



中山大學
SUN YAT-SEN UNIVERSITY

操作系统实验 1

编译内核/利用已有内核构建 OS

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：_____赵施琦_____

学生学号：_____23336324_____

实验地点：_____实验楼 B203_____

实验成绩：_____

报告时间：_____2025 年 2 月 28 日_____

Section 1 实验概述

- **实验任务 1：环境配置。**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。
- **实验任务 2：编译 Linux 内核、Qemu 启动内核并开启远程调试。**熟悉现有 Linux 内核的编译过程和启动过程，并在自行编译内核的基础上构建简单应用并启动。熟悉编译环境、相关工具集，并能够实现内核远程调试。
- **实验任务 3：制作 Initramfs。**熟悉制作 initramfs 的方法。编写简单应用程序随内核启动运行。
- **实验任务 4：编译并启动 Busybox。**会利用精简的 Busybox 工具集构建简单的 OS，熟悉现代操作系统的构建过程。
- **实验任务 5：完成 Linux 0.11 内核的编译。**编译完成后利用 gdb 进行远程调试，并查看磁盘的分区情况以及文件类型。创建本地挂载目录并创建文件，随后卸载后查看创建的文件是否还存在。

Section 2 实验步骤与实验结果

----- 实验任务 1 -----

1.任务要求：选择在 Ubuntu 系统上，或者在 Windows 系统上利用虚拟机进行实验。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。验证环境配置正确并确认配置 gcc、g++、nasm 等的版本号正确，为下一步实验做准备。

2.思路分析：由于在做实验的两节课内不足够完成实验内容，所以考虑在我的 Windows 笔记本电脑上使用 VMware 虚拟机运行 Ubuntu18 系统。选择 VMware 是因为该软件相比起 Virtual Box 更好使用，选择 Ubuntu18 系统是因为先前有在 Ubuntu18 上做项目的经验。搭建实验环境只需要使用 apt 就可方便快速的搭建。注意搭建之前可以进行换源从而加快配置速度。

3.实验步骤：配置虚拟机用户名为“yiqi”，用于排除抄袭嫌疑。随后开始配置实验环境。首先进行换源，换源可以跳过命令行进行手动换源，但是在换源之前要记得给虚拟机联网。之后可使用下列命令配置实验环境。

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

```
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
sudo apt install gcc-multilib
sudo apt install g++-multilib
```

安装完毕后可以使用下列命令查看版本号以验证安装成功

安装的名称 `--version`

此处还有一点需要注意，Ubuntu18 安装的 nasm 版本号为 2.13，而我们后续需要的 nasm 版本号为 2.15。在实验指导中找到 `env/nasm-2.15.05.tar.x` 后编译安装 nasm2.15。vscode 直接在软件里面手动进行下载，非常方便。

至此我们的实验环境配置就已经完成啦。

----- 实验任务 2 -----

1.任务要求：下载 linux 内核并编译，熟悉现有 Linux 内核的编译过程和启动过程，并在自行编译内核的基础上构建简单应用并启动。熟悉编译环境、相关工具集，并能够实现内核远程调试。使用 qemu 启动内核并使用 gdb 进行调试。

2.思路分析：注意内核要编译为 i386 32 位版本，编译完成后检查 Linux 压缩镜像 `linux-5.10.19/arch/x86/boot/bzImage` 和符号表 `linux-5.10.19/vmlinux` 是否已经生成。启动 qemu 后在另一个新的终端运行 gdb，注意不要关闭原来的终端，在 gdb 中设置断点并运行查看效果。

3.实验步骤：首先下载 linux-5.10 的内核并编译为 i386 32 位版本

```
wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.10.tar.xz
make i386_defconfig

make menuconfig

make -j8
```

观察发现 Linux 压缩镜像 `linux-5.10.19/arch/x86/boot/bzImage` 和符号表 `linux-5.10.19/vmlinux` 已经生成，说明编译内核步骤正确执行。然后使用 qemu

启动内核并开启远程调试。

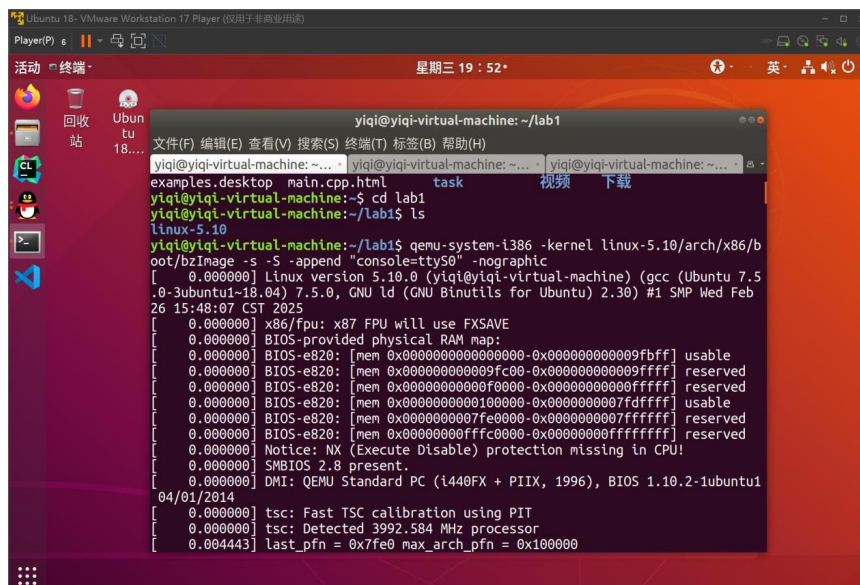
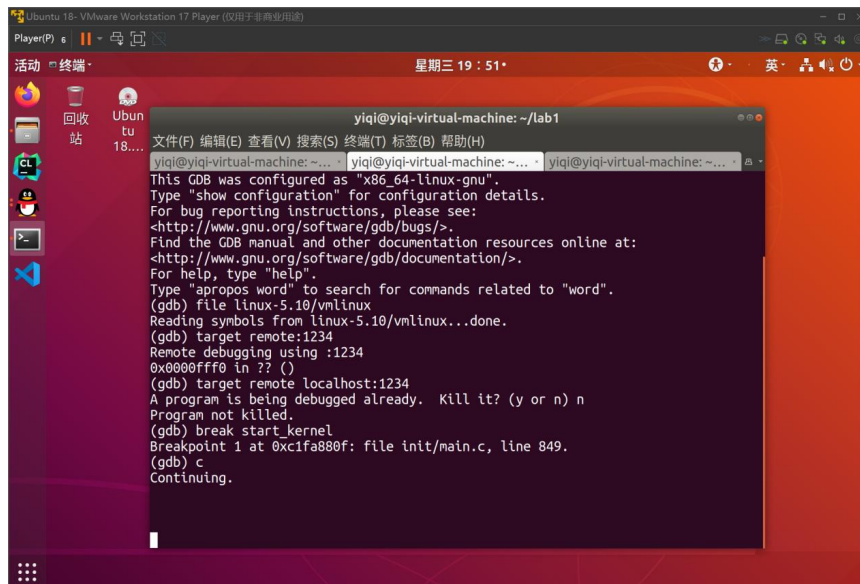
```
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -s -S -append  
"console=ttyS0" -nographic
```

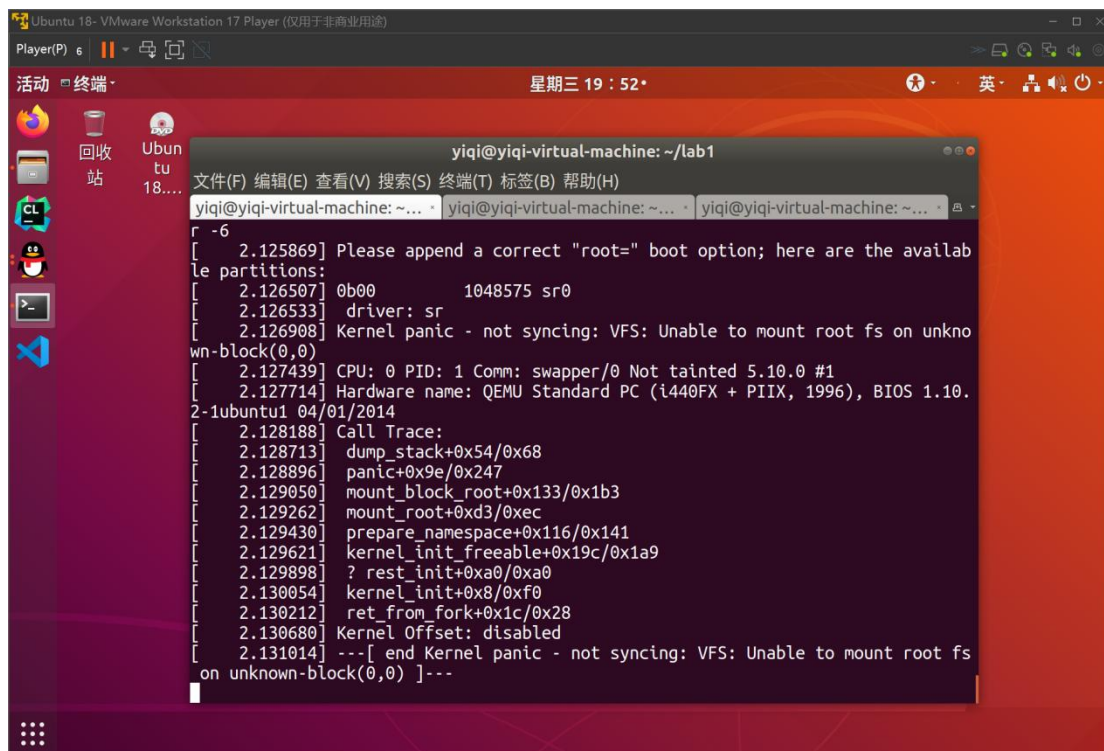
另开一个终端输入以下命令

```
gdb  
  
file linux-5.10.19/vmlinux  
  
target remote:1234  
  
break start_kernel  
  
c
```

返回 qemu 观察到系统正常运行

4.实验结果展示：通过执行前述代码，可得下图结果。





在 qemu 虚拟机里运行的 Linux 系统能成功启动，并且最终以 Kernel panic 宣告结束。

----- 实验任务 3 -----

1.任务要求：熟悉制作 initramfs 的方法。编写简单应用程序随内核启动运行。观察并确认我们编写的程序正确的随内核启动而运行。

2.思路分析：首先创建一个可执行文件并使用编译器编译为 32 位可执行文件。随后用 cpio 打包 initramfs 并启动内核，加载 initramfs。通过 gdb 调试后即可看到伴随内核运行我们程序执行的结果。注意输出应该在 qemu 命令执行的终端中显示。

3.实验步骤：首先创建一个.c 文件并在其中编写如下代码

```
#include <stdio.h>

void main(){

    printf("lab1: good game\n");

    fflush(stdout);

    while(1);

}
```

为将上面代码编译成 32 位可执行文件，我们使用 gcc 编译加上 -m32 确保编译为

32 为可执行文件。随后用 cpio 打包 initramfs，启动内核，并加载 initramfs。

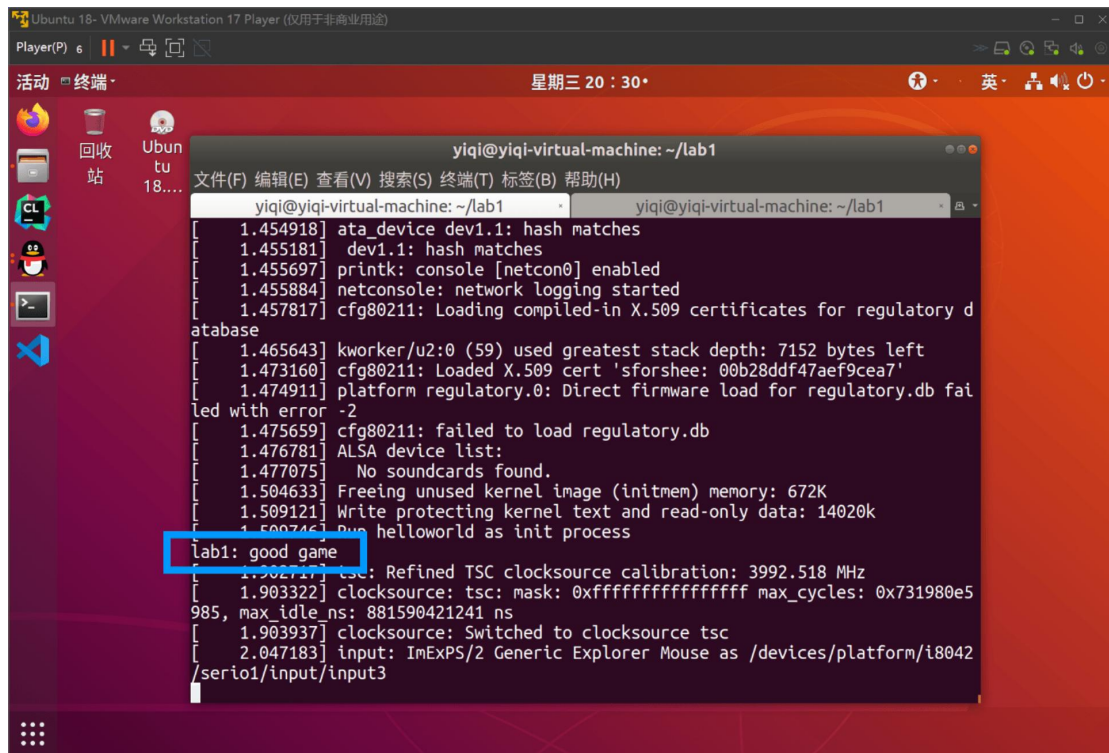
```
gcc -o helloworld -m32 -static helloworld.c

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
```

重复上述的 gdb 调试过程，观察终端发现终端成功打印出了“lab1: good game”的字样。

4. 实验结果展示：通过执行前述代码，可得下图结果。



从上图中可以看出终端成功打印出了“lab1: good game”的字样，这证明我们的 initramfs 制作成功。

----- 实验任务 4 -----

1.任务要求：会利用精简的 Busybox 工具集构建简单的 OS，熟悉现代操作系统的构建过程。最后使用命令行命令验证我们构建的 OS 可以正常运行。

2.思路分析：从课程网站处下载下载 Busybox，然后解压并编译。要记得为我们的 OS 制作 initramfs，不要忘了给文件加上执行权限。全部执行完成后便可以加载 busybox，使用命令行以确认我们的 OS 可以正常运行。

3.实验步骤：从课程网站处下载下载 Busybox，然后解压并编译。

```
tar -xf busybox-1.33.0.tar.bz2
```

```
make defconfig  
make menuconfig
```

在编译完后的图形界面内设置() Additional CFLAGS 和() Additional LDFLAGS 为(-m32 -march=i386) Additional CFLAGS、(-m32) Additional LDFLAGS。保存退出，然后编译。

```
make -j8  
make install
```

然后制作 Initramfs，在命令行输入以下命令

```
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
```

由于 initramfs 需要一个 init 程序,我们再写一个简单的 shell 脚本作为 init, 用 gedit 打开文件 init, 输入以下内容

```
#!/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
```

加上执行权限,并将 x86-busybox 下面的内容打包归档成 cpio 文件,以供 Linux 内核做 initramfs 启动执行。

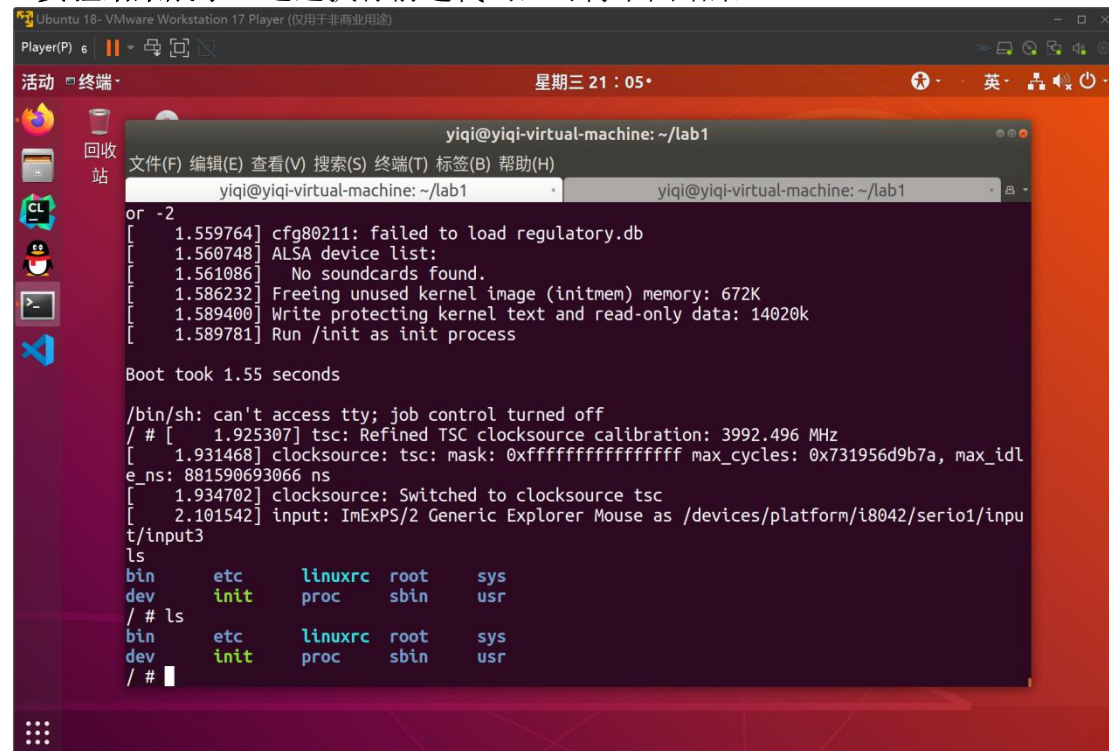
```
chmod u+x init  
find . -print0 | cpio --null -ov --format=newc | gzip -9 >  
~/lab1/initramfs-busybox-x86.cpio.gz
```

最后加载 busybox

```
qemu-system-i386 -kernel linux-5.10/arch/x86/boot/bzImage -initrd  
initramfs-busybox-x86.cpio.gz -nographic -append "console=ttyS0"
```

执行完命令后发现我们进入了一个酷似命令行的终端,在输入命令后可看到该终端正常反应。

4.实验结果展示：通过执行前述代码，可得下图结果。



```
yi qi@yi qi-virtual-machine: ~/lab1
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 标签(B) 帮助(H)
yi qi@yi qi-virtual-machine: ~/lab1
or -2
[ 1.559764] cfg80211: failed to load regulatory.db
[ 1.560748] ALSA device list:
[ 1.561086] No soundcards found.
[ 1.586232] Freeing unused kernel image (initmem) memory: 672K
[ 1.589400] Write protecting kernel text and read-only data: 14020k
[ 1.589781] Run /init as init process

Boot took 1.55 seconds

/bin/sh: can't access tty; job control turned off
/ # [ 1.925307] tsc: Refined TSC clocksource calibration: 3992.496 MHz
[ 1.931468] clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x731956d9b7a, max_idl
e_ns: 881590693066 ns
[ 1.934702] clocksource: Switched to clocksource tsc
[ 2.101542] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042/serio1/inpu
t/input3
ls
bin      etc      linuxrc  root     sys
dev      init     proc     sbin     usr
/ # ls
bin      etc      linuxrc  root     sys
dev      init     proc     sbin     usr
/ #
```

在上图中我们执行了 ls 命令，可正常查看到当前的文件夹。

----- 实验任务 5 -----

1.任务要求：完成 Linux 0.11 内核的编译。编译完成后进入 Linux 0.11 操作系统，熟悉该操作系统的命令，查看目录结构，运行程序等。随后利用 gdb 进行远程调试，并查看磁盘的分区情况以及文件类型，并创建本地挂载目录。创建本地挂载目录并创建文件，随后卸载后查看创建的文件是否还存在。

2.思路分析：找到正确的 linux-0.11 源码并编译为 32 位，编译完成后进入该操作系统运行一些基本命令和程序用来熟悉该操作系统。其中注意需要修改 Makefile 的内容使得操作系统可以调试并且被编译为 32 位。启动内核时先不要加上“-s -S”不然系统会停止等待调试连接，先熟悉命令后再进行调试连接。远程调试时注意符号表位于 system 内，注意设置源码目录和汇编代码的形式，在关键位置添加断点后即可查看寄存器内的内容。查看磁盘的分区情况以及文件类型时记得加上“-l”查看。挂载完磁盘镜像后需要查看挂载的实际设备名称，保存编写的文件。卸载文件系统后还要检查是否已经成功卸载，并查看编写的文件是否出现在/usr 目录下。

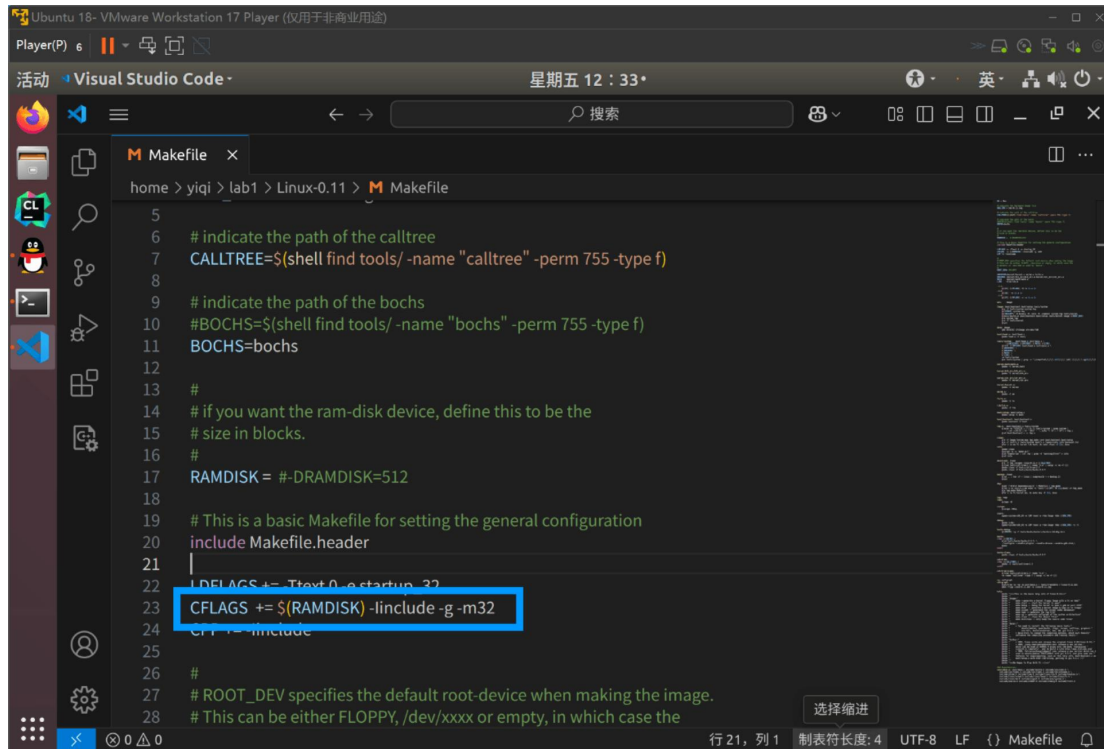
3.实验步骤：由于课程提供网址内的 linux-0.11 不正确，故从课程群内提供的源码中直接解压 linux-0.11 内核。


```
tar -xzvf linux-0.11.tar.gz
cd Linux-0.11
```

查看并修改 Makefile 文件确保编译时使用 32 位架构并且启用调试信息。

code Makefile

在 vscode 中修改 CFLAGS（编译器标志），使其包含 -m32 和 -g:



保存并退出，最后使用 make 编译。编译成功后使用 qemu-i386-system 加载、启动内核，运行简单的命令熟悉操作系统。

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

运行完命令后开始使用 gdb 进行远程调试

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

另一个终端:

gdb lab1/Linux-0.11/tools/system

directory /path/to/linux0.11

set disassembly-flavor intel

target remote :1234

break *0x7C00

break main

continue

x /2xw 0x7DFE
```

```
x /2xw 0x7DFF
continue
x /2xw 0x7DFE
x /2xw 0x7DFF
```

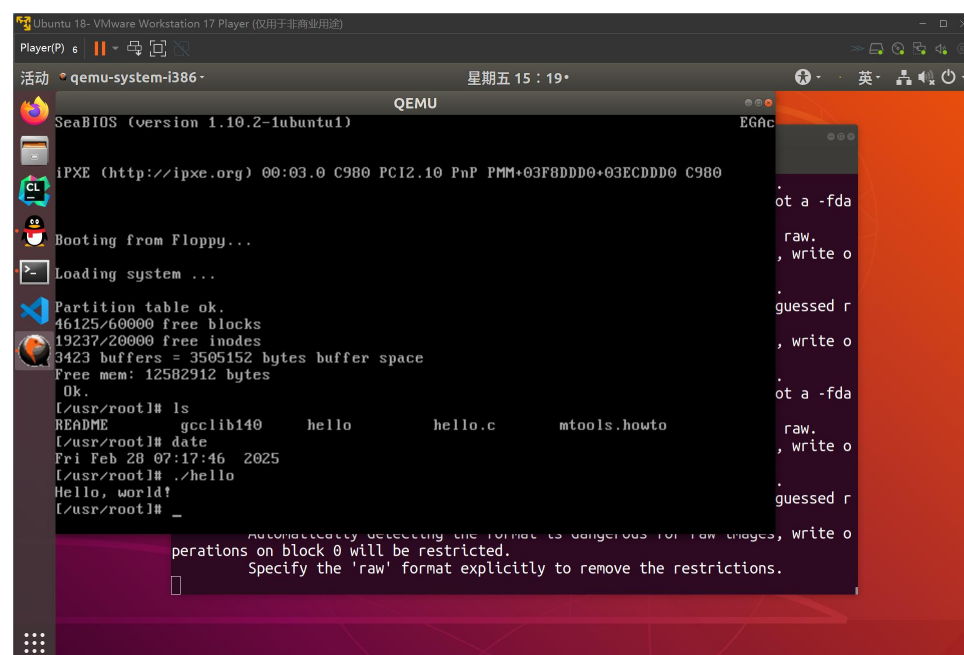
找到hdc-0.11.img 硬盘镜像文件,用fdisk 命令查看磁盘的分区情况以及文件类型(minix),然后创建本地挂载目录。挂载 linux 0.11 硬盘镜像,查看是否挂载成功,查看挂载后的 hdc 目录结构并在 hdc 中创建文件。然后卸载文件系统 hdc 重新用 qemu 启动 linux 0.11 观察/usr 目录下是否有 hello.txt 文件。

```
fdisk -l hdc-0.11.img
mkdir hdc
df -h
ll hdc
sudo touch hello.txt
sudo vim hello.txt
df -h
qemu-system-i386 -m 16 -boot a -fda Image -hda hdc-0.11.img
```

通过观察发现/usr 目录下已有 hello.txt 文件且文件内容与挂载时编写的内容一致。

4.实验结果展示：通过执行前述代码，可得下图结果。

在 linux-0.11 中执行基本指令：



使用 gdb 进行调式，查看寄存器内容：

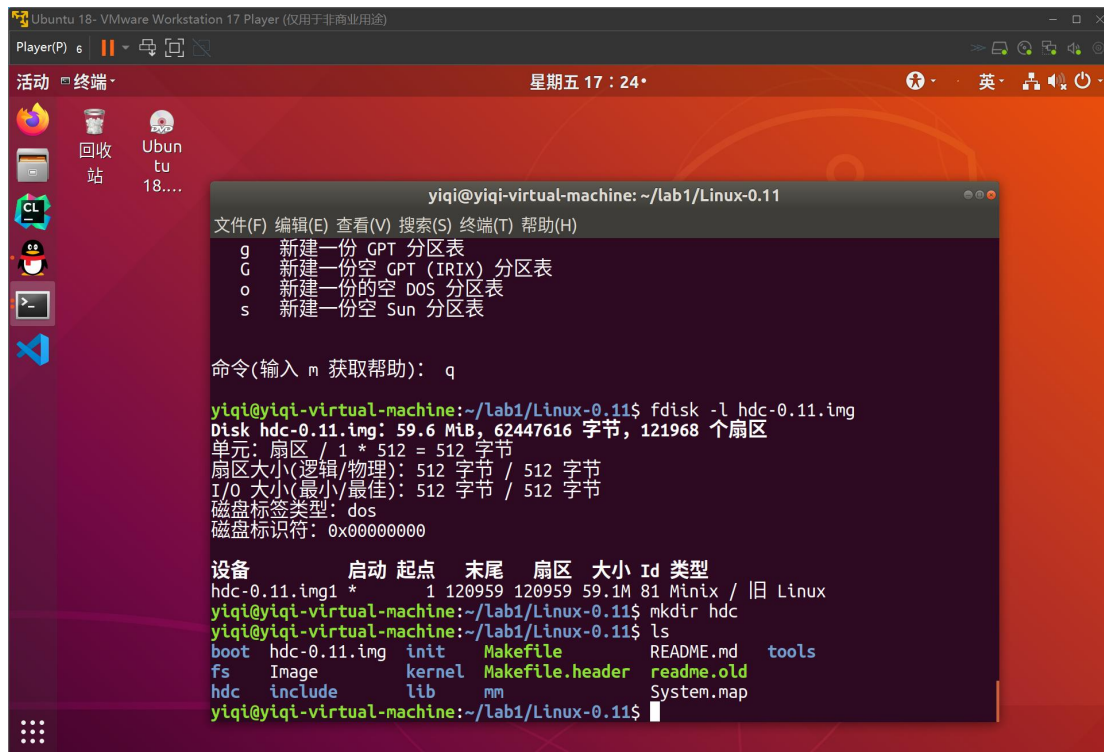
```
Ubuntu 18- VMware Workstation 17 Player (仅用于非商业用途)
Player(P) 6
活动 终端
星期五 17 : 17
yiqi@yiqi-virtual-machine: ~

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
Function "0x7c00" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) break *0x7c00
Breakpoint 1 at 0x7c00: file traps.c, line 76.
(gdb) set disassembly-flavor intel
(gdb) directory lab1/Linux-0.11
Source directories searched: /home/yiqi/lab1/Linux-0.11:$cd:$cd
(gdb) break *0x7c00
Note: breakpoint 1 also set at pc 0x7c00.
Breakpoint 2 at 0x7c00: file traps.c, line 76.
(gdb) break main
Breakpoint 3 at 0x6773: file init/main.c, line 107.
(gdb) continue
Continuing.

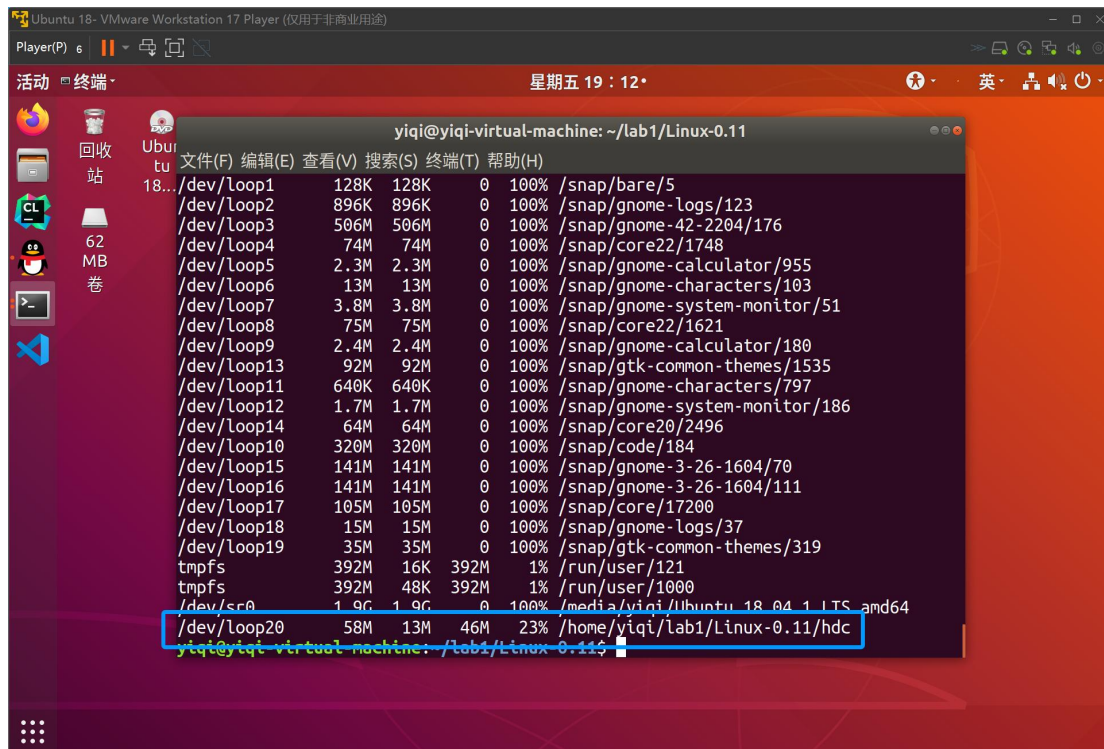
Breakpoint 1, 0x00007c00 in die (str=0x6f62 <sleep_on+45> "", esp_ptr=35543,
nr=0) at traps.c:76
76      printk("%p ",get_seg_long(0x17,i+(long *)esp[3]));
(gdb) x/16x 0x7DFF
0x7dfe <do_int3+123>: 0x0000aa55      0x00000000      0x00000000      0x00000000
0x7e0e <do_int3+139>: 0x00000000      0x00000000      0x00000000      0x00000000
0x7e1e <do_int3+155>: 0x00000000      0x00000000      0x00000000      0x00000000
0x7e2e <do_int3+171>: 0x00000000      0x00000000      0x00000000      0x00000000
(gdb) x/16x 0x7DFF
0x7dff <do_int3+124>: 0x000000aa      0x00000000      0x00000000      0x00000000
0x7e0f <do_int3+140>: 0x00000000      0x00000000      0x00000000      0x00000000
0x7e1f <do_int3+156>: 0x00000000      0x00000000      0x00000000      0x00000000
0x7e2f <do_int3+172>: 0x00000000      0x00000000      0x00000000      0x00000000
(gdb)

Breakpoint 3, main () at init/main.c:107
107 {
/* The startup routine assumes (well, ...) this */
(gdb) x/16x 0x7DFF
0x7dff <do_int3+124>: 0x00001007      0x8b20c483      0x83202444      0x088b08c0
0x7e0f <do_int3+140>: 0x2024448b      0x8b04c083      0x24448b10      0x51008b20
0x7e1f <do_int3+156>: 0x838d5052      0xffff82a8      0x0fdde850      0xc4830000
0x7e2f <do_int3+172>: 0xc4839010      0x83c35b18      0x8ce80cec      0x0500002e
(gdb) x/16x 0x7DFF
0x7dfe <do_int3+123>: 0x001007e8      0x20c48300      0x2024448b      0x8b08c083
0x7e0e <do_int3+139>: 0x24448b08      0x04c08320      0x448b108b      0x008b2024
0x7e1e <do_int3+155>: 0x8d505251      0xff82a883      0xdde850ff      0x8300000f
0x7e2e <do_int3+171>: 0x839010c4      0xc35b18c4      0xe80cec83      0x00002e8c
(gdb) continue
Continuing.
```

使用 fdisk 命令查看磁盘的分区情况以及文件类型(minix):

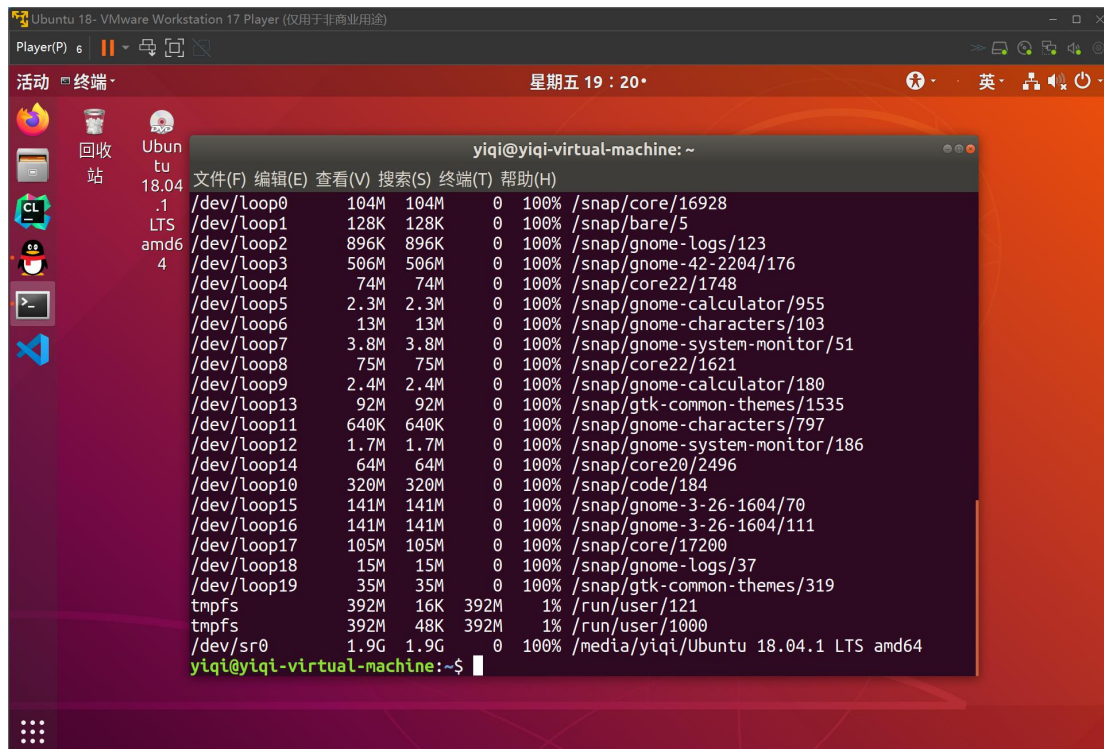


确认挂载成功:

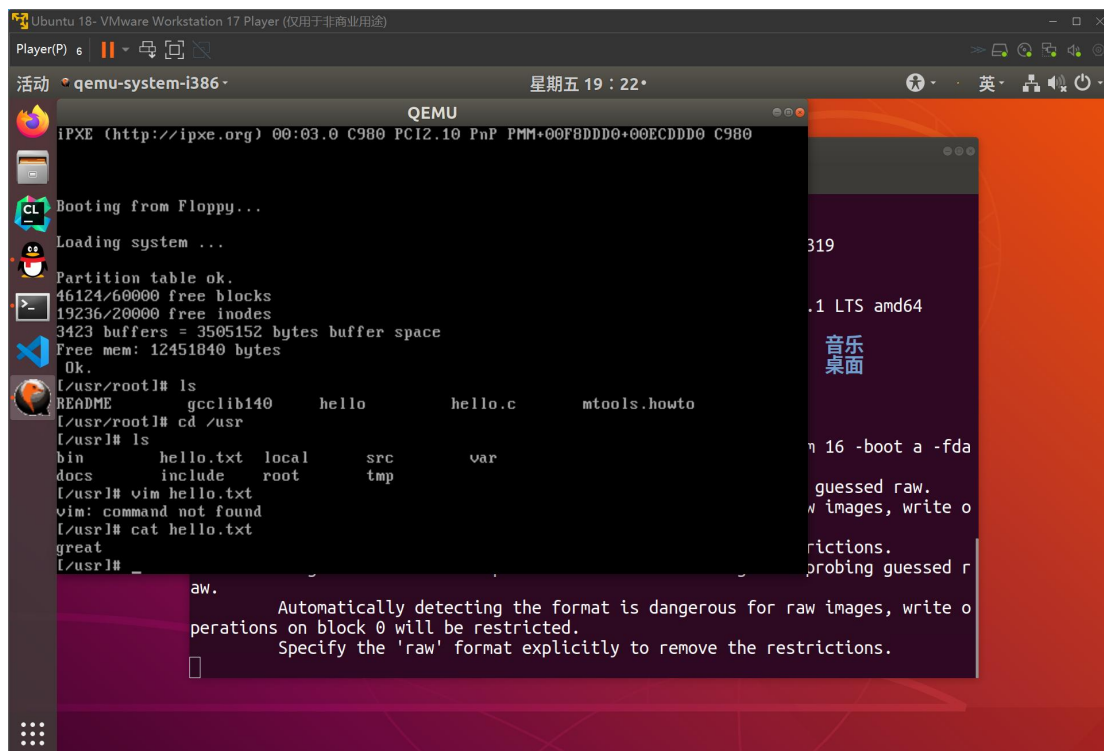


创建 hello.txt 文件并写入 “great”

确认卸载成功:



观察到/usr 目录下有 hello.txt 文件并且文件内容与写入内容一致



Section 3 实验总结与心得体会

通过本次实验，我发现我对 linux 命令、操作系统内核的编译、启动和调试拥有很大的知识漏洞。在很多时候面对实验不是不会，而是不知道该怎么办，不知道使用什么命令来完成实验，因此只能参照实验手册一步一步运行命令完成实

验。在第一遍运行时大多命令我只是知其然而不知其所以然，只是不动脑筋的照葫芦画瓢，完成实验后我发现自己并无收获，于是意识到我学习方法具有严重缺陷，决定从头开始了解每一行命令的含义。虽然基础不好，但我最终还是大体明白了上述所用到的命令是在干什么，各自有什么意义。

本次实验极大地加深了我对操作系统内核的了解，我从 0 开始完成了操作系统的编译、启动和调试，制作了 `initramfs`，尝试接触了 `qemu` 和 `gdb`。在这个过程中，我不仅掌握了编译内核的基本流程，还深入理解了操作系统启动过程中的各个关键环节，如引导加载程序、内核初始化、文件系统的挂载等。此外，通过 `qemu` 虚拟机，我能够快速测试和验证内核的功能，减少了在物理机上调试的时间成本。使用 `gdb` 进行调试让我能够更加清晰地跟踪内核执行过程，定位问题，并加深了我对操作系统内部机制的理解。

这次实验不仅让我更加熟悉了操作系统的底层实现，还让我意识到调试和优化内核的复杂性和挑战性。在接下来的学习中，我将继续深入探索操作系统的内核设计，尤其是在内存管理、进程调度以及设备驱动等方面，希望能够进一步提升我的实践能力和理论水平。

Section 4 对实验的改进建议和意见

在编译并启动 `Busybox` 实验时我不是很清楚如何编写 `initramfs` 以及 `initramfs` 需要的 `init` 程序，为什么要打包归档成 `cpio` 文件，如果可以的话希望老师可以在课上具体的讲解。

Section 5 附录：参考资料清单

<https://gitee.com/apshuang/sysu-2025-spring-operating-system>

<https://gitee.com/sysu2024-osta/sysu-2024-spring-operating-system>

<http://course.dds-sysu.tech/course/18/homework>