

《操作系统原理》实验报告 1

姓名	郭雪菲	学号	U202112131	专业班级	网安 2104	时间	2023.11.21
----	-----	----	------------	------	---------	----	------------

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 (推荐银河麒麟或优麒麟, 建议)

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 或银河麒麟或优麒麟) 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 或银河麒麟或优麒麟) 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 或银河麒麟或优麒麟) 或 Windows 下, 编写脚本或批处理。脚本参数 1 个: 指定目录。脚本的作用是把指定目录中的全部文件的文件名加后缀, 后缀是执行脚本时的日期和时分。例如: 文件名 “test” 变成 “test-2023-11-21-20-42”。

三、实验过程

3.1 编写 MBR 引导程序

1) 环境配置

实验环境: VMware Workstation Pro 17, 优麒麟 20.04

下载优麒麟的镜像文件:

1. 官网下载: <https://www.ubuntukylin.com/downloads/> (下载速度慢)
2. 镜像网站: 在官网底部有多重选择 (最后选择了华为云, 速度较快)

创建虚拟机, 内存设置为 4G, 硬盘空间尽量大, 防止后续编译内核因存储空间不足而失败。(之后做任务二时编译了几个小时, 怀疑是内存和硬盘开太小了, 扩展了硬盘后做任务三, 编译速度有所提高)

将使用下列设置创建虚拟机:

名称:	Ubuntukylin20.04
位置:	D:\文档\Virtual Machines\Ubuntukylin20.04
版本:	Workstation 17.x
操作系统:	Ubuntu 64 位
硬盘:	80 GB, 拆分
内存:	4096 MB
网络适配器:	NAT
其他设备:	2 个 CPU 内核, CD/DVD, USB 控制器, 打印机, 声卡

登录优麒麟后, 点击桌面的“安装 Ubuntu“, 否则试用版无法正常进行后续操作, 系统重启会进行清空 (密码 shiftw041)

安装 (以超级用户)

您是谁?

您的姓名: shiftw

您的计算机名: shiftw-virtual-machine
与其他计算机联网时使用的名称。

选择一个用户名: shiftw

选择一个密码: ●●●●●●●● 密码强度: 合理

确认您的密码: ●●●●●●●●

☐ 自动登录

☒ 登录时需要密码

后退(B) 继续

安装完成后登录, 打开终端, 使用如下命令更新软件源, 否则无法正常安装必要工具:

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

2) 安装 NASM

使用命令 `sudo apt install nasm`

3) 安装 Bochs

使用命令 `sudo apt-get install vgabios bochs bochs-x bximage`

4) 编写引导扇区程序

切换到 `usr/local/` 目录, 创建 `sudo vim boot.asm`, 编写所提供的引导程序

```
shiftw@shiftw-virtual-machine: /usr/local
文件(F) 编辑(E) 视图(V) 搜索(S) 终端(T) 帮助(H)
org 07c00h
mov ax, cs
mov ds, ax
mov es, ax
call DispStr
jmp $
DispStr:
mov ax, BootMessage
mov bp, ax
mov cx, 16
mov ax, 01301h
mov bx, 000ch
mov dl, 0
int 10h
ret
BootMessage: db "Hello, OS"
times 510 - ($-$$) db 0
dw 0xaa55
```

编写完成后使用 `sudo nasm boot.asm -o boot.bin` 命令编译

5) 创建虚拟软盘

通过如下 `sudo bxiimage` 命令创建虚拟软盘，选择生成类型为 `fd`，设置软盘映像文件名，当前目录成功生成 `a.img` 软盘映像：

```
shiftw@shiftw-virtual-machine: /usr/local$ sudo 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.
[1.44M] a.img
Your choice (a.img) did not match any of the choices:
160k, 180k, 320k, 360k, 720k, 1.2M, 1.44M, 1.68M, 1.72M, 2.88M

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.
[1.44M]

What should be the name of the image?
[a.img]

Creating floppy image 'a.img' with 2880 sectors

The following line should appear in your bochsrc:
  floppy0: image="a.img", status=inserted
```

6) 将引导扇区写入软盘

通过 `sudo dd if=boot.bin of=a.img bs=512 count=1 conv=notrunc` 命令将扇区写入软盘

```
shiftw@shiftw-virtual-machine:/usr/local$ sudo dd if=boot.bin of=a.img bs=512 count=1 conv=notrunc
记录了1+0 的读入
记录了1+0 的写出
512字节已复制, 0.000385961 s, 1.3 MB/s
```

7) 配置 Bochs

创建 `bochrs` 文件, 编写以下配置信息:

```
shiftw@shiftw-virtual-machine: /usr/local

文件(F) 编辑(E) 视图(V) 搜索(S) 终端(T) 帮助(H)

megs: 128
romimage: file=/usr/share/bochs/BIOS-bochs-latest
vgaromimage: file=/usr/share/vgabios/vgabios.bin
floppya: 1_44=a.img, status=inserted
boot: floppy
log: bochsout.txt
mouse: enabled=0
```

8) 启动 Bochs 虚拟机

通过 `sudo bochs -f bochrs` 命令执行启动后返回终端, 输入 `c` 按回车, 可以看到执行成功, 显示红色字体的 `Hello OS`

```
Bochs x86-64 emulator, http://bochs.sourceforge.net/
shiftw@shiftw-virtual-machine:/usr/local$ sudo bochs -f bochrs

Bochs x86 Emulator 2.6.11
Built from SVN snapshot on January 5, 2020
Timestamp: Sun Jan 5 08:36:00 CET 2020

000000000000i[ ] LTDL LIBRARY_PATH not set. using compile time default '/usr/li
000000000000i[ ] BXSHARE not set. using compile time default '/usr/share/bochs'
000000000000i[ ] lt_dldhandle is 0x55bf2e396be0
000000000000i[PLUGIN] loaded plugin libbx unmapped.so
000000000000i[ ] lt_dldhandle is 0x55bf2e397a00
000000000000i[PLUGIN] loaded plugin libbx biosdev.so
000000000000i[ ] lt_dldhandle is 0x55bf2e3983a0
000000000000i[PLUGIN] loaded plugin libbx speaker.so
000000000000i[ ] lt_dldhandle is 0x55bf2e39d490
000000000000i[PLUGIN] loaded plugin libbx extfpuiqrq.so
000000000000i[ ] lt_dldhandle is 0x55bf2e39dc60
000000000000i[PLUGIN] loaded plugin libbx parallel.so
000000000000i[ ] lt_dldhandle is 0x55bf2e39f8c0
000000000000i[PLUGIN] loaded plugin libbx serial.so
000000000000i[ ] lt_dldhandle is 0x55bf2e3a3cc0
000000000000i[PLUGIN] loaded plugin libbx gameport.so
000000000000i[ ] lt_dldhandle is 0x55bf2e3a44f0
000000000000i[PLUGIN] loaded plugin libbx iodebug.so
000000000000i[ ] reading configuration from bochrs
000000000000i[ ] lt_dldhandle is 0x55bf2e3a4fa0
000000000000i[PLUGIN] loaded plugin libbx x.so
000000000000i[ ] installing x module as the Bochs GUI
000000000000i[ ] using log file bochsout.txt
Next at t=0
(0) [0x0000ffffffffff] f000:fff0 (unk. ctxt): jmpf 0xf000:e05b ; ea5be000f0
<bochs:1> c

Hello, OS
Bios (PCI) current-svn 07 Jan 2020
This UGA/UBE Bios is released under the GNU LGPL

Please visit :
. http://bochs.sourceforge.net
. http://www.nongnu.org/vgabios

Bochs UBE Display Adapter enabled

Bochs 2.6.11 BIOS - build: 01/05/20
$Revision: 13752 $ $Date: 2019-12-30 14:16:18 +0100
Options: apmbios pcibios pnpbios eltorito rombios32

Press F12 for boot menu.

Booting from Floppy...
```

3.2 编译 linux 内核

1) 查看当前 linux 安装了哪些内核

`dpkg --get-selections | grep linux-image`

```
shiftw@shiftw-virtual-machine:/$ dpkg --get-selections | grep linux-image
ii linux-image-5.11.0-25-generic      5.11.0-25.27~20.04.1      amd64      Signed ke
rnel image generic
ii linux-image-5.4.0-166-generic     5.4.0-166.183             amd64      Signed ke
rnel image generic
ii linux-image-5.4.0-80-generic      5.4.0-80.90               amd64      Signed ke
rnel image generic
ii linux-image-generic               5.4.0.166.163             amd64      Generic L
linux kernel image
```

`uname -r` 查看正在使用的内核版本，后面下载的内核版本不能高于目前版本

```
shiftw@shiftw-virtual-machine:/$ uname -r
5.11.0-25-generic
```

2) 下载 linux 内核源码并解压

从网上下载不高于目前内核版本的 kernel 源代码。

1.官网: <https://www.kernel.org/> (下载速度慢)

2.镜像网站: <http://ftp.sjtu.edu.cn/sites/ftp.kernel.org/pub/linux/kernel/> (推荐)

此处选择下载 `linux-5.8.4.tar.xz`，之后移动压缩包到 `usr/src` 目录下，解压后删除。

`tar xf linux-5.8.4.tar.xz`

`sudo mv linux-5.8.4 /usr/src/`

`sudo rm linux-5.8.4.tar.xz`

3) 下载软件包

使用如下命令安装编译的必要工具 `gcc`、`gdb`、`bison`、`flex`、`libncurses5-dev`、`libssl-dev`、`libidn11` 以及虚拟机的必备工具：

`sudo apt-get install gcc gdb bison flex libncurses5-dev libssl-dev libidn11 build-essential`

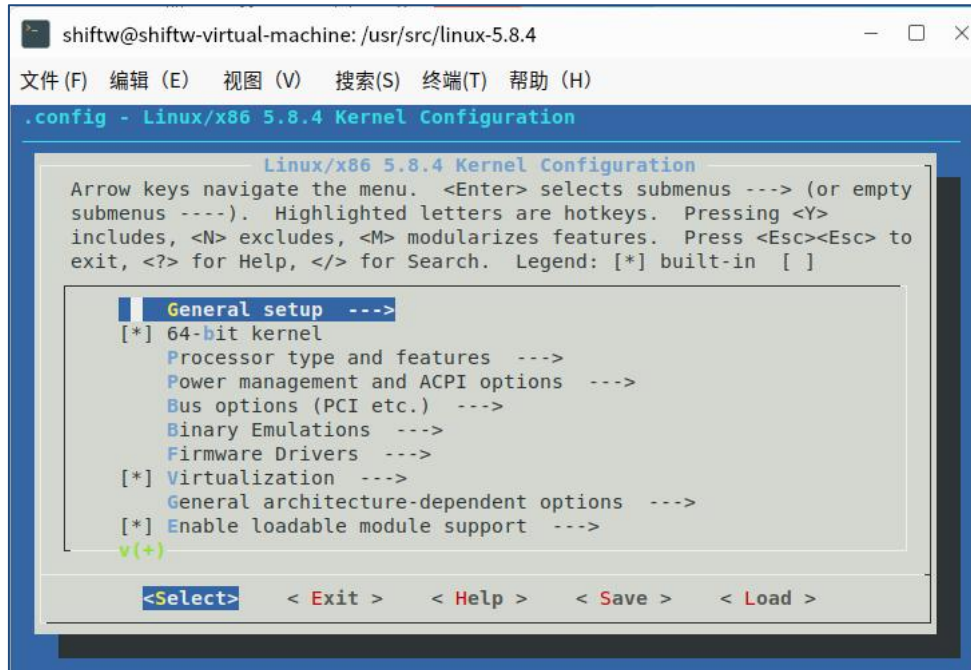
4) 配置内核

`sudo make mrproper` #清除残留的 `.config` 和 `.o` 文件

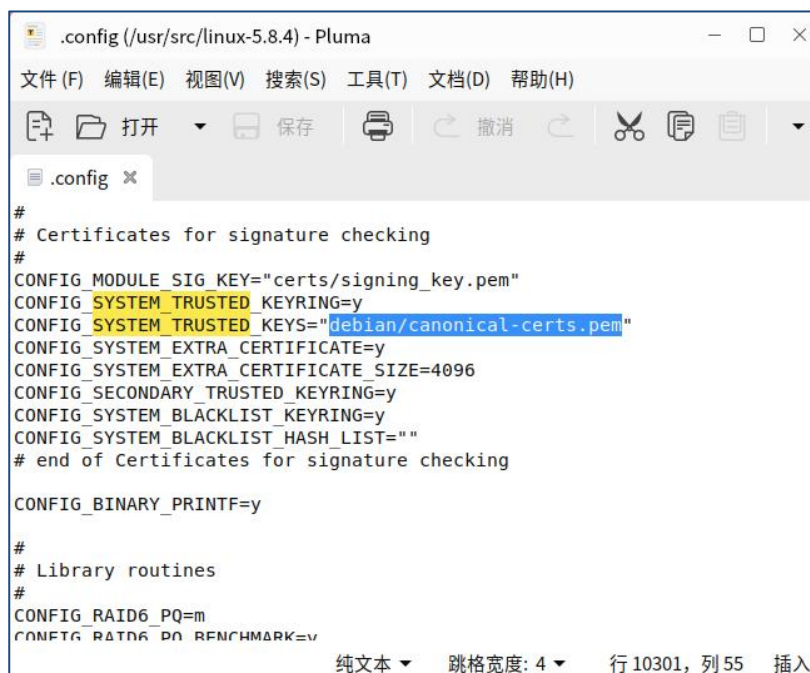
`sudo make clean`

`sudo make menuconfig` #打开配置内核的图形窗口使用个性化配置。

实际实验过程中无需修改，直接使用默认配置即可。显示如下界面后，点击 `<Exit>` 即可



修改配置文件 `sudo pluma .config`，将 `CONFIG_SYSTEM_TRUSTED_KEYS` 修改为空字符串



5) 编译内核

进入源码目录，使用 `make` 命令编译内核，同时借助 `-jn` 加快编译速度，采用 `sudo make -j4` 命令编译。`n` 是要生成的线程数，通常每个处理器产生一个或两个线程较合适，否则编译进度过慢或者发生内存崩溃。可以查看虚拟机硬件配置中的处理器总分配内核数。由于虚拟机配置较低，此编译阶段非常耗时，可以挂着虚拟机去做别的事情。

6) 安装内核模块

```
sudo make modules
```

```
sudo make modules_install
```

```
sudo make install
```

并将新安装的内核设置为引导，更新 grub 引导程序。

```
sudo update-initramfs -c -k 5.8.4
```

```
sudo update-grub2
```

7) 进入新内核

终端输入 **reboot** 重启虚拟机，并在重启过程中按下 **tab+shift** 进入高级启动项，发现多出了新的内核版本，即为编译过后的新 linux 内核



选择新内核进入，终端查看目前内核版本发现是新安装的 linux 版本，且创建时间与编译开始时间相同，说明的确是新内核。

```
shiftw@shiftw-virtual-machine:/$ uname -r
5.8.4
shiftw@shiftw-virtual-machine:/$ uname -a
Linux shiftw-virtual-machine 5.8.4 #1 SMP Tue Nov 21 07:52:05 CST 2023 x86_64 x86_64 GNU/Linux
```

3.3 系统调用添加

1) 添加系统调用号

通过如下命令进入系统调用号的配置

```
sudo vim /usr/src/linux-5.8.4/arch/x86/entry/syscalls/syscall_64.tbl
```

在末尾添加两个新调用号

```
64 add sys_add
```

```
64 max sys_max
```

```
548      64      add      sys_add
549      64      max      sys_max
<ch/x86/entry/syscalls/syscall_64.tbl" 407L, 14209C          407,31      底端
```

2) 添加系统调用服务函数的声明

通过如下命令进入函数声明配置

```
sudo vim /usr/src/linux-5.8.4/include/linux/syscalls.h
```

添加两条新的函数声明

```
asmlinkage int sys_add(int a, int b);
```

```
asmlinkage int sys_max(int a, int b, int c);
```

```
asmlinkage int sys_add(int a, int b);
asmlinkage int sys_max(int a, int b, int c);
#endif
"include/linux/syscalls.h" 1428L, 57290C          1428,6      底端
```

3) 实现自己的系统调用服务函数

通过如下命令进入对函数的定义

```
sudo vim /usr/src/linux-5.8.4/kernel/sys.c
```

对两条新函数进行定义

```
SYSCALL_DEFINE2(add,int,a,int,b){
    return a+b;
}
SYSCALL_DEFINE3(max,int,a,int,b,int,c){
    return (a>b?a:b)>c?(a>b?a:b):c;
}
#endif /* CONFIG_COMPAT */
-- 插入 --
2699,2      底端
```


4) 重新编译内核

参考任务二的步骤重新编译内核：

```
cd /usr/src/linux-5.8.4/
```

```
sudo make mrproper #清除残留的.config 和.o 文件
```

```
sudo make clean
```

```
sudo make menuconfig #打开配置内核的图形窗口使用个性化配置。
```

实际实验过程中无需修改，直接使用默认配置点击<Exit>即可。

修改配置文件 `sudo pluma .config`，将 `CONFIG_SYSTEM_TRUSTED_KEYS` 修改为空字符串

```
sudo make -j8 命令编译
```

```
sudo make modules
```

```
sudo make modules_install
```

```
sudo make install
```

```
sudo update-initramfs -c -k 5.8.4
```

```
sudo update-grub2
```

终端输入 `reboot` 重启虚拟机，并按下 `tab+shift` 进入高级启动项，选择新内核进入

```
shiftw@shiftw-virtual-machine:~/桌面$ uname -a
Linux shiftw-virtual-machine 5.8.4 #1 SMP Tue Nov 21 16:10:29 CST 2023 x86_64 x86_64 x86_64 GNU/Linux
```

5) 测试调用

编写 c 程序测试

```
#include <stdio.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <linux/kernel.h>
int main(){
    int nRet1 = syscall(548, 20,18); // nRet = 38 ,第一个参数是系统调用号
    printf("add_result=%d\n", nRet1);
    nRet1 = syscall(549, 20,18, 4); // nRet = 20
    printf("max_result=%d\n",nRet1);
    return 0;
}
```

```
sudo vim test.c
```

```
gcc test.c -o test
```

```
./test
```

3.4 编写追加后缀脚本

1) 编写脚本

编写如下所示脚本文件作为批处理程序 postfix.sh

```
#!/bin/bash
cd $1
for files in `ls -l | grep ^[^d] | awk '{print $9}'`
do
    final=${files##*.}
    filename=${files%.*}
    filename=${filename/-*-*-*-*-*}
    cur_date="- `date +%Y-%m-%d-%H-%M` "
    file=$filename$cur_date
    mv $files ${file}.$final
done
```

2) 测试程序

创建 test 文件夹存放测试文件若干进行测试，带参数执行批处理程序

```
sudo ./postfix.sh test/
```

尝试运行批处理程序失败，报错显示没有此操作。

ls -l 查看脚本文件权限

```
-rw-rw-r-- 1 shiftw shiftw 264 11月 21 12:16 postfix.sh
```

显示没有执行权限，故使用 chmod u+x postfix.sh 赋予权限

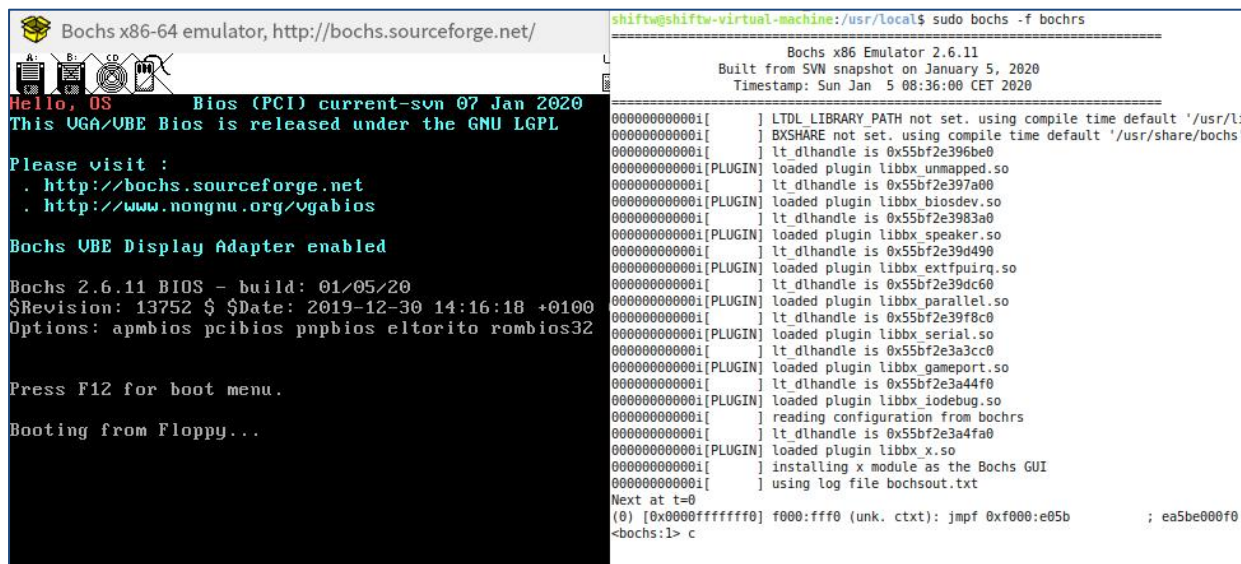
得到测试结果如下所示，可以看到 test 文件夹其中的两个子目录并没有被追加后缀，其余文件正常被追加后缀，追加时间格式为-年-月-日-时-分，重新运行批处理程序，后缀没有追加而是更新了时间。

```
shiftw@shiftw-virtual-machine:~$ sudo ./postfix.sh test/
shiftw@shiftw-virtual-machine:~$ ls test/
tes1-2023-11-21-12-24.txt  test3-2023-11-21-12-24.txt  test5
test2-2023-11-21-12-24.py  test4
shiftw@shiftw-virtual-machine:~$ sudo ./postfix.sh test/
shiftw@shiftw-virtual-machine:~$ la test/
tes1-2023-11-21-12-28.txt  test3-2023-11-21-12-28.txt  test5
test2-2023-11-21-12-28.py  test4
```

四、实验结果

4.1 编写 MBR 引导程序

可以观察到执行成功，黑色启动界面中首行显示红色的 Hello OS。



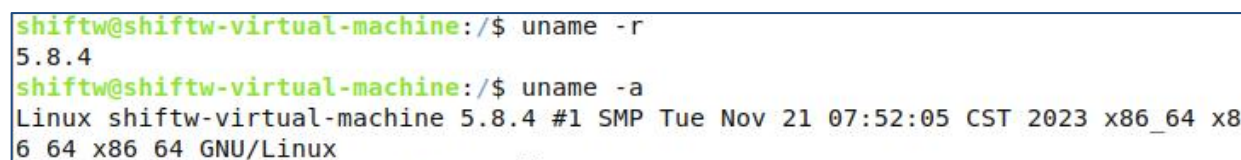
The screenshot shows the Bochs x86-64 emulator interface. The left pane displays the BIOS boot process with a black background and red text: "Hello, OS", "Bios (PCI) current-svn 07 Jan 2020", "This UGA/VE BIOS is released under the GNU LGPL", "Please visit : . http://bochs.sourceforge.net . http://www.nongnu.org/vgabios", "Bochs VBE Display Adapter enabled", "Bochs 2.6.11 BIOS - build: 01/05/20", "\$Revision: 13752 \$ \$Date: 2019-12-30 14:16:18 +0100", "Options: apmbios pcibios pnpbios eltorito rombios32", "Press F12 for boot menu.", and "Booting from Floppy...". The right pane shows the terminal output of the command "sudo bochs -f bochrs", displaying the Bochs x86 Emulator 2.6.11 version information, build date, and a list of loaded plugins and their handles.

4.2 编译 linux 内核

重启虚拟机并进入高级选项，发现出现了新的内核版本，即为编译过后的新 linux 内核。



选择新内核进入，查看内核版本发现是新安装的 linux 版本，且创建时间也是新的，说明更换新内核成功。



The screenshot shows the terminal output of the command "uname -r" and "uname -a". The output of "uname -r" is "5.8.4". The output of "uname -a" is "Linux shiftw-virtual-machine 5.8.4 #1 SMP Tue Nov 21 07:52:05 CST 2023 x86_64 x86_64 x86_64 GNU/Linux".

4.3 添加系统调用

编写 c 程序测试添加的系统调用函数，代码如下：

```
shiftw@shiftw-virtual-machine: /usr/src/linux-5.8.4
文件(F) 编辑(E) 视图(V) 搜索(S) 终端(T) 帮助(H)
#include <stdio.h>
#include <sys/syscall.h>
int main(){
    int nRet = syscall(548, 20, 18); // nRet = 38 ,第一个参数是系统调用号
    printf("add_result=%d\n", nRet);
    nRet = syscall(549, 20, 18, 4); // nRet = 20
    printf("max_result=%d\n", nRet);
    return 0;
}
```

```
sudo vim test.c
```

```
gcc test.c -o test
```

```
./test
```

操作结果如下所示，能正确输出加和与最大值，可见调用成功！

```
shiftw@shiftw-virtual-machine:/usr/src/linux-5.8.4$ sudo vim test.c
shiftw@shiftw-virtual-machine:/usr/src/linux-5.8.4$ gcc test.c -o test
shiftw@shiftw-virtual-machine:/usr/src/linux-5.8.4$ ./test
add_result=38
max_result=20
```

4.4 编写追加后缀脚本

创建 test 文件夹存放测试文件若干进行测试，带参数执行批处理程序

得到测试结果如下所示，可以看到 test 文件夹其中的两个子目录并没有被追加后缀，其余文件正常被追加后缀，追加时间格式为-年-月-日-时-分，重新运行批处理程序，后缀没有追加而是更新了时间，说明批处理程序设计成功！

```
shiftw@shiftw-virtual-machine:~$ sudo ./postfix.sh test/
shiftw@shiftw-virtual-machine:~$ ls test/
tes1-2023-11-21-12-24.txt  test3-2023-11-21-12-24.txt  test5
test2-2023-11-21-12-24.py  test4
shiftw@shiftw-virtual-machine:~$ sudo ./postfix.sh test/
shiftw@shiftw-virtual-machine:~$ la test/
tes1-2023-11-21-12-28.txt  test3-2023-11-21-12-28.txt  test5
test2-2023-11-21-12-28.py  test4
```


五、实验错误排查和解决方法

5.1 编写 MBR 引导程序

1. 进入引导程序后，没有出现预期的字样。因为还需要返回终端程序，执行按下 **c+enter** 的操作。

5.2 编译 linux 内核

1. 第一次尝试 **make** 编译的时候中间报错“没有规则可制作目标”，还需要更改源码文件夹里的 **.config** 配置文件，将 **CONFIG_SYSTEM_TRUSTED_KEYS** 修改为空字符串，才能正常编译。且在查找该字段的时候 **vim** 的查找功能失效，不知道是什么原因，改用 **pluma** 查找成功。

2. 第一次用 **make -j4** 编译过程很慢，以及磁盘分配空间不够大。查到资料应该在一开始就尽量分配较大的磁盘和内存，同时真是磁盘空间也要预留足够，并且分配 4 个以上的内核。这是本次实验最大的疏忽，以为内存够就行了，之后再使用磁盘扩展似乎也用处不大了，导致编译阶段耗时巨大。下次创建虚拟机的时候还是要注意尽量配置更好的性能和大内存。

3. **ubuntukylin** 虚拟机自动挂机锁屏，导致本就耗时巨大的编译过程频繁被打断。一开始尝试修改虚拟机电源相关选项，没有解决，搜索之后尝试关掉屏保的自动开启，成功解决问题。



4. 编译完成后用 **reboot** 命令重启，没有看到高级选项选择内核而是弹出登陆选项，先不要登录，而是点击右下角的电源键选择重启，按下 **tab+shift**，即可看到高级选项。

5.3 系统调用添加

1. 添加系统调用函数源码时对 `SYSCALL_DEFINE(n)` 中的 `n` 理解不清楚，`n` 实际上指明了系统调用的参数个数，不可随意修改。

2. 一开始系统调用号的添加、系统调用函数的声明以及系统调用函数实现三者中的函数名没有对应好，导致添加失败报错，之后注意细节更改了就成功了。

5.4 编写追加后缀脚本

1. 一开始疑惑于如何区分文件夹和文件，想到的操作是遍历目标路径，对每个项目都进行判断是文件夹还是文件。后来发现可以使用只显示文件而不显示文件夹的命令 `ls -l | grep ^[^\d] | awk '{print $9}'`。

2. 对于如何区分加没加过后缀的文件，从参考资料中得到一个思路即打标记，这样只需要根据标记的有无决定是更新后缀还是加后缀，但增加了冗余的信息且没有完全满足要求。之后想到其实不必要打标记区分，对每个文件都执行删除后缀再添加后缀的操作就可以了。

3. 在处理文件后缀名的时候，考虑到文件名中可能有多个“.”干扰文件后缀的识别，将 `${files/. *}` 改为 `${files%.*}`。

六、实验参考资料和网址

(1) 教学课件

(2) https://blog.csdn.net/qq_46106285/article/details/121507087

(3) <https://www.5lcto.com/article/663841.html>

(4) <https://www.cnblogs.com/xiaocen/p/3717993.html>

(5) <https://www.cnblogs.com/zhang-jun-jie/p/9266858.html>

(6) <https://www.jb51.net/article/114157.htm>