Lab1

吴非 519021910924

内核环境：5.13.0-35-generic

模块一

实验过程：

由于课上老师讲的比较浅，首先回顾一下课件，并从网上搜索类似的题目进行查看。然后开始代码编写。由于自己一直使用 C++,C 编程尤其是 linux 内核编程并不熟悉，因此这个看起来简单的模块实现花了很久。

一些具体实现细节：

考虑到 insmod 执行 init 函数，会立刻执行计算逻辑，而 echo 后又需要再计算一遍，因此我把计算的逻辑另写一个 calculate 函数，提高代码复用性。

实验截图：

模块 1 加法：

```
xp/CS353-2022-Spring/project1$ sudo insmod calc.ko operand1=2 operand2=1,2,3,4
operator=add
(base) wf@wf-virtual-machine:~/linux_files/linux_exp/CS353-Linux-Kernel/linux_e
xp/CS353-2022-Spring/project1$ dmesg
[ 1650.597281] welcome to this calculator
[ 1650.597284] operand1:2
[ 1650.597285] operator = add
[ 1650.597288] Create /proc/519021910924 Success!
[ 1650.597290] Create /proc/519021910924/calc Success!
[ 1650.597291] 加法完成
(base) wf@wf-virtual-machine:~/linux_files/linux_exp/CS353-Linux-Kernel/linux_e
xp/CS353-2022-Spring/project1$ cat /proc/519021910924/calc
3,4,5,6(base) wf@wf-virtual-machine:~/linux_files/linux_exp/CS353-Linux-Kernel/
linux_exp/CS353-2022-Spring/project1$ echo 3 > /proc/519021910924/calc
(base) wf@wf-virtual-machine:~/linux_files/linux_exp/CS353-Linux-Kernel/linux_e
xp/CS353-2022-Spring/project1$ cat /proc/519021910924/calc
4,5,6,7(base) wf@wf-virtual-machine:~/linux_files/linux_exp/CS353-Linux-Kernel/
linux_exp/CS353-2022-Spring/project1$
```

乘法：

```
xp/CS353-2022-Spring/project1$ sudo insmod calc.ko operand1=2 operand2=1,2,3,4
operator=mul
(base) wf@wf-virtual-machine:~/linux_files/linux_exp/CS353-Linux-Kernel/linux_e
xp/CS353-2022-Spring/project1$ dmesg
[ 1486.631611] welcome to this calculator
[ 1486.631613] operand1:2
[ 1486.631614] operator = mul
[ 1486.631618] Create /proc/519021910924 Success!
[ 1486.631620] Create /proc/519021910924/calc Success!
[ 1486.631621] 乘法完成
(base) wf@wf-virtual-machine:~/linux_files/linux_exp/CS353-Linux-Kernel/linux_e
xp/CS353-2022-Spring/project1$ cat /proc/519021910924/calc
2,4,6,8(base) wf@wf-virtual-machine:~/linux_files/linux_exp/CS353-Linux-Kernel/
linux_exp/CS353-2022-Spring/project1$ echo 3 > /proc/519021910924/calc
(base) wf@wf-virtual-machine:~/linux_files/linux_exp/CS353-Linux-Kernel/linux_e
xp/CS353-2022-Spring/project1$ cat /proc/519021910924/calc
(base) wf@wf-virtual-machine:~/linux_files/linux_exp/CS353-Linux-Kernel/linux_e
xp/CS353-2022-Spring/project1$ cat /proc/519021910924/calc
3,6,9,12(base) wf@wf-virtual-machine:~/linux_files/linux_exp/CS353-Linux-Kernel
/linux_exp/CS353-2022-Spring/project1$
```

（我注意到第一个 cat 没输出，因为我是直接复制很多命令一起执行，应该是这时候没有计算完成）

操作符合法性检查:

```
xp/CS353-2022-Spring/project1$ sudo insmod calc.ko operand1=2 operand2=1,2,3,4
operator=foo
(base) wf@wf-virtual-machine:~/linux_files/linux_exp/CS353-Linux-Kernel/linux_e
xp/CS353-2022-Spring/project1$ dmesg
[ 1699.368961] welcome to this calculator
[ 1699.368963] operand1:2
[ 1699.368964] operator = foo
[ 1699.368967] Create /proc/519021910924 Success!
[ 1699.368970] Create /proc/519021910924/calc Success!
[ 1699.368971] 要求的操作不是加也不是乘，出错！(操作符检查)
[ 1699.368971] do_init_module: 'calc'->init suspiciously returned 1, it should
follow 0/-E convention
               do_init_module: loading module anyway...
[ 1699.368974] CPU: 1 PID: 6924 Comm: insmod Tainted: G           OE     5.13.0
-35-generic #40~20.04.1-Ubuntu
[ 1699.368976] Hardware name: VMware, Inc. VMware Virtual Platform/440BX Deskto
p Reference Platform, BIOS 6.00 07/22/2020
[ 1699.368977] Call Trace:
[ 1699.368978]  <TASK>
```

另外一些细节在代码注释中解释。

模块二:

实验过程:

由于这个写在用户态，因此比模块二要简单一些，主要逻辑是通过 readdir 函数递归遍历 proc 下的目录，找到开头是数字的目录（进程 pid 号），打印目录名称的同时读出这个目录下我想要的特定文件内容并打印，最后一个循环结束打印一个\n 实现换行。

一些具体实现细节:

Isnull 函数判断 cmdline 文件是否为空，为空则读取 comm，否则读 cmdline；
findstat 函数从 stat 文件里逐个读取字符，直到读到右括号，右括号右侧第二个数据是我想要输出的数据。
另外一些细节在代码注释中解释。

实验截图：

```
pid: 7405       S       /usr/share/code/code
pid: 7452       S       /home/wf/.vscode/extensions/ms-vscode.cpptools-1.7.1/bi
n/cpptools
pid: 7491       S       /usr/bin/bash
pid: 7526       S       /bin/sh
pid: 7527       S       /home/wf/.vscode/extensions/ms-vsliveshare.vsliveshare-
1.0.5449/dotnet_modules/vsls-agent
pid: 7584       S       /usr/share/code/code
pid: 7612       S       /usr/share/code/code
pid: 7733       S       /usr/bin/bash
pid: 7807       S       /usr/share/code/code
pid: 8321       S       git
pid: 8322       S       /usr/lib/git-core/git-remote-https
pid:10351       I       kworker/0:0-events

pid:10352       I       kworker/1:1-events

pid:11077       I       kworker/2:2-rcu_par_gp

pid:11247       I       kworker/3:0-events
```

另外我注意到 comm 的输出会输出一个空行，

```
pid:11384       S       /usr/lib/cups/notifier/dbus
pid:11385       S       /usr/lib/cups/notifier/dbus
pid:11386       S       /usr/lib/cups/notifier/dbus
pid:11387       S       /usr/lib/cups/notifier/dbus
pid:11388       S       /usr/lib/cups/notifier/dbus
pid:12102       I       这是commkworker/2:0-events

pid:12103       I       这是commkworker/3:2-events

pid:12607       I       这是commkworker/u256:2-events_unbound
```

但是 cmdline 和 stat 不会，猜测是因为 comm 文件里带有空格行或是 proc/pid/comm 文件的特性。

以及本来我使用 fseek(fileStream, 0L, SEEK_END);fileSize = ftell(fileStream) + 1；来确定文件大小，fread 直接读取全部内容，但是它实际只读了一个字符，但是使用相同的代码读取普通创建的文件却可以读取全部内容，我猜测是由于 proc 是伪文件，运行在内存中，经查询得到以下信息：

> 文件实际大小为 0，调用 fopen 和 fseek 函数时，/proc 文件系统只是显示文件，而没有生成具体的内容，只有对/proc 目录下的文件进行编辑时，/proc 文件系统才会去根据内核信息建立对应文件。

（最后运行第二个模块的时候需要 rmmod calc，不然会报错（因为学号也是数字开头的）。

感想：

要实现的功能比较容易理解，但是在内核环境的限制下，有许多之前很少接触的 C 语言底层语法需要用到，比如 const char* char* char[] string 之间的相互转化,以及 char*和 int 的相互转换，这些问题至少花费了我几个小时的时间查询实现。

不同内核的函数也都有略微差别，看到网上别人的代码大致框架类似，但是老版本的内核函数也有所改变，更深刻感受到内核的复杂，但是更应该关注逻辑和原理，这些是次要。

由于执行在内核态，以及有很多的 sudo 命令，代码有稍微的不对就有可能 PF 内存溢出，系统卡死等情况，需要耐心重启,仔细排查。

意识到 window 的不方便，由于虚拟机比较卡，内核代码编程在 window,但是调试需要在虚拟机，这就造成了不便和冗余操作，看是否可以加速虚拟机运行（window 的文件系统不太行，不方便搭建环境）

最后是给自己看的知识点：

proc_read 的返回值表示你向用户空间复制了多少字节数据；

一般在 32 位及以上机器上，int 占四字节，char 占一字节，short 占二字节；

malloc 一个 char *之后指针位置移动了需要定义一个 char *copy 保存起始地址然后 kfree 它；

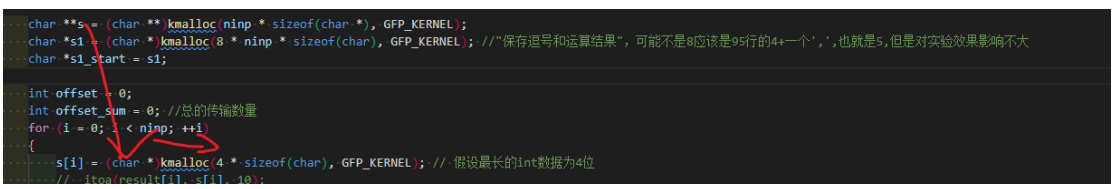
在 if 语句块里（局部作用域）里如果不给 char []整体赋初值，会有作用域提升的情况；

静态局部变量使用 static 修饰符定义，即使在声明时未赋初值，编译器也会把它初始化为 0。且静态局部变量存储于进程的全局数据区，即使函数返回，它的值也会保持不变。static 函数的访问区域为文件级的，普通的函数是项目级的，因此可以使用 static 来减少函数或变量的作用域（内核里面，有比较相似功能的.不同 c 文件，它们里面的变量名很可能相同）

注意到 static 的指针变量中，module_param 处理后不需要 kmalloc，也就是 module_param 应该给了指针动态的空间（根据 insmod 的参数数量）

char*以\0 结束，这也是 strlen 的计算底层逻辑；

Int 数组转化为 char 数组，



Int 如果是一维度的，char 应该是二维的，因为 int 可能有很多位数，而 char 只能表示

一个位。