

《操作系统原理》实验报告

姓名	鄢湧棚	学号	U201911741	专业班级	网安 1902	时间	2022.1.6
----	-----	----	------------	------	---------	----	----------

一、实验目的

- (1) 理解操作系统引导程序/BIOS/MBR 的概念和作用；
- (2) 理解并应用操作系统生成的概念和过程；
- (3) 理解并应用操作系统操作界面，系统调用概念
- (4) 掌握和推广国产操作系统（限“银河麒麟”，加 10 分，直到满分）

二、实验内容

- (1) 用 NASM 编写 MBR 引导程序，在 BOCHS 虚拟机中测试。
- (2) 在 Linux（建议 Ubuntu 或银河麒麟）下载剪和编译 Linux 内核，并启用新内核。（其他发行版本也可以）
- (3) 为 Linux 内核（建议 Ubuntu 或银河麒麟）增加 2 个系统调用，并启用新的内核，并编写应用程序测试。（其他发行版本也可以）
- (4) 在 Linux（建议 Ubuntu 或银河麒麟）或 Windows 下，编写脚本或批处理。
在指定目录中的全部 txt 文件末尾追加或更新：用户名:日期和时间.root:2021-11-23 09:50

三、实验过程

1) MBR 引导程序编写测试

在 Ubuntu 虚拟机中安装 BOCHS 和 nasm

先安装 BOCHS 环境：

```
sudo apt-get install build-essential xorg-dev libgtk2.0-dev
```

然后在 NASM 官网下载压缩包，解压后编译安装：

```
tar zxvf nasm-2.15.05.tar.gz
cd nasm-2.15.05
./configure
```

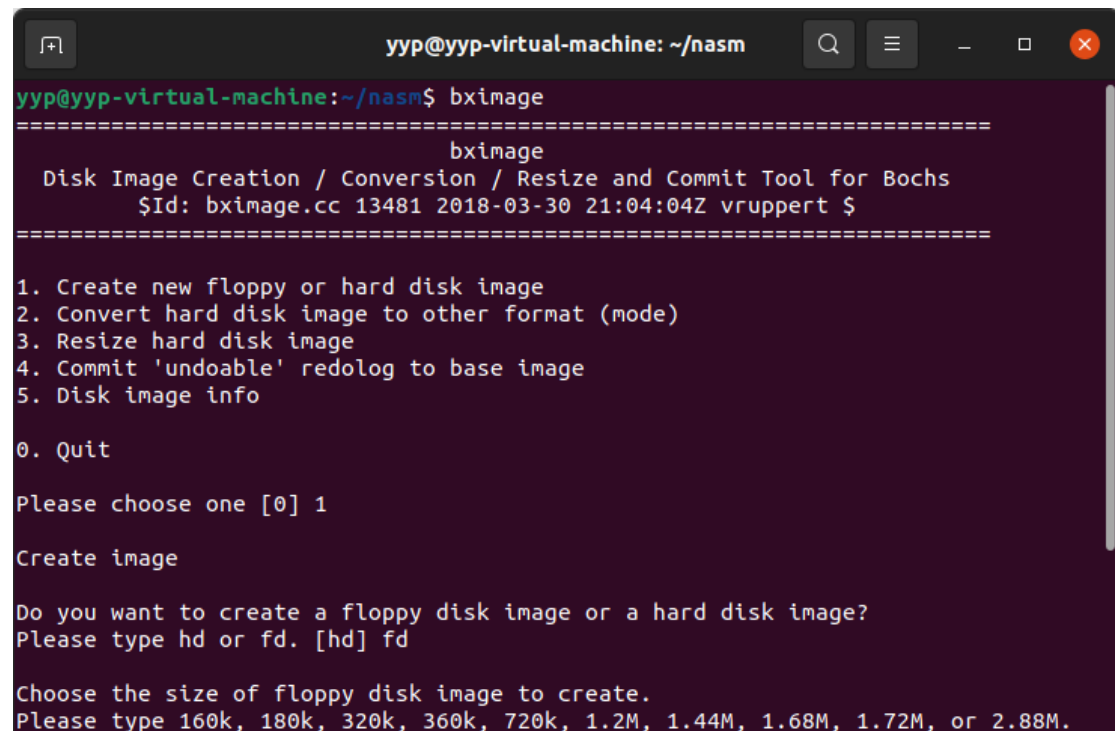
```
make  
sudo make install
```

接下来安装 Bochs，同样在官网下载，使用以下命令安装：

```
tar zxvf bochs-2.6.11.tar.gz  
cd bochs-2.6.11.tar.gz  
./configure --enable-debugger --enable-disasm  
make  
sudo make install
```

安装好后，接下来编写汇编代码，创建镜像，并将汇编代码编译如镜像中。

使用 bxiimage 生成镜像：



```
yyp@yyp-virtual-machine: ~/nasm  
yyp@yyp-virtual-machine:~/nasm$ bxiimage  
===== bxiimage =====  
Disk Image Creation / Conversion / Resize and Commit Tool for Bochs  
$Id: bxiimage.cc 13481 2018-03-30 21:04:04Z vruppert $  
=====
```

1. Create new floppy or hard disk image
2. Convert hard disk image to other format (mode)
3. Resize hard disk image
4. Commit 'undoable' redolog to base image
5. Disk image info

0. Quit

Please choose one [0] 1

Create image

Do you want to create a floppy disk image or a hard disk image?
Please type hd or fd. [hd] fd

Choose the size of floppy disk image to create.
Please type 160k, 180k, 320k, 360k, 720k, 1.2M, 1.44M, 1.68M, 1.72M, or 2.88M.

后面的一直回车就行，生成了一个 a.img 的镜像。

编写 boot.asm

```
org 07c00h          ; 告诉编译器程序加载到 7c00 处  
  
mov ax, cs  
  
mov ds, ax  
  
mov es, ax  
  
call DispStr        ; 调用显示字符串例程
```

```

    jmp $          ; 无限循环
DispStr:
    mov ax, BootMessage
    mov bp, ax      ; ES:BP = 串地址
    mov cx, 16      ; CX = 串长度
    mov ax, 01301h   ; AH = 13, AL = 01h
    mov bx, 000ch    ; 页号为 0 (BH = 0) 黑底红字 (BL = 0Ch, 高亮)
    mov dl, 0
    int 10h         ; 10h 号中断
    ret
BootMessage:        db "Hello, OS world!"
times 510-($-$$) db 0 ; 填充剩下的空间, 使生成的二进制代码恰好为
512 字节
dw 0xaa55          ; 结束标志

```

编译, 把程序写入镜像:

```

nasm -f bin code.asm -o code.bin -l code.lst
dd if=code.bin of=a.img

```

接下来要执行 NASM 代码, 在文件夹里创建 bochsrc.txt

```

megs:128
#模拟器的内存

romimage:file=/usr/local/share/bochs/BIOS-bochs-latest
#这个是 BIOS-bochs-latest 的路径, 可能不一样

vgaromimage:file=/usr/local/share/bochs/VGABIOS-lgpl-latest
#这个是 VGABIOS-lgpl-latest 的路径, 也可能不一样

floppya:1_44=a.img,status=inserted
#这个是启动软盘, 在当前目录下, 如果不在当前目录, 需要指明路径

```

boot:floppy

#表示从软盘启动

log:bochsout.txt

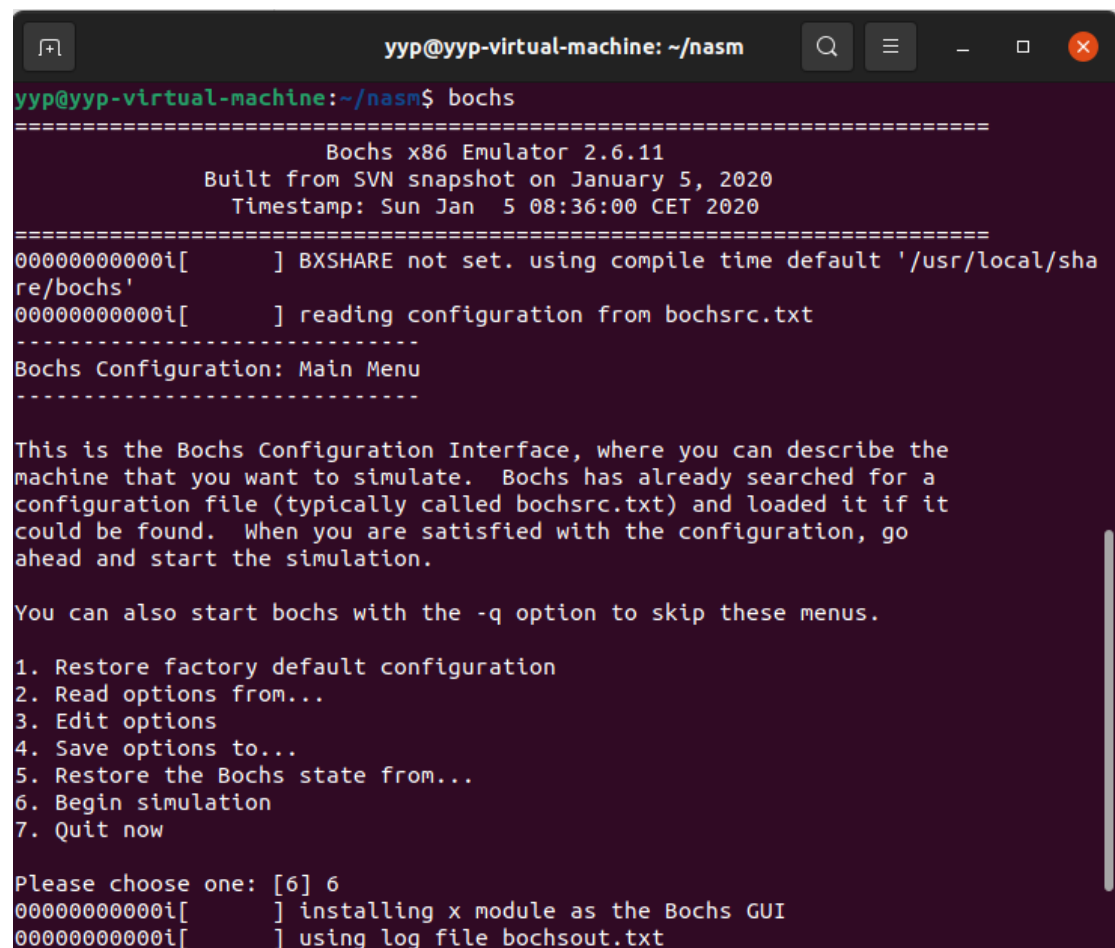
#日志输出文件

终端执行如下命令

```
bochs -f bochsrc
```

```
bochs
```

选择 6 开始模拟。

A terminal window titled 'yyp@yyp-virtual-machine: ~/nasm' showing the execution of 'bochs'. The output displays the Bochs version (2.6.11), build date (January 5, 2020), and timestamp. It then shows configuration messages: 'BXSHARE not set. using compile time default' and 'reading configuration from bochsrc.txt'. A 'Bochs Configuration: Main Menu' is displayed with a list of options: 1. Restore factory default configuration, 2. Read options from..., 3. Edit options, 4. Save options to..., 5. Restore the Bochs state from..., 6. Begin simulation, 7. Quit now. The user has selected option 6, and the terminal shows 'Please choose one: [6] 6' followed by 'installing x module as the Bochs GUI' and 'using log file bochsout.txt'.

```
yyp@yyp-virtual-machine: ~/nasm
yyp@yyp-virtual-machine:~/nasm$ bochs
=====
                Bochs x86 Emulator 2.6.11
          Built from SVN snapshot on January 5, 2020
          Timestamp: Sun Jan  5 08:36:00 CET 2020
=====
000000000000i[      ] BXSHARE not set. using compile time default '/usr/local/sha
re/bochs'
000000000000i[      ] reading configuration from bochsrc.txt
-----
Bochs Configuration: Main Menu
-----

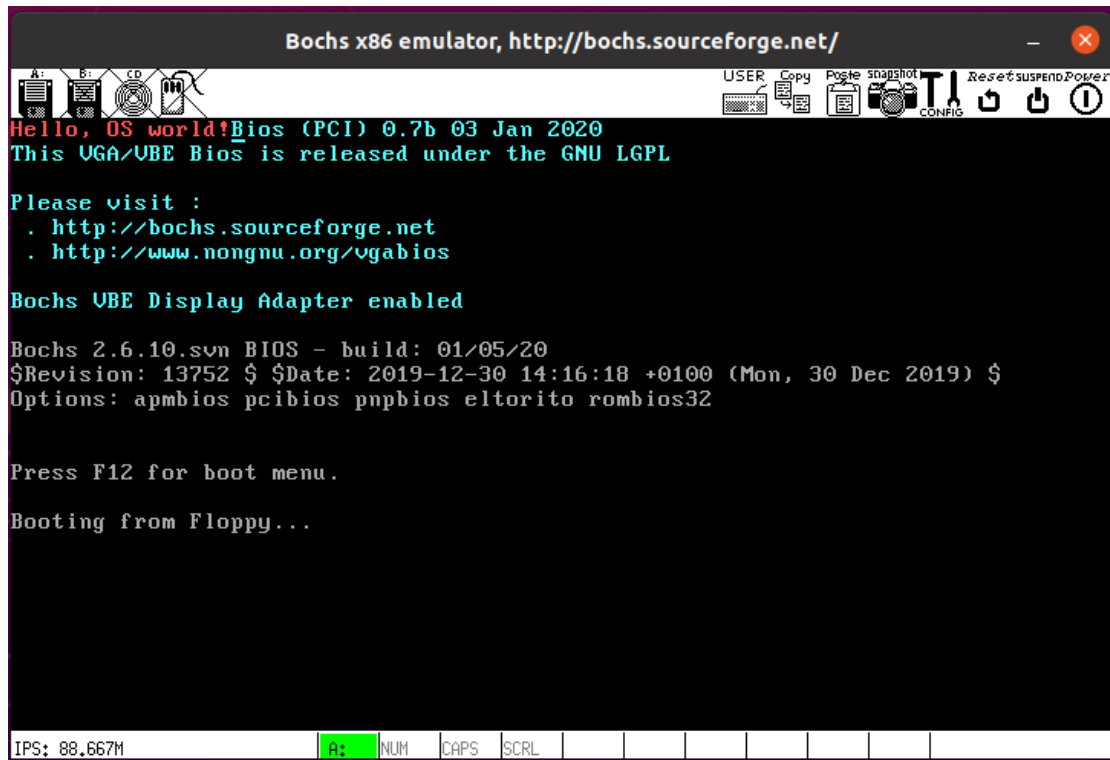
This is the Bochs Configuration Interface, where you can describe the
machine that you want to simulate.  Bochs has already searched for a
configuration file (typically called bochsrc.txt) and loaded it if it
could be found.  When you are satisfied with the configuration, go
ahead and start the simulation.

You can also start bochs with the -q option to skip these menus.

1. Restore factory default configuration
2. Read options from...
3. Edit options
4. Save options to...
5. Restore the Bochs state from...
6. Begin simulation
7. Quit now

Please choose one: [6] 6
000000000000i[      ] installing x module as the Bochs GUI
000000000000i[      ] using log file bochsout.txt
```

这时出现黑窗口，再在终端输入字符 ‘c’，显示如下



可以看到，系统启动过程中执行了 MBR 区的代码，打印了字符串。

2) ubuntu 内核编译以及添加系统调用

因为编译内核非常花时间，所以将 2, 3 任务合起来做，添加完系统调用后再编译。

先在 Linux 官网 (<https://www.kernel.org/>) 下载最新版的内核，本次实验使用的版本为 5.15.4，下载后在用户文件夹下解压。

```
tar -xvf linux-5.15.4.tar.xz
```

解压后使用如下命令安装依赖环境

```
sudo apt-get install libncurses5-dev openssl libssl-dev
sudo apt-get install build-essential openssl
sudo apt-get install pkg-config
sudo apt-get install libc6-dev
sudo apt-get install bison
sudo apt-get install flex
sudo apt-get install libelf-dev
sudo apt-get install zlib1g-dev
```

接下来添加系统调用，先在系统调用表中添加调用号和函数名，进入内核文件夹，使用如下指令打开调用表：

```
sudo gedit arch/x86/entry/syscalls/syscall_64.tbl
```

在末尾按照序号添加系统调用 add, max:

```
413 545      x32      execveat          compat_sys_execveat
414 546      x32      preadv2          compat_sys_preadv64v2
415 547      x32      pwritev2        compat_sys_pwritev64v2
416 548      64      add              sys_add
417 549      64      max              sys_max
418 550      64      hyc              sys_hyc
419 # This is the end of the legacy x32 range.  Numbers 548 and above are
420 # not special and are not to be used for x32-specific syscalls.
```

注意，不同版本的内核系统调用的数量不一样，按照上面的格式写就行。

使用如下命令，打开系统调用函数头文件

```
sudo gedit include/linux/syscalls.h
```

同样在末尾，添加函数声明：

```
1384
1385 asmlinkage int sys_add(int a, int b);
1386
1387 asmlinkage int sys_max(int a, int b, int c);
1388
1389 asmlinkage int sys_hyc(void);
1390 #endif
```

最后，需要编写函数原型，在如下文件中编写：

```
sudo gedit kernel/sys.c
```

同样在末尾添加：

```
~~~~~
2697
2698 SYSCALL_DEFINE2(add, int, a, int, b)
2699 {
2700     return a+b;
2701 }
2702
2703 SYSCALL_DEFINE3(max, int, a, int, b, int, c)
2704 {
2705     return (a>b?(a>c?a:c):(b>c?b:c));
2706 }
2707
2708 SYSCALL_DEFINE0(hyc)
2709 {
2710     printk("I love Hyc!!\\(>v<\\)\n");
2711     return 0;
2712 }
2713 #endif /* CONFIG_COMPAT */
```

注意，这里 SYSCALL_DEFINE 后面的数字是函数参数个数，一个参数就是 1，没有参数就是 0。后面括号里跟的，第一个是函数名，后面的是参数类型和参数名。

最后一个调用中，打印使用的是 `printk`，因为没有 `c` 基本输入输出头文件，功能是向内核信息区写入信息，可以在终端中使用 `sudo mesg` 命令查看。

接下来开始编译，先创建配置文件，可以直接使用当前使用内核的配置文件，在 `boot` 文件夹下找到版本最高的内核版本的配置，复制到内核文件夹下的 `.config` 文件。

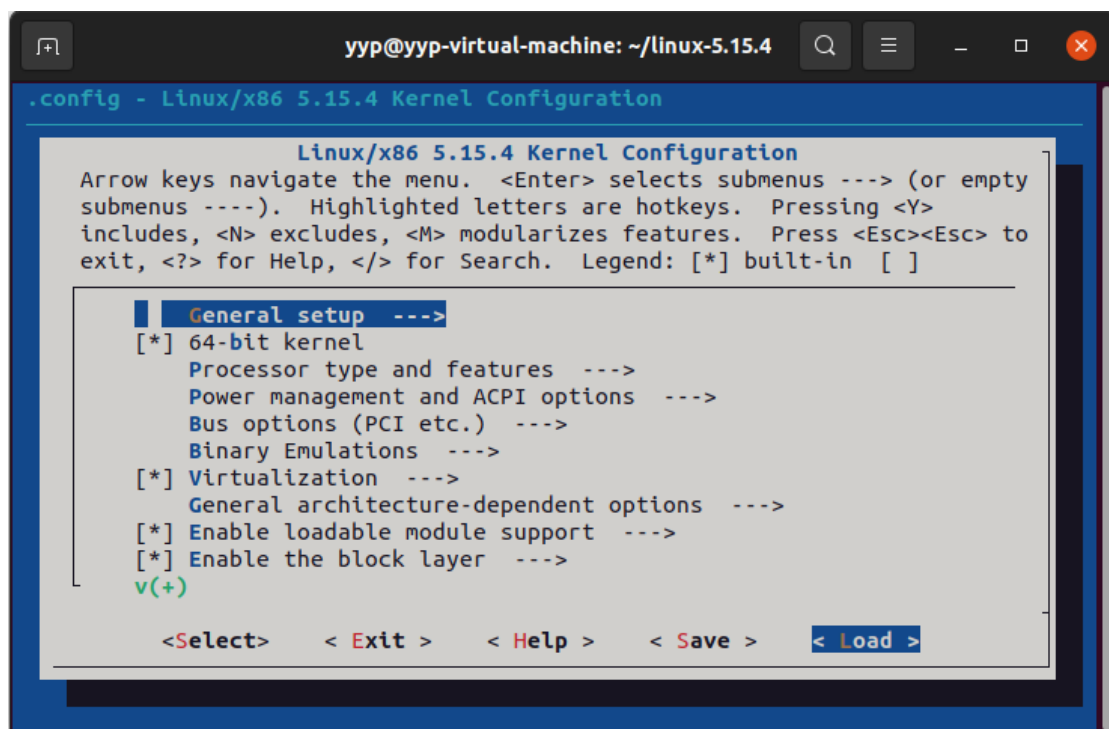
```
sudo cp /boot/config-5.11.0-40-generic ./config
```

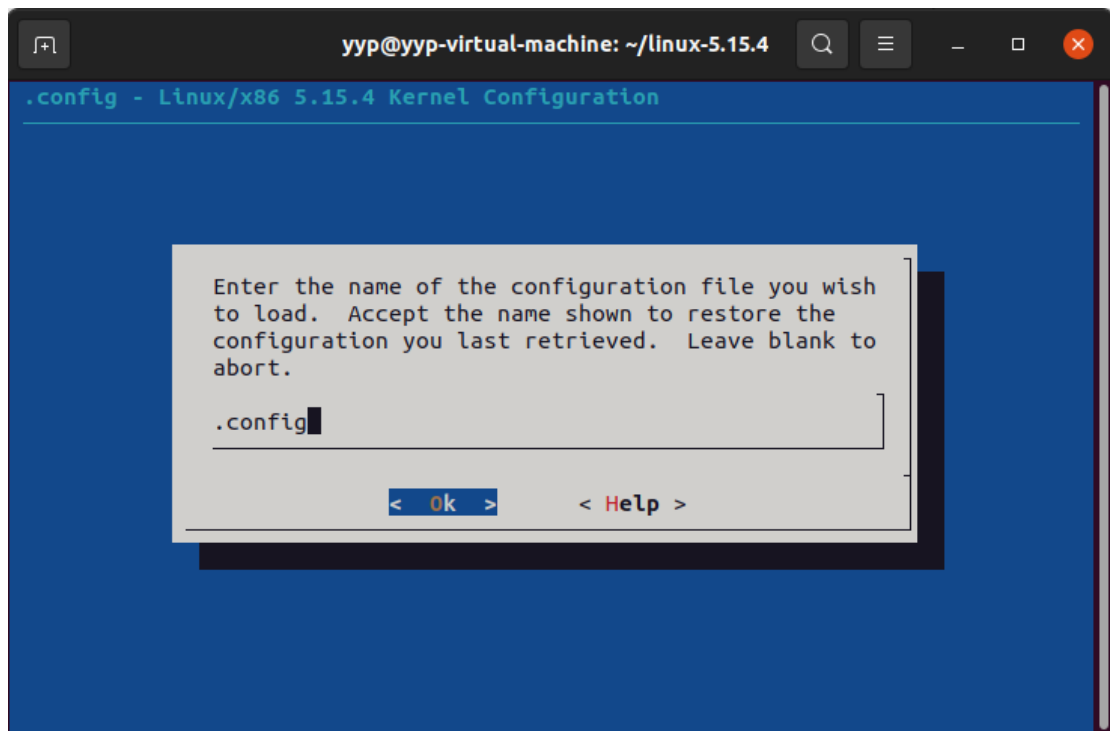
可能有的地方需要修改，不然编译的时候会报错，编译的时候可以根据错误进行修改。

然后控制台输入

```
sudo make menuconfig
```

跳出图形化配置界面，选择 `load`，输入 `.config`，选择 `OK`，最后 `save`





配置好后开始编译

```
sudo make -j16  
sudo make modules_install  
sudo make install
```

这里第一条命令-j 后面的数字是处理器逻辑核数，虚拟机分配了几个核就用这个数的两倍。编译过程中可能会提示缺少东西，缺了就 apt-get install，配置文件错了就改一下然后重新编译，重新编译前记得清除之前的文件，使用如下命令：

```
sudo make mrproper  
sudo make clean
```

全部编译成功后，重启。

重启过程中一直按 Shift 键（也有可能是 Esc 键），进入 Ubuntu 高级选项界面，即可见到编译好的内核，选择该内核启动。

3) 批处理文件

编写脚本文件如下：


```

#!/bin/bash

name=$USER

function read_dir() {
    for file in `ls $1`
    do
        if [ -d $1"/"$file ]
        then
            read_dir $1"/"$file
        else
            if [ "${file##*.}"x = "txt"x ]; then
                time=$(date "+%Y-%m-%d %H:%M:%S")
                addstr="\n$name\t\t$time"
                echo -e -n '\n\n' >> $1"/"$file
                sed -i ':n;/^\n*$/{$! N;$d;bn}' $1"/"$file
                tailstr=$(cat $1"/"$file|tail -n 1)
                if echo $tailstr|grep -Eq "[0-9]{4}-(0[1-9]|1[0-2])-(0[1-9]|1[0-2][0-9]|3[0-1]) ([0-1][0-9]|2[0-4]):([0-5][0-9]|60):([0-5][0-9]|60)"
                then
                    sed -i '$d' $1"/"$file
                    sed -i ':n;/^\n*$/{$! N;$d;bn}' $1"/"$file
                    echo -e -n $addstr >> $1"/"$file
                else
                    echo -e -n $addstr >> $1"/"$file
                fi
            fi
        fi
    done
}

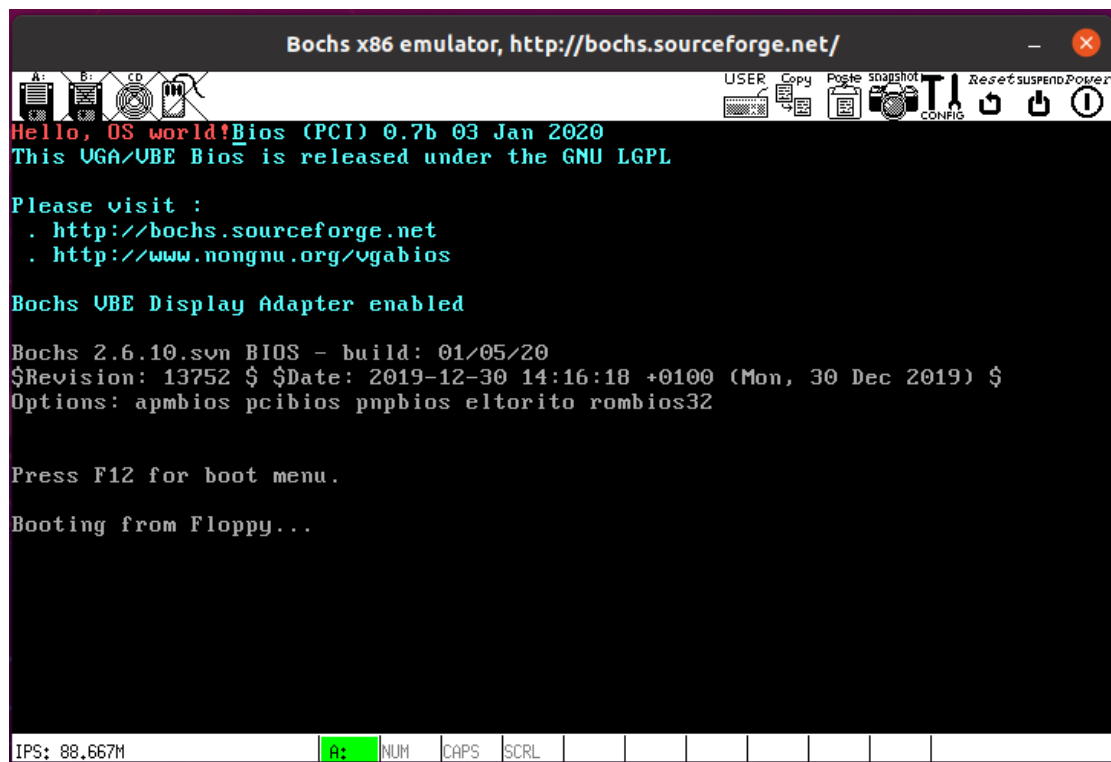
```

```
}
```

```
read_dir $1
```

四、 实验结果

1) MBR 引导程序编写测试



可以看到，系统启动过程中执行了 MBR 区的代码，打印了字符串。

2) ubuntu 内核编译以及添加系统调用

编译完成后显示如下：

```
yyp@yyp-virtual-machine: ~/linux-5.15.4
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 5.15.4 /boot/vmlinuz-5.15.4
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 5.15.4 /boot/vmlinuz-5.15.4
update-initramfs: Generating /boot/initrd.img-5.15.4
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 5.15.4 /boot/vmlinuz-5.15.4
run-parts: executing /etc/kernel/postinst.d/update-notifier 5.15.4 /boot/vmlinuz-5.15.4
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 5.15.4 /boot/vmlinuz-5.15.4
Sourcing file `/etc/default/grub'
Sourcing file `/etc/default/grub.d/init-select.cfg'
正在生成 grub 配置文件 ...
找到 Linux 镜像: /boot/vmlinuz-5.15.4
找到 initrd 镜像: /boot/initrd.img-5.15.4
找到 Linux 镜像: /boot/vmlinuz-5.11.0-40-generic
找到 initrd 镜像: /boot/initrd.img-5.11.0-40-generic
找到 Linux 镜像: /boot/vmlinuz-5.11.0-38-generic
找到 initrd 镜像: /boot/initrd.img-5.11.0-38-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
完成
yyp@yyp-virtual-machine:~/linux-5.15.4$
```

当前版本:

```
make: *** [_modinst_] Error 2
yyp@ubuntu:~/yyp/linux-5.4.161$ uname -r
5.4.0-84-generic
yyp@ubuntu:~/yyp/linux-5.4.161$
```

重启选用新内核

```
yyp@yyp-virtual-machine: ~/Desktop
yyp@yyp-virtual-machine:~/Desktop$ uname -r
5.15.4
yyp@yyp-virtual-machine:~/Desktop$
```

可以看到，内核版本已升级至 5.15.4。

对于系统调用，编写如下 c 程序进行验证

```
#include <stdio.h>

#include <linux/kernel.h>
```

```

#include <sys/syscall.h>
#include <unistd.h>
int main()
{
    /*548 549 550*/
    int a, b, c, d;
    a=10;b=20;c=30;
    d = syscall(548,a,b);
    printf("%d\n",d);
    d = syscall(549,a,b,c);
    printf("%d\n",d);
    d=syscall(550);
    return 0;
}

```

编译运行如下：

```

yyp@yyp-virtual-machine:~/Desktop$ ./syscalltest
30
30

```

查看内核缓冲区，可以看到 printk 的输出

```

.4.1 pid=90287 comm= cups-browsed Reque
uid=0
[127498.867888] audit: type=1400 audit(16
eration="capable" profile="/usr/sbin/cups
capability=23 capname="sys_nice"
[131599.500202] I love Hyc!!\(>v<\)
yyp@yyp-virtual-machine:~/Desktop$

```

3) 批处理文件

在当前文件夹下新建 a.txt，使用如下命令

```
sudo bash add.sh ./
```

结果如下：

```
1 阿巴阿巴
2 阿巴阿巴
3
4 root 2022-01-06 11:19:08
```

添加成功

五、 实验错误排查和解决方法

1) MBR 引导程序编写测试

有参考资料，没遇到什么问题。

2) ubuntu 内核编译以及添加系统调用

内核编译过程有非常多的错误，每次编译到一半都会提示缺少需要的文件或者配置文件出错，解决方法是少东西就装东西，配置文件出错百度搜出错原因就能找到对应的解决办法。

3) 批处理文件

bash 脚本的语法都能搜到，没遇到什么问题。

六、 实验参考资料和网址

<https://www.cnblogs.com/chengmf/p/12526821.html>

<https://www.cnblogs.com/LyShark/p/13353400.html>

https://blog.csdn.net/weixin_44224230/article/details/89945899