浙江大学实验报告

课程名称： 操作系统

实验项目名称：RV64环境搭建和内核编译

学生姓名：颜晗　　学号：3200105515

电子邮件地址：3050057413@qq.com

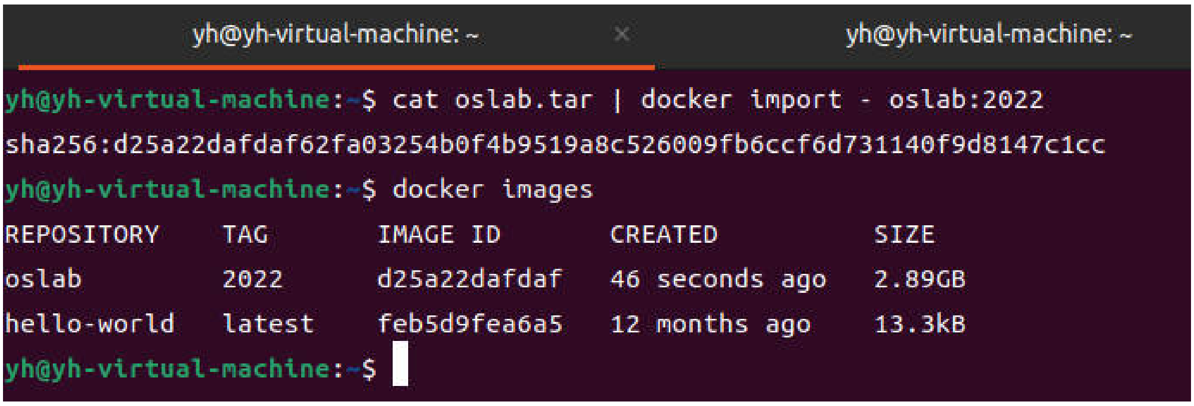
实验日期：2022-10

一、实验内容

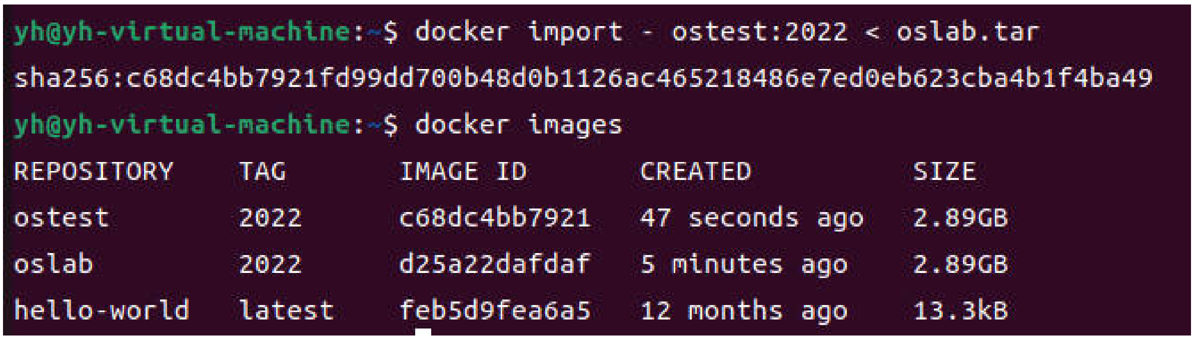
1. 搭建 Docker 环境

根据官方文档 选择自己喜欢的方法在`ubuntu 22.04`中安装`docker`环境并启动`docker`服务。

```
cat oslab.tar | docker import - oslab:2022
### 导入docker镜像命令
### 主要命令为 docker import 部分，‘-’后部分用于为新镜像命名为“oslab”并给予“2022”的tag
### 显然import缺少一个输入的归档文件用于创建镜像
### oslab.tar 即实验提供的归档文件.
### 通过管道连接 cat 和 import 命令，将cat命令的输出（oslab.tar文件内容）变为import的输入
### 同理，我们有另外的命令形式完成任务
docker import - ostest:2022 < oslab.tar
### 如上，使用重定向符号也可完成任务
docker images
### 可查询所建立的镜像
```

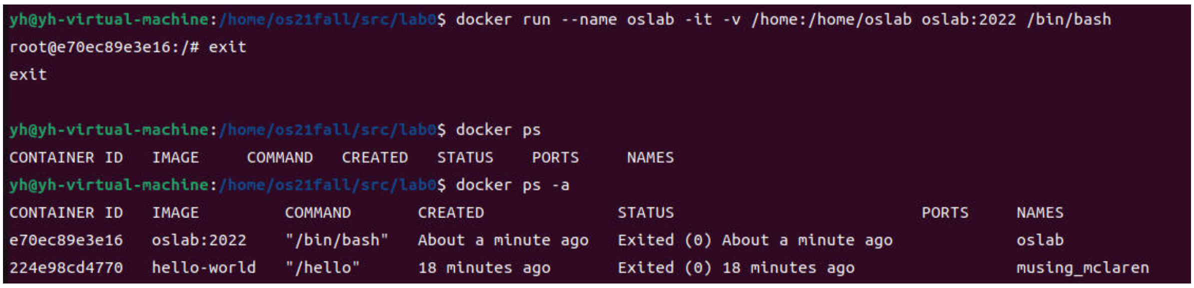


```
yh@yh-virtual-machine:~$ cat oslab.tar | docker import - oslab:2022
sha256:d25a22dafdaf62fa03254b0f4b9519a8c526009fb6ccf6d731140f9d8147c1cc
yh@yh-virtual-machine:~$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
oslab         2022      d25a22dafdaf   46 seconds ago   2.89GB
hello-world   latest    feb5d9fea6a5   12 months ago    13.3kB
yh@yh-virtual-machine:~$
```



```
yh@yh-virtual-machine:~$ docker import - ostest:2022 < oslab.tar
sha256:c68dc4bb7921fd99dd700b48d0b1126ac465218486e7ed0eb623cba4b1f4ba49
yh@yh-virtual-machine:~$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
ostest        2022      c68dc4bb7921   47 seconds ago   2.89GB
oslab         2022      d25a22dafdaf   5 minutes ago    2.89GB
hello-world   latest    feb5d9fea6a5   12 months ago    13.3kB
```

```
docker run --name oslab -it oslab:2022 /bin/bash
### run 命令从镜像中创建容器
### --name oslab 选项，将容器命名为oslab
### -i 交互模式启动容器 -t 为容器分配一个伪输入终端，两个选项通常搭配同时使用
### /bin/bash 容器应保证至少有一个进程在运行，该参数表示容器创建（启动）后执行bash命令
### 即整条命令使用镜像oslab:2022以交互模式创建一个命名为oslab的容器，并在容器中执行/bin/bash命令

docker exec -it oslab /bin/bash
###参数意义同run命令，在退出容器后重新进入 or 在多个终端进入容器

docker run --name oslab -it -v /home:/home/oslab oslab:2022 /bin/bash
### -v参数使得创建容器时挂载主机的一个目录，如上命令将主机的/home 目录挂载至docker容器中/home/oslab 目录下，即通过访问docker中的/home/oslab目录我们可以访问到主机的/home目录
### 方便我们在主机使用git后使用docker访问与修改文件
```

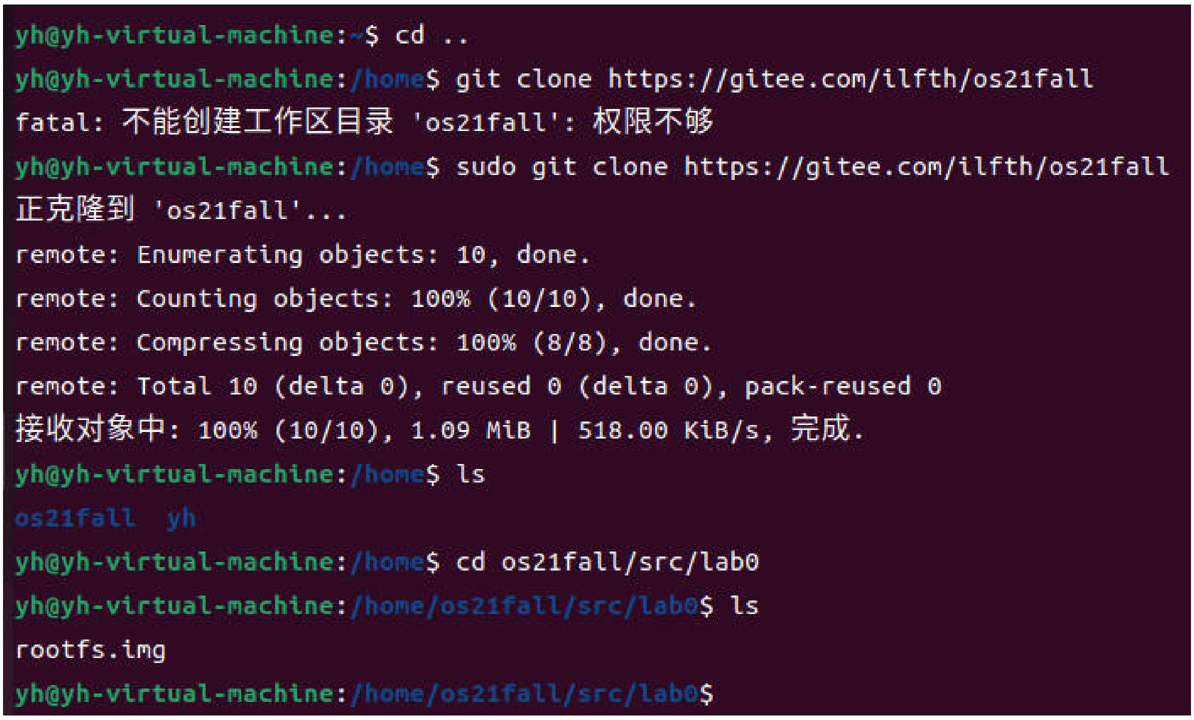


```
yh@yh-virtual-machine:/home/os21fall/src/lab0$ docker run --name oslab -it -v /home:/home/oslab oslab:2022 /bin/bash
root@e70ec89e3e16:/# exit
exit

yh@yh-virtual-machine:/home/os21fall/src/lab0$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
yh@yh-virtual-machine:/home/os21fall/src/lab0$ docker ps -a
CONTAINER ID   IMAGE         COMMAND       CREATED              STATUS                          PORTS     NAMES
e70ec89e3e16   oslab:2022    "/bin/bash"   About a minute ago   Exited (0) About a minute ago             oslab
224e98cd4770   hello-world   "/hello"      18 minutes ago       Exited (0) 18 minutes ago                 musing_mclaren
```

2. 获取 `Linux` 源码和已经编译好的文件系统

进入 `/home` 目录并克隆实验仓库，内含根文件系统的镜像。

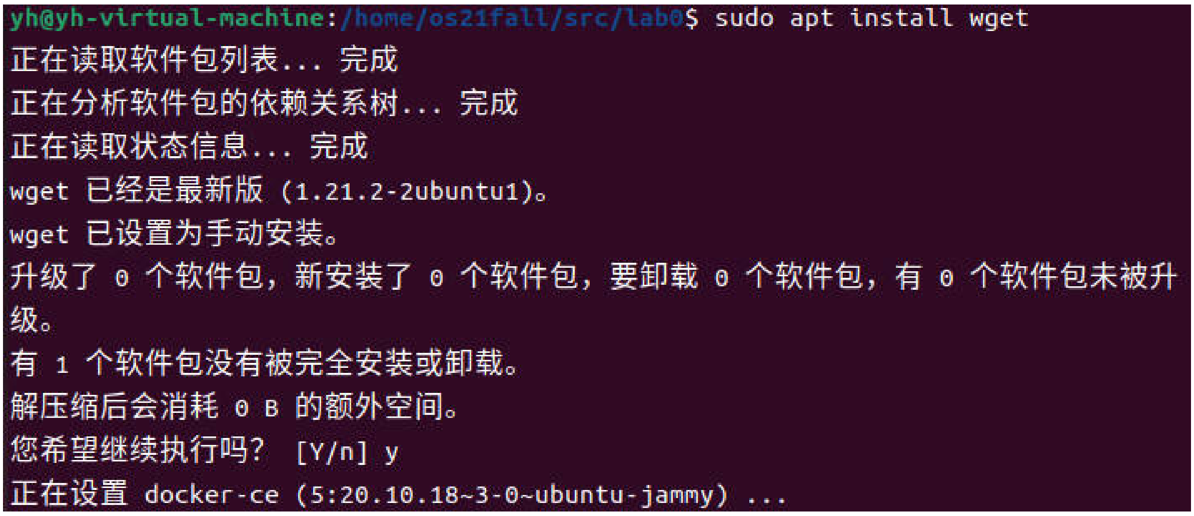
```
cd /home #进入/home目录
sudo git clone https://gitee.com/zjusec/os21fall
#/home目录下普通用户无权限建立新目录，需要超级用户权限

```



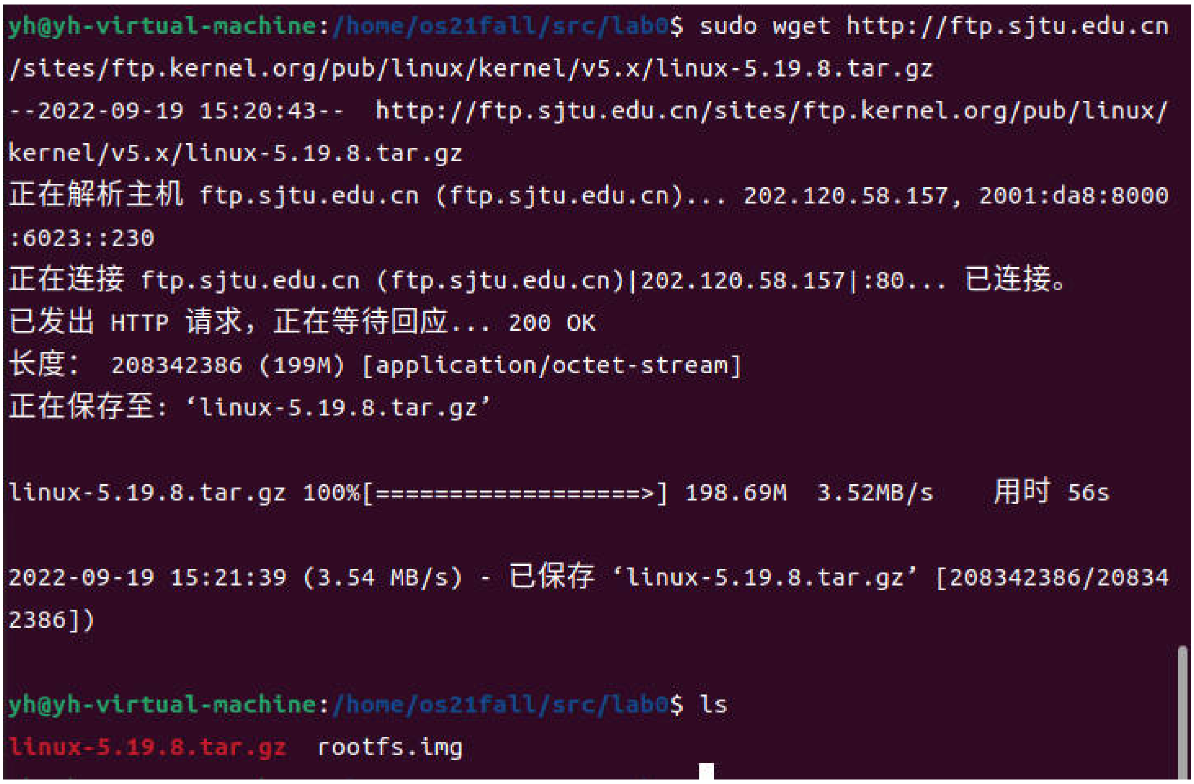
```
yh@yh-virtual-machine:~$ cd ..
yh@yh-virtual-machine:/home$ git clone https://gitee.com/ilfth/os21fall
fatal: 不能创建工作区目录 'os21fall': 权限不够
yh@yh-virtual-machine:/home$ sudo git clone https://gitee.com/ilfth/os21fall
正克隆到 'os21fall'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 10 (delta 0), reused 0 (delta 0), pack-reused 0
接收对象中: 100% (10/10), 1.09 MiB | 518.00 KiB/s, 完成.
yh@yh-virtual-machine:/home$ ls
os21fall  yh
yh@yh-virtual-machine:/home$ cd os21fall/src/lab0
yh@yh-virtual-machine:/home/os21fall/src/lab0$ ls
rootfs.img
yh@yh-virtual-machine:/home/os21fall/src/lab0$
```

在`/home/os21fall/src/lab0`目录下下载`linux`源码。

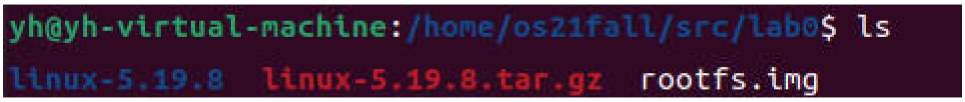
```
sudo apt install wget   ### 安装下载工具
sudo wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.19.8.tar.xz
### 国内备用下载地址
sudo wget http://ftp.sjtu.edu.cn/sites/ftp.kernel.org/pub/linux/kernel/v5.x/linux-5.19.8.tar.gz

tar -zxvf linux-5.19.8.tar.gz
### 下载的源文件为压缩包，需要使用命令解压缩
```



```
yh@yh-virtual-machine:/home/os21fall/src/lab0$ sudo apt install wget
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
wget 已经是最新版 (1.21.2-2ubuntu1)。
wget 已设置为手动安装。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 0 个软件包未被升级。
有 1 个软件包没有被完全安装或卸载。
解压缩后会消耗 0 B 的额外空间。
您希望继续执行吗？ [Y/n] y
正在设置 docker-ce (5:20.10.18~3-0~ubuntu-jammy) ...
```



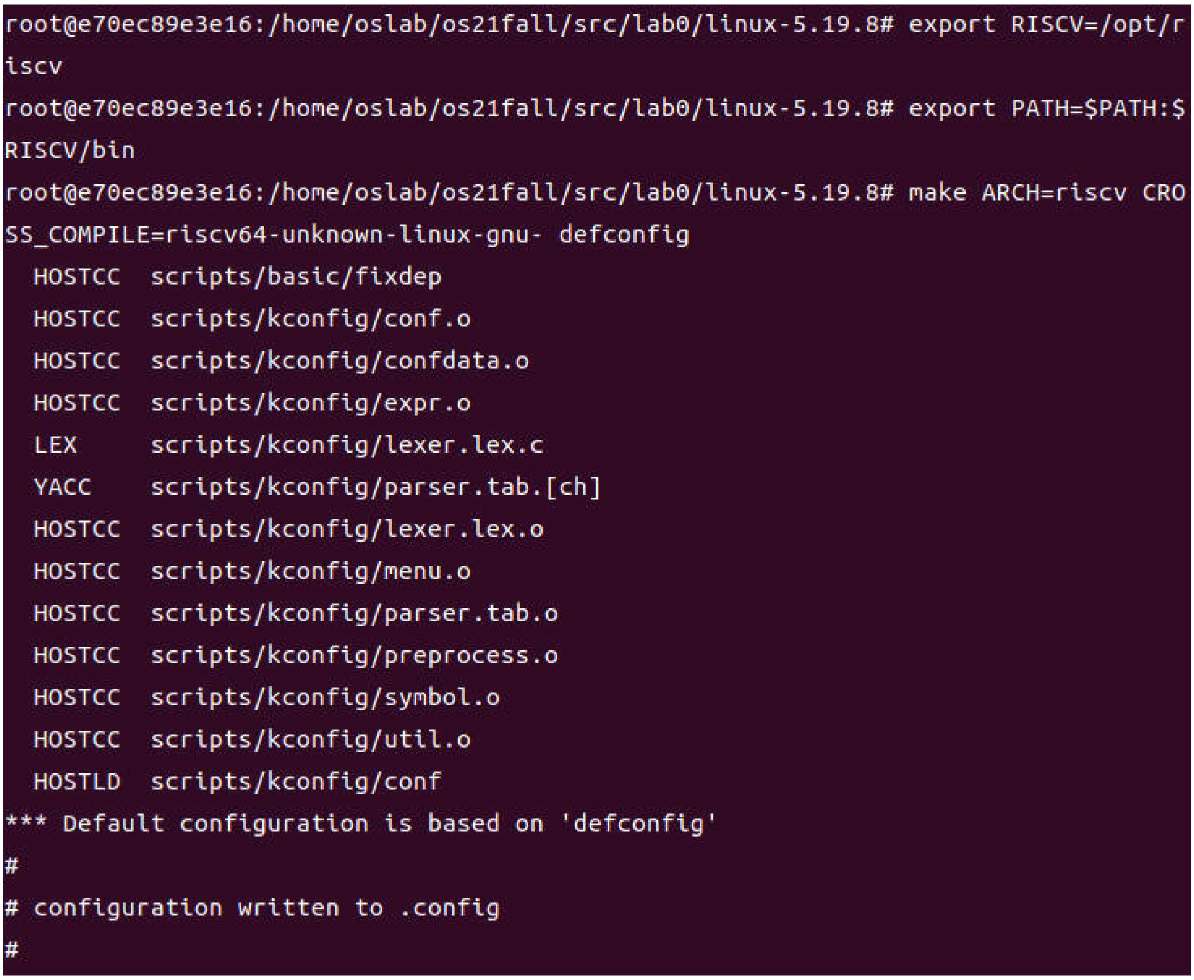
```
yh@yh-virtual-machine:/home/os21fall/src/lab0$ sudo wget http://ftp.sjtu.edu.cn
/sites/ftp.kernel.org/pub/linux/kernel/v5.x/linux-5.19.8.tar.gz
--2022-09-19 15:20:43--  http://ftp.sjtu.edu.cn/sites/ftp.kernel.org/pub/linux/
kernel/v5.x/linux-5.19.8.tar.gz
正在解析主机 ftp.sjtu.edu.cn (ftp.sjtu.edu.cn)... 202.120.58.157, 2001:da8:8000
:6023::230
正在连接 ftp.sjtu.edu.cn (ftp.sjtu.edu.cn)|202.120.58.157|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度： 208342386 (199M) [application/octet-stream]
正在保存至: ‘linux-5.19.8.tar.gz’

linux-5.19.8.tar.gz 100%[===================>] 198.69M  3.52MB/s    用时 56s

2022-09-19 15:21:39 (3.54 MB/s) - 已保存 ‘linux-5.19.8.tar.gz’ [208342386/20834
2386])

yh@yh-virtual-machine:/home/os21fall/src/lab0$ ls
linux-5.19.8.tar.gz  rootfs.img
```



```
yh@yh-virtual-machine:/home/os21fall/src/lab0$ ls
linux-5.19.8  linux-5.19.8.tar.gz  rootfs.img
```

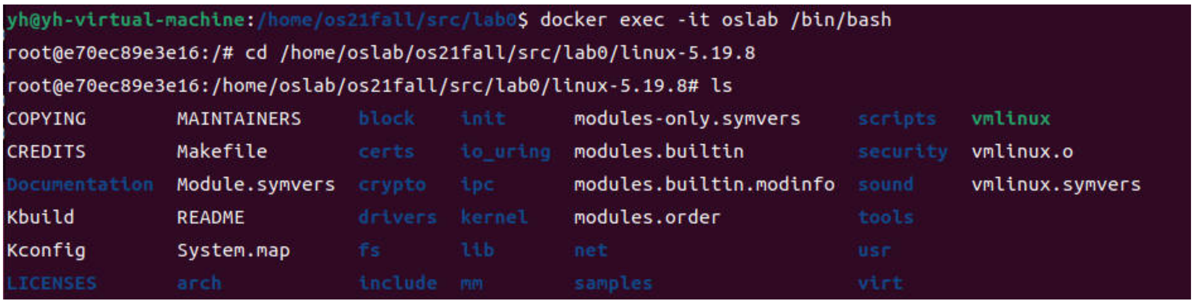
3. 编译`linux`内核

```
docker exec -it oslab /bin/bash
### 接下来的步骤在docker中执行，需要进入docker

export PATH=$PATH:/opt/riscv/bin
### 设置环境变量，使得后面的编译命令可以不必重复地址
make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu- defconfig
### 生成配置
make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu- -j$(nproc)
### 编译
```



```
root@e70ec89e3e16:/home/oslab/os21fall/src/lab0/linux-5.19.8# export RISCV=/opt/riscv
root@e70ec89e3e16:/home/oslab/os21fall/src/lab0/linux-5.19.8# export PATH=$PATH:$RISCV/bin
root@e70ec89e3e16:/home/oslab/os21fall/src/lab0/linux-5.19.8# make ARCH=riscv CROSS_COMPILE=riscv64-unknown-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
#
```

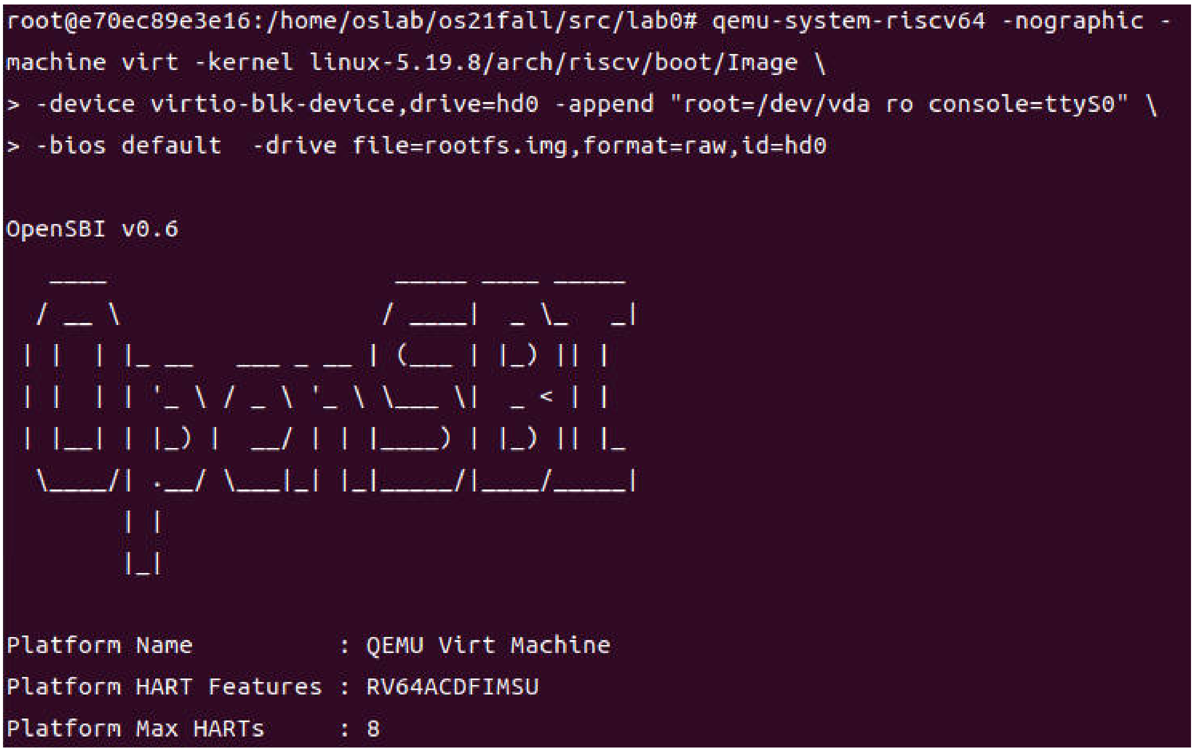
编译后的内核目录



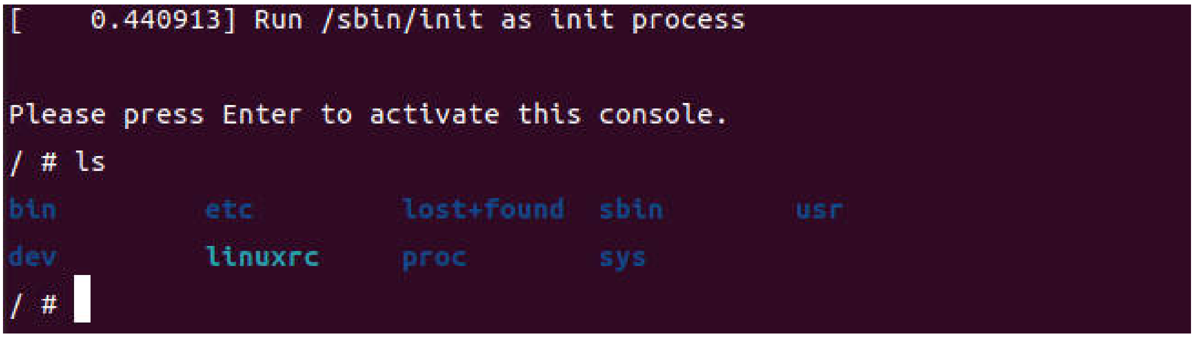
```
yh@yh-virtual-machine:/home/os21fall/src/lab0$ docker exec -it oslab /bin/bash
root@e70ec89e3e16:/# cd /home/oslab/os21fall/src/lab0/linux-5.19.8
root@e70ec89e3e16:/home/oslab/os21fall/src/lab0/linux-5.19.8# ls
COPYING        MAINTAINERS    block    init      modules-only.symvers     scripts   vmlinux
CREDITS        Makefile       certs    io_uring  modules.builtin          security  vmlinux.o
Documentation  Module.symvers crypto   ipc       modules.builtin.modinfo  sound     vmlinux.symvers
Kbuild         README         drivers  kernel    modules.order            tools
Kconfig        System.map     fs       lib       net                      usr
LICENSES       arch           include  mm        samples                  virt
```

4. 使用QEMU运行内核

```
cd  /home/os21fall/src/lab0/
### 内含linux源代码目录
qemu-system-riscv64 -nographic -machine virt -kernel linux-5.19.8/arch/riscv/boot/Image \
-device virtio-blk-device,drive=hd0 -append"root=/dev/vda ro console=ttyS0" \
-bios default -drivefile=rootfs.img,format=raw,id=hd0
```



```
root@e70ec89e3e16:/home/oslab/os21fall/src/lab0# qemu-system-riscv64 -nographic -
machine virt -kernel linux-5.19.8/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

OpenSBI v0.6
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name          : QEMU Virt Machine
Platform HART Features : RV64ACDFIMSU
Platform Max HARTs     : 8
```

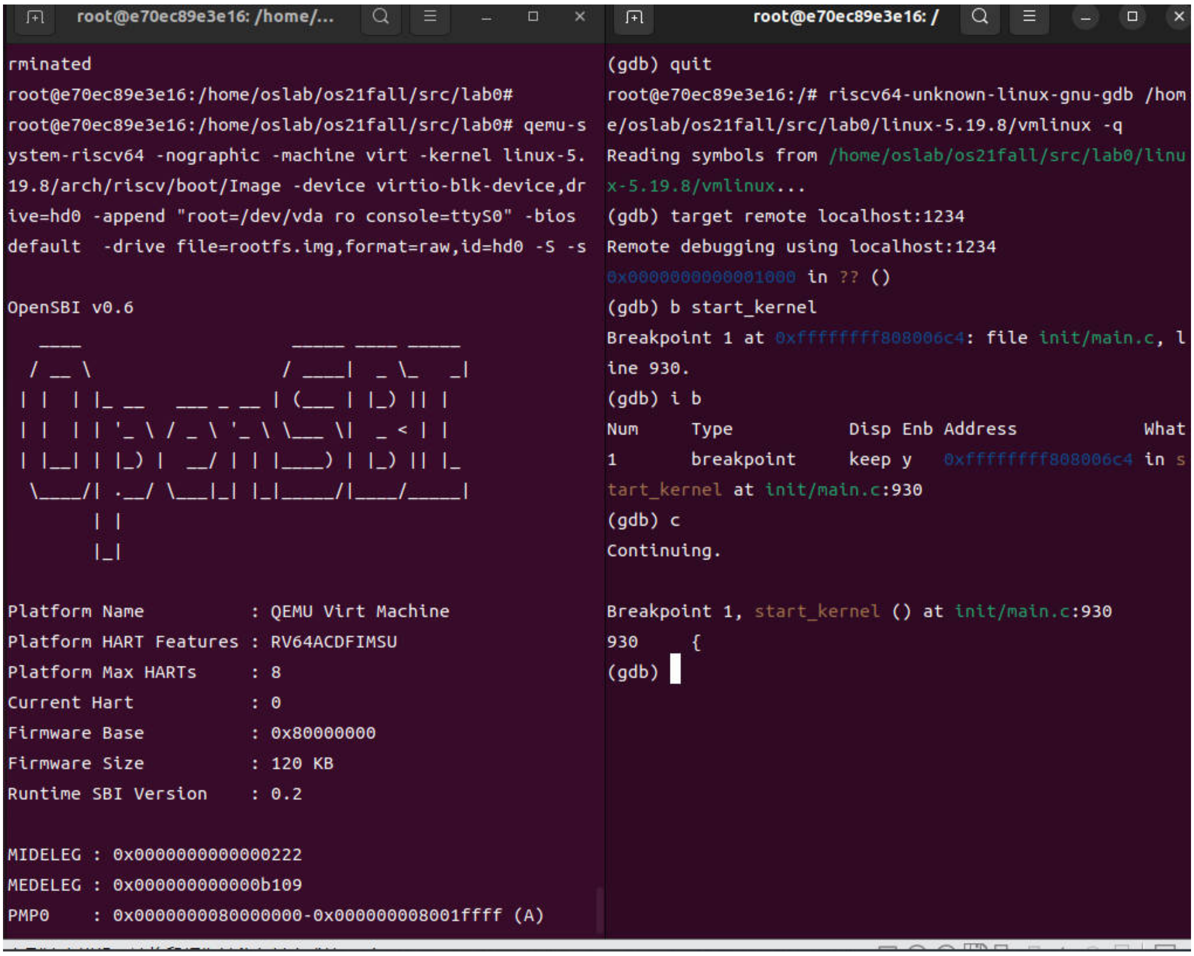


```
[    0.440913] Run /sbin/init as init process

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

5. 使用`gdb`调试内核

```
### 同时开启两个终端并同时运行docker，分别启动qemu与gdb
### Terminal 1
cd /home/os21fall/src/lab0
qemu-system-riscv64 -nographic -machine virt -kernel linux-5.19.8/arch/riscv/boot/Image \
-device virtio-blk-device,drive=hd0 -append"root=/dev/vda ro console=ttyS0" \
-bios default -drivefile=rootfs.img,format=raw,id=hd0 -S -s
### 最后添加的-S -s参数用于在启动后qemu不立即运行guest，而等待主机gdb发起连接，可以方便进行调试

### Terminal 2
export PATH=$PATH:/opt/riscv/bin
riscv64-unknown-linux-gnu-gdb /home/oslab/os21fall/src/lab0/linux-5.19.8/vmlinux

### 可能出现Reading symbols from vmlinux...(No debugging symbols found in vmlinux)信息，需要在内核Makefile的KBUILD_CFLAGS上添加-g选项，然后重新编译内核，继续运行上述命令行启动gdb开始调试。
### 该信息并不算警告或错误，可能会淹没在gdb的启动信息中，要仔细查看，否则无调试信息
```

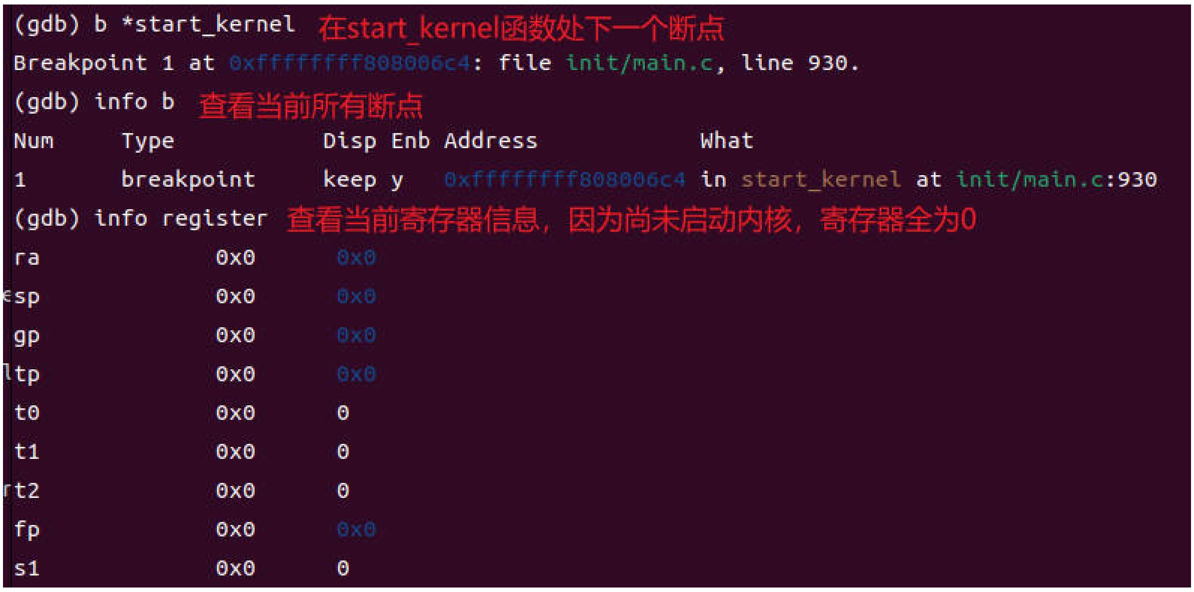
启动并连接的图示。



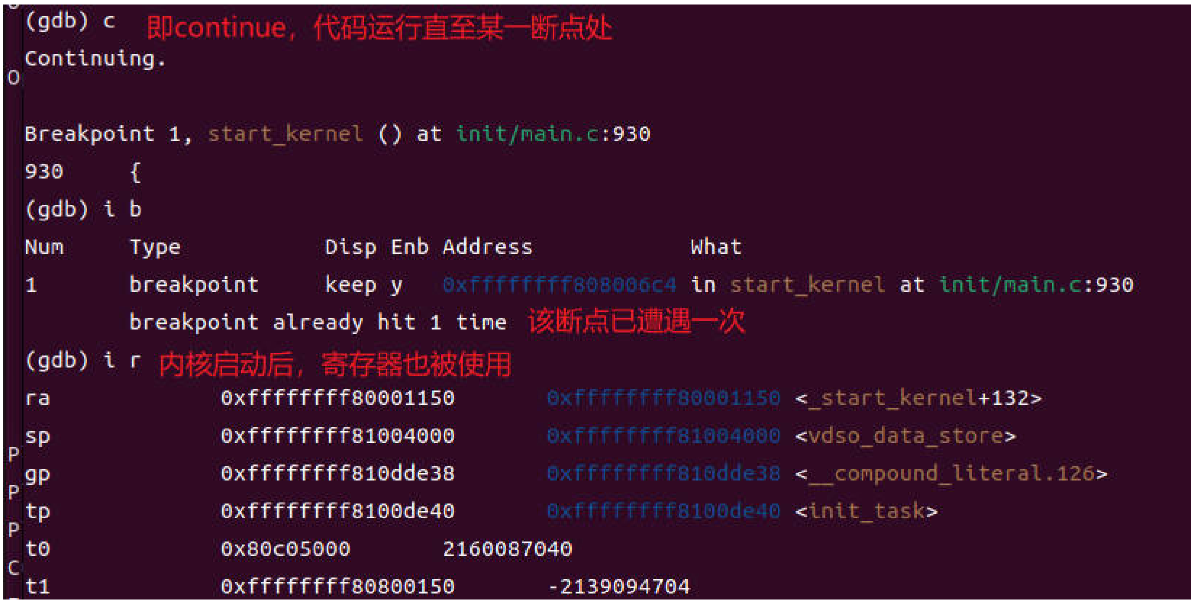
```
rminated
root@e70ec89e3e16:/home/oslab/os21fall/src/lab0#
root@e70ec89e3e16:/home/oslab/os21fall/src/lab0# qemu-system-riscv64 -nographic -machine virt -kernel linux-5.19.8/arch/riscv/boot/Image -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0" -bios default  -drive file=rootfs.img,format=raw,id=hd0 -S -s

OpenSBI v0.6
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name          : QEMU Virt Machine
Platform HART Features : RV64ACDFIMSU
Platform Max HARTs     : 8
Current Hart           : 0
Firmware Base          : 0x80000000
Firmware Size          : 120 KB
Runtime SBI Version    : 0.2

MIDELEG : 0x0000000000000222
MEDELEG : 0x000000000000b109
PMP0    : 0x0000000080000000-0x000000008001ffff (A)
```

```
(gdb) quit
root@e70ec89e3e16:/# riscv64-unknown-linux-gnu-gdb /home/oslab/os21fall/src/lab0/linux-5.19.8/vmlinux -q
Reading symbols from /home/oslab/os21fall/src/lab0/linux-5.19.8/vmlinux...
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
0x0000000000001000 in ?? ()
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff808006c4: file init/main.c, line 930.
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0xffffffff808006c4 in start_kernel at init/main.c:930
(gdb) c
Continuing.

Breakpoint 1, start_kernel () at init/main.c:930
930     {
(gdb)
```

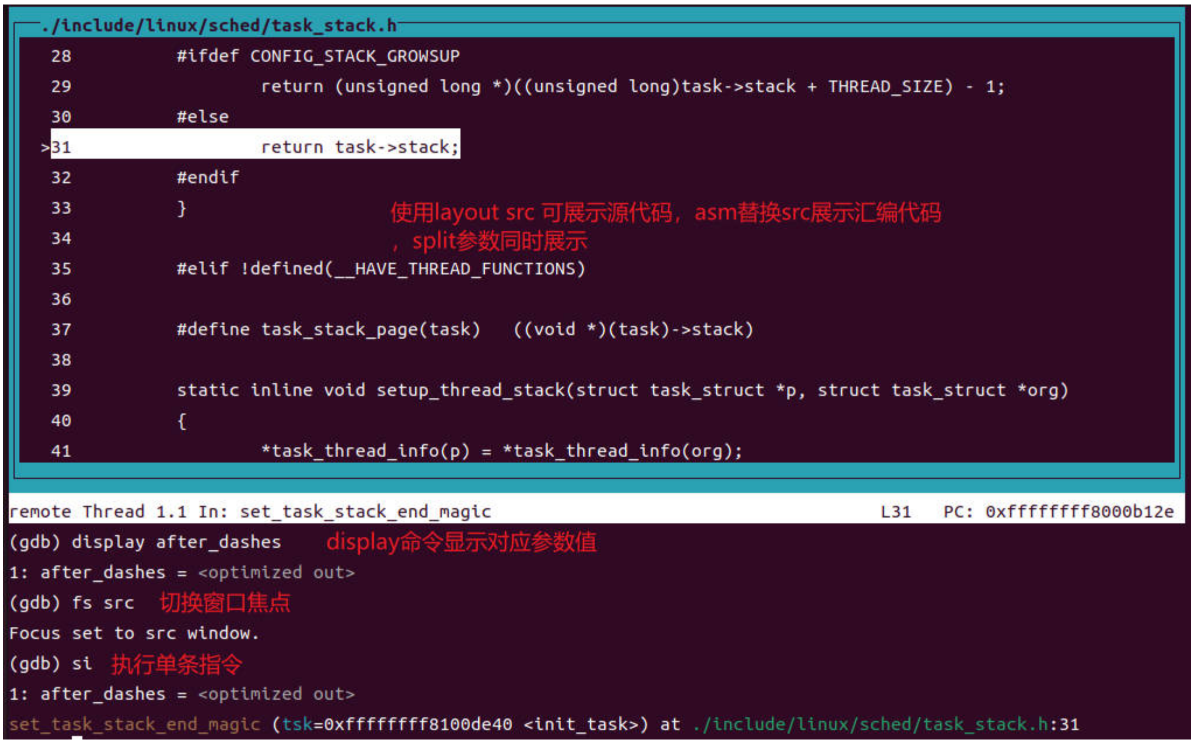
部分指令执行



```
(gdb) b *start_kernel  在start_kernel函数处下一个断点
Breakpoint 1 at 0xffffffff808006c4: file init/main.c, line 930.
(gdb) info b  查看当前所有断点
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0xffffffff808006c4 in start_kernel at init/main.c:930
(gdb) info register  查看当前寄存器信息，因为尚未启动内核，寄存器全为0
ra             0x0      0x0
sp             0x0      0x0
gp             0x0      0x0
tp             0x0      0x0
t0             0x0      0
t1             0x0      0
t2             0x0      0
fp             0x0      0x0
s1             0x0      0
```

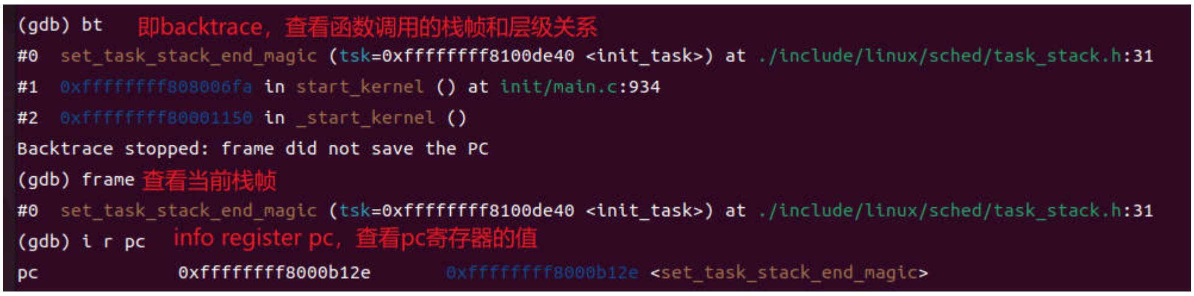


```
(gdb) c  即continue，代码运行直至某一断点处
Continuing.

Breakpoint 1, start_kernel () at init/main.c:930
930     {
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0xffffffff808006c4 in start_kernel at init/main.c:930
        breakpoint already hit 1 time  该断点已遭遇一次
(gdb) i r  内核启动后，寄存器也被使用
ra             0xffffffff80001150       0xffffffff80001150 <_start_kernel+132>
sp             0xffffffff81004000       0xffffffff81004000 <vdso_data_store>
gp             0xffffffff810dde38       0xffffffff810dde38 <__compound_literal.126>
tp             0xffffffff8100de40       0xffffffff8100de40 <init_task>
t0             0x80c05000       2160087040
t1             0xffffffff80800150       -2139094704
```



```
./include/linux/sched/task_stack.h
        #ifdef CONFIG_STACK_GROWSUP
                return (unsigned long *)((unsigned long)task->stack + THREAD_SIZE) - 1;
        #else
                return task->stack;
        #endif
        }
        
        #elif !defined(__HAVE_THREAD_FUNCTIONS)

        #define task_stack_page(task)   ((void *)(task)->stack)

        static inline void setup_thread_stack(struct task_struct *p, struct task_struct *org)
        {
                *task_thread_info(p) = *task_thread_info(org);
remote Thread 1.1 In: set_task_stack_end_magic                          L31   PC: 0xffffffff8000b12e
(gdb) display after_dashes   display命令显示对应参数值
1: after_dashes = <optimized out>
(gdb) fs src   切换窗口焦点
Focus set to src window.
(gdb) si  执行单条指令
1: after_dashes = <optimized out>
set_task_stack_end_magic (tsk=0xffffffff8100de40 <init_task>) at ./include/linux/sched/task_stack.h:31
```

使用layout src 可展示源代码，asm替换src展示汇编代码，split参数同时展示

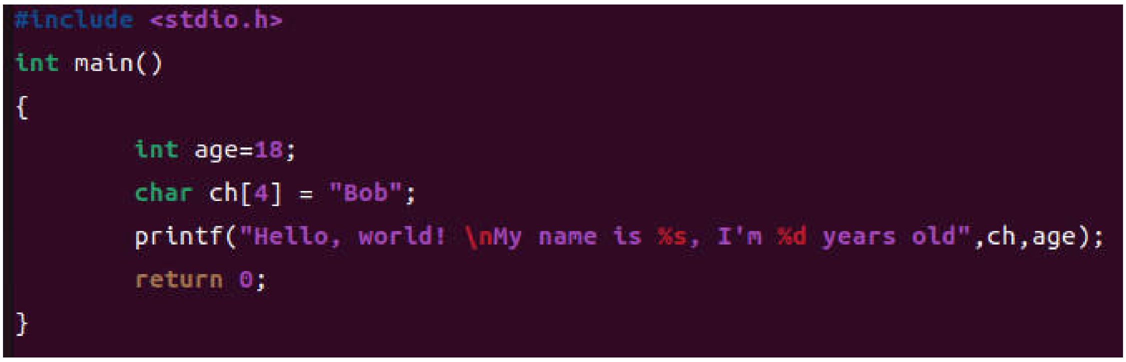


```
(gdb) bt   即backtrace，查看函数调用的栈帧和层级关系
#0  set_task_stack_end_magic (tsk=0xffffffff8100de40 <init_task>) at ./include/linux/sched/task_stack.h:31
#1  0xffffffff808006fa in start_kernel () at init/main.c:934
#2  0xffffffff80001150 in _start_kernel ()
Backtrace stopped: frame did not save the PC
(gdb) frame 查看当前栈帧
#0  set_task_stack_end_magic (tsk=0xffffffff8100de40 <init_task>) at ./include/linux/sched/task_stack.h:31
(gdb) i r pc   info register pc，查看pc寄存器的值
pc             0xffffffff8000b12e       0xffffffff8000b12e <set_task_stack_end_magic>
```

二、思考题

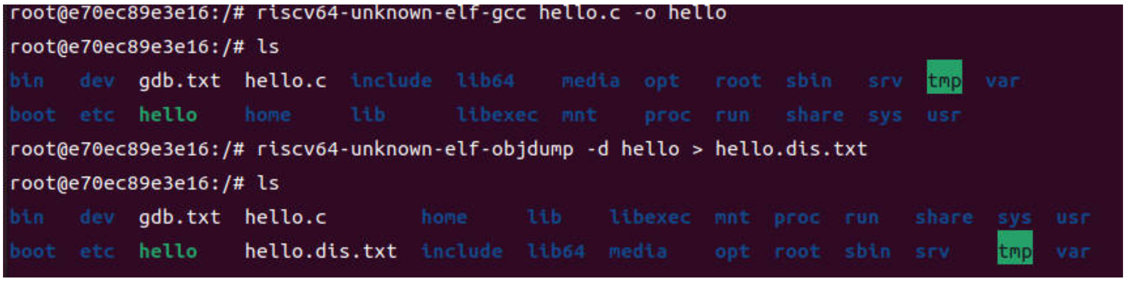
1. 使用`riscv64-unknown-elf-gcc`编译单个`.c`文件
2. 使用`riscv64-unknown-elf-objdump`反汇编`1`中得到的编译产物。

> - 简单编写一个c文件

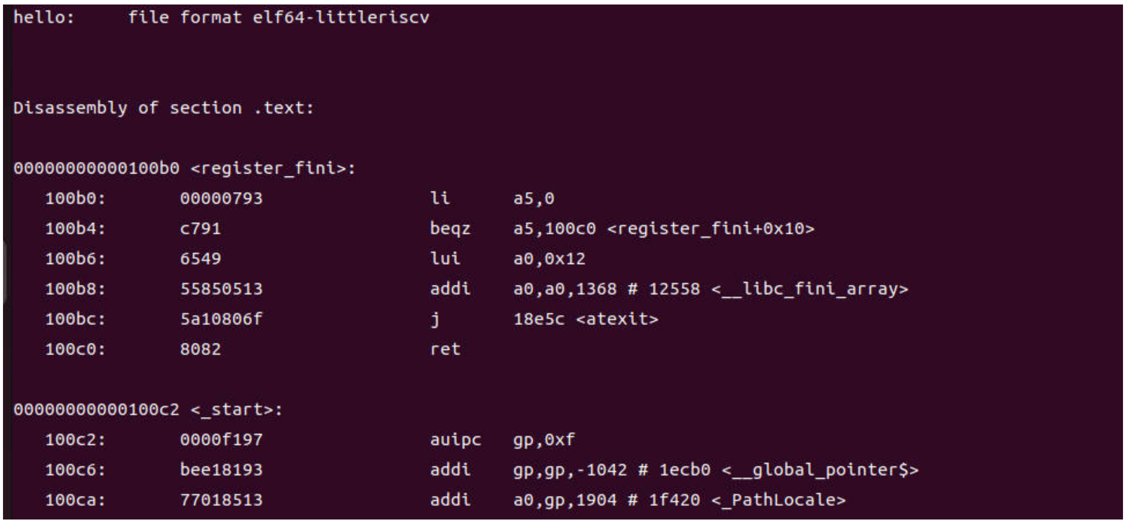


```c
#include <stdio.h>
int main()
{
        int age=18;
        char ch[4] = "Bob";
        printf("Hello, world! \nMy name is %s, I'm %d years old",ch,age);
        return 0;
}
```

- 编译与反汇编



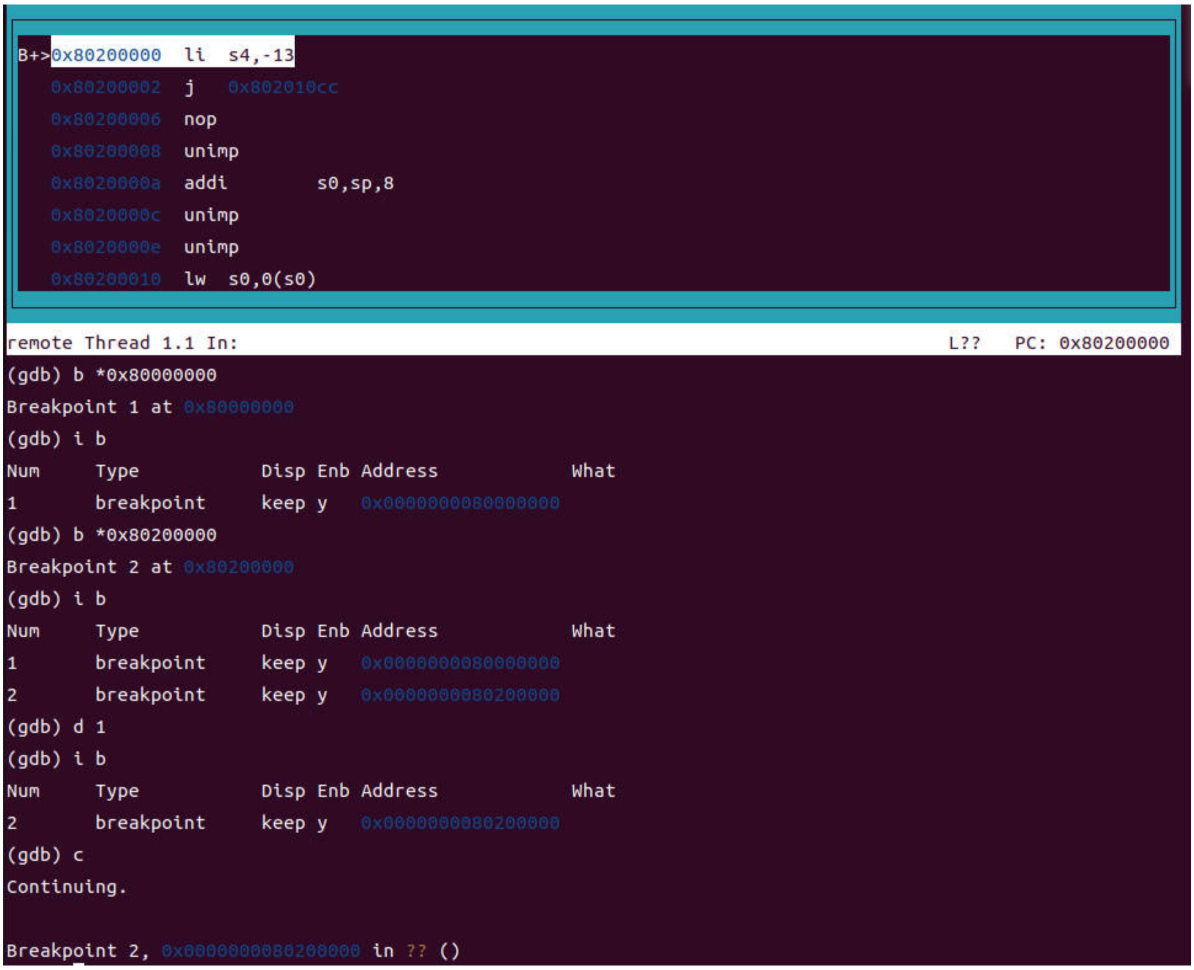
```
root@e70ec89e3e16:/# riscv64-unknown-elf-gcc hello.c -o hello
root@e70ec89e3e16:/# ls
bin   dev  gdb.txt  hello.c  include  lib64    media  opt   root  sbin   srv  tmp  var
boot  etc  hello    home     lib      libexec  mnt    proc  run   share  sys  usr
root@e70ec89e3e16:/# riscv64-unknown-elf-objdump -d hello > hello.dis.txt
root@e70ec89e3e16:/# ls
bin   dev  gdb.txt  hello.c        home     lib    libexec  mnt  proc  run   share  sys  usr
boot  etc  hello    hello.dis.txt  include  lib64  media    opt  root  sbin  srv    tmp  var
```



```
hello:     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:       6549                    lui     a0,0x12
   100b8:       55850513                addi    a0,a0,1368 # 12558 <__libc_fini_array>
   100bc:       5a10806f                j       18e5c <atexit>
   100c0:       8082                    ret

00000000000100c2 <_start>:
   100c2:       0000f197                auipc   gp,0xf
   100c6:       bee18193                addi    gp,gp,-1042 # 1ecb0 <__global_pointer$>
   100ca:       77018513                addi    a0,gp,1904 # 1f420 <_PathLocale>
```

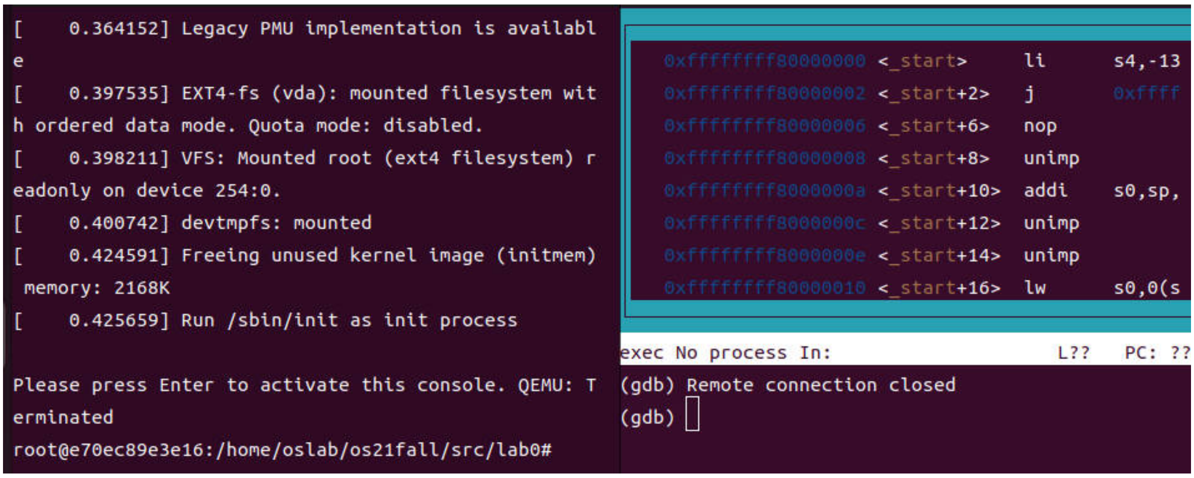
3.

```
layout asm # 查看汇编代码
b *0x80000000 #在0x80000000处下断点
i b #查看所有已下断点
b *0x80200000 #在0x80200000处下断点
d 1(需删断点对应的编号) # 删除编号为1的断点，此处为0x80000000
c # 继续运行至触发0x80200000
n 1 # 单步调试一次
```



```
B+>0x80200000  li   s4,-13
   0x80200002  j    0x802010cc
   0x80200006  nop
   0x80200008  unimp
   0x8020000a  addi        s0,sp,8
   0x8020000c  unimp
   0x8020000e  unimp
   0x80200010  lw   s0,0(s0)

remote Thread 1.1 In:                                                      L??   PC: 0x80200000
(gdb) b *0x80000000
Breakpoint 1 at 0x80000000
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000080000000
(gdb) b *0x80200000
Breakpoint 2 at 0x80200000
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000080000000
2       breakpoint     keep y   0x0000000080200000
(gdb) d 1
(gdb) i b
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x0000000080200000
(gdb) c
Continuing.

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

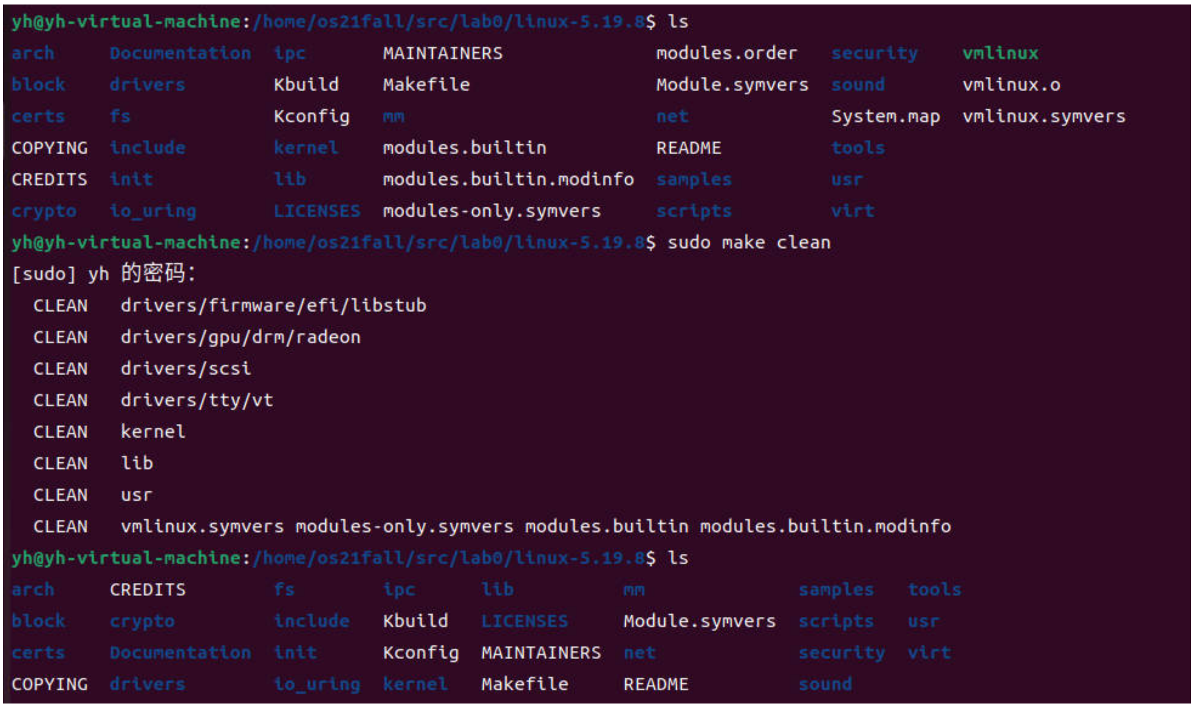
退出QEMU



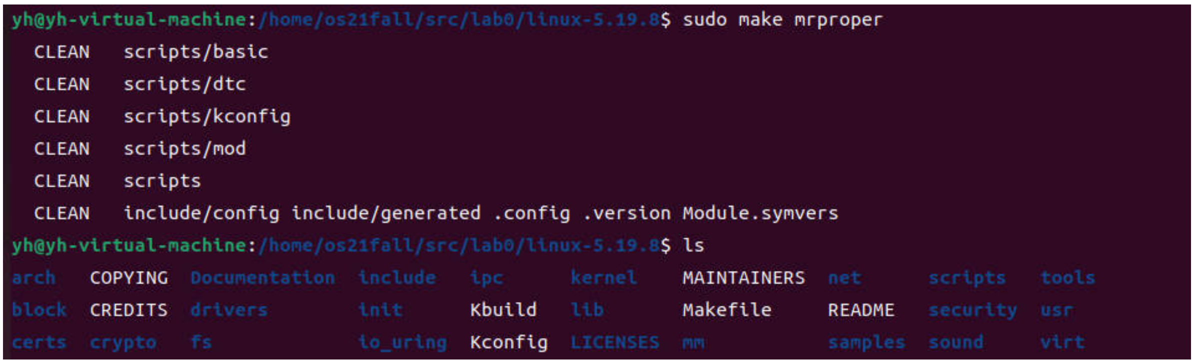
```
[    0.364152] Legacy PMU implementation is availabl
e
[    0.397535] EXT4-fs (vda): mounted filesystem wit
h ordered data mode. Quota mode: disabled.
[    0.398211] VFS: Mounted root (ext4 filesystem) r
eadonly on device 254:0.
[    0.400742] devtmpfs: mounted
[    0.424591] Freeing unused kernel image (initmem)
 memory: 2168K
[    0.425659] Run /sbin/init as init process

Please press Enter to activate this console. QEMU: T
erminated
root@e70ec89e3e16:/home/oslab/os21fall/src/lab0#
```

```
   0xffffffff80000000 <_start>      li     s4,-13
   0xffffffff80000002 <_start+2>    j      0xffff
   0xffffffff80000006 <_start+6>    nop
   0xffffffff80000008 <_start+8>    unimp
   0xffffffff8000000a <_start+10>   addi   s0,sp,
   0xffffffff8000000c <_start+12>   unimp
   0xffffffff8000000e <_start+14>   unimp
   0xffffffff80000010 <_start+16>   lw     s0,0(s

exec No process In:                           L??   PC: ??
(gdb) Remote connection closed
(gdb) 
```

4. 使用`make`工具清除`linux`的构建产物



```
yh@yh-virtual-machine:/home/os21fall/src/lab0/linux-5.19.8$ ls
arch     Documentation  ipc       MAINTAINERS              modules.order   security    vmlinux
block    drivers        Kbuild    Makefile                 Module.symvers  sound       vmlinux.o
certs    fs             Kconfig   mm                       net             System.map  vmlinux.symvers
COPYING  include        kernel    modules.builtin          README          tools
CREDITS  init           lib       modules.builtin.modinfo  samples         usr
crypto   io_uring       LICENSES  modules-only.symvers     scripts         virt
yh@yh-virtual-machine:/home/os21fall/src/lab0/linux-5.19.8$ sudo make clean
[sudo] yh 的密码：
  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.modinfo
yh@yh-virtual-machine:/home/os21fall/src/lab0/linux-5.19.8$ ls
arch     CREDITS        fs        ipc      lib          mm              samples   tools
block    crypto         include   Kbuild   LICENSES     Module.symvers  scripts   usr
certs    Documentation  init      Kconfig  MAINTAINERS  net             security  virt
COPYING  drivers        io_uring  kernel   Makefile     README          sound
```



```
yh@yh-virtual-machine:/home/os21fall/src/lab0/linux-5.19.8$ sudo make mrproper
  CLEAN   scripts/basic
  CLEAN   scripts/dtc
  CLEAN   scripts/kconfig
  CLEAN   scripts/mod
  CLEAN   scripts
  CLEAN   include/config include/generated .config .version Module.symvers
yh@yh-virtual-machine:/home/os21fall/src/lab0/linux-5.19.8$ ls
arch   COPYING  Documentation  include   ipc      kernel    MAINTAINERS  net     scripts   tools
block  CREDITS  drivers        init      Kbuild   lib       Makefile     README  security  usr
certs  crypto   fs             io_uring  Kconfig  LICENSES  mm           samples sound     virt
```

5. `vmlinux` 和 `Image` 的关系和区别

> 两者都是 `linux` 内核映像，
>
> `vmlinux` 是编译出来的最原始的未压缩的文件，为 `ELF` 格式文件，该映像可用于定位内核问题，但不能直接引导Linux系统启动，可用于 `debug`。
>
> `Image` 是使用 `objcopy` 处理 `vmlinux` 丢弃多余信息后的完全的二进制文件，未经过压缩，可以直接引导 `Linux` 内核启动。
>
> 两者的关系在于同为 `Linux` 内核映像文件，`Image` 为 `vmlinux` 处理后的产物；
>
> 区别为两者文件格式不同，`Image` 可用于 `Linux` 系统启动，`vmlinux` 不能。

三、讨论、心得

本次实验的指导非常详细，基本按照命令一步步执行下去即可，`gdb` 调试部分看给出的参考书也可有一个基本的了解。也增强了安装和使用环境的能力。但是作为导引除了调试技术实在看不出来和后面的实验的关联部分，给出的实验资料也比较杂，让人依旧摸不着头脑。