



# 本科生实验报告

实验课程：\_\_\_\_操作系统原理实验\_\_\_\_

实验名称：\_\_\_\_编译内核/利用已有内核创建 OS\_\_\_\_

专业名称：\_\_\_\_计算机科学与技术\_\_\_\_

学生姓名：\_\_\_\_熊彦钧\_\_\_\_

学生学号：\_\_\_\_23336266\_\_\_\_

实验地点：\_\_\_\_实验楼 B203\_\_\_\_

实验成绩：\_\_\_\_

报告时间：\_\_\_\_2023 年 4 月 28 日\_\_\_\_

## Section 1 实验概述

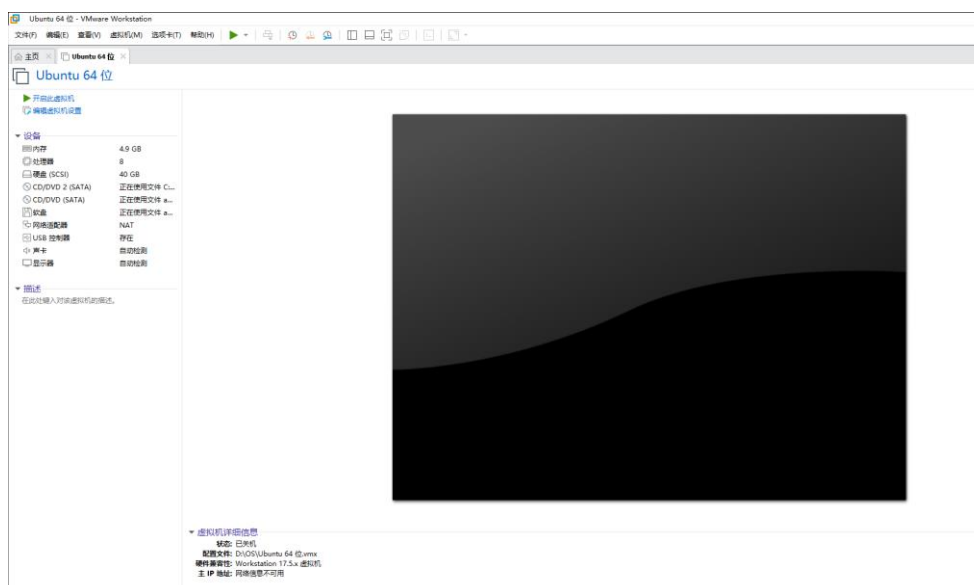
- 1.搭建 OS 内核开发环境包括：代码编辑环境、编译环境、运行环境、调试环境等。
- 2.下载并编译 i386（32 位）内核，并利用 qemu 启动内核。
- 3.熟悉制作 initramfs 的方法。
- 4.编写简单应用程序随内核启动运行。
- 5.编译 i386 版本的 Busybox，随内核启动，构建简单的 OS。
- 6.开启远程调试功能，进行调试跟踪代码运行。
- 7.撰写实验报告

## Section 2 实验步骤与实验结果

### ----- 实验任务 0（课程实验前置准备） -----

- 任务要求：配置操作系统实验所需的环境
- 实验步骤：

1.下载并启动虚拟机：由于本人在使用 virtualbox 时遇到了诸多 bug，经同学推荐，使用软件 VMware 来启动虚拟机。本人下载的是学校镜像网站 matrix 上的 Ubuntu22.04.5。经过一系列虚拟机设置后，虚拟机的界面如下：



2.配置 C/C++环境：这一步骤所需代码如下：

```
sudo apt install binutils
sudo apt install gcc
```

在虚拟机的命令行中输入代码并观察输出：

```

jhinx@jhinx-virtual-machine:~$ sudo apt install binutils
[sudo] password for jhinx:
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
将会同时安装下列软件：
  binutils-common binutils-x86-64-linux-gnu libbinutils libctf0
建议安装：
  binutils-doc
下列软件包将被升级：
  binutils binutils-common binutils-x86-64-linux-gnu libbinutils libctf0
升级了 5 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 206 个软件包未被升级。
需要下载 3,317 kB 的归档。
解压后会消耗 4,096 B 的额外空间。
您希望继续执行吗？ [Y/n] Y
获取:1 http://mirrors.tuna.tsinghua.edu.cn/ubuntu jammy-updates/main amd64 libctf0 amd64 2.38-4ubuntu2.7 [103 kB]
获取:2 http://mirrors.tuna.tsinghua.edu.cn/ubuntu jammy-updates/main amd64 binutils-x86-64-linux-gnu amd64 2.38-4ubuntu2.7 [2,326 kB]
获取:3 http://mirrors.tuna.tsinghua.edu.cn/ubuntu jammy-updates/main amd64 binutils-common amd64 2.38-4ubuntu2.7 [222 kB]
获取:4 http://mirrors.tuna.tsinghua.edu.cn/ubuntu jammy-updates/main amd64 binutils amd64 2.38-4ubuntu2.7 [3,196 B]
获取:5 http://mirrors.tuna.tsinghua.edu.cn/ubuntu jammy-updates/main amd64 libbinutils amd64 2.38-4ubuntu2.7 [662 kB]
已下载 3,317 kB，耗时 3秒 (1,207 kB/s)
(正在读取数据库 ... 系统当前共安装有 211247 个文件和目录。)
准备解压 .../libctf0_2.38-4ubuntu2.7_amd64.deb ...
正在解压 libctf0:amd64 (2.38-4ubuntu2.7) 并覆盖 (2.38-4ubuntu2.6) ...
准备解压 .../binutils-x86-64-linux-gnu_2.38-4ubuntu2.7_amd64.deb ...
正在解压 binutils-x86-64-linux-gnu (2.38-4ubuntu2.7) 并覆盖 (2.38-4ubuntu2.6) ..
准备解压 .../binutils-common_2.38-4ubuntu2.7_amd64.deb ...
正在解压 binutils-common:amd64 (2.38-4ubuntu2.7) 并覆盖 (2.38-4ubuntu2.6) ...
准备解压 .../binutils_2.38-4ubuntu2.7_amd64.deb ...
正在解压 binutils (2.38-4ubuntu2.7) 并覆盖 (2.38-4ubuntu2.6) ...
准备解压 .../libbinutils_2.38-4ubuntu2.7_amd64.deb ...
正在解压 libbinutils:amd64 (2.38-4ubuntu2.7) 并覆盖 (2.38-4ubuntu2.6) ...
正在设置 binutils-common:amd64 (2.38-4ubuntu2.7) ...
正在设置 libbinutils:amd64 (2.38-4ubuntu2.7) ...
正在设置 libctf0:amd64 (2.38-4ubuntu2.7) ...
正在设置 binutils-x86-64-linux-gnu (2.38-4ubuntu2.7) ...
正在设置 binutils (2.38-4ubuntu2.7) ...
正在处理用于 libc-bin (2.35-0ubuntu3.8) 的触发器 ...
正在处理用于 man-db (2.10.2-1) 的触发器 ...

```

```

jhinx@jhinx-virtual-machine:~$ sudo apt install gcc
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
gcc 已经是最新版 (4:11.2.0-1ubuntu1)。
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 206 个软件包未被升级。

```

## GCC -V

```

jhinx@jhinx-virtual-machine:~$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/11/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none:amdgc-n-andhsa
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 11.4.0-1ubuntu1-22.04' --with-bug
url=file:///usr/share/doc/gcc-11/README.Bugs --enable-languages=c,ada,c++,go,brig,d,fortran,objc
,obj-c++,m2 --prefix=/usr --with-gcc-major-version-only --program-suffix=-11 --program-prefix=x8
6_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-includ
e-d-gettext --enable-threads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap --enable-cloc
ale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --
enable-gnu-unique-object --disable-vtable-verify --enable-plugin --enable-default-pie --with-sys
tem-zlib --enable-libphobos-checking=release --with-target-system-zlib=auto --enable-objc-gc=aut
o --enable-multiarch --disable-werror --enable-cet --with-arch-32=i686 --with-abi=m64 --with-mul
tilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-offload-targets=nvptx-non
e=/build/gcc-11-XeT9LY/gcc-11-11.4.0/debian/tmp-nvptx/usr,amdgc-n-andhsa=/build/gcc-11-XeT9LY/gcc
-11-11.4.0/debian/tmp-gcn/usr --without-cuda-driver --enable-checking=release --build=x86_64-lin
ux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu --with-build-config=bootstrap-lto-lean
--enable-link-serialization=2
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 11.4.0 (Ubuntu 11.4.0-1ubuntu1-22.04)

```

## 3.安装其他工具：

```
sudo apt install nasm
sudo apt install qemu
sudo apt install cmake
sudo apt install libncurses5-dev
sudo apt install bison
sudo apt install flex
sudo apt install libssl-dev
sudo apt install libc6-dev-i386
```

以上指令的作用，都是在虚拟机上下载和操作系统实验相关的软件/工具。由于篇幅问题，此处就不展示相关截图了。

## ----- 实验任务 1-----

- 任务要求：在虚拟机中编译 Linux 内核
- 实验步骤：

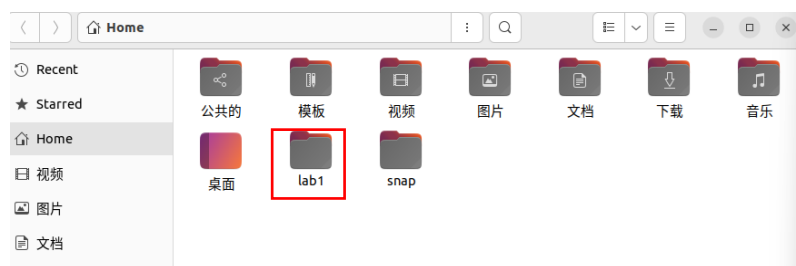
### 1. 下载内核：

按照实验指导，我们首先创建文件夹 lab1 并进入，随后下载内核到文件夹 lab1，然后解压并进入，以上步骤所需的代码如下：

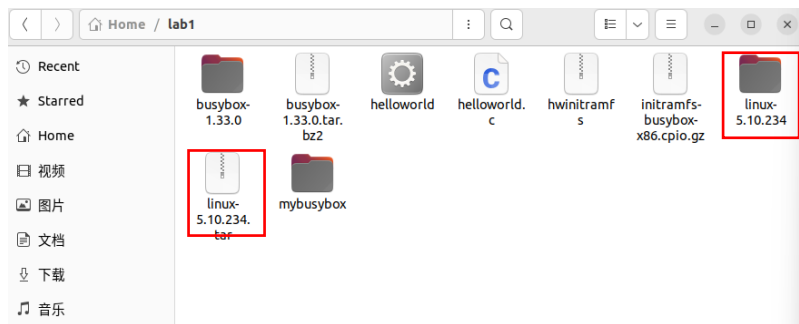
```
mkdir ~/lab1
cd ~/lab1
xz -d linux-5.10.234.tar.xz
tar -xvf linux-5.10.234.tar
cd linux-5.10.234
```

（注，由于本人下载的是 5.10.234 版本的 Linux，因此代码需要全部改成 5.10.234）

创建文件夹后，可以在 File 中看到这里多了一个 lab1 文件夹



下载和解压后，lab1 会多出这两个文件：

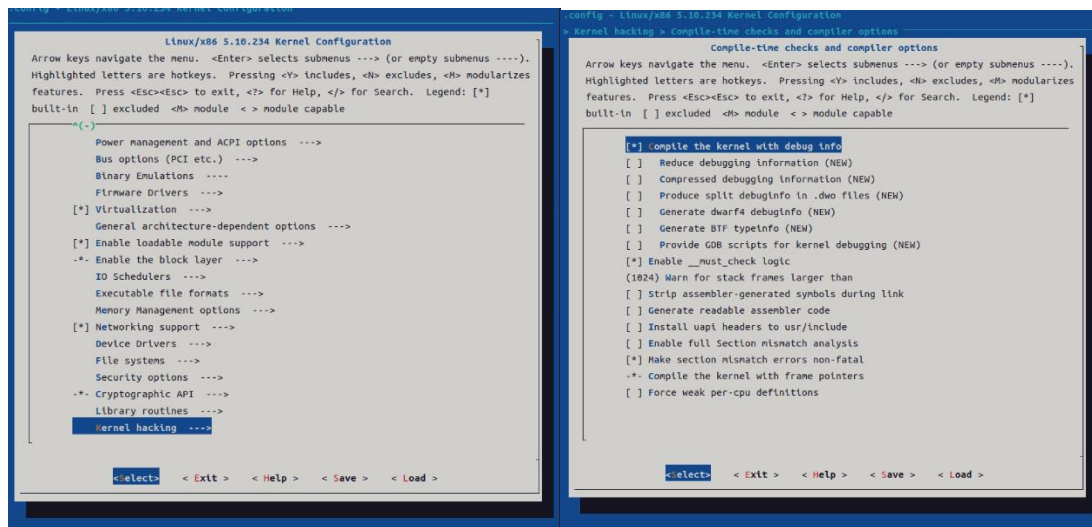


2.编译内核：按照要求将内核编译成 i386 32 位版本，首先设置编译条件：

```
make i386_defconfig
make menuconfig
```

```
jhinx@jhinx-virtual-machine:~/lab1$ cd linux-5.10.234
jhinx@jhinx-virtual-machine:~/lab1/linux-5.10.234$ make i386_defconfig
#
# configuration written to .config
#
jhinx@jhinx-virtual-machine:~/lab1/linux-5.10.234$ make menuconfig
*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.
jhinx@jhinx-virtual-machine:~/lab1/linux-5.10.234$
```

在打开的图像界面中依次选择 Kernel hacking、Compile-time checks and compiler options，最后在[ ] Compile the kernel with debug info 输入 Y 勾选，保存退出



最后一步，编译内核：

```
make -j8
```

```
jhinx@jhinx-virtual-machine:~/lab1/linux-5.10.234$ make -j8
SYNC include/config/auto.conf.cmd
CALL scripts/atomic/check-atomics.sh
CALL scripts/checksyscalls.sh
CHK include/generated/compile.h
kernel: arch/x86/boot/bzImage is ready (#2)
```

（注，由于之前已经编译过一次，因此截图中的输出行数较少）

## 实验任务 2

- 任务要求：使用 qemu 和 gdb 启动刚刚编译好的 Linux-5.10.234 内核并进行调试。
- 实验步骤：

### 1. 启动 qemu:

```
cd ~/lab1
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage
-s -S -append "console=ttyS0" -nographic
```

```
jhinx@jhinx-virtual-machine:~$ cd ~/lab1
jhinx@jhinx-virtual-machine:~/lab1$ qemu-system-i386 -kernel linux-5.10.234/arch
/x86/boot/bzImage -s -S -append "console=ttyS0" -nographic
```

此时，qemu 并未输出任何信息，这是因为我们开启了 gdb 调试，而 qemu 在等待 gdb 输入的指令后才能继续执行。接下来我们启动 gdb，通过 gdb 来告诉 qemu 应该怎么做。

**2.gdb 调试:** 在另外一个 Terminal 下启动 gdb，注意，不要关闭 qemu 所在的 Terminal。gdb 调试的代码如下：

```
cd ~/lab1 (要在 lab1 中运行)
gdb (启动 gdb)
file linux-5.10.234/vmlinux (加载符号表)
target remote:1234 (连接 qemu)
break start_kernel (设置断点)
c (运行)
```

```
jhinx@jhinx-virtual-machine:~$ cd ~/lab1
jhinx@jhinx-virtual-machine:~/lab1$ gdb
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/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:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<https://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file linux-5.10.234/vmlinux
Reading symbols from linux-5.10.234/vmlinux...
(gdb) target remote:1234
Remote debugging using 1234
*0x00000000 in ?? ()
(gdb) break start_kernel
Breakpoint 1 at 0xc21297d2: file init/main.c, line 849.
(gdb) c
continuing.
(gdb) |
```

```
2.767024] PMU: No PMU found: 121102110
2.769048] printk: console [netcon0] enabled
2.769547] netconsole: network logging started
2.776556] cfg80211: Loading compiled-in X.509 certificates for regulatory e
2.807846] kworker/u2:1 (59) used greatest stack depth: 7164 bytes left
2.830855] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
2.835499] cfg80211: Loaded X.509 cert 'wens: 61c838651aabdca94b0bdc7fff06c7'
2.839814] platform regulatory.8a: Direct firmware load for regulatory.db fa2
2.840339] cfg80211: failed to load regulatory.db
2.841421] clk: Disabling unused clocks
2.842024] ALSA device list:
2.842487]   No soundcards found.
3.299857] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8043
3.301818] nd: Waiting for all devices to be available before autodetect
3.303536] nd: If you don't use raid, use raid=noautodetect
3.304720] nd: Autodetecting RAID arrays.
3.306123] nd: autorn ---
3.306333] nd: ... autorn DONE.
3.309312] VFS: Cannot open root device "(null)" or unknown-block(0,0): errr
3.309806] Please append a correct "root=" boot option; here are the availa:
3.311002] 0b00 1048575 sr0
3.311035] driver: sr
3.311896] Kernel panic - not syncing: VFS: Unable to mount root fs on unkn)
3.313338] CPU: 0 PID: 1 Comm: swapper/0 Not tainted 5.10.234 #2
3.313962] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), B105 1.154
3.314567] Call Trace:
3.317260] dump_stack+0x54/0x68
3.317640] panic+0xb1/0x261
3.318029] mount_block_root+0x145/0x1af
3.318431] mount_root+0x1/0xe9
3.318532] prepare_namespace+0x116/0x141
3.318635] kernel_init_freeable+0x1cf/0x1dc
3.318793] ? rest_init+0x92/0x92
3.318963] kernel_init+0x0/0x0e
3.319075] ret_from_kernel_init+0x28
3.320053] Kernel Offset: disabled
3.320935] ---[ end Kernel panic - not syncing: VFS: Unable to mount root f-
```

(注：左为 gdb 调试，右为 qemu 输出)

根据 qemu 输出的内容，Linux 系统可以成功启动

**3.该实验注意事项：** 由于 Linux 采用的是相对寻址，gdb 必须在 lab1 中运行，否则在加载符号表时，会因为找不到符号表而报错：

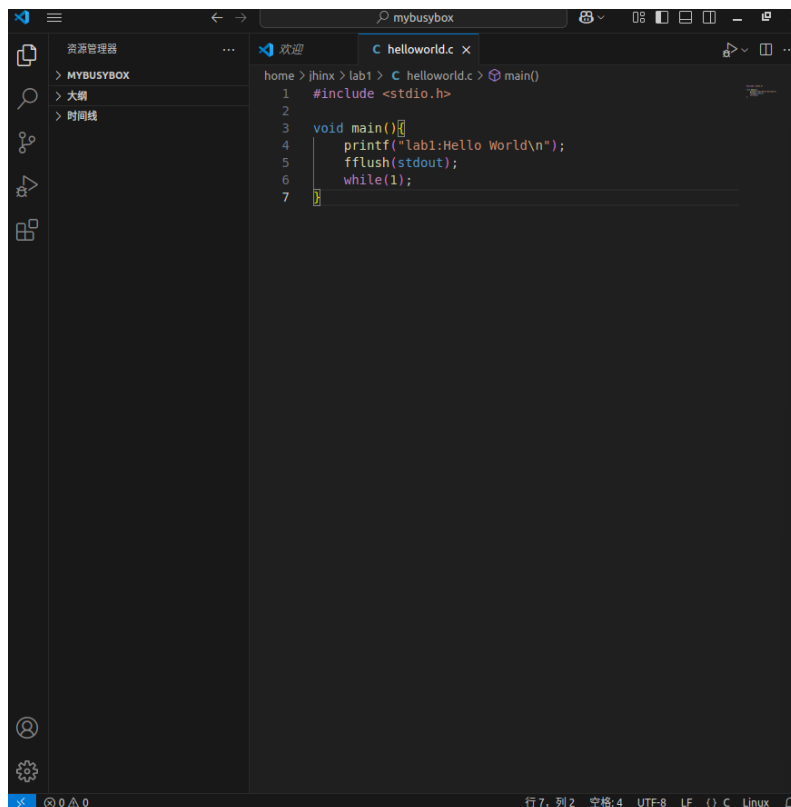
```
jhinx@jhinx-virtual-machine:~$ gdb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to 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:
<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".
(gdb) file linux-5.10.234/vmlinux
linux-5.10.234/vmlinux: No such file or directory.
(gdb)
```

### ----- 实验任务 3 -----

- 任务要求：制作一个 Initramfs 并在 qemu 中加载
- 实验步骤：

**1.制作 initramfs：** 首先制作一个 Hello World initramfs，这里我们通过 Linux 中下载的编辑器 VScode（在 Linux 中被命名为 Code.）来新建一个.c 形式的文件，并将他保存在 lab1 文件夹中。



随后我们将上面代码编译成 32 位可执行文件。

```
cd ~/lab1
gcc -o helloworld -m32 -static helloworld.c
```

## 2.加载 initramfs:

```
echo helloworld | cpio -o --format=newc > hwinitramfs (用 cpio 打包 initramfs)
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -
initrd hwinitramfs -s -S -append "console=ttyS0 rdinit=helloworld" -
nographic (启动内核, 并加载 initramfs)
```

```
jhinx@jhinx-virtual-machine:~$ cd ~/lab1
jhinx@jhinx-virtual-machine:~/lab1$ gcc -o helloworld -m32 -static helloworld.c
jhinx@jhinx-virtual-machine:~/lab1$ echo helloworld | cpio -o --format=newc > hw
initramfs
1459 blocks
jhinx@jhinx-virtual-machine:~/lab1$ qemu-system-i386 -kernel linux-5.10.234/arch
/x86/boot/bzImage -initrd hwinitramfs -s -S -append "console=ttyS0 rdinit=hellow
orld" -nographic
```

随后重复 gdb 调试过程

```
jhinx@jhinx-virtual-machine:~$ cd ~/lab1
jhinx@jhinx-virtual-machine:~/lab1$ gdb
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/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:
<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".
(gdb) file linux-5.10.234/vmlinux
Reading symbols from linux-5.10.234/vmlinux...
(gdb) target remote:1234
Remote debugging using :1234
*OoOoOoOoOo in ?? ()
(gdb) break start_kernel
Breakpoint 1 at 0xc21297d: file init/main.c, line 849.
(gdb) c
Continuing.
lab1:Hello World
(gdb) [ 2.403961] sched_clock: Marking stable (2422309915, -19091247)->(2517444294)
[ 2.406270] registered taskstats version 1
[ 2.406610] Loading compiled-in X.509 certificates
[ 2.414165] PM: Magic number: 13:269:779
[ 2.415024] printk: console [netcon0] enabled
[ 2.415363] netconsole: network logging started
[ 2.420267] cfg80211: Loading compiled-in X.509 certificates for regulatory da
[ 2.445499] tsc: Refined TSC clocksource calibration: 2688.129 MHz
[ 2.446522] clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x26bf7085
[ 2.447000] clocksource: Switched to clocksource tsc
[ 2.451163] kworker/u2:1 (59) used greatest stack depth: 7172 bytes left
[ 2.469834] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[ 2.473791] cfg80211: Loaded X.509 cert 'wens: 61c038651aabdcf94bd0ac7ff06c7'
[ 2.476807] platform regulatory.0: Direct firmware load for regulatory.db fa2
[ 2.477892] cfg80211: failed to load regulatory.db
[ 2.478757] clk: Disabling unused clocks
[ 2.479440] ALSA device list:
[ 2.481860] No soundcards found.
[ 2.585709] Freeing unused kernel image (initmem) memory: 684K
[ 2.591431] Write protecting kernel text and read-only data: 15556k
[ 2.593489] Run helloworld as init process
lab1:Hello World
[ 2.998828] input: InExPS/2 Generic Explorer Mouse as /devices/platform/i8043
```

(注: 左为 gdb 调试, 又为 qemu 输出内容)

可以看到成功输出了 lab1: Hello World\n

## 实验任务 4

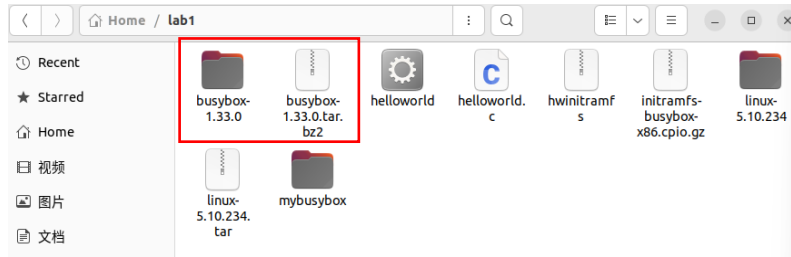
- 任务要求: 编译并启动 Busybox
- 实验步骤:

1.编译 busybox: 从课程网站下载 busybox 到 lab1 文件夹, 并解压

```
cd ~/lab1
tar -xjf busybox-1.33.0.tar.bz2
```

下载、解压后, 可以看到 lab1 文件夹中多出了这两个文件

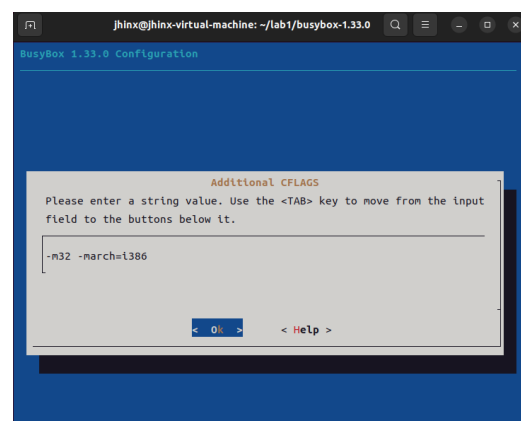
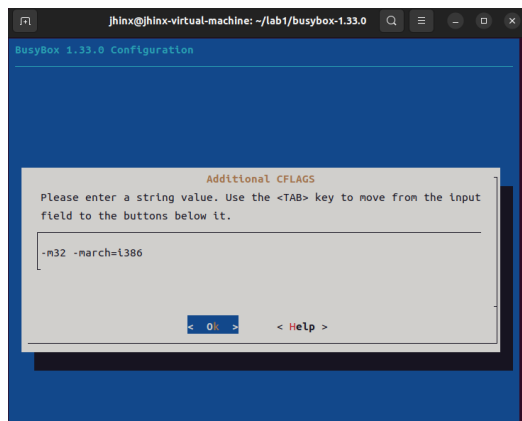




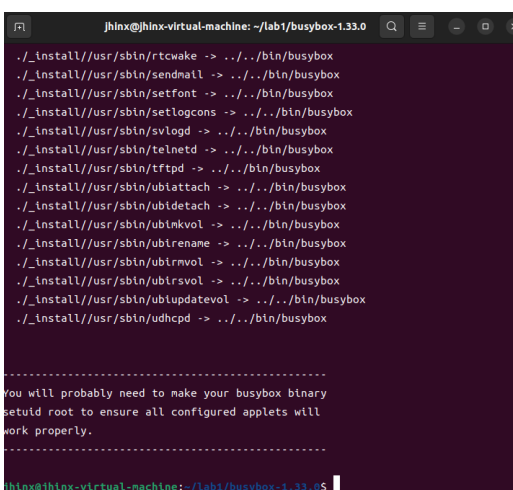
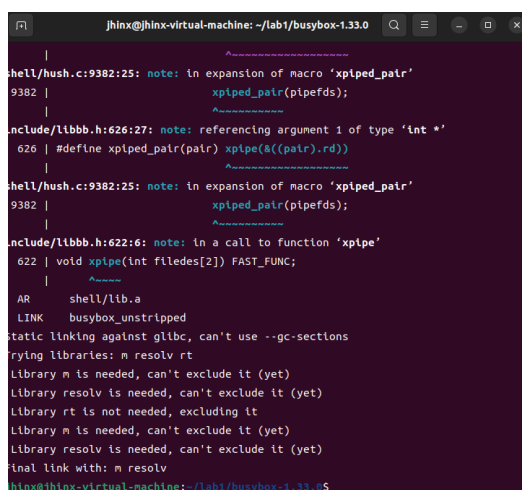
随后编译 busybox

```
make defconfig
make menuconfig
make -j8
make install
```

在 make menuconfig 后，会进入 setting 界面，按照实验指导，需要进行如下设置，以保证编译后的 busybox 可以在 i386 内核中执行 32 位文件。



上面是 make menuconfig 的设置，下面是编译内核后的输出



（注：在编译 busybox 时，有一处地方极易犯错，在上面我们提到，Linux 默认采用的是相对寻址，因此在编译 busybox 时，我们需要 cd 到 busybox 的文件夹中，否则编译代码将会报错）



```
Open  [icon] *init
~/lab1/mybusybox Save [icon] [icon] [icon] [icon]

1 #!/bin/sh
2 mount -t proc none /proc
3 mount -t sysfs none /sys
4 echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime) seconds\n"
5 exec /bin/sh
```

随后，执行以下代码：

```
chmod u+x init (为 init 加上执行权限)
find . -print0 | cpio --null -ov --format=newc | gzip -9 >
~/lab1/initramfs-busybox-x86.cpio.gz (将 x86-busybox 下面的内容打包归档成
cpio 文件)
cd ~/lab1
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -initrd
initramfs-busybox-x86.cpio.gz -nographic -append "console=ttyS0" -m
size=2048 (加载 busybox)
ls (查看当前文档)
```

```
jhinx@jhinx-virtual-machine: ~/lab1/mybusybox$ chmod u+x init
jhinx@jhinx-virtual-machine:~/lab1/mybusybox$ find . -print0 | cpio --null -ov -
-format=newc | gzip -9 > ~/lab1/initramfs-busybox-x86.cpio.gz
.
./usr
./usr/sbin
./usr/sbin/nandwrite
./usr/sbin/deluser
./usr/sbin/nanddump
./usr/sbin/rdate
./usr/sbin/nbd-client
./usr/sbin/partprobe
./usr/sbin/fsfreeze
./usr/sbin/chroot
./usr/sbin/setlogcons
./usr/sbin/i2cdump
./usr/sbin/fbset
./usr/sbin/add-shell
./usr/sbin/ubinkvol
./usr/sbin/ubiattach
./usr/sbin/telnetd
```

```
jhinx@jhinx-virtual-machine: ~/lab1 [icon] [icon] [icon] [icon] [icon] [icon]
[ 2.579565] clocksource: tsc: mask: 0xfffffffffffffff max_cycles: 0x26bf11fs
[ 2.581558] clocksource: Switched to clocksource tsc
[ 2.586403] cfg80211: Loading compiled-in X.509 certificates for regulatory e
[ 2.662788] modprobe (59) used greatest stack depth: 6908 bytes left
[ 2.694019] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[ 2.698339] cfg80211: Loaded X.509 cert 'wens: 61c038651aabdca94bd0ac7ff06c7'
[ 2.704844] platform regulatory.0: Direct firmware load for regulatory.db fa2
[ 2.707008] clk: Disabling unused clocks
[ 2.708726] cfg80211: failed to load regulatory.db
[ 2.710141] ALSA device list:
[ 2.710409] No soundcards found.
[ 2.830434] Freeing unused kernel image (initmem) memory: 684K
[ 2.835660] Write protecting kernel text and read-only data: 15556k
[ 2.836584] Run /init as init process
[ 2.871721] mount (63) used greatest stack depth: 6860 bytes left

Boot took 2.92 seconds

/bin/sh: can't access tty; job control turned off
/ # [ 3.121103] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/3
ls
bin      etc      linuxrc  root    sys
dev      init     proc    /sbin   /usr
/ #
```

## -----实验任务 5-----

- 任务要求： 完成 Linux 0.11 内核的编译、启动和调试
- 实验步骤：

1. 下载 Linux 0.11 内核代码： 通过微信文件传输助手，将课程群中 Linux0.11 压缩包传到虚拟机中，将压缩包复制到 lab1 中并解压。

由于压缩包的格式为.gz，因此使用以下指令进行解压

```
tar -xzf Linux-0.11-lab1.tar.gz
```

```
jhinx@jhinx-virtual-machine:~/桌面$ cd ~/lab1
jhinx@jhinx-virtual-machine:~/lab1$ tar -xzf Linux-0.11-lab1.tar.gz
Linux-0.11/
Linux-0.11/fs/
Linux-0.11/fs/buffer.c
Linux-0.11/fs/file_table.c
Linux-0.11/fs/char_dev.c
Linux-0.11/fs/Makefile
Linux-0.11/fs/block_dev.c
Linux-0.11/fs/truncate.c
Linux-0.11/fs/exec.c
Linux-0.11/fs/inode.c
Linux-0.11/fs/fcntl.c
Linux-0.11/fs/pipe.c
Linux-0.11/fs/super.c
Linux-0.11/fs/ioctl.c
Linux-0.11/fs/open.c
Linux-0.11/fs/file_dev.c
Linux-0.11/fs/bitmap.c
Linux-0.11/fs/stat.c
Linux-0.11/fs/read_write.c
Linux-0.11/fs/namei.c
Linux-0.11/hdc-0.11.img
Linux-0.11/Makefile
```

2. 编译 32 位版本的 Linux 0.11 内核： 找到 Makefile，查看里面的内容，通过 make 工具进行编译；另外，因为需要调试，所以需要在 gcc 编译命令中添加 -g 参数，产生内核的符号表；编译 32 位版本的内核，需要添加 -m32 参数；

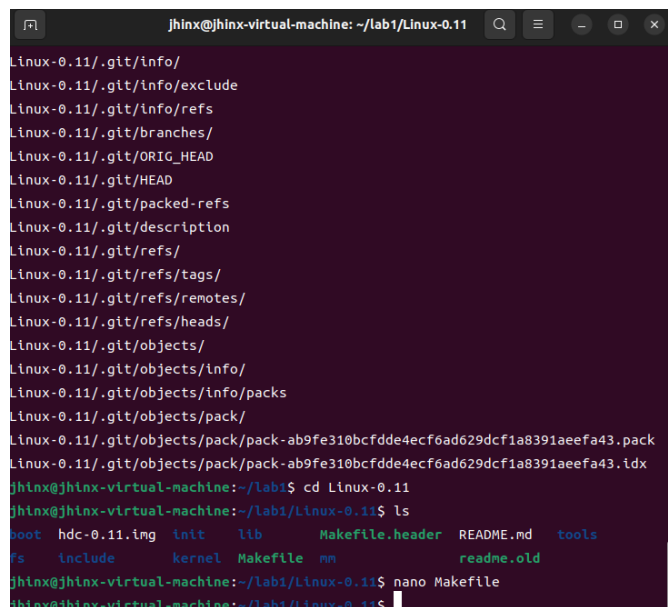
### (1) 查看 makefile

```
Open  Makefile
~/zhaojiaoyan/Linux0.11  Save  [Icons]

1 OS = Mac
2
3 # Indicate the Hardware Image file
4 HDA_IMG = hdc-0.11.img
5
6 # Indicate the path of the calltree
7 CALLTREE=$(shell find tools/ -name "calltree" -perm 755 -type f)
8
9 # Indicate the path of the bochs
10 BOCHS=$(shell find tools/ -name "bochs" -perm 755 -type f)
11 BOCHS=bochs
12
13 #
14 # If you want the ran-disk device, define this to be the
15 # size in blocks.
16 #
17 RANDISK = #-DRANDISK=512
18
19 # This is a basic Makefile for setting the general configuration
20 include Makefile.header
21
22 LDFLAGS += -Ttext 0 -e startup_32
23 CFLAGS += $(RANDISK) -Iinclude
24 CPP += -Iinclude
25
26 #
27 # ROOT_DEV specifies the default root-device when making the image.
28 # This can be either FLOPPY, /dev/xxx or empty, in which case the
29 # default of /dev/hd6 is used by 'build'.
30 #
31 ROOT_DEV= #FLOPPY
32
33 ARCHIVES=kernel/kernel.o mm/mm.o fs/fs.o
34 DRIVERS=kernel/bk_drv/bk_drv.a kernel/chr_drv/chr_drv.a
35 MATH=kernel/math/math.a
36 LIBS=lib/lib.a
37
38 .c.o:
39     $(CC) $(CFLAGS) -S -o $*.s $<
40 .s.o:
41     $(AS) -o $*.o $<
42 .c.o:
43     $(CC) $(CFLAGS) -c -o $*.o $<
44
45 all: Image
```

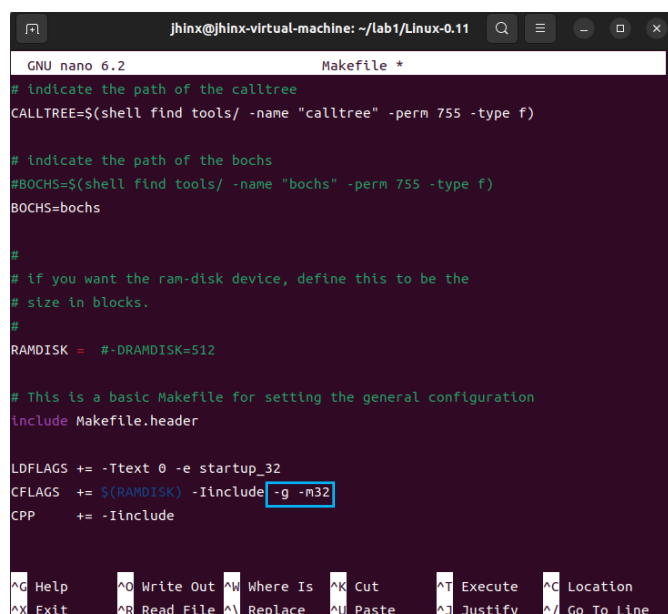
(2) 在 gcc 编译命令中添加参数：经过上网搜索，了解到可以通过以下方式修改：先通过以下代码进入 Makefile

```
cd Linux-0.11
nano Makefile
```



```
jhinx@jhinx-virtual-machine: ~/lab1/Linux-0.11
Linux-0.11/.git/info/
Linux-0.11/.git/info/exclude
Linux-0.11/.git/info/refs
Linux-0.11/.git/branches/
Linux-0.11/.git/ORIG_HEAD
Linux-0.11/.git/HEAD
Linux-0.11/.git/packed-refs
Linux-0.11/.git/description
Linux-0.11/.git/refs/
Linux-0.11/.git/refs/tags/
Linux-0.11/.git/refs/remotes/
Linux-0.11/.git/refs/heads/
Linux-0.11/.git/objects/
Linux-0.11/.git/objects/info/
Linux-0.11/.git/objects/info/packs
Linux-0.11/.git/objects/pack/
Linux-0.11/.git/objects/pack/pack-ab9fe310bcfdde4ecf6ad629dcf1a8391aeefa43.pack
Linux-0.11/.git/objects/pack/pack-ab9fe310bcfdde4ecf6ad629dcf1a8391aeefa43.idx
jhinx@jhinx-virtual-machine:~/lab1$ cd Linux-0.11
jhinx@jhinx-virtual-machine:~/lab1/Linux-0.11$ ls
boot  hdc-0.11.img  init  lib  Makefile.header  README.md  tools
fs     include  kernel  Makefile  mm  readme.old
jhinx@jhinx-virtual-machine:~/lab1/Linux-0.11$ nano Makefile
jhinx@jhinx-virtual-machine:~/lab1/Linux-0.11$
```

随后，找到“CFLAGS +=”，在后面加上“-g -m32”，然后保存退出



```
GNU nano 6.2 Makefile *
# indicate the path of the calltree
CALLTREE=$(shell find tools/ -name "calltree" -perm 755 -type f)

# indicate the path of the bochs
#BOCHS=$(shell find tools/ -name "bochs" -perm 755 -type f)
BOCHS=bochs

#
# If you want the ram-disk device, define this to be the
# size in blocks.
#
RAMDISK = #-DRAMDISK=512

# This is a basic Makefile for setting the general configuration
include Makefile.header

LDFLAGS += -Ttext 0 -e startup_32
CFLAGS += $(RAMDISK) -include -g -m32
CPP += -include

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^_ Replace ^U Paste ^J Justify ^_ Go To Line
```

接着输入“make”编译内核

```

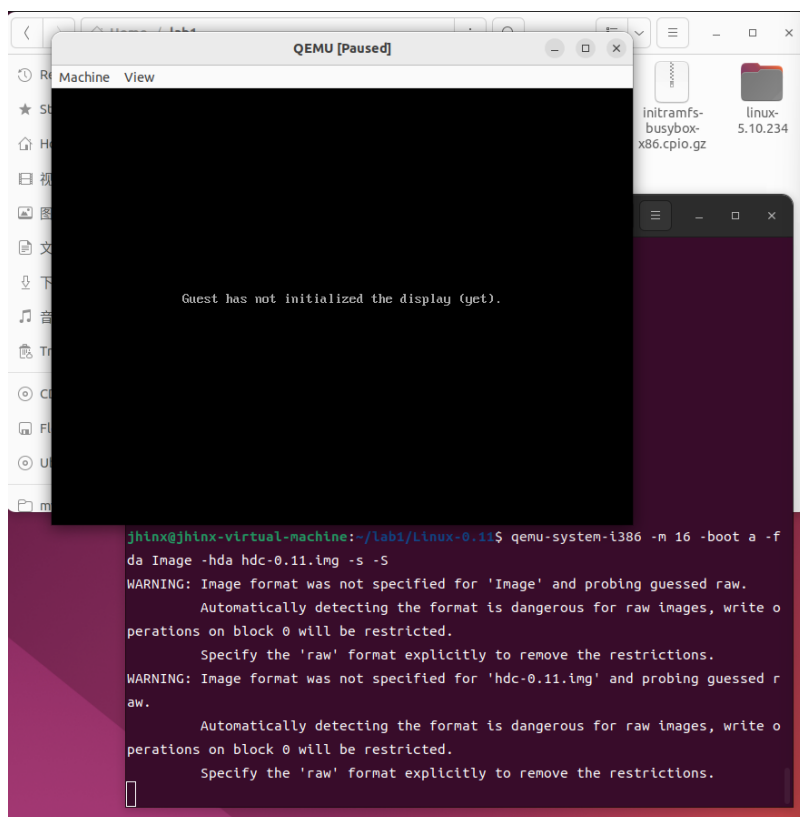
jhx@jhx-virtual-machine:~/lab1/Linux-0.11$ nano Makefile
jhx@jhx-virtual-machine:~/lab1/Linux-0.11$ make
make[1]: Entering directory '/home/jhx/lab1/Linux-0.11/boot'
make[1]: Leaving directory '/home/jhx/lab1/Linux-0.11/boot'
make[1]: Entering directory '/home/jhx/lab1/Linux-0.11/boot'
make[1]: Leaving directory '/home/jhx/lab1/Linux-0.11/boot'
make[1]: Entering directory '/home/jhx/lab1/Linux-0.11/boot'
make[1]: Leaving directory '/home/jhx/lab1/Linux-0.11/boot'
init/main.c:23:15: warning: type defaults to 'int' in declaration of 'fork' [-Wimplicit-int]
    23 | static inline fork(void) __attribute__((always_inline));
        |           ^~~~~
init/main.c:24:15: warning: type defaults to 'int' in declaration of 'pause' [-Wimplicit-int]
    24 | static inline pause(void) __attribute__((always_inline));
        |           ^~~~~
make[1]: Entering directory '/home/jhx/lab1/Linux-0.11/kernel'
In file included from traps.c:13:
../include/string.h:46:22: warning: inline function 'memchr' declared but never defined
    46 | extern inline void * memchr(const void * cs, char c, int count);
        |           ^~~~~

```

### 3.使用 qemu-system-i386 加载、启动内核:

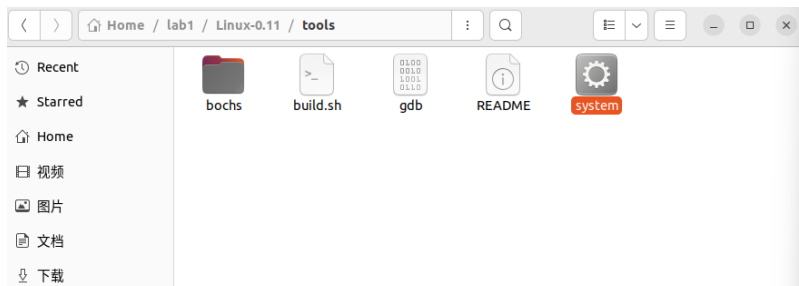
```
qemu-system-i386 -m 16 -boot a -fda Image -hda hdc-0.11.img -s -S
```

此时会打开一个黑色窗口，里面并不会有任何东西，这跟我们前面编译 Linux 5.10.19 是类似的，因为我们开启了 gdb 调试，而 qemu 在等待 gdb 输入的指令后才能继续执行。所以接下来我们需要使用 gdb 进行调试。



### 4.利用 gdb 进行调试远程调试:

(1) 找到 Linux 0.11 的符号表



## (2) 启动 gdb, 加载符号表, 远程连接 qemu 调试

```
cd ~/lab1
gdb
file Linux-0.11/tools/system (与前面 gdb 调试的代码有所不同, 因为
符号表的路径不一样)
target remote:1234
```

```
jhinx@jhinx-virtual-machine: $ cd ~/lab1
jhinx@jhinx-virtual-machine: ~/lab1$ gdb
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/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:
<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".
(gdb) file vmlinux
vmlinux: No such file or directory.
(gdb) file Linux-0.11/tools/system
Reading symbols from Linux-0.11/tools/system...
(gdb) target remote:1234
Remote debugging using :1234
0x0000ffff in do_execve (elf=0x0 <startup_32>, tnp=0,
    filename=0x0 <startup_32>, argv=0x0 <startup_32>, envp=0x0 <startup_32>)
    at exec.c:212
212         if (current->uid == inode->i_uid)
```

此时会打开一个黑色窗口, 里面并不会有任何东西, 这跟我们前面编译 Linux5.10.19 是类似的, 因为我们开启了 gdb 调试, 而 qemu 在等待 gdb 输入的指令后才能继续执行。所以接下来我们需要使用 gdb 进行调试。

## (3) 使用 gdb 调试:

```
directory /home/jhinx/lab1/Linux-0.11 (设置源码目录: directory
linux0.11 的源码路径)
set disassembly-flavor intel (设置汇编代码的形式 (intel))
break *0x7c00 (设置断点)
x/4xw 0x7DFE (观察地址内容)
x/4xw 0x7DFF
```

```
QEMU [Paused]
Machine View
SeaBIOS (version 1.15.0-1)
iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+00F8B590+00ECB590 CA00
Booting from Floppy...

jhinx@jhinx-virtual-machine: ~/lab1
Remote debugging using :1234
0x0000ffff in do_execve (eip=0x0 <startup_32>, tmp=0,
  filename=0x0 <startup_32>, argv=0x0 <startup_32>, envp=0x0 <startup_32>)
  at exec.c:212
212      if (current->euid == inode->i_uid)
(gdb) directory /home/jhinx/lab1/Linux-0.11
Source directories searched: /home/jhinx/lab1/Linux-0.11:$cd:$cd
(gdb) set disassembly-flavor intel
(gdb) break *0x7c00
Breakpoint 1 at 0x7c00: file traps.c, line 76.
(gdb) continue
Continuing.

Breakpoint 1, die (str=0x6f5e <sleep_on+24> "", esp_ptr=31941, nr=0) at traps.c:
76
76      printk("%p ", get_seg_long(0x17,i+(long *)esp[3]))
);
(gdb) x/4xw 0x7DFF
0x7dfe <do_int3+105>: 0x0000aa55 0x00000000 0x00000000 0x000000
00
(gdb) x/4xw 0x7DFF
0x7dff <do_int3+106>: 0x000000aa 0x00000000 0x00000000 0x000000
00
(gdb)
```

#### (4) 在 Linux-0.11 上挂载磁盘:

```
cd ~/lab1/Linux-0.11
fdisk hdc-0.11.img (用 fdisk 命令查看磁盘的分区情况以及文件类型(minix), 按照指引, 输入 p 查看)
cd ~/lab1
mkdir hdc (创建本地挂载目录)
df -h (显示磁盘空间)
sudo mount -t minix -o loop,offset=512 /home/jhinx/lab1/Linux-0.11/hdc-0.11.img /home/jhinx/lab1/hdc (第一个路径为硬盘的原路径, 第二个路径为镜像的目标路径)
df -h (在 ubuntu22.04 中执行, 查看是否出现挂载的 hdc 路径)
ll hdc (查看挂载后的 hdc 目录结构)
```

```
jhinx@jhinx-virtual-machine: ~/lab1/Linux-0.11
jhinx@jhinx-virtual-machine:~$ cd lab1/Linux-0.11
jhinx@jhinx-virtual-machine:~/lab1/Linux-0.11$ fdisk hdc-0.11.img

Welcome to fdisk (util-linux 2.37.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): p

Disk hdc-0.11.img: 59.55 MiB, 62447616 bytes, 121968 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x00000000

Device           Boot Start    End Sectors  Size Id Type
hdc-0.11.img1    *      1 120959   120959  59.1M 81 Minix / old Linux

Command (m for help): m
```



```

jhinx@jhinx-virtual-machine:~/lab1$ mkdir hdc
jhinx@jhinx-virtual-machine:~/lab1$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           482M  2.2M  480M   1% /run
/dev/sda3       39G   19G   19G  51% /
tmpfs           2.4G   0  2.4G   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           2.4G   0  2.4G   0% /run/qemu
/dev/sda2       512M  6.1M  506M   2% /boot/efi
tmpfs           482M  124K  482M   1% /run/user/1000
/dev/sr1        4.5G  4.5G   0 100% /media/jhinx/Ubuntu 22.04.5 LTS
amd64
/dev/sr0        157M  157M   0 100% /media/jhinx/CDROM

```

```

jhinx@jhinx-virtual-machine:~/桌面$ cd ~/lab1
jhinx@jhinx-virtual-machine:~/lab1$ sudo mount -t minix -o loop,offset=512 /home
/jhinx/lab1/Linux-0.11/hdc-0.11.img /home/jhinx/lab1/hdc
[sudo] password for jhinx:
jhinx@jhinx-virtual-machine:~/lab1$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           482M  2.2M  480M   1% /run
/dev/sda3       39G   19G   19G  51% /
tmpfs           2.4G   0  2.4G   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           2.4G   0  2.4G   0% /run/qemu
/dev/sda2       512M  6.1M  506M   2% /boot/efi
tmpfs           482M  124K  482M   1% /run/user/1000
/dev/sr1        4.5G  4.5G   0 100% /media/jhinx/Ubuntu 22.04.5 LTS amd64
/dev/sr0        157M  157M   0 100% /media/jhinx/CDROM
/dev/loop15     58M   13M   46M  23% /home/jhinx/lab1/hdc

```

```

jhinx@jhinx-virtual-machine:~/lab1$ ll hdc
total 13
drwxr-xr-x 10 root root 176 3月 22 2004 ./
drwxrwxr-x 7 jhinx jhinx 4096 3月 2 12:20 ../
drwxr-xr-x 2 root root 912 3月 22 2004 bin/
drwxr-xr-x 2 root root 336 3月 22 2004 dev/
drwxr-xr-x 2 root root 224 3月 22 2004 etc/
drwxr-xr-x 8 root root 128 3月 22 2004 image/
drwxr-xr-x 2 root root 32 3月 22 2004 mnt/
drwxr-xr-x 2 root root 64 3月 22 2004 tmp/
drwxr-xr-x 10 root root 192 3月 30 2004 usr/
drwxr-xr-x 2 root root 32 3月 22 2004 var/

```

（以上为各个指令的输入输出示例）

#### （5）在 hdc 中创建文件：

```

cd hdc/usr
sudo touch hello.txt
sudo vim hello.txt （编辑文件）

```

本人在执行上述代码时，出现了一些问题，输入 `sudo vim hello.txt` 时出现了报错，经检查是因为虚拟机环境没有安装 `vim` 所导致，在安装 `vim` 后问题得到了解决

```

jhinx@jhinx-virtual-machine:~/lab1$ cd hdc/usr
jhinx@jhinx-virtual-machine:~/lab1/hdc/usr$ sudo touch hello.txt
jhinx@jhinx-virtual-machine:~/lab1/hdc/usr$ sudo vim hello.txt
sudo: vim: command not found
jhinx@jhinx-virtual-machine:~/lab1/hdc/usr$ sudo vim hello.txt
sudo: vim: command not found
jhinx@jhinx-virtual-machine:~/lab1/hdc/usr$ sudo apt update
命中:1 http://mirrors.tuna.tsinghua.edu.cn/ubuntu jammy InRelease
命中:2 http://mirrors.tuna.tsinghua.edu.cn/ubuntu jammy-updates InRelease
命中:3 http://mirrors.tuna.tsinghua.edu.cn/ubuntu jammy-backports InRelease
获取:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
获取:5 http://security.ubuntu.com/ubuntu jammy-security/main amd64 DEP-11 Metadata [43.1 kB]
获取:6 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 DEP-11 Metadata [208 B]
获取:7 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 DEP-11 Metadata [125 kB]
获取:8 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 DEP-11 Metadata [208 B]
已下载 298 kB，耗时 2秒 (127 kB/s)
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

```

```

jhinx@jhinx-virtual-machine:~/lab1/hdc/usr$ sudo apt install vim
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
将会同时安装下列软件：
  vim-runtime
建议安装：
  ctags vim-doc vim-scripts
下列【新】软件包将被安装：
  vim vim-runtime
升级了 0 个软件包，新安装了 2 个软件包，要卸载 0 个软件包，有 66 个软件包未被升级。
需要下载 8,565 kB 的归档。
解压缩后会消耗 37.6 MB 的额外空间。
您希望继续执行吗？ [Y/n] Y
获取:1 http://mirrors.tuna.tsinghua.edu.cn/ubuntu jammy-updates/main amd64 vim-runtime all 2:8.2.3995-1ubuntu2.23 [6,833 kB]
获取:2 http://mirrors.tuna.tsinghua.edu.cn/ubuntu jammy-updates/main amd64 vim-a

```

```
jhinx@jhinx-virtual-machine: ~/lab1/hdc/usr
正在选中未选择的软件包 vim。
准备解压 .../vim_2%3a8.2.3995-1ubuntu2.23_amd64.deb ...
正在解压 vim (2:8.2.3995-1ubuntu2.23) ...
正在设置 vim-runtime (2:8.2.3995-1ubuntu2.23) ...
正在设置 vim (2:8.2.3995-1ubuntu2.23) ...
update-alternatives: 使用 /usr/bin/vim.basic 来在自动模式中提供 /usr/bin/vim (vim)
update-alternatives: 使用 /usr/bin/vim.basic 来在自动模式中提供 /usr/bin/vimdiff (vimdiff)
update-alternatives: 使用 /usr/bin/vim.basic 来在自动模式中提供 /usr/bin/rvim (rvim)
update-alternatives: 使用 /usr/bin/vim.basic 来在自动模式中提供 /usr/bin/rview (rview)
update-alternatives: 使用 /usr/bin/vim.basic 来在自动模式中提供 /usr/bin/vi (vi)
update-alternatives: 使用 /usr/bin/vim.basic 来在自动模式中提供 /usr/bin/view (view)
update-alternatives: 使用 /usr/bin/vim.basic 来在自动模式中提供 /usr/bin/ex (ex)
正在处理用于 man-db (2.10.2-1) 的触发器 ...
jhinx@jhinx-virtual-machine:~/lab1/hdc/usr$ sudo vim hello.txt
```

卸载文件系统 hdc 的指令如下：

```
sudo umount /dev/loop (查看具体的 loop 设备)
df -h (查看是否已经卸载)
```

注意，此处 loop 后需要输入具体的数字，否则指令无效。根据前面步骤中 df -h 指令的输出可以看到，hdc 下载到了 loop15 中，因此此处的代码也应为 loop15

另外，在执行 umount 时，应该使用 cd 指令使命令行跳转到别处，否则将会出现类似我们平时遇到的无法卸载正在打开的文件夹这一问题

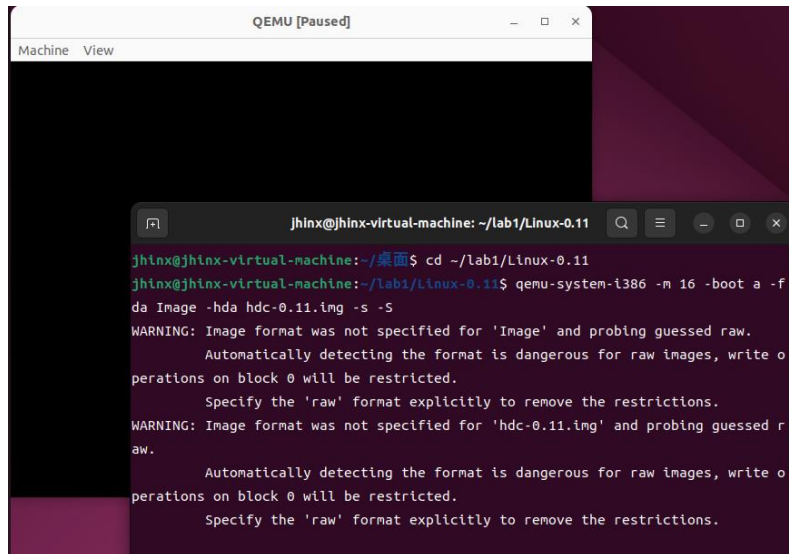
```
jhinx@jhinx-virtual-machine:~/lab1/hdc/usr$ sudo vim hello.txt
jhinx@jhinx-virtual-machine:~/lab1/hdc/usr$ sudo umount /dev/loop
umount: /dev/loop: no mount point specified.
jhinx@jhinx-virtual-machine:~/lab1/hdc/usr$ sudo umount /dev/loop15
umount: /home/jhinx/lab1/hdc: target is busy.
```

```
jhinx@jhinx-virtual-machine: ~/桌面
jhinx@jhinx-virtual-machine:~/桌面$ sudo umount /dev/loop15
[sudo] password for jhinx:
jhinx@jhinx-virtual-machine:~/桌面$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           482M  2.2M  480M   1% /run
/dev/sda3        39G   19G   19G  51% /
tmpfs           2.4G   0  2.4G   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           2.4G   0  2.4G   0% /run/qemu
/dev/sda2       512M  6.1M  506M   2% /boot/efi
tmpfs           482M  128K  482M   1% /run/user/1000
/dev/sr1         4.5G  4.5G   0 100% /media/jhinx/Ubuntu 22.04.5 LTS amd64
/dev/sr0        157M  157M   0 100% /media/jhinx/CDROM
```

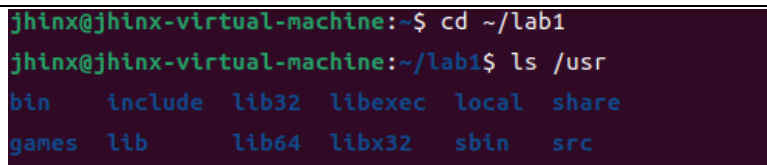
由 df -h 的输出可以看到，此时 hdc 已经被卸载

重新用 qemu 启动 linux 0.11，发现依然可以正常启动

```
cd ~/lab1/Linux-0.11
qemu-system-i386 -m 16 -boot a -fda Image -hda hdc-0.11.img -s -S
```



```
cd ~/lab1
ls /usr(观察/usr 目录下是否有 hello.txt 文件)
```

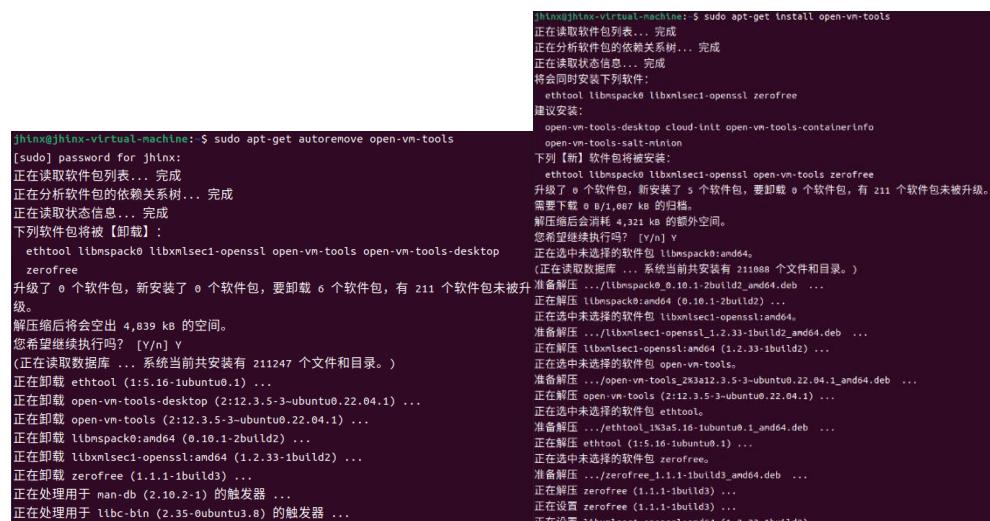


经观察，hello.txt 文件已经被删除，证明文件系统 hdc 卸载成功。

## Section 5 实验总结与心得体会

1.在实验过程中，发现如果每一行代码都手敲的话会特别麻烦，因此上网搜索发现有工具可以实现 Linux 和 Windows 的复制粘贴互通，实现方法如下：

```
sudo apt-get autoremove open-vm-tools
sudo apt-get install open-vm-tools
sudo apt-get install open-vm-tools-desktop
```



```

jhinx@jhinx-virtual-machine:~$ sudo apt-get install open-vn-tools-desktop
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
下列【新】软件包将被安装：
  open-vn-tools-desktop
升级了 0 个软件包，新安装了 1 个软件包，要卸载 0 个软件包，有 211 个软件包未被升级。
需要下载 0 B/139 kB 的归档。
解压缩后会消耗 518 kB 的额外空间。
正在选中未选择的软件包 open-vn-tools-desktop。
(正在读取数据库 ... 系统当前共安装有 211230 个文件和目录。)
准备解压 .../open-vn-tools-desktop_2%3a12.3.5-3-ubuntu0.22.04.1_amd64.deb ...
正在解压 open-vn-tools-desktop (2:12.3.5-3-ubuntu0.22.04.1) ...
正在设置 open-vn-tools-desktop (2:12.3.5-3-ubuntu0.22.04.1) ...
正在处理用于 man-db (2.10.2-1) 的触发器 ...

```

网址如下：

[https://blog.csdn.net/weixin\\_74149432/article/details/144148245?ops\\_request\\_misc=%257B%2522request%255Fid%2522%253A%25226ce1dae763e06018c4cd71aa3b091805%2522%252C%2522scm%2522%253A%25220140713.130102334..%2522%257D&request\\_id=6ce1dae763e06018c4cd71aa3b091805&biz\\_id=0&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~sobaiduend~default-2-144148245-null-null.142^v101^pc\\_search\\_result\\_base3&utm\\_term=vmware%E5%A4%8D%E5%88%B6%E7%B2%98%E8%B4%B4&spm=1018.2226.3001.4187](https://blog.csdn.net/weixin_74149432/article/details/144148245?ops_request_misc=%257B%2522request%255Fid%2522%253A%25226ce1dae763e06018c4cd71aa3b091805%2522%252C%2522scm%2522%253A%25220140713.130102334..%2522%257D&request_id=6ce1dae763e06018c4cd71aa3b091805&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-144148245-null-null.142^v101^pc_search_result_base3&utm_term=vmware%E5%A4%8D%E5%88%B6%E7%B2%98%E8%B4%B4&spm=1018.2226.3001.4187)

2.实验过程中遇到的问题，为了方便展示问题以及解决方法之间的区别，已经直接安插在了实验步骤中，如有需要请在实验步骤中查看。

3.在本次实验中，因为初次接触操作系统，因此对指令的作用等了解甚微。本人借助了 ChatGPT 来帮助自己理解每一个指令的作用，以及各个指令中参数的格式和使用方法，但并未通过 ChatGPT 直接代替人脑的思考得到实验结果，只是借助 gpt 来增进了对操作系统的理解。

4.在实验的截图中，均为带有“jhinx”字样的图片，jhinx 为本人虚拟机的用户名，也是本人的微信名，本人以此来作为标识，证明实验为本人亲手所做而非抄袭。

## Section 6 附录：代码清单

本节为可选章节，如果无法在压缩包中提供代码的.c/.cpp 文件或.asm 文件，请务必在这里提供完整的实验代码。格式要求和第 3 节相同。

【实验任务 0】完整的汇编程序如下：

```

sudo apt install binutils
sudo apt install gcc

```

```
gcc -v
sudo apt install nasm
sudo apt install qemu
sudo apt install cmake
sudo apt install libncurses5-dev
sudo apt install bison
sudo apt install flex
sudo apt install libssl-dev
sudo apt install libc6-dev-i386
```

【实验任务 1】完整的汇编程序如下：

```
mkdir ~/lab1
cd ~/lab1
xz -d linux-5.10.234.tar.xz
tar -xvf linux-5.10.234.tar
cd linux-5.10.234
make i386_defconfig
make menuconfig
make -j8
```

【实验任务 2】完整的汇编程序如下：

```
cd ~/lab1
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -s -
S -append "console=ttyS0" -nographic
cd ~/lab1
gdb
file linux-5.10.234/vmlinux
target remote:1234
break start_kernel
c
```

【实验任务 3】完整的汇编程序如下：

```
cd ~/lab1
gcc -o helloworld -m32 -static helloworld.c
echo helloworld | cpio -o --format=newc > hwinitramfs
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -
initrd hwinitramfs -s -S -append "console=ttyS0 rdinit=helloworld" -
nographic
cd ~/lab1
gdb
file linux-5.10.234/vmlinux
target remote:1234
break start_kernel
c
```

【实验任务 4】完整的汇编程序如下：

```
cd ~/lab1
tar -xjf busybox-1.33.0.tar.bz2
make defconfig
make menuconfig
make -j8
make install
cd ~/lab1
mkdir mybusybox
mkdir -pv mybusybox/{bin,sbin,etc,proc,sys,usr/{bin,sbin}}
cp -av busybox-1.33.0/_install/* mybusybox/
cd mybusybox
#!/bin/sh
mount -t proc none /proc
mount -t sysfs none /sys
echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime) seconds\n"
exec /bin/sh
chmod u+x init (为 init 加上执行权限)
find . -print0 | cpio --null -ov --format=newc | gzip -9 >
~/lab1/initramfs-busybox-x86.cpio.gz (将 x86-busybox 下面的内容打包归档
成 cpio 文件)
cd ~/lab1
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -
initrd initramfs-busybox-x86.cpio.gz -nographic -append
"console=ttyS0" -m size=2048 (加载 busybox)
ls
```

【实验任务 5】完整的汇编程序如下：

```
cd ~/lab1
tar -xzvf Linux-0.11-lab1.tar.gz
cd Linux-0.11
nano Makefile
qemu-system-i386 -m 16 -boot a -fda Image -hda hdc-0.11.img -s -
S
cd ~/lab1
gdb
file Linux-0.11/tools/system
target remote:1234
directory /home/jhinx/lab1/Linux-0.11 (设置源码目录: directory
linux0.11 的源码路径)
set disassembly-flavor intel (设置汇编代码的形式 (intel))
break *0x7c00 (设置断点)
x/4xw 0x7DFE (观察地址内容)
x/4xw 0x7DFF
```

```
cd ~/lab1/Linux-0.11
fdisk hdc-0.11.img
cd ~/lab1
mkdir hdc
df -h
sudo mount -t minix -o loop,offset=512 /home/jhinx/lab1/Linux-
0.11/hdc-0.11.img /home/jhinx/lab1/hdc
df -h
ll hdc
cd hdc/usr
sudo touch hello.txt
sudo vim hello.txt
sudo umount /dev/loop
df -h
cd ~/lab1/Linux-0.11
qemu-system-i386 -m 16 -boot a -fda Image -hda hdc-0.11.img -s -
S
cd ~/lab1
ls /usr
```