`

shell 实习报告

杨东升 1400012898

2017.6.5

目录

内容一：总体概述

本次试验要设计实现一个用户程序 shell，通过./nachos -x shell 进入用户交互界面中。在该界面中可以查询支持的功能、可以创建删除文件或目录、可以执行另一个用户程序并输出运行结果，类似 Linux 上跑的 bash。本实验所修改的代码包括内核和用户程序两部分。

内容二：任务完成情况

任务完成列表（Y/N）

	Exercise1	Exercise2	Exercise3	Exercise4	Exercise5
第一部分	Y	Y			

（总之就是都完成了）

具体 Exercise 的完成情况

Exercise 1　修改内核代码

Shell 的所有操作需要通过系统调用来实现，所以必须先修改系统调用的代码，来满足 shell 的需求。

第一个需要修改的部分是读写函数。需要根据描述符的值判断是文件还是命令行。如果描述符是 0，则是从命令行读；如果描述符是 1，则是向命令行写。

不同于文件读写，向命令行读写需要调用 getchar 和 putchar 函数，代码实现如下：

```
else if ((which == SyscallException) && (type == SC_Write)) {
    int offset = machine->ReadRegister(4) ;
    int size = machine->ReadRegister(5) ;
    int id = machine->ReadRegister(6) ;
    char * content = new char[size] ;
    for( int i = 0 ; i < size; i ++ )
        machine -> ReadMem(offset+i, 1, (int*)(content+i)) ;
    if( id == ConsoleOutput )
    {
        for(int i = 0 ; i < size ; i ++ )
            putchar(content[i]) ;
    }
    else
    {
        OpenFile * openfile = dcrpMap[id] ;
        openfile -> Write(content, size) ;
    }
```

```
else if ((which == SyscallException) && (type == SC_Read)) {
    int offset = machine->ReadRegister(4) ;
    int size = machine->ReadRegister(5) ;
    int id = machine->ReadRegister(6) ;
    char * content = new char[size] ;
    if( id == ConsoleInput )
        for( int i = 0 ; i < size ; i ++ )
            content[i] = getchar() ;
    else
    {
        OpenFile * openfile = dcrpMap[id] ;
        openfile -> Read(content, size) ;
    }
    content[size] = '\0' ;
    int * temp = new int[size] ;
    for( int i = 0 ; i < size; i ++ )
    {
        temp[i] = content[i] ;
        machine -> WriteMem(offset+i, 1, temp[i]) ;
        machine -> ReadMem(offset+i, 1, (int*)(content+i)) ;
    }
```

除此之外，我还在 Close 函数中埋了一个后门，用于打印帮助信息。用户程序中无法对字符串方便地赋值，所以打印输出比较困难。为了解决这个问题，我借用内核函数 Close 来打印信息。当以参数-1 调用 Close 时，Close 不关闭文件，只打印帮助信息。代码如下：

```
//Close
else if ((which == SyscallException) && (type == SC_Close)) {
    int id = machine->ReadRegister(4) ;
    if( id == -1 )
    {
        printf("h: help\n") ;
        printf("c filename: create a file\n") ;
        printf("o filename: open a file\n") ;
        printf("w filenum size: write into a file\n") ;
        printf("r filenum size: read from a file\n") ;
        printf("x filenum: close a file\n") ;
        printf("e filename: execute a file\n") ;
        printf("q: quit\n") ;
    }
    else
    {
        OpenFile * openfile = dcrpMap[id] ;
        currentThread -> space -> filemap -> Clear(id) ;
        dcrpMap.erase(id) ;
        printf("close: %d\n", id) ;
    }
```

Exercise 2　编写用户程序代码

在 nachos 的 test 文件夹中，原本就有 shell.c 文件，所以我在 shell.c 的基础上进行修改和扩充，这样不需要更改 makefile。

首先打印两个横线作为提示符，然后读取一行命令。接下来根据命令的第一个字符，来判断用户的操作。命令的通用格式是：空格+命令字符+参数。

第一个参数在基址+3 处，第二个参数在基址+5 处。

```
do {

    Read(&buffer[i], 1, input);

} while( buffer[i++] != '\n' );

buffer[--i] = '\0';

if( buffer[1] == 'e' ) {
    newProc = Exec(buffer+3);
    Join(newProc);
}
else if( buffer[1] == 'c' ) {
    Create(buffer+3) ;
}
else if( buffer[1] == 'o' ) {
    fd = Open(buffer+3) ;
}
```

e 代表执行程序，参数是文件名。

c 代表创建文件，参数是文件名

o 代表打开文件，参数是文件名，返回描述符

```
else if( buffer[1] == 'r' ) {
    int j = 0 ;
    do {
        Read(&content[j], 1, buffer[3]-'0');
    } while( j++ < buffer[5]-'0' );
    Write(content, buffer[5]-'0', output);
}
else if( buffer[1] == 'w' ) {
    int j = 0 ;
    do {
        Read(&content[j], 1, input);
        Write(&content[j], 1, buffer[3]-'0');
    } while( j++ < buffer[5]-'0' );
}
```

r 代表读取文件，第一个参数是文件描述符，第二个参数是读取长度。通过 Read 系统调用从文件中读出对应的字符串，再通过 Write 系统调用向控制台输出字符串。

w 代表写文件，第一个参数是文件描述符，第二个参数是写长度。通过 Read 系统调用

从命令行中读出对应的字符串，再通过 Write 系统调用写字符串到文件。

```
else if( buffer[1] == 'x' ) {
    Close(buffer[3]-'0') ;
}
else if( buffer[1] == 'h' ) {
    Close(-1) ;
}
else if( buffer[1] == 'q' ) {
    break ;
}
```

x 代表关闭文件，参数是描述符

h 代表帮助信息

q 代表退出控制台

展示实验结果：

关于文件的命令：创建 myfile 文件，打开 myfile，写入 hello!；关闭 myfile，再次打开 myfile，读取 6 个字符，关闭 myfile

```
vagrant@precise32:/vagran
-- c myfile
myfile create: myfile
using Linux fileSystem
-- o myfile
open myfile into 2
-- w 2 6
hello!
-- x 2
close: 2
-- o myfile
open myfile into 2
-- r 2 6
hello!-- x 2
close: 2
```

其他命令：执行 halt 程序，显示帮助信息，退出命令行

```
-- e ../test/halt
ready to exec ../test/halt
wait for prog 0
here: ../test/halt
start program ../test/halt
prog yield
prog exit with 0
-- h
h: help
c filename: create a file
o filename: open a file
w filenum size: write into a file
r filenum size: read from a file
x filenum: close a file
e filename: execute a file
q: quit
-- q
prog exit with 0
```

内容三：遇到的困难以及解决方法

困难 1：用户程序无法对字符串赋值

h 命令输出帮助信息，但是由于无法对字符串整体赋值，所以难以在用户程序中写这些帮助信息，解决办法是通过系统调用陷入内核，在内核的函数中打印帮助信息。

困难 2：描述符起始编号问题

最开始我让文件描述符从 0 开始，发生的现象是写到文件中的内容没有真正被写入。经过分析，我发现#define ConsoleInput 0 定义了 0 号描述符是控制台输入，因此写函数实际执行的是 getchar 这一分支。为了改正这一问题，需要让文件描述符从 2 开始。为此我通过静态变量调用了两次 bitmap->find()，把 0 和 1 的位置占用了，文件的编号于是从 2 开始。

困难 3：参数解析

多个参数时，不容易确定各个参数的边界。为了实现的方便，我做了如下规定：命令长度都是 1 个字符；操作的字符串长度都是 1 个字符；文件描述符都是 1 个字符。

内容四：收获及感想

我认为 shell 是系统调用的延伸，系统调用接口只能通过程序员的代码访问，但是 shell 交互性更强，可以通过命令行进行操作，可以方便普通用户使用系统。这次 lab 使我对于系统调用的理解更深入了，对于设计 shell 命令也有了更多的经验。

内容五：对课程的意见和建议

无。

内容六：参考文献

【1】现代操作系统　陈向群