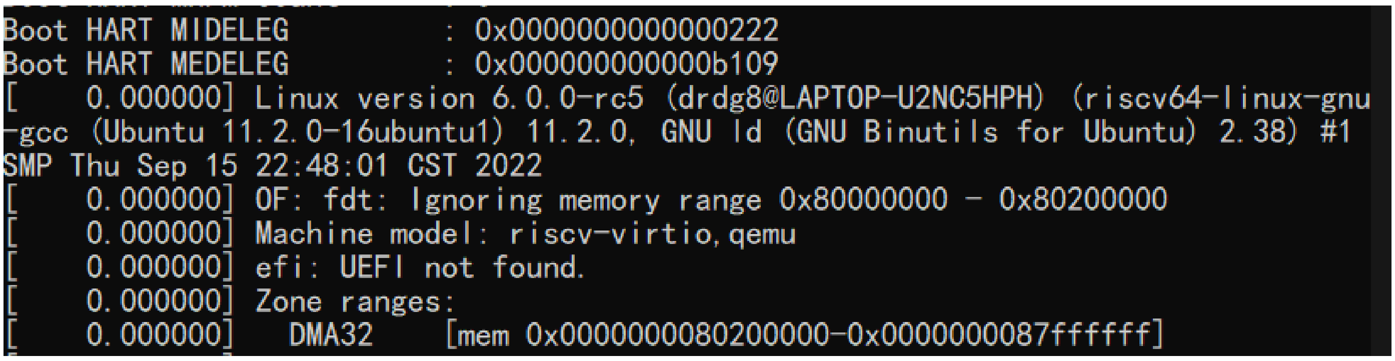
```
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/linux-6.0-rc5$ make ARCH=riscv CROSS_COMPILE
=riscv64-linux-gnu- -j$(nproc)
  WRAP    arch/riscv/include/generated/uapi/asm/errno.h
  WRAP    arch/riscv/include/generated/uapi/asm/fcntl.h
  WRAP    arch/riscv/include/generated/uapi/asm/ioctl.h
  WRAP    arch/riscv/include/generated/uapi/asm/ioctls.h
  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
```

查阅``make help``信息可知，``ARCH=``指定架构，``CROSS_COMPILE=``指定交叉编译选项。前一个命令是生成默认配置，后一个命令是构建内核。

使用QEMU调试内核

根据我的下载路径将命令改成如下：

> qemu-system-riscv64 -nographic -machine virt -kernel /home/drdg8/os22fall-stu/linux-6.0-rc5/arch/riscv/boot/Image -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0" -bios default -drive file=/home/drdg8/os22fall-stu/src/lab0/rootfs.img,format=raw,id=hd0



```
Boot HART MIDELEG         : 0x0000000000000222
Boot HART MEDELEG         : 0x000000000000b109
[    0.000000] Linux version 6.0.0-rc5 (drdg8@LAPTOP-U2NC5HPH) (riscv64-linux-gnu
-gcc (Ubuntu 11.2.0-16ubuntu1) 11.2.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #1
SMP Thu Sep 15 22:48:01 CST 2022
[    0.000000] OF: fdt: Ignoring memory range 0x80000000 - 0x80200000
[    0.000000] Machine model: riscv-virtio,qemu
[    0.000000] efi: UEFI not found.
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x0000000080200000-0x0000000087ffffff]
```

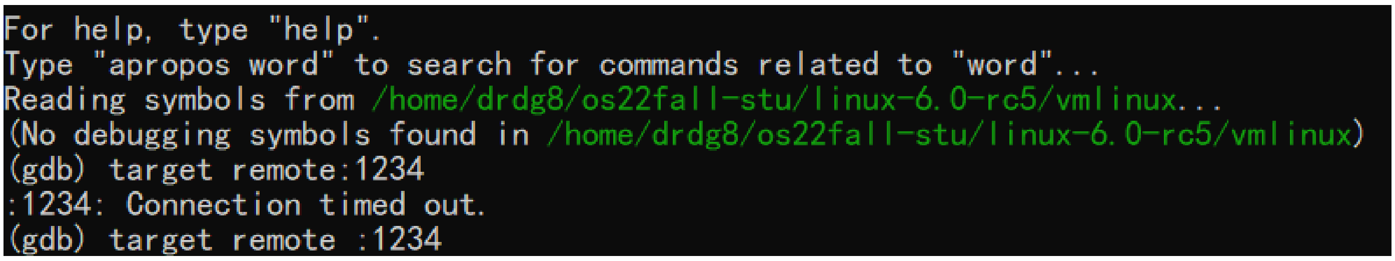
可以看到，内核已经正常运行了起来。截图是内核信息。

使用GDB调试内核

在上一步直接运行gdb调试功能时：

> gdb-multiarch /home/drdg8/os22fall-stu/linux-6.0-rc5/vmlinux

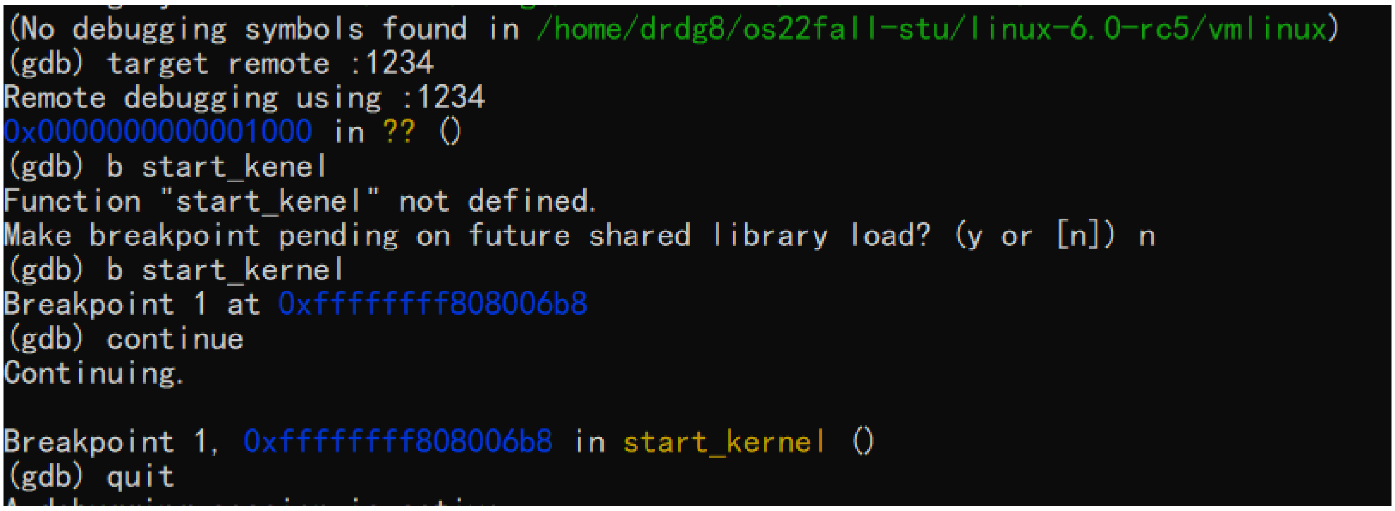
```
发现连接不上，显示``connection timed out``:
```



```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from /home/drdg8/os22fall-stu/linux-6.0-rc5/vmlinux...
(No debugging symbols found in /home/drdg8/os22fall-stu/linux-6.0-rc5/vmlinux)
(gdb) target remote:1234
:1234: Connection timed out.
(gdb) target remote :1234
```

```
仔细检查发现，原来是没有加``-S -s``，其中``-S``表示暂停CPU执行，``-s``表示tcp:1234开放:
```

> qemu-system-riscv64 -nographic -machine virt -kernel /home/drdg8/os22fall-stu/linux-6.0-rc5/arch/riscv/boot/Image -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0" -bios default -drive file=/home/drdg8/os22fall-stu/src/lab0/rootfs.img,format=raw,id=hd0 -S -s

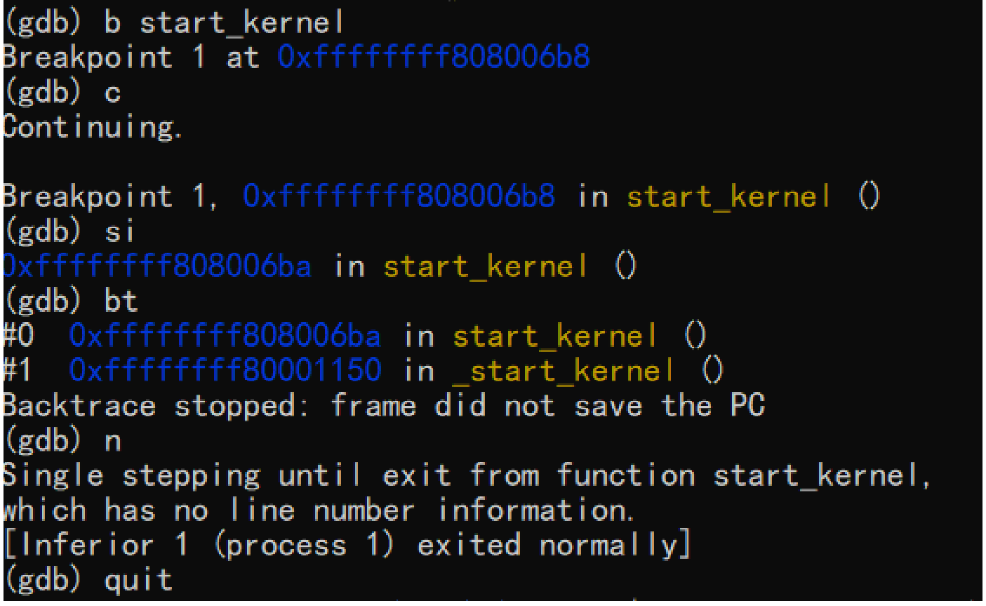


```
(No debugging symbols found in /home/drdg8/os22fall-stu/linux-6.0-rc5/vmlinux)
(gdb) target remote :1234
Remote debugging using :1234
0x0000000000001000 in ?? ()
(gdb) b start_kenel
Function "start_kenel" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff808006b8
(gdb) continue
Continuing.

Breakpoint 1, 0xffffffff808006b8 in start_kernel ()
(gdb) quit
```

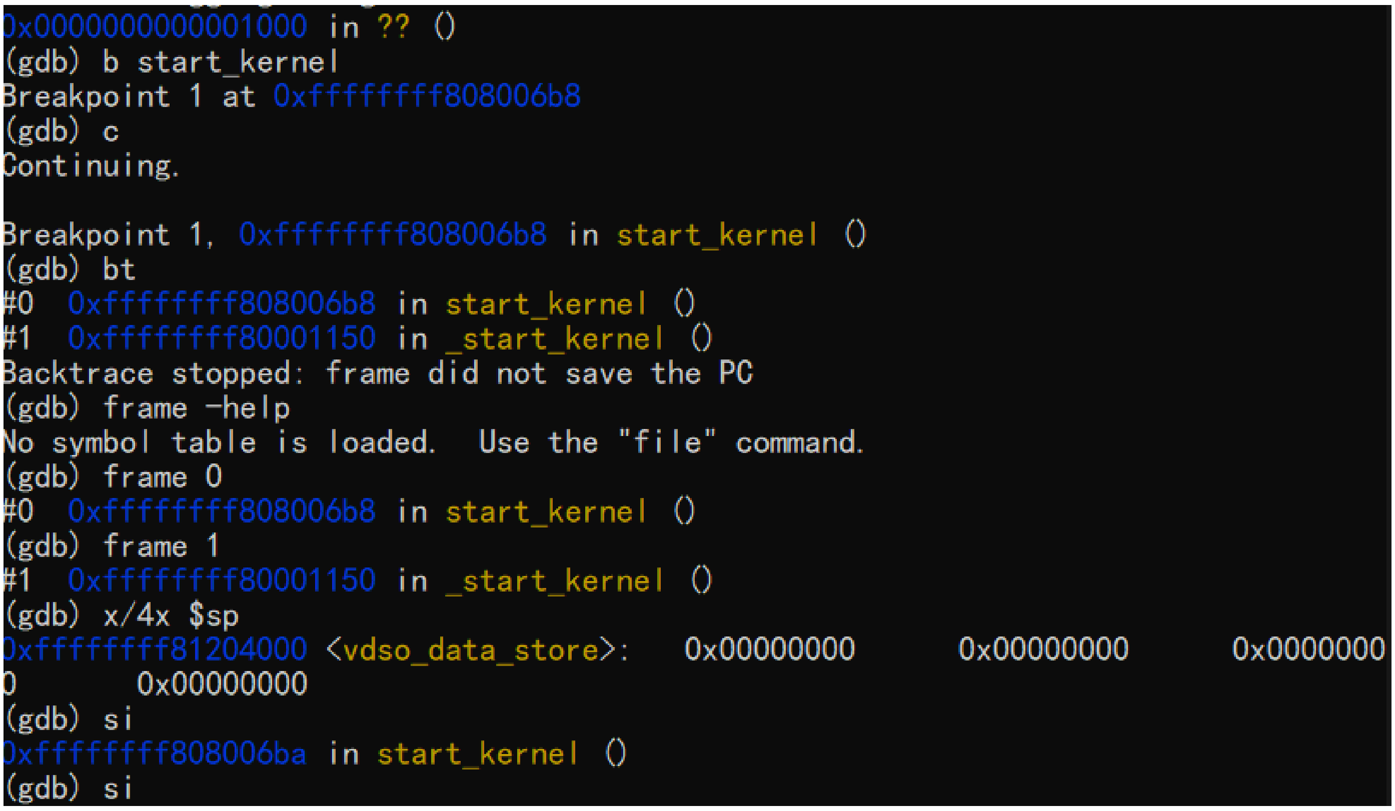
```
命令改好后就能正常调试了。
```

使用各种命令

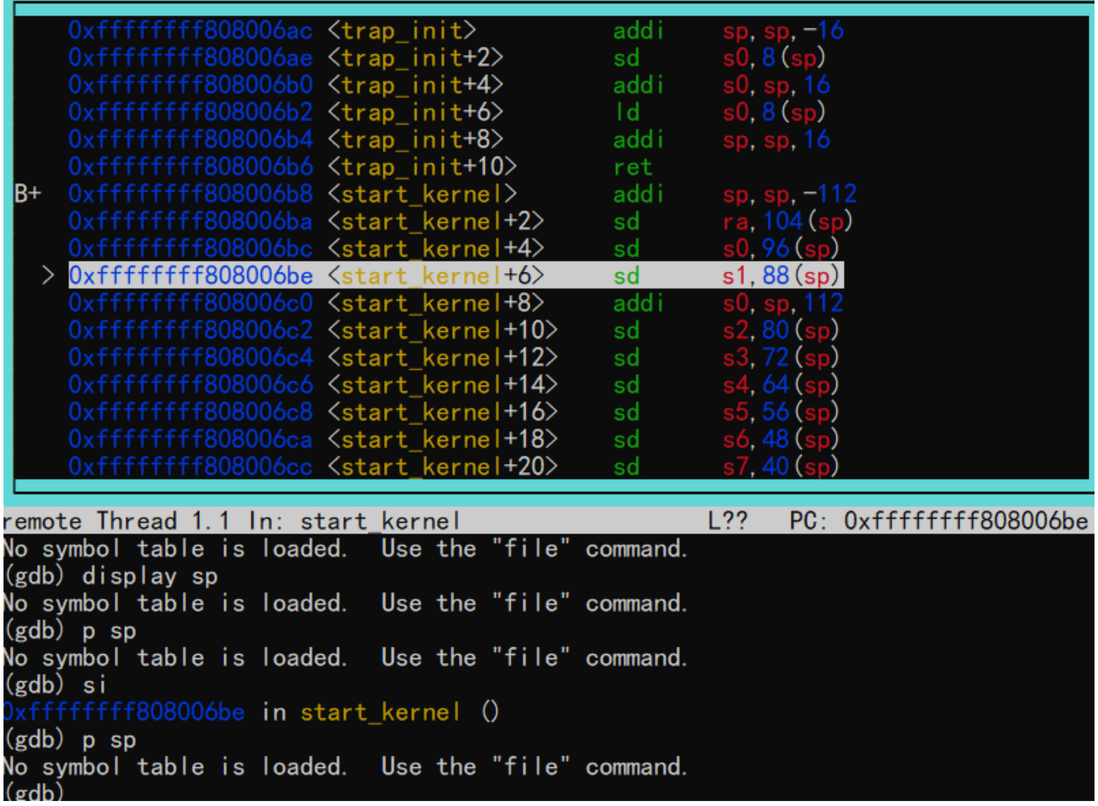
```
在默认生成的内核中没有调试信息，所以``n/p/display``等命令都不能正常使用.
```



```
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff808006b8
(gdb) c
Continuing.

Breakpoint 1, 0xffffffff808006b8 in start_kernel ()
(gdb) si
0xffffffff808006ba in start_kernel ()
(gdb) bt
#0  0xffffffff808006ba in start_kernel ()
#1  0xffffffff80001150 in _start_kernel ()
Backtrace stopped: frame did not save the PC
(gdb) n
Single stepping until exit from function start_kernel,
which has no line number information.
[Inferior 1 (process 1) exited normally]
(gdb) quit
```

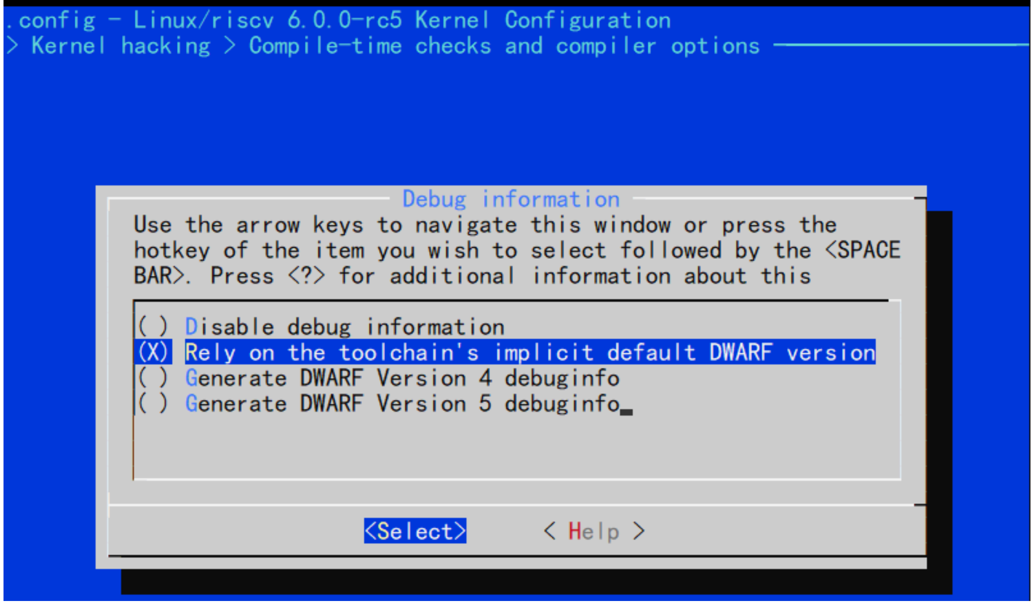


```
0x0000000000001000 in ?? ()
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff808006b8
(gdb) c
Continuing.

Breakpoint 1, 0xffffffff808006b8 in start_kernel ()
(gdb) bt
#0  0xffffffff808006b8 in start_kernel ()
#1  0xffffffff80001150 in _start_kernel ()
Backtrace stopped: frame did not save the PC
(gdb) frame -help
No symbol table is loaded.  Use the "file" command.
(gdb) frame 0
#0  0xffffffff808006b8 in start_kernel ()
(gdb) frame 1
#1  0xffffffff80001150 in _start_kernel ()
(gdb) x/4x $sp
0xffffffff81204000 <vdso_data_store>:   0x00000000      0x00000000      0x0000000
0       0x00000000
(gdb) si
0xffffffff808006ba in start_kernel ()
(gdb) si
```

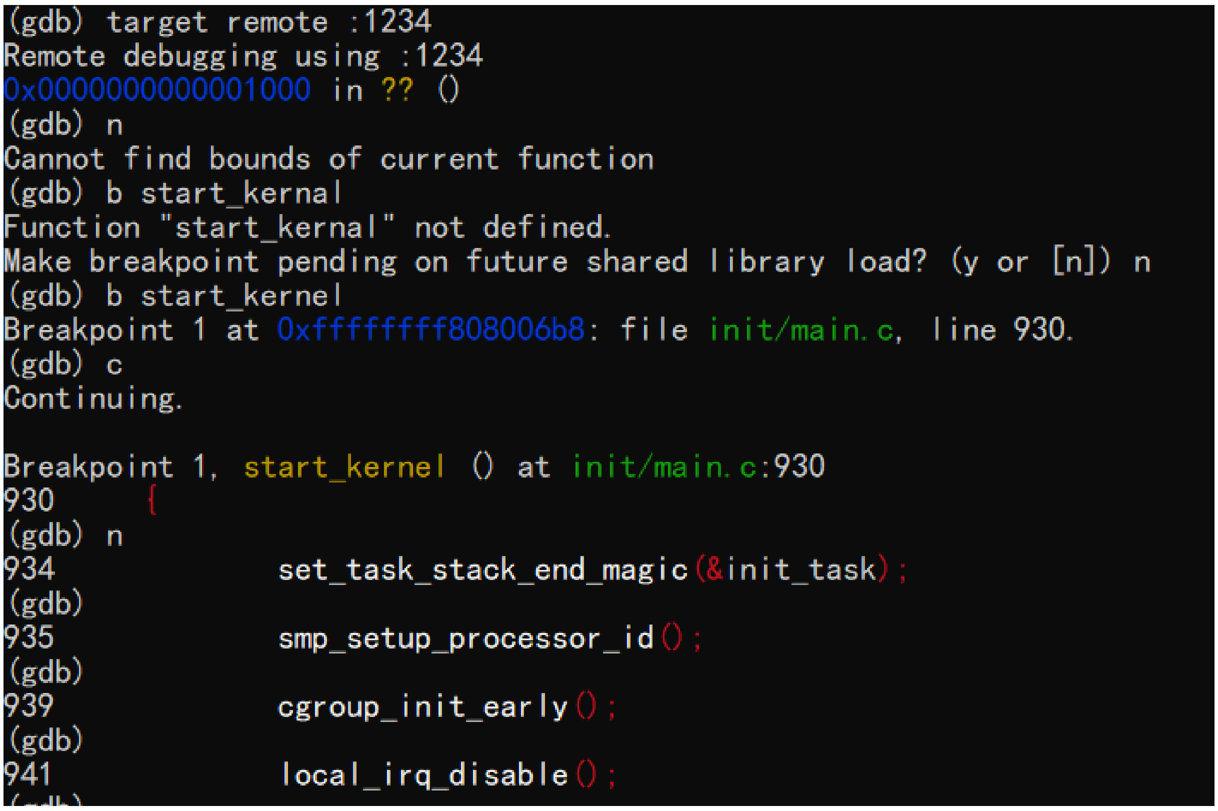


```
    0xffffffff808006ac <trap_init>         addi    sp,sp,-16
    0xffffffff808006ae <trap_init+2>       sd      s0,8(sp)
    0xffffffff808006b0 <trap_init+4>       addi    s0,sp,16
    0xffffffff808006b2 <trap_init+6>       ld      s0,8(sp)
    0xffffffff808006b4 <trap_init+8>       addi    sp,sp,16
    0xffffffff808006b6 <trap_init+10>      ret
B+  0xffffffff808006b8 <start_kernel>      addi    sp,sp,-112
    0xffffffff808006ba <start_kernel+2>    sd      ra,104(sp)
    0xffffffff808006bc <start_kernel+4>    sd      s0,96(sp)
  > 0xffffffff808006be <start_kernel+6>    sd      s1,88(sp)
    0xffffffff808006c0 <start_kernel+8>    addi    s0,sp,112
    0xffffffff808006c2 <start_kernel+10>   sd      s2,80(sp)
    0xffffffff808006c4 <start_kernel+12>   sd      s3,72(sp)
    0xffffffff808006c6 <start_kernel+14>   sd      s4,64(sp)
    0xffffffff808006c8 <start_kernel+16>   sd      s5,56(sp)
    0xffffffff808006ca <start_kernel+18>   sd      s6,48(sp)
    0xffffffff808006cc <start_kernel+20>   sd      s7,40(sp)

remote Thread 1.1 In: start_kernel                    L??   PC: 0xffffffff808006be
No symbol table is loaded.  Use the "file" command.
(gdb) display sp
No symbol table is loaded.  Use the "file" command.
(gdb) p sp
No symbol table is loaded.  Use the "file" command.
(gdb) si
0xffffffff808006be in start_kernel ()
(gdb) p sp
No symbol table is loaded.  Use the "file" command.
(gdb)
```

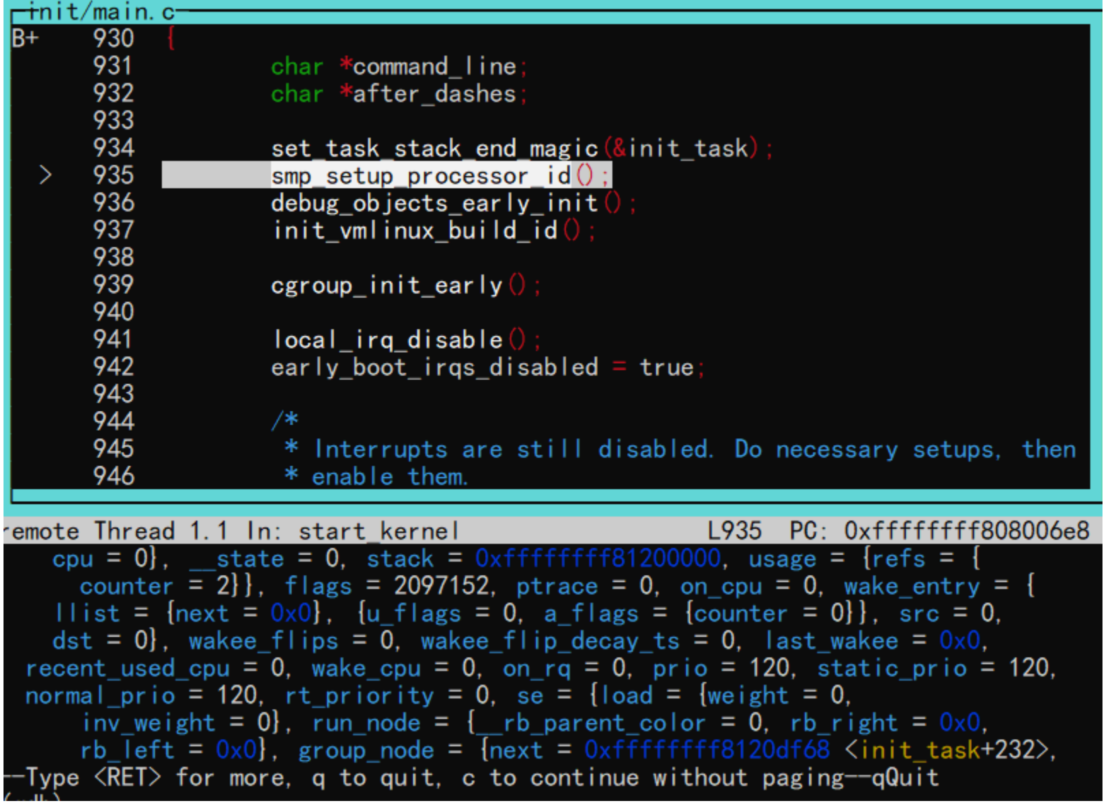
报错因为是gcc编译,编译选项中没有开启-g，所以编译信息不足。
第三步上方的汇编代码是``layout asm``命令生成的。
可以看到，``backtrace/break/strp instruction/continue``等等命令都可以使用，许多命令的使用方法由于实验指导里说的很详细了，我就不再赘述。



```
.config - Linux/riscv 6.0.0-rc5 Kernel Configuration
> Kernel hacking > Compile-time checks and compiler options
                     Debug information
  Use the arrow keys to navigate this window or press the
  hotkey of the item you wish to select followed by the <SPACE
  BAR>. Press <?> for additional information about this

  ( ) Disable debug information
  (X) Rely on the toolchain's implicit default DWARF version
  ( ) Generate DWARF Version 4 debuginfo
  ( ) Generate DWARF Version 5 debuginfo

                  <Select>      < Help >
```



```
(gdb) target remote :1234
Remote debugging using :1234
0x0000000000001000 in ?? ()
(gdb) n
Cannot find bounds of current function
(gdb) b start_kernal
Function "start_kernal" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff808006b8: file init/main.c, line 930.
(gdb) c
Continuing.

Breakpoint 1, start_kernel () at init/main.c:930
930     {
(gdb) n
934             set_task_stack_end_magic(&init_task);
(gdb)
935             smp_setup_processor_id();
(gdb)
939             cgroup_init_early();
(gdb)
941             local_irq_disable();
```



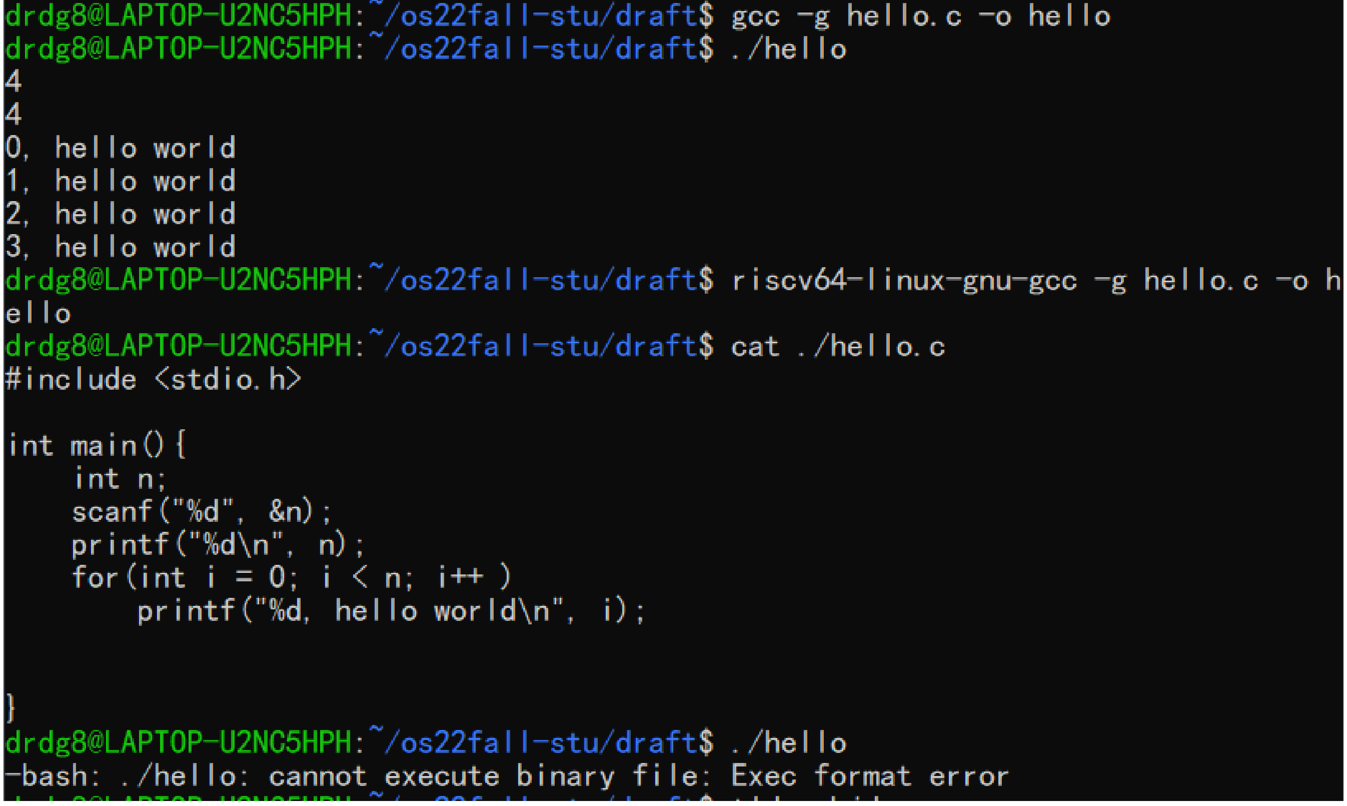
```
┌─init/main.c─
B+      {
                char *command_line;
                char *after_dashes;

                set_task_stack_end_magic(&init_task);
  >             smp_setup_processor_id();
                debug_objects_early_init();
                init_vmlinux_build_id();

                cgroup_init_early();

                local_irq_disable();
                early_boot_irqs_disabled = true;

                /*
                 * Interrupts are still disabled. Do necessary setups, then
                 * enable them.
```

```
remote Thread 1.1 In: start_kernel                         L935   PC: 0xffffffff808006e8
    cpu = 0}, __state = 0, stack = 0xffffffff81200000, usage = {refs = {
      counter = 2}}, flags = 2097152, ptrace = 0, on_cpu = 0, wake_entry = {
    llist = {next = 0x0}, {u_flags = 0, a_flags = {counter = 0}}, src = 0,
    dst = 0}, wakee_flips = 0, wakee_flip_decay_ts = 0, last_wakee = 0x0,
  recent_used_cpu = 0, wake_cpu = 0, on_rq = 0, prio = 120, static_prio = 120,
  normal_prio = 120, rt_priority = 0, se = {load = {weight = 0,
      inv_weight = 0}, run_node = {__rb_parent_color = 0, rb_right = 0x0,
      rb_left = 0x0}, group_node = {next = 0xffffffff8120df68 <init_task+232>,
--Type <RET> for more, q to quit, c to continue without paging--qQuit
```

之后重新用``make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- menuconfig``生成内核，并在上图一的设置中改成``rely on the toolchain's implict default DWARF version``就可以了。
可以看到，``n/p``等命令都可以使用了，其中上图二也能显示file地址与函数名称了，上图三是使用``p init_task``的结果。
到此，试验任务完成。

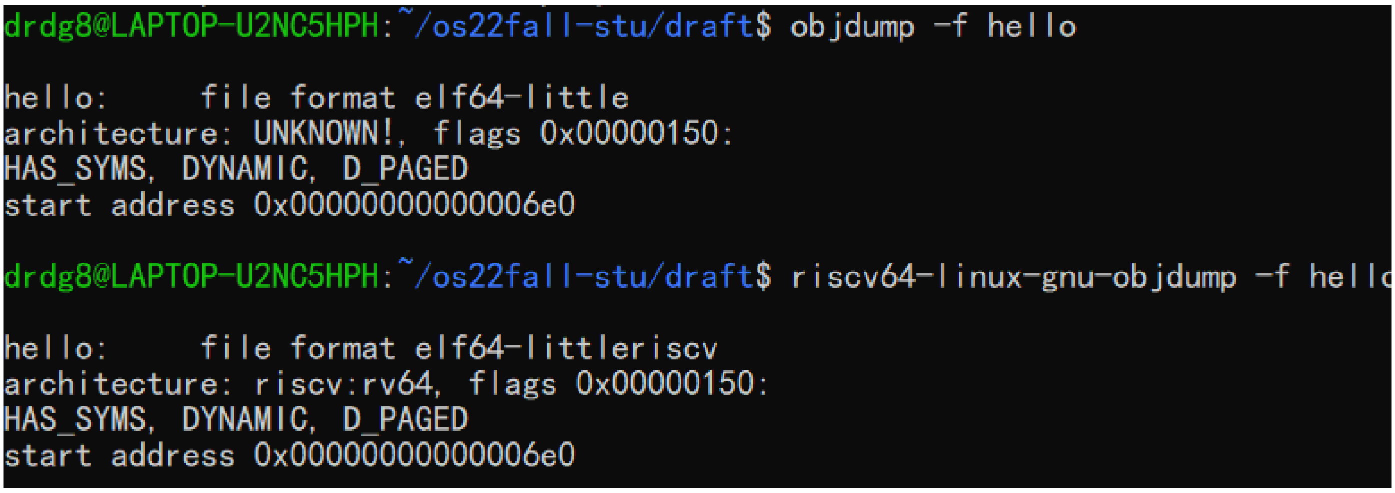
思考题

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


```
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/draft$ gcc -g hello.c -o hello
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/draft$ ./hello
4
4
0, hello world
1, hello world
2, hello world
3, hello world
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/draft$ riscv64-linux-gnu-gcc -g hello.c -o h
ello
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/draft$ cat ./hello.c
#include <stdio.h>

int main(){
    int n;
    scanf("%d", &n);
    printf("%d\n", n);
    for(int i = 0; i < n; i++ )
        printf("%d, hello world\n", i);


}
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/draft$ ./hello
-bash: ./hello: cannot execute binary file: Exec format error
```

上图中，我分别用``gcc``与``riscv64-linux-gnu-gcc``编译了一个``hello.c``，可以看到直接用``gcc``编译的可执行文件可以直接执行，而``riscv64-linux-gnu-gcc``编译的则不能，报错``Exec format error``.这是``x86``的架构不能执行``ARM``也就是``RISCV``指令集产生的文件。

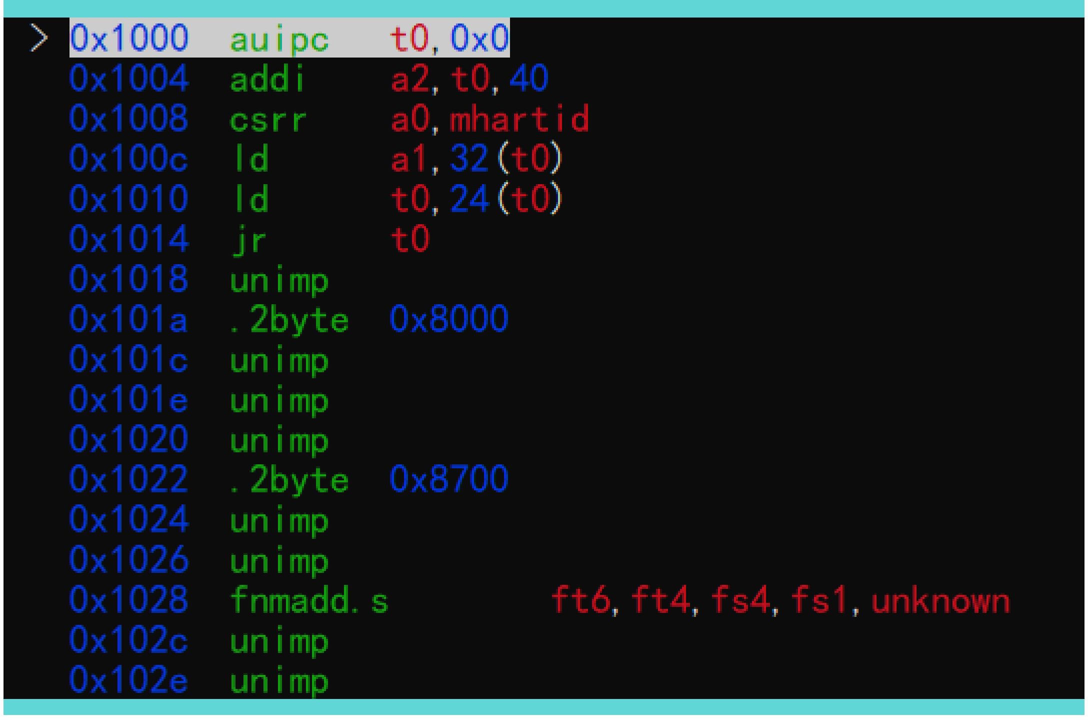
2. 使用`riscv64-linux-gnu-objdump`反汇编 1 中得到的编译产物


```
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/draft$ objdump -f hello

hello:     file format elf64-little
architecture: UNKNOWN!, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x00000000000006e0

drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/draft$ riscv64-linux-gnu-objdump -f hello

hello:     file format elf64-littleriscv
architecture: riscv:rv64, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x00000000000006e0
```

可以看到objdump都能反编译文件，但riscv64的能检测出架构。

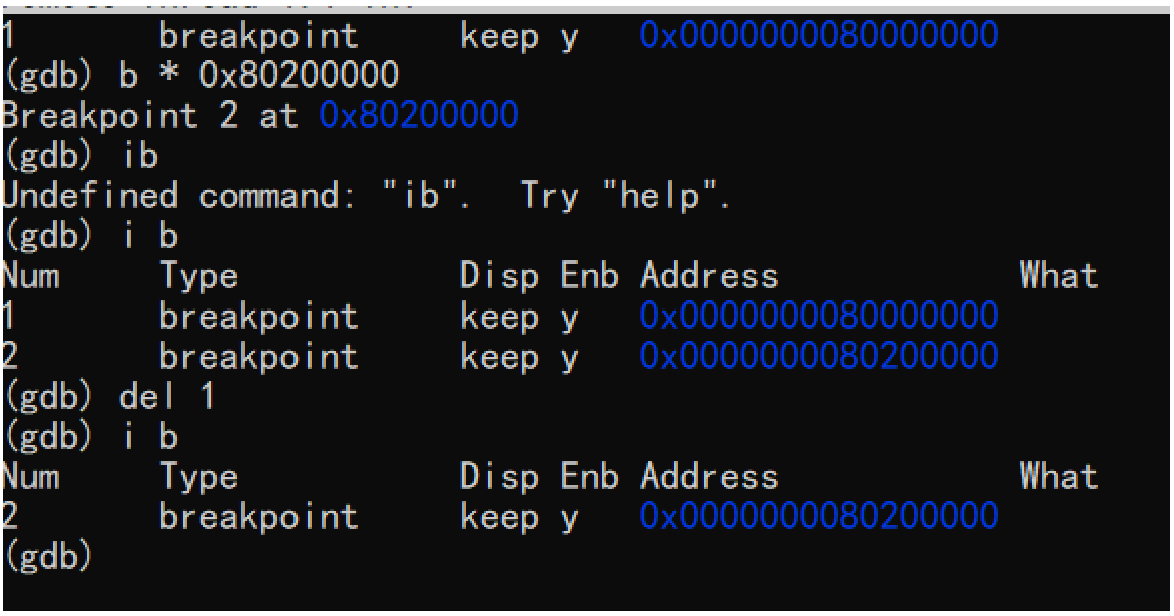
3. 调试Linux:

   a. 使用 layout asm 显示汇编代码



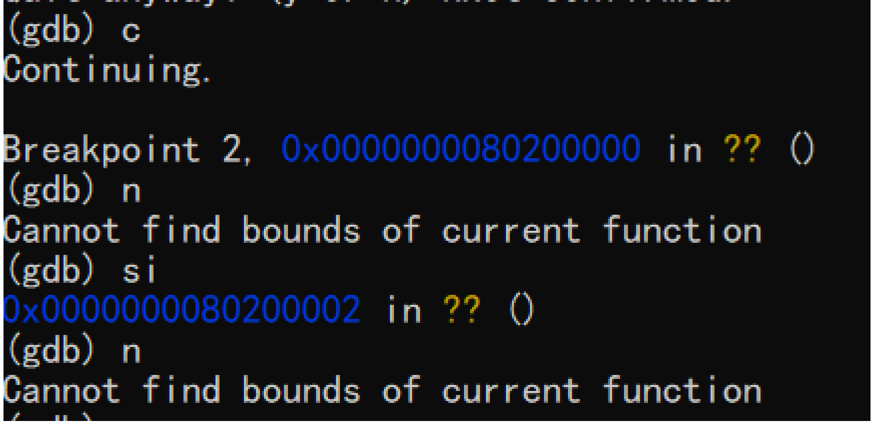
```
> 0x1000  auipc   t0,0x0
  0x1004  addi    a2,t0,40
  0x1008  csrr    a0,mhartid
  0x100c  ld      a1,32(t0)
  0x1010  ld      t0,24(t0)
  0x1014  jr      t0
  0x1018  unimp
  0x101a  .2byte  0x8000
  0x101c  unimp
  0x101e  unimp
  0x1020  unimp
  0x1022  .2byte  0x8700
  0x1024  unimp
  0x1026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

```
b.c.d.e. 用 b * 0x80000000 设下断点，用 info breakpoint(i b) 查看断电， 用 del 2 来删除第几个断点。
```



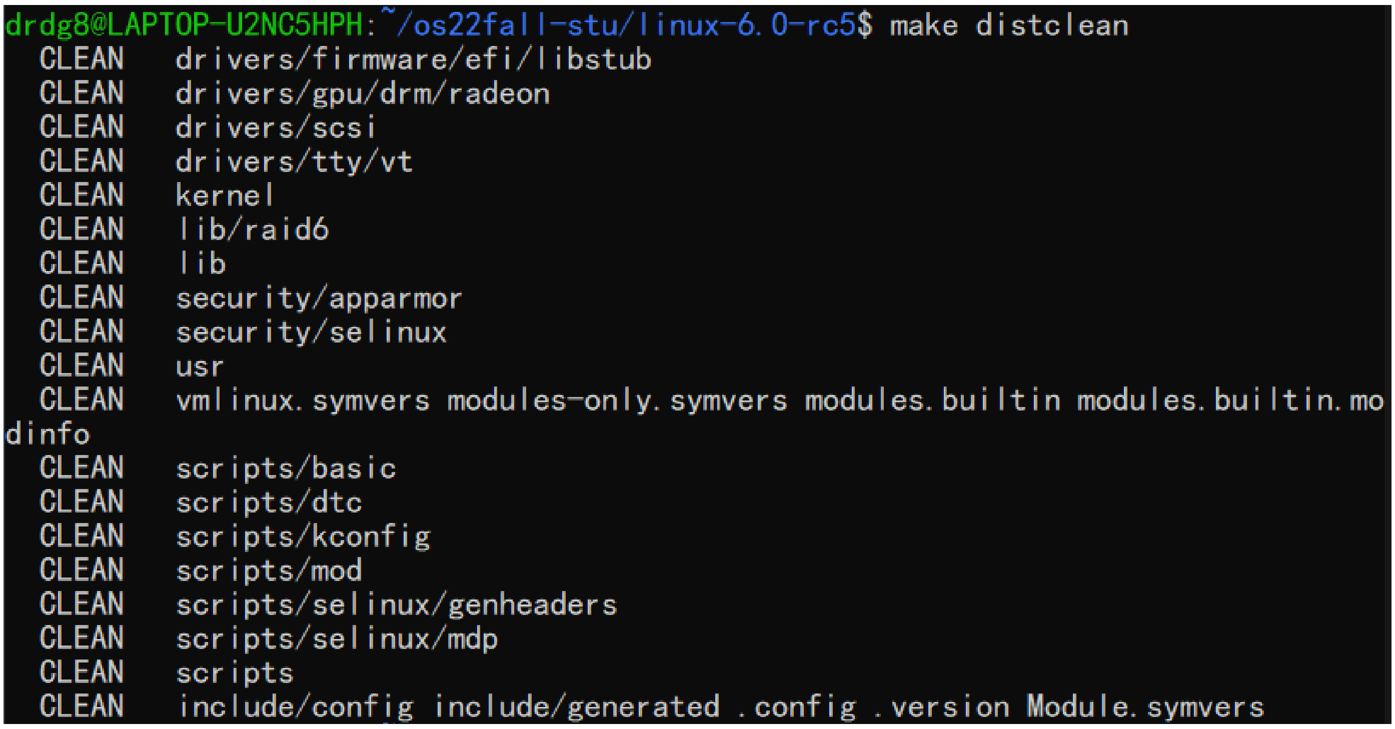
```
1       breakpoint     keep y   0x0000000080000000
(gdb) b * 0x80200000
Breakpoint 2 at 0x80200000
(gdb) ib
Undefined command: "ib".  Try "help".
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000080000000
2       breakpoint     keep y   0x0000000080200000
(gdb) del 1
(gdb) i b
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x0000000080200000
(gdb)
```

```
f.g.h. 之后continue到指定断点，用si可以执行单条指令，但n不行，提示``cannot find the bounds of function``。这是之前默认配置时做的。
```


```
(gdb) c
Continuing.

Breakpoint 2, 0x0000000080200000 in ?? ()
(gdb) n
Cannot find bounds of current function
(gdb) si
0x0000000080200002 in ?? ()
(gdb) n
Cannot find bounds of current function
```

4. 用make清除构建产物

   查询make help可知，make distclean能清除配置文件与构建产物.


```
drdg8@LAPTOP-U2NC5HPH:~/os22fall-stu/linux-6.0-rc5$ make distclean
  CLEAN   drivers/firmware/efi/libstub
  CLEAN   drivers/gpu/drm/radeon
  CLEAN   drivers/scsi
  CLEAN   drivers/tty/vt
  CLEAN   kernel
  CLEAN   lib/raid6
  CLEAN   lib
  CLEAN   security/apparmor
  CLEAN   security/selinux
  CLEAN   usr
  CLEAN   vmlinux.symvers modules-only.symvers modules.builtin modules.builtin.mo
dinfo
  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
```

5. vmlinux 和 Image 的关系和区别是什么？

   这是网上查到的信息：

   > vmlinux：Linux内核编译出来的原始的内核文件，elf格式，未做压缩处理。
   > Image：Linux内核编译时，使用objcopy处理vmlinux后生成的二进制内核映像。

   > Image: The generic Linux kernel binary image file. vmlinux: This is the Linux kernel in a statically linked executable file format.

   在之前的实验中，我们在直接运用根文件系统或者qemu启动linux时，用的是image格式，而用gdb调试时则用的时vmlinux，因此，vmlinux一定包含着更多的调试信息，便于我们学习。