操作系统实验报告

xv6-shell改进

组长 ： 徐毅 2012013300

组员 ： 林杨湄 2012013279

赵志恒 2012013316

陈鸣海 2012013336

目录

一、概述

1.1 实验选题

对xv6的shell功能进行改进，xv6源码来源:git://pdos.csail.mit.edu/xv6/xv6.git。

1.2开发环境及开发工具

开发环境：Ubuntu 14.04

开发工具：gcc，qemu

开发语言：C、汇编

版本管理：github

1.3 团队分工

徐毅	历史记录、通配符、新增 tac , head , tail , mv 命令
林杨湄	新增rename，cp，help命令，功能测试
赵志恒	光标移动，vim命令
陈鸣海	path环境变量，重定向，tab自动补全，键盘响应事件系统调用，ctrl +c，cat命令改进

二、键盘响应

一些应用程序需要对键盘某些输入及时响应，如shell中tab键要引发自动补全、上下键查看历史记录等等。

第一种思路是直接在console.c中添加对某些按键的响应事件，这是最简单的实现，但是不同的进程的响应事件不同，这样做会引发冲突。

第二种思路是在第一种的基础上，通知内核当前运行在用户空间的是哪一个用户进程，以保证后续所有的键盘响应事件都是独立而不相互干扰的，这是学长的实现方法。但是内核需要做大量的判断：根据当前用户空间进程和按下的键来返回不同的数据，触发用户进程的响应。这种方法的问题是，每新增一个用户程序，如果它对某些按键有特殊响应，都需要修改内核代码，这样的操作系统是不成熟的。

我们采用的第三种思路是让内核代码固定下来，提供一些系统调用给用户程序使用，用户程序使用我们提供的接口，不需要修改内核代码。具体实现中遇到的问题是：

xv6将不同字符和特殊按键视作同类，都放入键盘缓冲区内供用户程序读取。默认在输入'\n'，C('D')时将input.w设置为input.e，再唤醒consoleread将缓冲区里的数据交给用户程序。这样的无法做到对特殊按键的及时响应。如果对每一个字符都唤醒consoleread，倒是可以做到对每一个字符及时响应，但是已发送的字符不可编辑，即不能用退格等修改了。所以我们首先只对特殊字符唤醒consoleread，将控制权交给用户程序；用户程序对自己有特殊响应的按键进行响应处理，而对没有响应事件的特殊字符，刷新键盘缓冲区，使之前的普通字符在控制台中依然处于可编辑状态。为此添加了两个系统调用clearc和insertc，使用户程序可以操作键盘缓冲区。

三、用户交互改进

3.1 历史记录

3.1.1 改进目的

我们希望每次用户在shell上输入相关命令之后，系统能够自动保存该命令，用户之后可以通过上下键查找自己输入命令的历史记录。

3.1.2 实现思路

首先在history.h中定义了保存历史记录的结构体，在sh.c中定义全局变量hs用户保存所有记录。每次用户执行一条命令之后将其保存在
hs中，并将其写入到.history中。每次shell启动过后将所有历史记录读入在hs中，这样用户按上下键就可以通过键盘的系统调用实现历史记录的查找。

3.1.3 具体实现（参见sh.c）

```
//将历史记录cmd写回到文件.history中
void setHistory(char* cmd)
//从.history中读出历史记录保存到hs中
void getHistory(struct history* hs)
//将记录cmd保存在hs中
void addHistory(struct history* hs, char* cmd)
//初始化历史记录hs
void initHistory(struct history* hs)
```

3.2 tab键自动补全

3.2.1功能

补全命令或参数。如果只找到一个匹配的文件，则自动补全；否则将所有匹配的文件列表列出。

3.2.2实现

利用键盘响应事件，shell收到tab后，截取出选取的参数，利用通配符*的查找方法进行查找即可。

3.3 ctrl+c

3.3.1 含义

强制关闭当前进程。当进程出现错误、陷入入死循环、进行行耗时操作而用户不愿意等待时，需要关闭程序。这时不能靠进程自身来监听终止止信号，要靠内核来操作。

3.3.2 实现思路

在console.c的consoleintr中，得到用户输入control+C之后，调用sendsignal(1)，这个函数添加在proc.c中，具体实现是：由于shell的特殊性，前台活跃子进程是唯一，所以只要kill掉pid最大的进程即可，通过遍历进程池ptable.proc实现。kill的方法是将对应进程的killed设为1，但是仅仅这样无法达到立即kill的作用，因为这个进程可能处于堵塞状态，无法立刻在trap中调用用exit()。所以判断如果它是SLEEPING状态，设为RUNNABLE, 这样下一一次调度到它就可以kill了。

四、PATH环境变量

4.1 含义

xv6并不支支持path变量，要执行行命令的只能在放在当前目录下，或者以绝对路径给出。随着系统的扩展,要么把各种程序都堆在一一个目录下，很冗杂，要么就要记着每个程序在哪个目录下，这很不实用。

我们添加了一个path文件，用户可以编辑path文件，自主添加shell搜索的目录。在执行命令时，shell 会在path文件中的目录一个个寻找要执行的命令文文件。

4.2实现思路

修改exec.c的实现。

首先打开path文件，按“;”分隔取出一个个目录。将目录拼接在命令名称之前，以组合出的带路径的命令文件名去查找，如果找到对应文件，则载入执行,否则尝试下一个目录。

注意，在内核中读写文件时，不能直接引用user.h使用里面的读写函数(里面有些函数的命名有冲突),可以使用底层一点的，先用namei()找到文件对应的node，然后再用用readi读取。这里要注意加锁和解锁。

这样，只需要修改系统的exec.c，用户程序不用做任何修改。而学长先前的实现是修改cd，有逻辑地址和实际地址，实现很麻麻烦，而而且改变了exec()的参数，每一个用用户程序(如rm)都要做相应的修改，耦合性太强。我们这样修改底层，保持对用用户的便利简洁，应该更好。

五、通配符

5.1 含义

通配符包含*.?.[]三种类型，其中*表示可以匹配任意包括零个字符；？ 表示可以匹配零个或者一个字符；[]表示匹配出现在里面的一个字符。特别 [1-6] 表示匹配 1 到 6 之间的某个字符，[a-z]同理。

5.2 改进目的

希望增加通配符能够让使用者很轻松的对多个文件进行操作。

5.3 实现思路

实现通配符功能主要分为两个部分。

首先需要一个匹配算法，判断一个带通配符的文件名和另一个正常的文件是否匹配，具体实现在sh.c中 int matchesPattern(char* input, char* pattern)函数

另一方面需要获取当前目录下的文件列表，将带有通配符的文件一一进行比较，具体实现在sh.c中void getFilelist(char *path,struct fileList *fl)函数和void getMatchList(char* filename,struct fileList *allfile,struct fileList *matchfile)

最后将所有满足匹配条件的文件名传入到执行文件的参数中，具体实现在sh.c的void runcmd(struct cmd *cmd)中case EXEC。

5.4 具体实现（参见sh.c）

//判断文件input和带通配符的文件pattern时候匹配，0表示不匹配，1表示匹配

int matchesPattern(char* input, char* pattern)

//从path路径下获取文件列表并保存在fl中

void getFilelist(char *path,struct fileList *fl)

六、重定向

6.1含义

完善了xv6的重定向,可以支支持>>,可以选择将stdout还是stdin重定向。

6.2 实现思路

xv6的重定向的实现是在runcmd中,如果类型是REDIR,则将fd关闭(xv6固定关掉stdout即fd=1)并且打开目标文件。我们要支支持诸如cmd >> file，cmd 1>> file，cmd 2> file等，首先在sh.c的gettoken中，对“>>” “1>” “2>>”等做出相应判断，根据1、2选择关闭的fd(未指明则默认为1)，根据>或>>来选择是覆盖还是追加。

另一个问题是xv6对文件的写入不支持覆盖和追加，只能从文件头开始重写。因此在fcntl.h中另外声明了O_ADD和O_OVER两种模式。写文件的模式在哪里实现?是记录在进程的打开文件表里的。这样实现：对于O_OVER(覆盖)模式，将初始f->size设为0，对于O_ADD(追加)模式，将f->off设为f->size即可。

七、xv6命令改进与添加

7.1 head，tail

7.1.1 含义

head用于显示文件前10行的内容，tail显示后10行的内容。

7.1.2 实现思路

两者实现的方法比较相似，都是通过open，read等系统调用读取文件的内容。唯一不同的是head每读一行显示一行，显示完10行为止，tail使用一个数组循环保存每行读到的内容，最后将后10行的内容显示出来。

7.2 cat，tac

7.2.1 含义

cat用于显示一个文件的内容，也可以cat > filename的方式创建一个文件并写入内容。tac与cat用法相似，但是从最后往前显示文件内容。

7.2.2 cat改进

在重定向和ctrl+c工作的基础上，cat可以实现创建文件(并编辑初始内容)，显示文文件，复制文件，追加文件，合并文件等。

创建文件：cat > filename，然后输入文件内容，结束按ctrl+c即可

复制文件：cat f1 > f2

追加文件：cat f2 >> f1

合并文件：cat f1 f2 > f3

7.2.3 tac实现思路

在tac.c中实现了一个链表结构，按倒序的方式存储文件每行内容，最后将其显示出来。

7.3 rename，cp，mv

7.3.1 含义

重命名，复制，移动文件

7.3.2 实现思路

通过open，read，write进行文件读写操作，使用link和unlink建立或者取消文件与文件名关联。

7.4 help

7.4.1 含义

获取指令帮助。

八、特色功能 vim编辑器

程序完全仿照Vim进行设计，实现了Vim的部分功能，具体使用方法参见README.pdf

8.1 实现思路

由于Vim需要完全控制Console，因此添加了一条系统调用setconsole，用来进行Console的写入和光标的控制，以及设置Console的模式。

在程序中主要将文本存储在一个缓冲区textbuf中，插入和删除文本时都对textbuf进行相应更改，并更新界面。

当Console模式为2时，onsole中每接到一个字符输入立即将字符传给进程，本身不做任何处理。在Vim程序中读取每一个字符进行相应的处理。

处理的逻辑为：

1、首先判断字符是不是特殊的控制字符，如键盘的上下左右键，Esc、Insert、Home、End键等，如果是，则进行相应的操作，否则转到2

2、判断当前的输入模式，0为控制模式，1和2为输入模式。如果是控制模式，则转到3，否则转到4

3、在控制模式下，指令分为2种，单字符指令和多字符指令，其中r和:开头的指令是多字符指令，其余为单字符指令。对于单字符指令，输入后立即执行并清除指令缓冲区。对于r指令，直到检测到第二个输入的字符才执行并清除指令缓冲区。对于:开头的指令，当用户输入回车时才进行执行。

4、在输入模式下，处理的方式和普通的文本编辑器基本一致。

8.2 实现细节

1、光标移动的实现。光标的移动只能在已有的行间进行，每一行只能移动到最后一个字符的下一个位置。当光标移动到最上方或最下方时需要进行翻页处理。程序中用一个startline来记录当前屏幕上第0行对应的是textbuf中的第几行，翻页操作只需要修改startline并刷新页面即可实现。

2、插入和删除的实现。插入和删除有两个子功能：插入一行和删除一行。当用户输入回车时，则要将本行光标后的内容移到新建的下一行。当用户在行首输入空格时，要将本行的内容接到上一行并删除本行。具体的实现都是采用数组循环复制的方法，例如要

删除一行的第i个字符，则将第i+1个字符复制到位置i，第i+2个字符复制到位置i+1，以此类推。插入的实现类似。

3、信息栏的实现。程序单独将最下面一行留作信息栏，显示提示信息、:开头的命令以及光标所在的位置，程序采用了两个函数进行信息和光标位置的设定。

九、总结

本次实验我们对xv6的shell部分进行了改进，最开始选择这个题目的时候我们其实并不知道从哪里入手，因此前期我们花费很多时间来阅读理解原先的代码，这过程中也用到了很多平时上课和作业中的知识。我们发现xv6最底层的如键盘响应，屏幕输出等相关实现理解起来比较困难，上层应用的功能理解起来倒是相对轻松，等我们攻克了底层实现这样一个难题之后，开发任务相对变得简单了一些，或者说转到我们熟悉的层面了。但很高兴能够有一个这样的机会直接接触底层的代码，在底层编写代码还是很考验我们的，经常会出现一些意想不到的错误，我们的调试主要也是在这方面。最后我们实现vim命令的时候遇到了很多问题，我们觉得可能和xv6系统本身的实现有关系，通过这次开发也让我们体会到一个好的系统设计的重要性，希望我们吸取这次开发过程中的经验和教训，在以后的开发过程中做一个更加健壮的系统。