# 《操作系统原理》实验报告

姓名	熊睿	学号	U202011107	专业班级	网安 2004	时间	2022.12.25
----	----	----	------------	------	---------	----	------------

# 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)下裁剪和编译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-2022-11-21-20-42。

#### 三、实验过程

#### 3.1 linux 内核编译

1) 获取内核源码

在官网下载 linux-5.10.155 内核源码压缩包, 拖入虚拟机中, 使用 tar 命令解压进/usr/src 目录。

2) 配置内核特性

进入解压后生成的/usr/src/linux-5.10.155 文件夹使用 make menuconfig 命令配置内核特性,不用做修改。

#### 3)编译内核

使用 make-j 8 命令编译内核, 8 为分配的物理内核数量两倍,可以提高编译速度。

# 图 1 编译内核

	crc32defs.h	.decompress unlz4.o.cmd	.idr.o.cmd	locking-selftest-mutex.h	objagg.ko	.sbitmap.o.d	test parman.c
	crc32.0	decompress unlama.c	.idr.o.d	locking-selftest-rlock.h	.obiagg.ko.cmd	scatterlist.c	test printf.c
42	.crc32.o.cmd	decompress unlzma.o	inflate.c	locking-selftest-rlock-harding.h	objagg, mod	scatterlist.o	test rhashtable.c
rgv split.c	.crc32.o.d	.decompress unlzma.o.cmd	interval tree.c	locking-selftest-rlock-softing.h	objagg.mod.c	.scatterlist.o.cmd	test siphash.c
rgv split.o	crc32table.h	decompress unlzo.c	interval tree.o	locking-selftest-rsem.h	.objagg.mod.cmd	.scatterlist.o.d	test sort.c
argy split.o.cmd	crc32test.c	decompress unlzo.o	.interval tree.o.cmd	locking-selftest-rtmutex.h	objagg.mod.o	seg buf.c	test stackinit.c
argy split.o.d	crc4.c	,decompress unlzo,o,cmd	interval tree.o.d	locking-selftest-softirg.h	. obiagg.mod.o.cmd	seg buf.o	test static key bas
shldi3.c	crc4.ko	decompress unxz.c	interval tree test.c	locking-selftest-spin.h	objagg.o	.seg buf.o.cmd	test static keys.c
shrdi3.c	.crc4.ko.cmd	decompress unxz.o	iomap.c	locking-selftest-spin-harding.h	.objagg.o.cmd	.seg buf.o.d	test string.c
snl decoder.c	crc4.mod	.decompress unxz.o.cmd	iomap copy.c	locking-selftest-spin-softirg.h	.objagg.o.d	sg pool.c	test-string helpers
sn1 decoder.o	crc4.mod.c	decompress unzstd.c	iomap copy.o	locking-selftest-wlock,h	of-reconfig-notifier-error-inject.c	sg pool.o	test strscpv.c
asn1 decoder.o.cmd	,crc4.mod.cmd	decompress unzstd.o	.iomap copy.o.cmd	locking-selftest-wlock-harding.h	oid registry.c	,sq pool,o,cmd	test sysctl.c
asn1 decoder.o.d	crc4.mod.o	.decompress unzstd.o.cmd	.iomap copy.o.d	locking-selftest-wlock-softing.h	oid registry data.c	.sq pool.o.d	test ubsan.c
ssoc array.c	.crc4.mod.o.cmd	devres.c	iomap.o	locking-selftest-wsem.h	oid registry.o	sq split.c	test user copy.c
ssoc array.o	crc4.o	devres.o	.iomap.o.cmd	lockref.c	.oid registry.o.cmd	shal.c	test uuid.c
assoc array.o.cmd	.crc4.o.cmd	.devres.o.cmd	.iomap.o.d	lockref.o	.oid registry.o.d	shal.o	test vmalloc.c
tomic64.c	.crc4.o.d	.devres.o.d	iommu-helper.c	.lockref.o.cmd	once.c	,shal,o,cmd	test xarrav.c
tomic64 test.c	crc64.c	digsig.c	iommu-helper.o	.lockref.o.d	once.o	,shal,o,d	textsearch, c
udit.c	crc64.ko	digsig.o	.iommu-helper.o.cmd	logic pio.c	.once.o.cmd	show mem.c	textsearch.o
cd.c	.crc64.ko.cmd	.digsig.o.cmd	iov iter.c	logic pio.o	.once.o.d	show mem.o	.textsearch.o.cmd
icd.o	crc64.mod	.digsig.o.d	iov iter.o	.logic pio.o.cmd	packing.c	.show mem.o.cmd	.textsearch.o.d
bcd.o.cmd	crc64.mod.c	dim	.iov iter.o.cmd	lru cache.c	packing.o	siphash.c	timerqueue.c
bcd.o.d	.crc64.mod.cmd	dump stack.c	.iov iter.o.d	lru cache.ko	.packing.o.cmd	siphash.o	timerqueue.o
ch.c	crc64.mod.o	dump stack.o	irg poll.c	.lru cache.ko.cmd	.packing.o.d	.siphash.o.cmd	.timerqueue.o.cmd
ch.ko	.crc64.mod.o.cmd	.dump stack.o.cmd	irg poll.o	lru cache.mod	parman.c	.siphash.o.d	.timerqueue.o.d
bch.ko.cmd	crc64.o	.dump stack.o.d	.irq poll.o.cmd	lru cache.mod.c	parman.ko	smp processor id.c	ts bm.c
ch.mod	.crc64.o.cmd	dynamic debug.c	.irq poll.o.d	.lru cache.mod.cmd	.parman.ko.cmd	sort.c	ts bm.ko
ch.mod.c	.crc64.o.d	dynamic debug.o	irq regs.c	lru cache.mod.o	parman.mod	sort.o	.ts bm.ko.cmd
bch.mod.cmd	crc64table.h	.dynamic_debug.o.cmd	irq_regs.o	.lru_cache.mod.o.cmd	parman.mod.c	.sort.o.cmd	ts_bm.mod
ch.mod.o	crc7.c	.dynamic debug.o.d	.irq regs.o.cmd	lru cache.o	.parman.mod.cmd	.sort.o.d	ts bm.mod.c
bch.mod.o.cmd	crc7.ko	dynamic queue limits.c	.irq regs.o.d	.lru cache.o.cmd	parman.mod.o	stackdepot.c	.ts bm.mod.cmd
ch.o	.crc7.ko.cmd	dynamic queue limits.o	is single threaded.c	.lru cache.o.d	.parman.mod.o.cmd	stmp device.c	ts bm.mod.o
bch.o.cmd	crc7.mod	.dynamic queue limits.o.cmd	is single threaded.o	lshrdi3.c	parman.o	string.c	.ts bm.mod.o.cmd
bch.o.d	crc7.mod.c	.dynamic queue limits.o.d	.is single threaded.o.cmd	Tz4	.parman.o.cnd	string helpers.c	ts bm.o
itfield kunit.c	.crc7.mod.cmd	earlycpio.c	kasprintf.c	lzo	.parman.o.d	string helpers.o	.ts bm.o.cmd
itmap.c	crc7.mod.o	earlycpio.o	kasprintf.o	Makefile	parser.c	.string helpers.o.cmd	ts_fsm.c
itmap.o	.crc7.mod.o.cmd	.earlycpio.o.cmd	.kasprintf.o.cmd	math	parser.o	.string helpers.o.d	ts fsm.ko
bitmap.o.cmd	crc7.o	errname.c	.kasprintf.o.d	memcat p.c	.parser.o.cmd	string.o	.ts fsm.ko.cmd
bitmap.o.d	.crc7.o.cmd	errname.o	Kconfig	memcat p.o	.parser.o.d	.string.o.cmd	ts fsm.mod
itrev.c	.crc7.o.d	.errname.o.cmd	Kconfig.debug	.memcat_p.o.cmd	pci_iomap.c	.string.o.d	ts_fsm.mod.c
itrev.o	crcB.c	error-inject.c	Kconfig.kasan	memcat p.o.d	pci igmap.o	strncov from user.c	.ts fsm.mod.cmd

图 2 编译完成

# 4) 安装内核、安装内核模块

使用 make modules\_install 和 make install 命令安装内核、安装内核模块。



图 3 模块编译完成

#### 5) 重启后选择新内核

重启后选择新内核,并用 uname -mrs 命令验证新内核是否已经启用。

```
certs COPYING fs init lib .m:
root@boots-VMware-Virtual-Platform:/usr/src/linux-5.10.155# uname -mrs
Linux 5.10.155 x86_64
root@boots-VMware-Virtual-Platform:/usr/src/linux-5.10.155# S
```

图 4 新内核版本



图 5 开机时可以选择启动哪个内核



图 6 原来版本内核内核

# 3.2 添加 linux 系统调用

# 1)编写函数调用

在/usr/src/linux-5.10.155/kernel 中的 sys.c 中增加函数调用。

```
return -EFAULT;
    return 0;
}
SYSCALL_DEFINE2(SSD_Add,int,x,int,y){
        printk("%d",x+y);
        return 0;
}
SYSCALL_DEFINE3(SSD_Max,int,a,int,b,int,c){
        if(a>b)b=a;
        if(b>c)c=b;
        printk("%d",c);
        return 0;
}
#endif /* CONFIG_COMPAT */
```

图 7 新增函数调用

#### 2) 系统调用函数声明

在/usr/src/linux-5.10.155/include/linux 中的 syscalls.h 中增加函数声明。

图 8 新增函数声明

#### 3)声明系统调用编号

在/usr/src/linux-5.10.155/arch/x86/entry/syscalls 的 syscall 64.tbl 中新增声明。

```
sys_pidfd getfd
438
                pidfd getfd
        common
439
                faccessat2
        common
                                        sys_faccessat2
                process madvise
                                        sys_process madvise
498
                SSD Add
                                        sys SSD Add
499
        64
                SSD Max
                                        sys SSD Max
# Due to a historical design error, certain syscalls are numbered differently
# in x32 as compared to native x86_64. These syscalls have numbers 512-547.
# Do not add new syscalls to this range. Numbers 548 and above are available
# for non-x32 use.
```

图 9 注册新调用

```
__SYSCALL(_NR_faccessat2, sys_faccessat2)
#define __NR_process_madvise 440
__SYSCALL(_NR_process_madvise, sys_process_madvise)
#define __NR_ssd_add 498
__SYSCALL(_NR_ssd_add, sys_SSD_Add)
#define __NR_ssd_max 499
__SYSCALL(_NR_ssd_max, sys_SSD_Max)

#undef __NR_syscalls
#define __NR_syscalls 441
```

图 10 ID 声明

4)编写应用程序测试新系统调用

图 11 测试程序编写

#### 图 12 运行测试程序

# 3.3 编写批处理程序

#### 1)编写批处理程序

```
for each in $1/*
       #文件夹,不改名
      if [ -d $each ]; then
               echo "$each is a directory."
       #文件,改名
       elif [ -f $each ]; then
               echo "$each is a normal file."
               #通配符判断是否已经有日期后缀
               if [[\$each == *-20[0-9][0-9]-[012][0-9]-[0123][0-9]-[012][0-9]-
[0-5][0-9]-[0-5][0-9]]; then
                      echo "$each >> ${each%-*}-`date +%Y-%m-%d-%H-%M-%S`"
                      mv $each ${each%%-*}-`date +%Y-%m-%d-%H-%M-%S`
               else
                      echo "$each >> $each-`date +%Y-%m-%d-%H-%M-%S`"
                      mv $each $each-`date +%Y-%m-%d-%H-%M-%S`
               fi
       fi
done
```

图 13 编写批处理程序

# 2)运行批处理程序

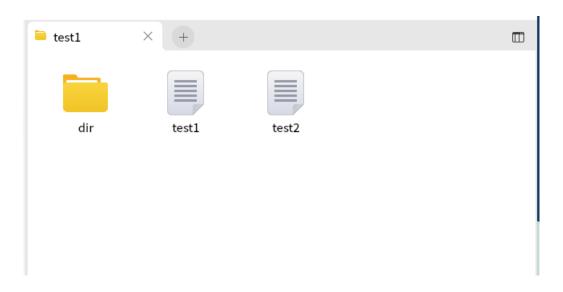


图 14 运行之前

```
oots@boots-VMware-Virtual-Platform:~/果面$ ./change.sh ./test1/test1/a-2022-11-23-14-53-34 is a normal file.
/test1/a-2022-11-23-14-53-34 >> ./test1/a-2022-12-26-13-51-15
/test1/b is a directory.
/test1/c-2022-11-23-14-53-34 is a normal file.
/test1/c-2022-11-23-14-53-34 >> ./test1/c-2022-12-26-13-51-15
oots@boots-VMware-Virtual-Platform:~/桌面$ ./change.sh ./test1
/test1/dir is a directory.
/test1/test1 is a normal file.
/test1/test2 is a normal file.
/test1/test2 is a normal file.
/test1/test2 >> ./test1/test2-2022-12-26-13-55-15
```

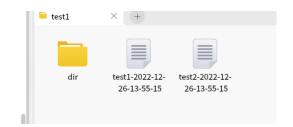


图 15 运行之后

# 四、实验结果

## 4.1 Linux 内核编译

```
certs COPYING fs init lib .m:
root@boots-VMware-Virtual-Platform:/usr/src/linux-5.10.155# uname -mrs
Linux 5.10.155 x86_64
root@boots-VMware-Virtual-Platform:/usr/src/linux-5.10.155# S
```

图 16 新内核编译成功版本

## 4.2 添加 Linux 系统调用

图 17 添加 Linux 系统调用

#### 4.3 编写批处理程序

```
oots@boots-VMware-Virtual-Platform:~/桌面$ ./change.sh ./test1
/test1/a-2022-11-23-14-53-34 is a normal file.
/test1/a-2022-11-23-14-53-34 >> ./test1/a-2022-12-26-13-51-15
/test1/b is a directory.
/test1/c-2022-11-23-14-53-34 >> ./test1/c-2022-12-26-13-51-15
oots@boots-VMware-Virtual-Platform:-/桌面$ ./change.sh ./test1
/test1/dir is a directory.
/test1/test1 is a normal file.
/test1/test1 >> ./test1/test1-2022-12-26-13-55-15
/test1/test2 is a normal file.
/test1/test2 >> ./test1/test2-2022-12-26-13-55-15
```

图 18 添批处理程序运行成功

# 五、实验错误排查和解决方法

#### 5.1 Linux 内核编译

#### 1) 内核选择错误

我的优麒麟的 Linux 版本是 Linux-5.10.0, 而我一开始选择的编译的内核是 Linux-5.9.6, 导致 make modules\_install 时发生错误,提示没有 modules.order 文件或是目录,更换内核 5.10.155 后可以执行成功,所以编译的时候尽量选择版本相近的内核.

```
sed: 无法读取 modules.order: 没有那个文件或目录 make: *** [Makefile:1416: modinst] 错误 2
```

图 19 编译失败错误

# 2) 读写权限错误

根据提示可以看到 lib/raid6/mktables:Permission denied,需要给 lib/raid6/mktables 这个文件读写权限,利用 chmod 777 指令进行操作,或者直接用 root 账号进行实验。

```
scripts/checksyscalls.sh
         CALL
         CALL
                 scripts/atomic/check-atomics.sh
         DESCEND objtool
INTAINERS
         CHK
                 include/generated/compile.h
         CHK
                 kernel/kheaders data.tar.xz
         TABLE
                lib/raid6/tables.c
        /bin/sh: 1: lib/raid6/mktables: Permission denied
       make[2]: *** [lib/raid6/Makefile:90: lib/raid6/tables.c] 错误 126
       make[2]: *** 正在删除文件"lib/raid6/tables.c"
       make[1]: *** [scripts/Makefile.build:500: lib/raid6] 错误 2
       make: *** [Makefile:1784: lib] 错误 2
```

图 20 读写权限错误

#### 5.2 添加 Linux 系统调用

#### 1) 不理解如何观察 Linux 系统调用

按照实验要求,进行 test.c 的编写和编译生成 exe 文件,执行 exe 文件后没有在终端中直接观察到变化和输出,通过和同学交流讨论,才知道,可以用 dmesg 和 dmesg -c 来观察,并且运行程序要用./(指代当前目录)来运行。

```
boots@boots-VMware-Virtual-Platform:~/桌面$ gcc test.c -o 111
boots@boots-VMware-Virtual-Platform:~/桌面$ 111
Could not find command-not-found database. Run 'sudo apt update' to populate it.
111: 未找到命令
boots@boots-VMware-Virtual-Platform:~/桌面$ ./111
ret:0
ret:0
boots@boots-VMware-Virtual-Platform:~/桌面$
```

图 21 无法观察输出

#### 5.3 编写批处理程序

1) 脚本文件执行命令不熟悉

还是脚本命令执行不是很熟悉,在查找相关资料后了解,对于 sh 文件的执行,需要指定一个文件路径,例如./test1 表面当前路径下的 test1 文件夹,如果只执行./change.sh 默认对/路径进行执行显示如下图所示。

```
boots@boots-VMware-Virtual-Platform:~/桌面$ ./change.sh
/backup is a directory.
/bin is a directory.
/boot is a directory.
/cdrom is a directory.
/data is a directory.
/dev is a directory.
/etc is a directory.
/home is a directory.
/lib is a directory.
/lib32 is a directory.
/lib64 is a directory.
/libx32 is a directory.
/lost+found is a directory.
/media is a directory.
/mnt is a directory.
/opt is a directory.
/proc is a directory.
/root is a directory.
/run is a directory.
/sbin is a directory.
/srv is a directory.
/sys is a directory.
```

图 22 change.sh 执行输出

我们需要对自己创建的文件夹里的内容进行修改,文件夹不修改,文件修改成执行是的日期名字,所以使用./change.sh./test1命令,观察到预期的结果。

图 23 ./change.sh ./test1 执行输出

# 六、实验参考资料和网址

- (1) 教学课件
- (2) Adding a New System Call The Linux Kernel documentation
- (3) https://blog.csdn.net/qq 46106285/article/details/121507087

(4) 《一个操作系统的实现》