实验 5 内核定制与编程

一、预备知识

1、 编译内核

（1） Linux 内核源文件缺省位置：/usr/src/linux。

（2） make mrproper 该命令确保源代码目录下没有不正确的.o 文件以及文件的互相依赖。

（3） make config 或者 make menuconfig 或者 make xconfig 配置内核。选择相应的配置时，有三种选择，它们分别代表的含义如下：

Y－将该功能编译进内核

N－不将该功能编译进内核

M－将该功能编译成可以在需要时动态插入到内核中的模块

（4） make dep 生成相关性。

（5） make clean 清除一些现有文件。

（6） make bzImage 编译内核，花费时间较长。

（7） 可选 #make modules 生成相应的模块。

#make modules_install 把模块拷贝到需要的目录中

#depmod –a 生成模块间的依赖关系

注意：在进行配置的过程中，只有回答 Enable loadable module support (CONFIG_MODULES)时选了"Yes"，该选项才是必要的。

（8） 最终将生成/linux/arch/i386/boot/bzImage，把 bzImage 拷贝到/home 中。

（9） 创建引导。如果采用 LILO，那么在/etc/lilo.conf 文件中添加 image=/home/bzImage Label=newLinux。

（10） [root@主机名]# /sbin/lilo –v ，确认对/etc/lilo.conf 的编辑无误。

（11） [root@主机名]# shutdown -r now，重启后出现 LILO 时按 TAB 键，输入 newLinux，此时新的内核发挥作用。

2、 创建 Linux 系统调用

（1） 可以通过编辑/kernel 目录下的 makefile 文件来添加一个包含我们自己的函数的文件，但更简单的方法是在源代码树中已经存在的文件内加入我们的函数代码。

在 kernel/sys.c 文件的某个位置添加以下代码

```
asmlinkage long sys_ourcall(long num)
{
printk(“Inside our syscall num=%d \n”,num);
return (num+1);
}
```

（2） 修改系统调用表的内容：

[root@主机名]# vi arch/i386/kernel/entity.S

……

.long sys_ourcall /*our syscall will be 274*/

……

（3） 文件 include/asm/unistd.h 将系统调用和它们在 sys_call_table 中的位置编号联系起来。

……

#define _NR_ourcall 274

……

（4） 创建一个用户程序来检测这个新的系统调用。

[root@主机名]# vi ourusercall.c

#include <stdio.h>

#include <stdlib.h>

#include “/usr/include/asm/unistd.h”

_syscall(long,ourcall,long,num);

/*使用了/unistd.h 文件中的_syscall(type,name,type1,name1)宏，这个宏借助适当的参数来处理 int 0x80 调用*/

（5） 测试

[root@主机名]# vi mytest.c

main()

{

printf(“our syscall → num in=5,num out=%d\n”),ourcall(5));

}

二、实验内容

1、 在官网中下载一个较新的稳定的内核版本。

2、 编译定制一个新的内核，写出编译过程。

3、 在新内核中创建一个系统调用 ourcall，要求调用时能够输出自己的姓名和学号，并写出过程和测试结果。