操作系统课程设计报告

信息与科学技术学院

张鹏鹏 040630416

2009-6-25

一，个人背景

记得大一时借过一本书：《自己动手写操作系统》，那时，对于一个刚学 C 语言的我来说，阅读都有难度。一颗好奇心在作祟，却让我第一次了解到操作系统的结构。同时，我也知道了，操作系统其实并不神秘。一直想自己动手写个操作系统，怎奈何，知识水平有限，未能如愿。

一晃大三了，这学期的操作系统课程让我有幸系统地学习了操作系统的相关理论。而这次课设正好是练兵的好机会，可惜，还是下不了决心，先完成老师布置的作业再说。

一直喜欢玩玩系统，虚拟机给了我放手去干的机会。试过几款虚拟机，最终还是选择了 VirtualBox，因为：一，开源，免费；二，支持的 hosts 广；三，支持的 guest os 多；四，运行速度快。我电脑是 Ubuntu+winXP 双系统，两系统下都装了 VirtualBox，共享虚拟硬盘，这样，两个系统里可以启用同一个客户机。而且我在虚拟机里装个 XP，那速度比我在实际机器里装还要快，怪哉。

而这次实验中用的 QEMU 虚拟机，我却不怎么了解。不过，幸好 QEMU 也有 linux 版（压缩包中 scripts 文件夹里的一大堆 sh 文件给了我提示），所以我完全可以在 Ubuntu 中做这个课设，不用回到 windows。本来想在 ubuntu 里 mount 那三个 img 文件，直接访问里面的东西，这样就可以用 ubuntu 里的 vi，gcc 和 gdb，可惜我对 mount 命令不熟，没有成功，而 gdb 在 qemu 的宿主机里没给，所以我调试花了好大力气。

VI，我非常喜欢的一个编辑器，ubuntu 和 XP 里都装了，有人说，这会变成一种信仰。还好，我仅仅写一些小的 C/C++程序时用到，方便，而且功能强大！给的 qemu 宿主机中的 vi 已经相当不错了，欣慰。

强调一点，我没有读过 linux 源代码。课设中我还发现，linux 2.4 和 2.6 的源代码有很大的区别，至少这次课设中牵扯的那些代码是这样的。

二，课设中遇到的问题，解决方法以及感想

1.UNIX 系统编程环境介绍

这部分我之前基本已经掌握，主要学习了这三个命令：sync,dd,touch。

2.实验环境介绍

这部分按部就班去做就行了，我是在 Ubuntu Linux 下装的 qemu，所以我修改并使用这两个文件：

launch-host.sh ： qemu -m 64 -fda ../floppy.img -hda ../host.img -hdb ../target.img -boot c

launch-target.sh：qemu -m 64 -fda ../floppy.img -hda ../target.img -boot a

3.生成目标机

分区命令以前用过，学习了一下格式化，没有遇到什么问题。

4.Hello World!

```
UnixLite: A Light Weight Operating System Written in C++
COPYRIGHT (C) 2009 NUAA CS DEPT.
Total Memory Size is 64 Meg
Hi,I am Roc! My name is zhangpengpeng:040630416.
bios32 service directory structure at 0lxf9ef0
bios32 service directory at 0xf9f00
PCI BIOS revision 2.10 entry at 0xc00f9f40
current time is 2009/6/25 13:24:52
ne2k.cc: PCI BIOS report RealTek-8029 at IO=0xc100 IRQ=11
hda cyl:130, head:16, sect:63 (logical parameter)
mount root fs ok
start /sbin/init ... ok
executing /etc/rc
SIOCSIFADDR: Invalid argument
login: _
```

5.系统调用的原理

虽然写了个加法的系统调用，实际是让系统自己做了一下 a+b 。系统调用就是把自己写的一个函数封装起来，作为系统调用让应用程序去调用。

这部分由于给内核增加了一个 plugin 文件夹，所以编译时先 make dep。下面几题都改变了编译依赖关系，所以都要加 make dep

6.UnixLite 内部函数接口

7.实现信号灯操作的系统调用

include 时，头文件顺序不要改变，有同学改变了就遇到问题，改回来问题解决。我估计这几个头文件里，有那么几个变量和函数，有着先声明后实现而后调用的依赖关系，所以顺序不能变。具体是哪几个，我没有去找。

由于没有 gdb，这部分调试非常费力。我先把源文件写好，再把与系统调用的有关的注释掉，然后作为普通的应用函数，在宿主机里写测试主函数进行调试，这里调试时，加了很多 printf 语句，最多时，执行一句 printf 一下。待到没有问题时，再注释掉调试语句，改回来，编译内核，写目标机，很顺利。

```
semaphore_t *semap;
semap=semaphore_list;
//printf("1\n");

while(semap!=NULL && semap->next!=NULL) //find the same name node
{
        if(strcmp(semap->name,name)==0)
                return -1;
        semap=semap->next;
}
//printf("2\n");
if(semap!=NULL && strcmp(semap->name,name)==0)  //test the last one nod
        return -1;
//printf("3\n");
semaphore_t *s =new semaphore_t;
//printf("4\n");
//printf("%s %s \n",s->name,name);
strcpy(s->name,name);
//printf("%s\n",s->name);
s->value=value;
s->next=NULL;
//printf("5\n");
```

写系统调用的测试程序时，在文件操作中，一开始忘记用 rewind()回调文件读写指针，为了这个不起眼的小 bug，我调试了将近两小时，近乎崩溃。

为了方便调试，我还加了个 show_sema()系统调用，用于查看所有的信号灯。

```
bash# cd test/testsema/
bash# ls
buffer      cpt.sh      destroy     product     showsema
consume     create      destroy.c   product.c   showsema.c
consume.c   create.c    makefile    sema.h
bash# make c
./create mutex 1
success to create semaphore mutex , 1
./create empty 8
success to create semaphore empty , 8
./create full 0
success to create semaphore full , 0
./showsema
semaphore:      mutex,1
semaphore:      empty,8
semaphore:       full,0
```

```
bash# make g1
./product 10 &
./consume 10
        Produce:0
        Produce:1
        Produce:2
        Produce:3
        Produce:4
        Produce:5
        Produce:6
        Produce:7
                Consume:0
                Consume:1
                Consume:2
                Consume:3
                Consume:4
                Consume:5
                Consume:6
                Consume:7
        Produce:8
        Produce:9
                Consume:8
                Consume:9
```

8.实现进程间通信的系统调用

这个和第 7 题差不多，主要问题是对状态的控制。比如 in++之后与 out 相等了，表明已经满了。这时需要阻塞进程，问题是应该什么时候阻塞。如果马上阻塞，那么发送 10 个 接受 10 个后 本来测试完成了，可接受程序会被阻塞，无法正常结束。如果循环体开始时检查 in 和 out 判断阻塞与否，那么一开始两个测试程序就什么也不做，都被阻塞住。问题出在 in 和 out 相等时无法判定是空还是满。

我的解决方法是，增加一个 state 变量，0 空，1 不空不满，2 满，循环体开始时根据 state 判定是否阻塞，循环体结束时，更改 state 状态，问题解决。

```
asmlinkage int send_msg(char *name , int msg)
{
        mail_box_t *mb;
        mb=mail_box_list;
        while(mb!=NULL) //test each node
        {
                if(strcmp(mb->name,name)==0)
                {
                        if(mb->state==2)        //full
                        {
                                //printf("full\n");
                                mb->waitq.wait();
                        }
                        mb->state=1;
                        mb->msgq[mb->in]=msg;
                        printf("Send_msg:%d\n",msg);
                        mb->in=(mb->in+1)%CAPACITY;
                        if(mb->in==mb->out)
                                mb->state=2;
                        mb->waitq.broadcast();
                        return 0;
                }
                mb=mb->next;
        }
```

```
./send mybox 11 &
Send_msg:0
Send_msg:1
Send_msg:2
Send_msg:3
Send_msg:4
Send_msg:5
Send_msg:6
Send_msg:7
./recv mybox 11
Receive_msg:0
Receive_msg:1
Receive_msg:2
Receive_msg:3
Receive_msg:4
Receive_msg:5
Receive_msg:6
Receive_msg:7
Send_msg:8
Receive_msg:8
Send_msg:9
Receive_msg:9
Send_msg:10
Receive_msg:10
bash#
```

9.Ram 盘驱动程序

改错：# mknod /dev/ram b 8 0 把 8 改成 7，看这一句

return (major(dev) < MAXBLKDEV) ? blkdevvec[major(dev)] : NULL;

8 时返回 NULL，7 时才能正常返回。

最容易忘记的是：

// 注意：在 init/main.cc 中调用该函数,必须在 doglobalctors 和 freebm 这两个函数之间调用

void rddevinit()

另外 read,write 系统调用返回的是实际读写的字节量，所以要求返回 3，只有两种办法：一，ram 大小为 3byte，只能写三个 byte；二，read,write 函数的第三个参数改为 3，只要求写 3 个 byte，我用的是第二种。

另外一切顺利。

```
bash# make
./write
write 3 bytes
./read
read 3 bytes,[XYZ]
bash#
```

最后说明一下，我所以给目标机的测试函数，都是在宿主机里写，专门放到一个 test 文件夹中，再 cp 给目标机

```
(none) login: root
No mail.
[root@(none) root]# ls
floppy.img  kernel/  make/  rootfs.tar.gz  test/
[root@(none) root]# cd test/
[root@(none) test]# ls
msgdbg/  semadbg/  testadd/  testasm/  testmsg/  testsema/
[root@(none) test]# _
```

三，结语

这次课设收获很大，让我对操作系统有了更深层次的理解。提高了动手能力，还不得已了看了一部分 linux 的核心代码。将来有机会自己动手写个操作系统，简单点的也行。