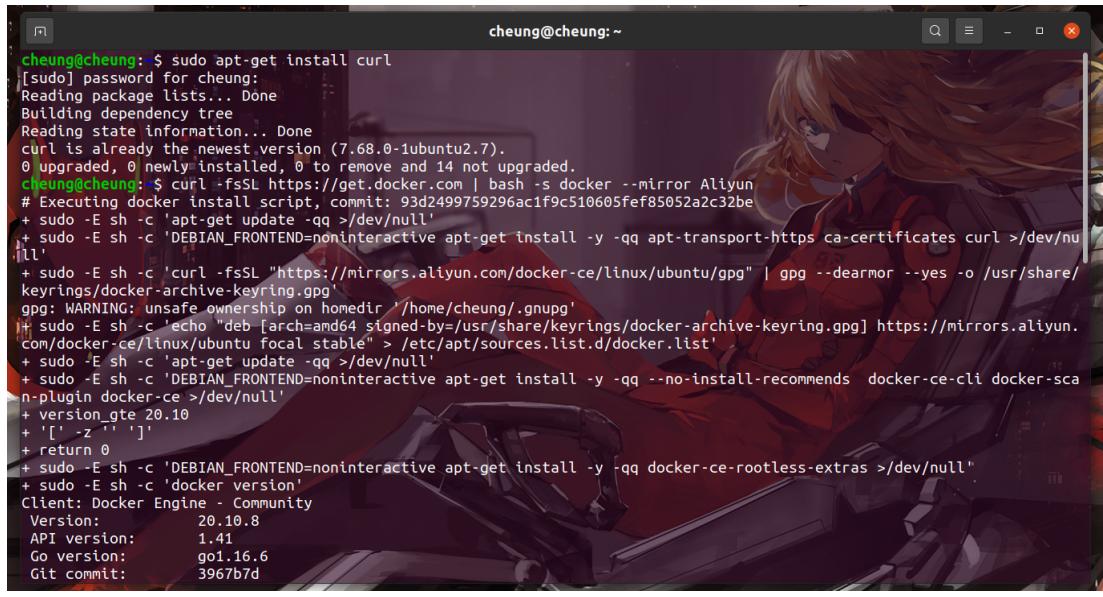


浙江大学实验报告

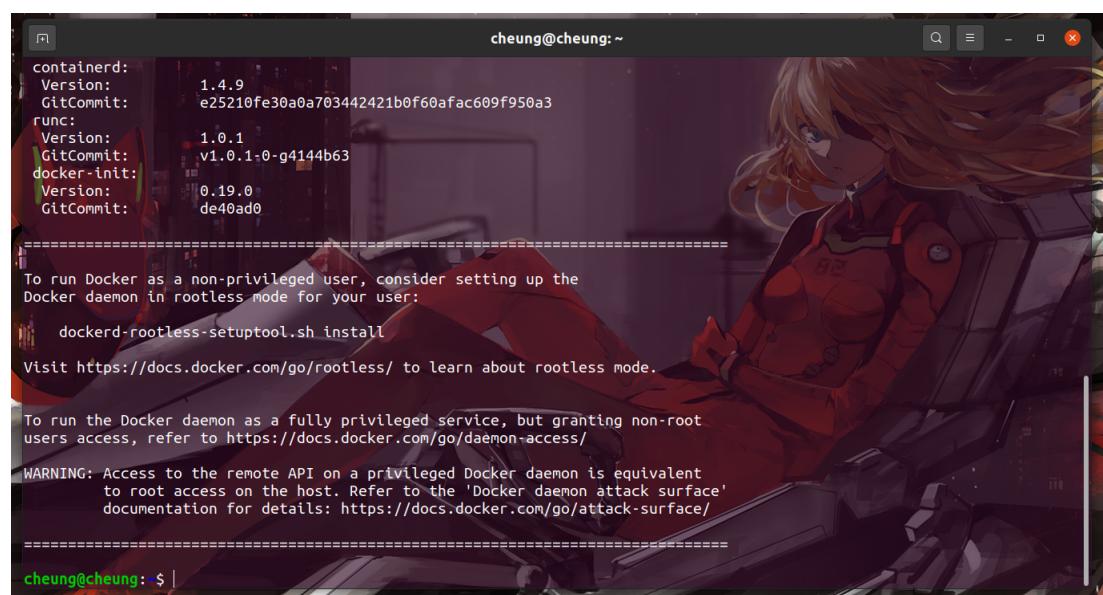
一、实验内容

搭建docker环境

首先，根据实验指导，安装curl和docker。如下图，系统提示我已经安装过了curl，并且在安装docker时输出了安装所使用的命令、docker的版本信息等信息。最后，还输出了docker的使用注意事项。



```
cheung@cheung:~$ sudo apt-get install curl
[sudo] password for cheung:
Reading package lists... Done
Building dependency tree...
Reading state information... Done
curl is already the newest version (7.68.0-1ubuntu2.7).
0 upgraded, 0 newly installed, 0 to remove and 14 not upgraded.
cheung@cheung:~$ curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun
# Executing docker install script, commit: 93d2499759296ac1f9c510605fef85052a2c32be
+ sudo -E sh -c 'apt-get update <qq >/dev/null'
+ sudo -E sh -c 'DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transport-https ca-certificates curl >/dev/null'
+ sudo -E sh -c 'curl -fsSL "https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg" | gpg --dearmor --yes -o /usr/share/keyrings/docker-archive-keyring.gpg'
gpg: WARNING: unsafe ownership of homedir '/home/cheung/.gnupg'
+ sudo -E sh -c 'echo "[arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://mirrors.aliyun.com/docker-ce/linux/ubuntu focal stable" > /etc/apt/sources.list.d/docker.list'
+ sudo -E sh -c 'apt-get update <qq >/dev/null'
+ sudo -E sh -c 'DEBIAN_FRONTEND=noninteractive apt-get install -y -qq --no-install-recommends docker-ce-cli docker-scenario-plugin docker-ce >/dev/null'
+ version_gte 20.10
+ '[' -z '' ']'
+ return 0
+ sudo -E sh -c 'DEBIAN_FRONTEND=noninteractive apt-get install -y -qq docker-ce-rootless-extras >/dev/null'
+ sudo -E sh -c 'docker version'
Client: Docker Engine - Community
Version:           20.10.8
API version:      1.41
Go version:       go1.16.6
Git commit:        3967b7d
```

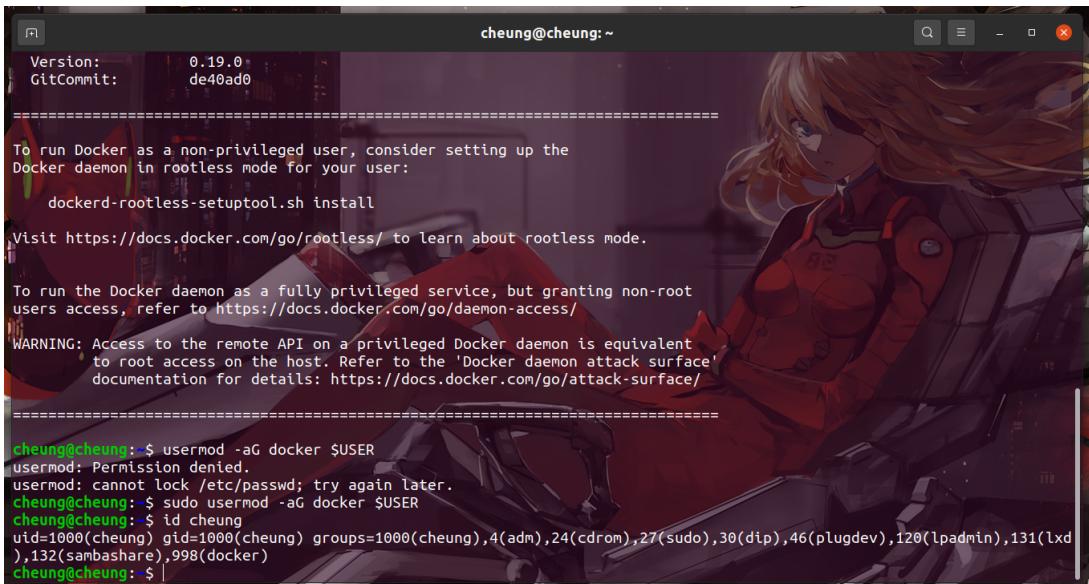
```
cheung@cheung:~$ docker daemon
containerd:
  Version:          1.4.9
  GitCommit:        e25210fe30a0a703442421b0f60afac609f950a3
runc:
  Version:          1.0.1
  GitCommit:        v1.0.1-0-g4144b63
docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

=====
To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:
  dockerd-rootless-setup.sh install
Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.

To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

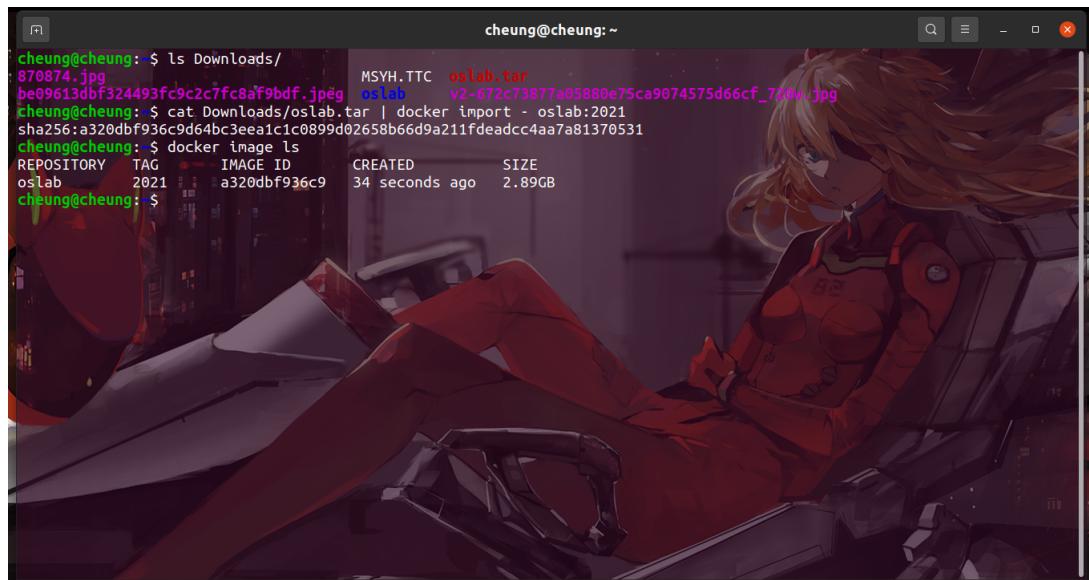
WARNING: Access to the remote API on a privileged Docker daemon is equivalent
to root access on the host. Refer to the 'Docker daemon attack surface'
documentation for details: https://docs.docker.com/go/attack-surface/
=====
```

将当前用户 cheung 加入docker的用户组，这里在实践中发现需要给予管理者权限。加入用户组后，利用 id 命令查看当前用户的信息，可以看到当前用户所属的用户组中已经包括了 docker。



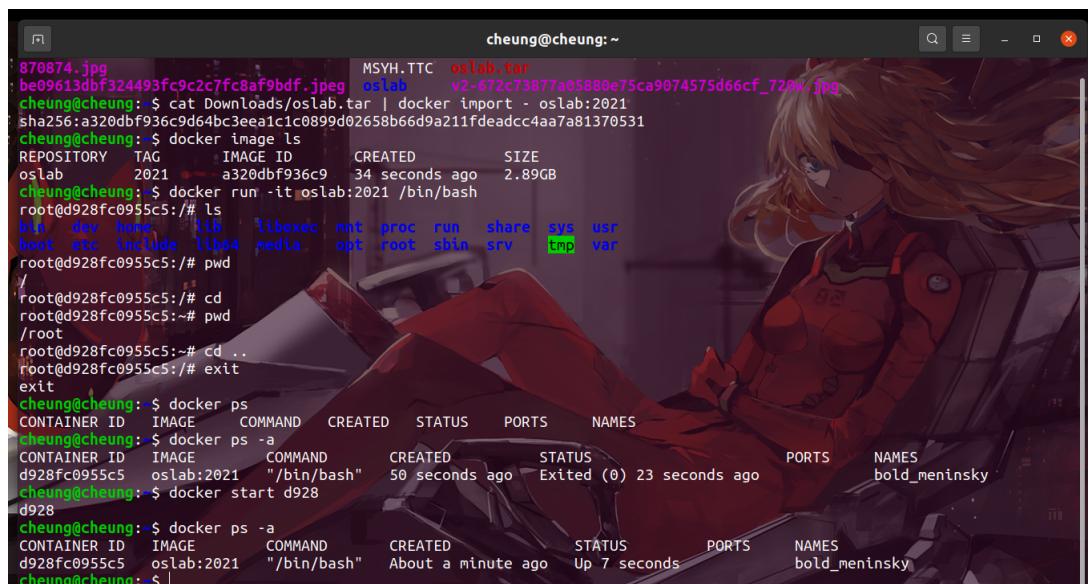
```
cheung@cheung:~  
Version:          0.19.0  
GitCommit:        de40ad0  
=====  
To run Docker as a non-privileged user, consider setting up the  
Docker daemon in rootless mode for your user:  
dockerd-rootless-setuptool.sh install  
Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.  
=====  
To run the Docker daemon as a fully privileged service, but granting non-root  
users access, refer to https://docs.docker.com/go/daemon-access/  
WARNING: Access to the remote API on a privileged Docker daemon is equivalent  
to root access on the host. Refer to the 'Docker daemon attack surface'  
documentation for details: https://docs.docker.com/go/attack-surface/  
=====  
cheung@cheung: $ usermod -aG docker $USER  
usermod: Permission denied.  
usermod: cannot lock /etc/passwd; try again later.  
cheung@cheung: $ sudo usermod -aG docker $USER  
cheung@cheung: $ id cheung  
uid=1000(cheung) gid=1000(cheung) groups=1000(cheung),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),998(docker)  
cheung@cheung: $ |
```

如下图，重启电脑后将老师下发的docker镜像导入。然后通过命令查看镜像，可以看到镜像已经被成功导入。



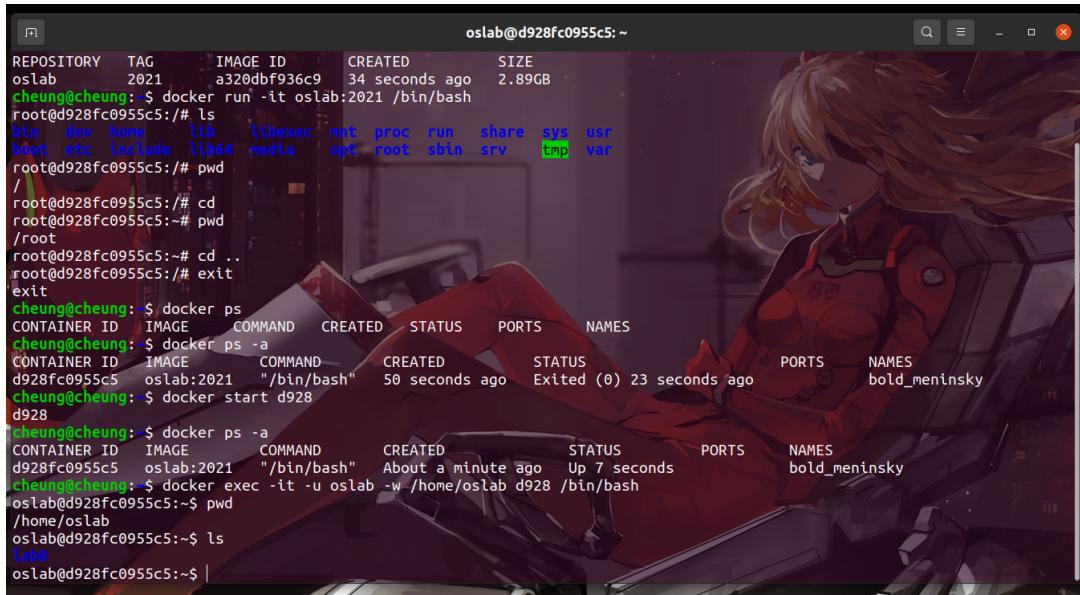
```
cheung@cheung:~  
cheung@cheung: $ ls Downloads/  
870874.jpg          MSYH.TTC oslab.tar  
be09613dbf324493fc9c2c7fc8af9bdf.jpeg oslab v2-672c73877a05880e75ca9074575d66cf_720w.jpg  
cheung@cheung: $ cat Downloads/oslab.tar | docker import - oslab:2021  
sha256:a320dbf936c9d64bc3eeac1c0899d02658b66d9a211fdeadcc4aa7a81370531  
cheung@cheung: $ docker image ls  
REPOSITORY TAG IMAGE ID CREATED SIZE  
oslab 2021 a320dbf936c9 34 seconds ago 2.89GB  
cheung@cheung: $ |
```

如下图，创建一个容器，在容器中简单执行若干条命令后退出，再查看所有容器的状态，启动容器后再次查看所有容器的状态。可以看到创建的容器的id为d928开头的字符串。在容器再次启动前，容器的状态为 `Exited(0)`，启动后状态为 `Up`。



```
cheung@cheung:~  
cheung@cheung: $ ls Downloads/  
870874.jpg          MSYH.TTC oslab.tar  
be09613dbf324493fc9c2c7fc8af9bdf.jpeg oslab v2-672c73877a05880e75ca9074575d66cf_720w.jpg  
cheung@cheung: $ cat Downloads/oslab.tar | docker import - oslab:2021  
sha256:a320dbf936c9d64bc3eeac1c0899d02658b66d9a211fdeadcc4aa7a81370531  
cheung@cheung: $ docker image ls  
REPOSITORY TAG IMAGE ID CREATED SIZE  
oslab 2021 a320dbf936c9 34 seconds ago 2.89GB  
cheung@cheung: $ docker run -it oslab:2021 /bin/bash  
root@d928fc0955c5:/# ls  
bin dev home lib libexec mnt proc run share sys usr  
boot etc include lib64 media opt root sbin srv tmp var  
root@d928fc0955c5:/# pwd  
/root  
root@d928fc0955c5:/# cd ..  
root@d928fc0955c5:/# exit  
exit  
cheung@cheung: $ docker ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
cheung@cheung: $ docker ps -a  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
d928fc0955c5 oslab:2021 "/bin/bash" 50 seconds ago Exited (0) 23 seconds ago bold_meninsky  
cheung@cheung: $ docker start d928  
d928  
cheung@cheung: $ docker ps -a  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
d928fc0955c5 oslab:2021 "/bin/bash" About a minute ago Up 7 seconds bold_meninsky  
cheung@cheung: $ |
```

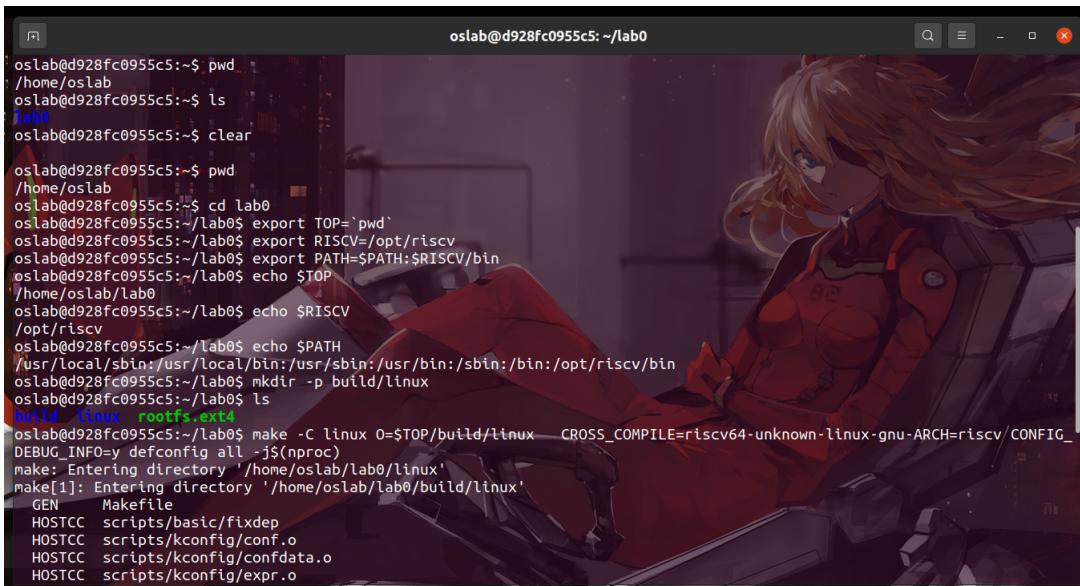
如下图，进入正在运行的容器。登入时，我们以 oslab 用户的身份登入，可以看到登入后显示的用户名为 oslab。简单运行两条命令，容器运行正常。



```
oslab@d928fc0955c5: ~
REPOSITORY TAG IMAGE ID CREATED SIZE
oslab 2021 a320dbf936c9 34 seconds ago 2.89GB
cheung@cheung: $ docker run -it oslab:2021 /bin/bash
root@d928fc0955c5:~# ls
bin dev home lib libexec mnt proc run share sys usr
boot etc include lib64 media opt root sbin srv tmp var
root@d928fc0955c5:~# pwd
/
root@d928fc0955c5:~# cd
root@d928fc0955c5:~# pwd
/root
root@d928fc0955c5:~# cd ..
root@d928fc0955c5:~# exit
exit
cheung@cheung: $ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
cheung@cheung: $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
d928fc0955c5 oslab:2021 "/bin/bash" 50 seconds ago Exited (0) 23 seconds ago bold_meninsky
cheung@cheung: $ docker start d928
d928
cheung@cheung: $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
d928fc0955c5 oslab:2021 "/bin/bash" About a minute ago Up 7 seconds bold_meninsky
cheung@cheung: $ docker exec -it -u oslab -w /home/oslab d928 /bin/bash
oslab@d928fc0955c5:~$ pwd
/home/oslab
oslab@d928fc0955c5:~$ ls
lab0
oslab@d928fc0955c5:~$ |
```

编译 linux 内核

如下图，根据实验指导执行一系列命令。在环境变量创建后通过 echo 命令检验其创建成功。在编译时，系统会输出许多信息。



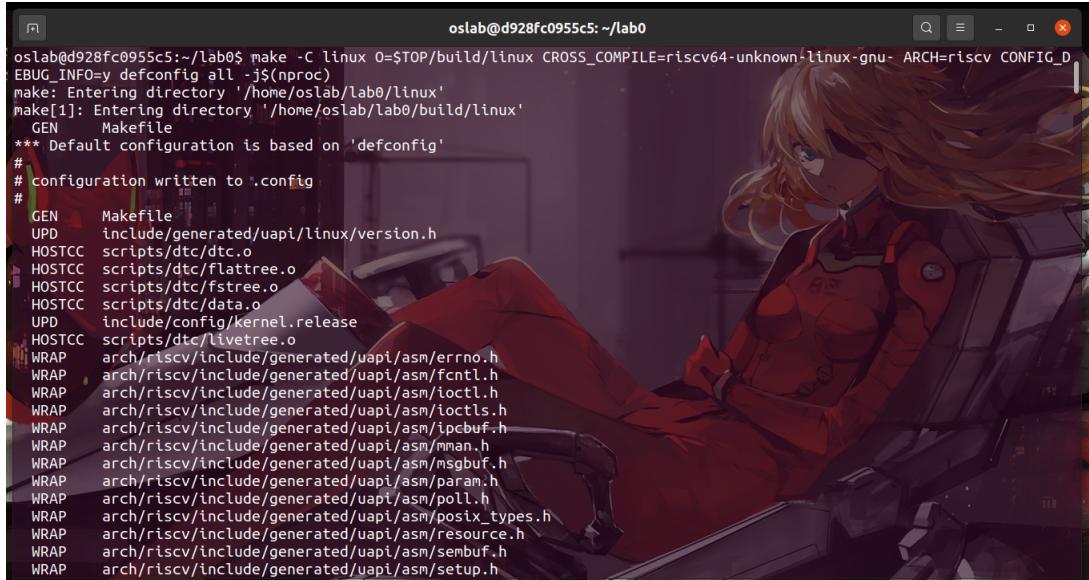
```
oslab@d928fc0955c5:~/lab0
oslab@d928fc0955c5:~$ pwd
/home/oslab
oslab@d928fc0955c5:~$ ls
lab0
oslab@d928fc0955c5:~$ clear
oslab@d928fc0955c5:~$ pwd
/home/oslab
oslab@d928fc0955c5:~$ cd lab0
oslab@d928fc0955c5:~/lab0$ export TOP=`pwd`
oslab@d928fc0955c5:~/lab0$ export RISCV=/opt/riscv
oslab@d928fc0955c5:~/lab0$ export PATH=$PATH:$RISCV/bin
oslab@d928fc0955c5:~/lab0$ echo $TOP
/home/oslab/lab0
oslab@d928fc0955c5:~/lab0$ echo $RISCV
/opt/riscv
oslab@d928fc0955c5:~/lab0$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/opt/riscv/bin
oslab@d928fc0955c5:~/lab0$ mkdir -p build/linux
oslab@d928fc0955c5:~/lab0$ ls
build linux rootfs ext4
oslab@d928fc0955c5:~/lab0$ make -C linux O=$TOP/build/linux CROSS_COMPILE=riscv64-unknown-linux-gnu-ARCH=riscv CONFIG_DEBUG_INFO=defconfig all -j$(nproc)
make: Entering directory '/home/oslab/lab0/linux'
make[1]: Entering directory '/home/oslab/lab0/build/linux'
  GEN  Makefile
  HOSTCC scripts/basic/fixdep
  HOSTCC scripts/kconfig/conf.o
  HOSTCC scripts/kconfig/confdata.o
  HOSTCC scripts/kconfig/expr.o
```

如下图，第一次执行的时候，编译并未成功，报错为找不到编译器。经过检查，发现是我误解实验指导 pdf 文件的排版所致。命令中的一个空格恰好出现在了 pdf 文件中的空格位置，我输入时并未发现这个问题，导致命令找不到对应的 gcc。

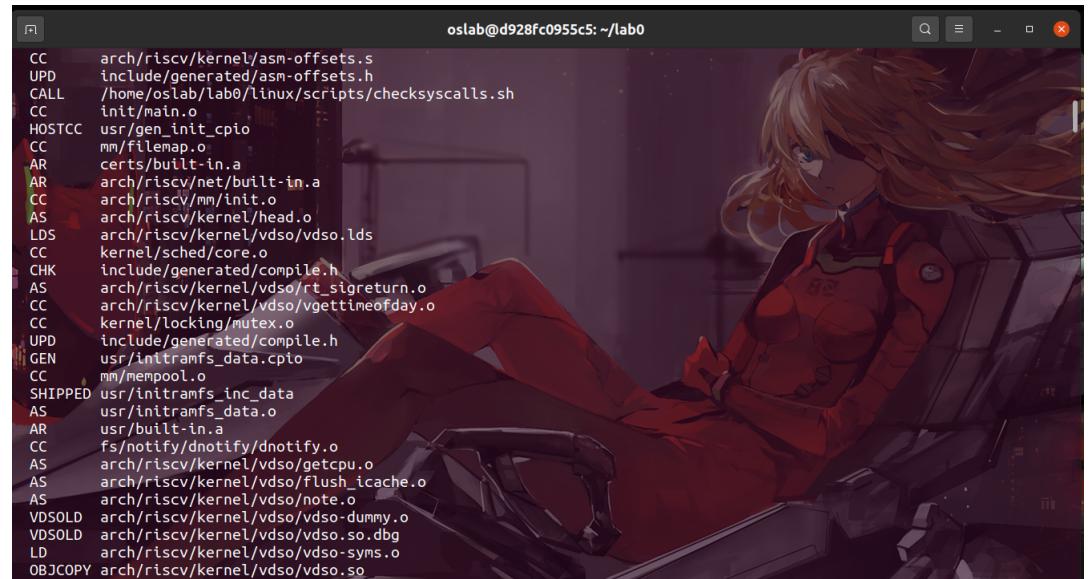


```
oslab@d928fc0955c5:~/lab0$ make -C linux O=$TOP/build/linux CROSS_COMPILE=riscv64-unknown-linux-gnu-ARCH=riscv CONFIG_DEBUG_INFO=y defconfig all -j$(nproc)
make: Entering directory '/home/oslab/lab0/linux'
make[1]: Entering directory '/home/oslab/lab0/build/linux'
  GEN  Makefile
 HOSTCC scripts/basic/fixedep
 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/preprocess.o
 HOSTCC scripts/kconfig/symbol.o
 HOSTCC scripts/kconfig/util.o
 HOSTCC scripts/kconfig/lexer.lex.o
 HOSTCC scripts/kconfig/parser.tab.o
 HOSTLD scripts/kconfig/conf
*** Default configuration is based on 'x86_64_defconfig'
scripts/Kconfig.include:39: compiler 'riscv64-unknown-linux-gnu-ARCH=riscvgcc' not found
/home/oslab/lab0/linux/scripts/kconfig/Makefile:80: recipe for target 'defconfig' failed
make[3]: *** [defconfig] Error 1
/home/oslab/lab0/linux/Makefile:606: recipe for target 'defconfig' failed
make[2]: *** [defconfig] Error 2
/home/oslab/lab0/linux/Makefile:336: recipe for target '__build_one_by_one' failed
make[1]: *** [__build_one_by_one] Error 2
make[1]: Leaving directory '/home/oslab/lab0/build/linux'
Makefile:185: recipe for target '__sub-make' failed
make: *** [__sub-make] Error 2
make: Leaving directory '/home/oslab/lab0/linux'
oslab@d928fc0955c5:~/Lab0$ |
```

如下图，重新输入命令并注意ARCH前的空格。这一次，可以看到命令顺利执行，出现了大量输出信息。



```
oslab@d928fc0955c5:~/lab0$ make -C linux O=$TOP/build/linux CROSS_COMPILE=riscv64-unknown-linux-gnu- ARCH=riscv CONFIG_DEBUG_INFO=y defconfig all -j$(nproc)
make: Entering directory '/home/oslab/lab0/linux'
make[1]: Entering directory '/home/oslab/lab0/build/linux'
  GEN  Makefile
*** Default configuration is based on 'defconfig'
#
# configuration written to '.config'
#
  GEN  Makefile
  UPD  include/generated/uapi/linux/version.h
  HOSTCC scripts/dtc/dtc.o
  HOSTCC scripts/dtc/flattree.o
  HOSTCC scripts/dtc/fstree.o
  HOSTCC scripts/dtc/data.o
  UPD  include/config/kernel.release
  HOSTCC scripts/dtc/livetree.o
  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
  WRAP arch/riscv/include/generated/uapi/asm/paran.h
  WRAP arch/riscv/include/generated/uapi/asm/poll.h
  WRAP arch/riscv/include/generated/uapi/asm posix_types.h
  WRAP arch/riscv/include/generated/uapi/asm/resource.h
  WRAP arch/riscv/include/generated/uapi/asm/sembuf.h
  WRAP arch/riscv/include/generated/uapi/asm/setup.h
```



```
oslab@d928fc0955c5:~/lab0
CC      arch/riscv/kernel/asm-offsets.s
UPD    include/generated/asm-offsets.h
CALL   /home/oslab/lab0/linux/scripts/checksyscalls.sh
CC      init/main.o
HOSTCC user/gen_init_cpio
CC      mm/filemap.o
AR     certs/built-in.a
CC      arch/riscv/net/built-in.a
CC      arch/riscv/mm/init.o
AS     arch/riscv/kernel/head.o
LDS   arch/riscv/kernel/vdso/vdso.lds
CC      kernel/sched/core.o
CHK   include/generated/compile.h
AS     arch/riscv/kernel/vdso/rt_sigreturn.o
CC      arch/riscv/kernel/vdso/gettimeofday.o
CC      kernel/locking/mutex.o
UPD   include/generated/compile.h
GEN   usr/initramfs_data.cpio
CC      mm/mempool.o
SHIPPED usr/initramfs_inc_data
AS     usr/initramfs_data.o
AR     usr/built-in.a
CC      fs/notify/dnotify.o
AS     arch/riscv/kernel/vdso/getcpu.o
AS     arch/riscv/kernel/vdso/flush_icache.o
AS     arch/riscv/kernel/vdso/note.o
VDSOLD arch/riscv/kernel/vdso/vdso-dummy.o
VDSOLD arch/riscv/kernel/vdso/vdso.so.debug
LD     arch/riscv/kernel/vdso/vdso-syms.o
OBJCOPY arch/riscv/kernel/vdso/vdso.so
```

等待命令执行完毕，进入 build/linux/arch/riscv/boot 目录下查看，可以看到出现了内核文件。编译linux内核成功。

```
CC  drivers/gpu/drm/radeon/r100.o
CC  drivers/gpu/drm/radeon/r300.o
CC  drivers/gpu/drm/radeon/r420.o
AR  drivers/gpu/drm/radeon/built-in.a
AR  drivers/gpu/drm/built-in.a
AR  drivers/gpu/built-in.a
AR  drivers/built-in.a
GEN  .version
CHK  include/generated/compile.h
LD   vmlinux.o
MODPOST vmlinux.symvers
MODINFO modules.builtin.modinfo
GEN   modules.builtin
LD    .tmp_vmlinux.kallsyms1
KSYM .tmp_vmlinux.kallsyms1.o
LD    .tmp_vmlinux.kallsyms2
KSYM .tmp_vmlinux.kallsyms2.o
LD    vmlinux
SYMSMAP 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'
oslab@d928fc0955c5:~/Lab0$ ls build/linux/arch/riscv/boot
Image Image.gz dts
oslab@d928fc0955c5:~/Lab0$ |
```

使用QEMU运行内核

输入命令，使用QEMU运行内核。系统会输出许多初始化信息。

```
oslab@d928fc0955c5:~/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

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 : 0x0000000000000022
MEDELEG : 0x000000000000b109
PMP0    : 0x0000000080000000-0x000000008001ffff (A)
PMP1    : 0x0000000000000000-0xffffffffffff (A,R,W,X)
[ 0.00000] OF: fdt: Ignoring memory range 0x80000000 - 0x80200000
[ 0.00000] Linux version 5.8.11 (oslab@d928fc0955c5) (riscv64-unknown-linux-gnu-gcc (GCC) 10.1.0, GNU ld (GNU Binutils) 2.35) #1 SMP Thu Sep 10 12:33:20 UTC 2021
```

等待系统提示输入用户名，以 `root` 身份登入后，可以简单执行几条命令。由命令行的提示符可以看出，已经进入了OEMU模拟器。最后退出模拟器。

```
oslab@d928fc0955c5: ~/lab0
Starting klogd: OK
Running sysctl: OK
Starting mddev... OK
modprobe: can't change directory to '/lib/modules': No such file or directory
Initializing random number generator: OK
Saving random seed: [  5.712431] random: 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

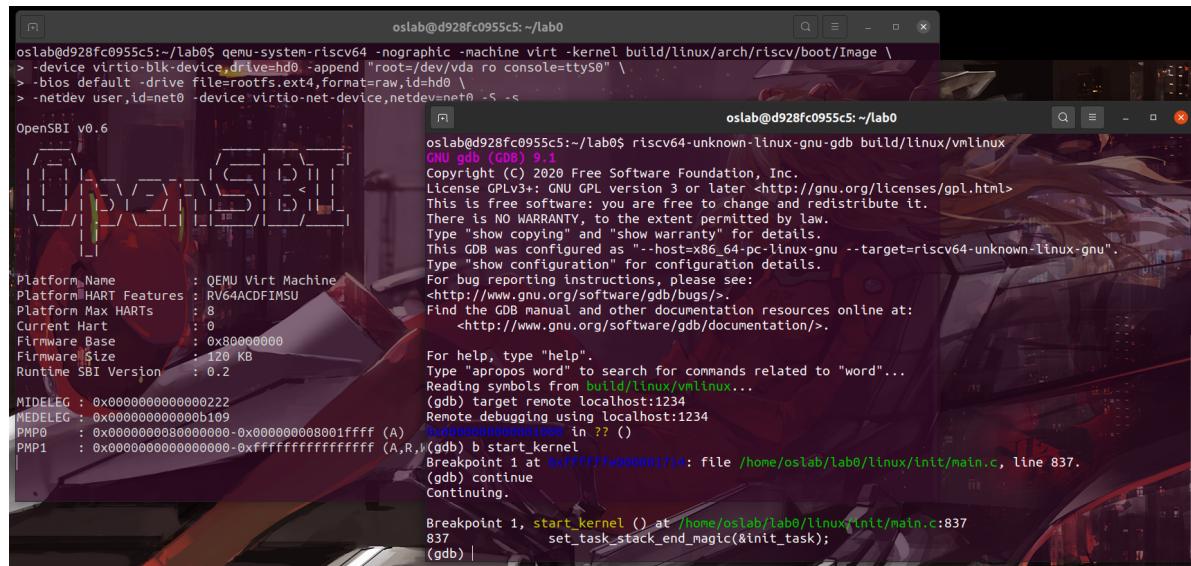
Welcome to Buildroot
buildroot login: root
#|pwd
/root
# ls
# cd ..
# ls
bin      lib      lost+found  opt      run      tmp
dev      lib64    media       proc    sbin      usr
etc      linuxrc   mnt       root    sys      var
# pwd
/
# [ 190.986158] random: fast init done
QEMU: Terminated
osLab@d928fc0955c5:~/lab0$ |
```

使用 gdb 对内核进行调试

在一个终端中进入模拟器，可以看到因为指定了 `-S -s` 参数，模拟器没有运行，没有任何输出。



根据实验指导，在另一个终端进入容器。连接上QEMU后，设置断点并继续执行。运行至断点处，可以看到因为断点，模拟器在输出初始化信息的过程中停了下来。同时，gdb显示断点在 `main.c` 中的837行。



根据gdb给出的信息，打开源文件。可以看到断点设置在了该函数的第一个执行语句处。

文件 编辑 选择 查看 转到 运行 终端 帮助

受限模式旨在实现安全地浏览代码。信任此窗口以启用所有功能。 管理 了解详细信息

```

825 }
826
827 void __init __weak arch_call_rest_init(void)
828 {
829     rest_init();
830 }
831
832 asmlinkage __visible void __init start_kernel(void)
833 {
834     char *command_line;
835     char *after_dashes;
836
837     set_task_stack_end_magic(&init_task);
838     smp_setup_processor_id();
839     debug_objects_early_init();
840
841     cgroup_init_early();
842
843     local_irq_disable();
844     early_boot_irqs_disabled = true;
845
846     /*
847         * Interrupts are still disabled. Do necessary setups, then

```

查看目录下的其他源文件，再选取一两个位置设置断点。除了实验指导中的断点，我还在 `version.c` 的 43 行和 `main.c` 的 852 行设置了断点。但是因为不甚了解源文件的结构，`version.c` 断点处似乎并不会被执行，因此我没有让它在未来链接时设置断点，而只增加设置了第二个断点。内核初始化至第一个断点时，通过 `backtrace` 或者 `b` 命令，可以看到此处有两层函数调用。

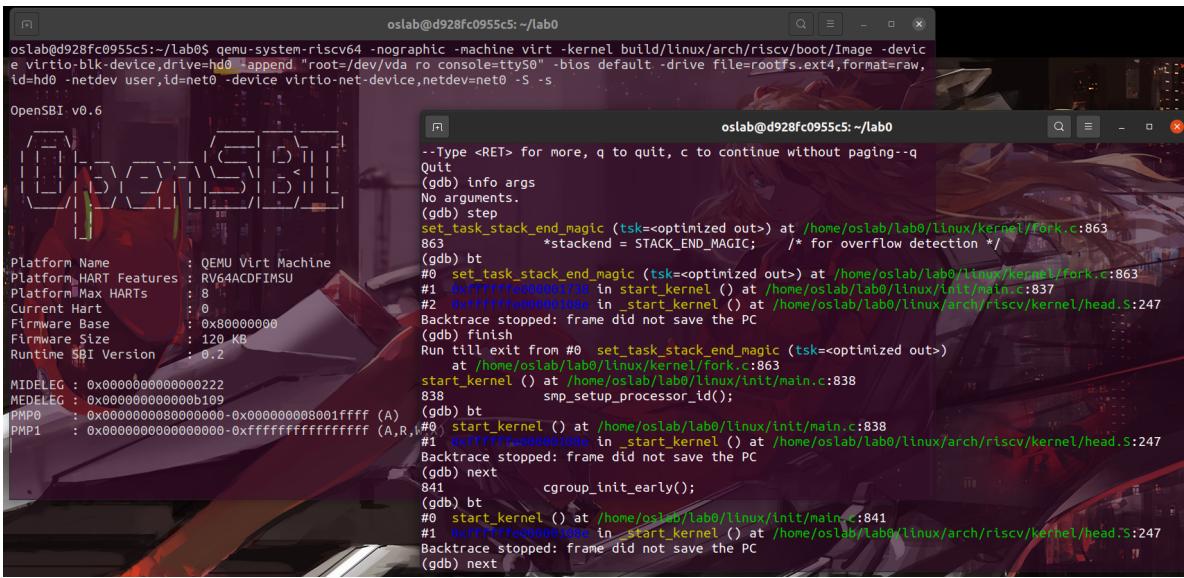
```

oslab@d928fc0955c5:~/lab0
oslab@d928fc0955c5:~/lab0
Type "apropos word" to search for commands related to "word"...
Reading symbols from build/linux/vmlinux...
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
0xb000000000001000 ln ?? ()
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff0000001714: file /home/oslab/lab0/linux/init/main.c, line 837.
(gdb) b version.c:43
No line 43 in file "version.c".
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) b main.c:852
Breakpoint 2 at 0xffffffff000001752: file /home/oslab/lab0/linux/init/main.c, line 852.
(gdb) continue
Undefined command: "continue". Try "help".
(gdb) continue
Continuing.

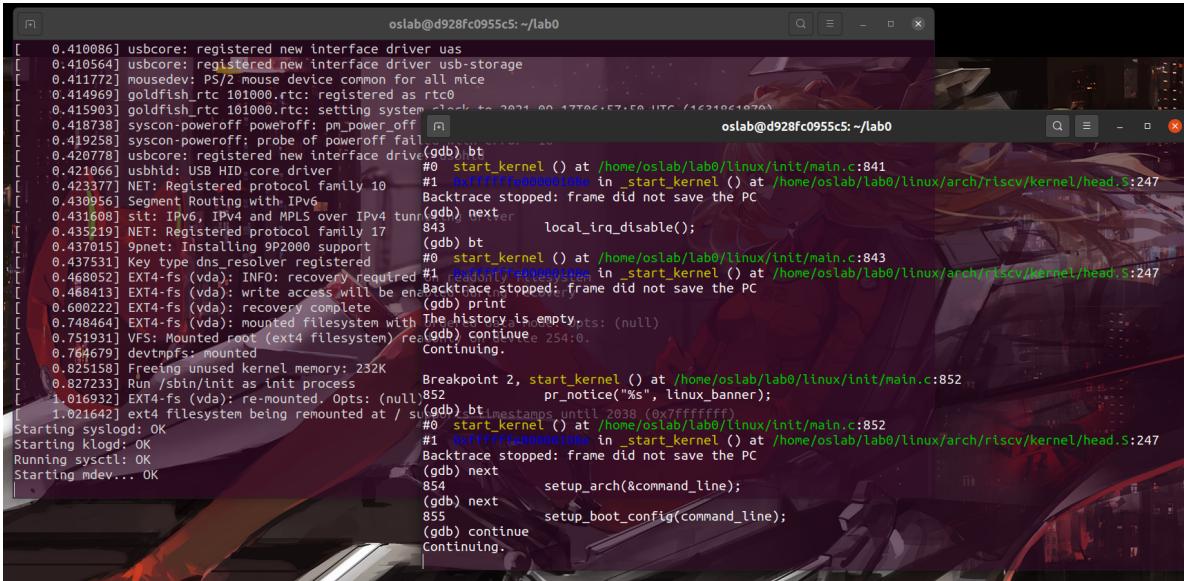
Breakpoint 1, start_kernel () at /home/oslab/lab0/linux/init/main.c:837
837             set_task_stack_end_magic(&init_task);
(gdb) backtrace
#0 start_kernel () at /home/oslab/lab0/linux/init/main.c:837
#1 0xffffffff00000108e in start_kernel () at /home/oslab/lab0/linux/arch/riscv/kernel/head.s:247
Backtrace stopped: frame did not save the PC
(gdb) info
"info" must be followed by the name of an info command.
List of Info subcommands:
info address -- Describe where symbol SYM is stored.

```

在此处，通过 `info args` 查看该函数的参数，显示没有参数，这与上述源代码中的截图一致（该函数没有传入参数）。通过 `step` 或 `s` 命令进入函数，再次查看函数调用层级，可以看到比 `start_kernel` 中的多了一层。通过 `finish` 命令结束该函数的运行后，再次查看调用函数层级，可以看到又回到了两层。通过 `next` 或 `n` 命令单步执行，可以看到行数的变化。



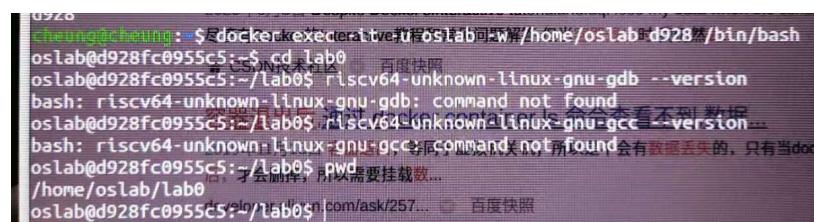
通过 `continue` 命令一直执行，执行到第二个断点处停下。简单查看信息后，再次 `continue`。此后不再有断点，可以看到模拟器正常执行至出现命令行提示符。



二、讨论心得

本次实验主要遇到两个问题，第一个问题是命令的空格问题。由于pdf的排版原因，编译命令与启动QEMU命令都有空格出现在回车处，一开始因为没有彻底理解命令每个参数的意义，我没有键入空格，导致错误。在了解命令的使用方法，以及各参数的意义后，我才加入空格，正确执行了命令。在这个过程中，实验指导中的链接很好地帮助了我理解命令的结构、docker以及qemu。

第二个问题是容器的环境问题。在进行调试时我发现，退出一个容器后再登入，容器会无法使用gcc等工具。经过查找资料与询问助教，我发现这是因为容器的环境变量又回到了初始状态。此前步骤中我手动添加的三个环境变量都被重置了，在我重新添加三个环境变量后，模拟器又可以正常运行。此外，我还检查了之前编译内核时产生的文件，发现文件没有被删除。似乎，重新进入容器时，**环境变量等环境会被重置，但是容器中的文件不会被重置**。



而要解决这个问题，助教给我的解决方案是将环境变量设置挂载到一个外部目录。一个我正在采用且不太直接的方案是，准备一个脚本文件 `pre.sh`，内有每次进入容器都要执行的命令如下：

```
cd lab0
export TOP=`pwd`/
export RISCV=/opt/riscv
export $PATH=$PATH:$RISCV/bin
```

进入容器后，首先执行 `source pre.sh` 命令，即可利用脚本将环境变量设置好。

与之类似的一个方法是直接在 `.bashrc` 文件内写好命令。