Linux 内核实验报告

实验题目：shell 命令解释系统设计实验

学号：200900301236　　（辅修号：　）
日期：**2012.4.21**　　班级：09 软 1　　姓名：　王添枝
Email：wangtianzhi@163.com

实验目的：练习怎样编写与内核的接口程序，深入了解和挖掘内核的各种技术。

硬件环境：微软实验室
软件环境：gcc4.6.1
ubuntu11.10
Linux version 3.0.0-12-generic
vi 7.3.154

实验步骤：

1、 设计说明

读取环境变量，分割环境变量的各个路径。用户输入命令后，对命令进行解析，根据不同的命令类型进行相应的处理。若命令中有‘&’符号则表示后台运行，‘<’符号表示输入重定向，‘>’符号表示输出重定向，‘|’符号表示管道，‘；’符号用来多条命令之间隔开。

对于输入输出重定向，截取重定向的文件；对于管道，分别管道左边和右边的命令；对于‘；’多命令符号，切割各条命令。解析各条命令，获取命令名和参数。取得命令所在的绝对路径，创建子进程用来执行命令。如果为管道，则在子进程写入管道，在父进程从管道读取。如果有后台执行符号‘&’则父进程不用等待子进程执行完，否则等待子进程执行完。

2、 调试记录

（1）管道功能

```
myshell$: ls | grep exp
exp1.doc
```

exp1_proc

exp1_proc.c

exp2

exp2.c

exp2.doc

exp2_myshell.c

exp2test

exp2 存在的问题.txt

（2）输出输入重定向

myshell$: ls > ls.txt

myshell$: cat <ls.txt

200900301236-实验一王添枝.pdf
exp1.doc
exp1_proc
exp1_proc.c
exp2
exp2.c
exp2.doc
exp2_myshell.c
exp2test
exp2 存在的问题.txt
Linux(旧版).pdf
linux 内核实验报告模板.doc
linux 内核实验教材.pdf
linux 内核实验教学大纲.pdf
ls.txt
各个文件的内容.txt
网络资源

（3）处理多条命令

多条命令之间通过“；”符号隔开

myshell$: pwd;date

/home/wangtianzhi/桌面/linux 内核实验

2012 年 04 月 21 日 星期六 21:44:03 CST

结论分析与体会:

通过此次实验，对怎样编写与内核的接口程序有了较深的了解。在实验过程中通过网上查阅了很多的资料，掌握了很多新的知识。老师已在实验要求中给出了一部

分程序范例，根据这些前期准备进行了实验。实验过程中，实验目的基本完成，但是代码没有进行优化，可能运行性能比较低。

程序完整源代码：

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/wait.h>
#include <sys/stat.h>
#define MAXARG 20
#define LS 80
#define SHELL_NAME "myshell$:"
#define PROMPT_ENVIRONMENT_VARIABLE "PROMPT"
char *prompt,**fullpath;
int k=0;
struct command_struct{
    char *name;
    int argc;//参数个数
    char *argv[];//存放参数（argv[0]为命令名）
}*command,*command2;

void splitCmd(char buffer[], struct command_struct* command) {
     int l = 0;
     command->argc=0;
         char *p;
    command->argv[l]=malloc(LS);
    command->argv[l++]=strtok(buffer," ");
    while(p=strtok(NULL," ")){
        command->argv[l]=malloc(LS);
        sprintf(command->argv[l++],"%s",p);
        command->argc++;
    }
    command->name=command->argv[0];
    command->argv[l]=NULL;
    //printf("argc=%d\n",command->argc);
}

/**
   查找命令所在路径
   fullPathFile：存放命令所在的路径
```

```
   command：要查找的路径
**/
void getfullPath(char fullPathFile[], struct command_struct* command)
{
    int  c;
    for (c = 0; c < k; c++) {
        sprintf(fullPathFile, "%s/%s", fullpath[c], command->name);

        //access()  checks  whether the calling process can access the
file path -
                      //name.   If pathname is a symbolic link, it is
dereferenced.
        // F_OK tests  for  the  existence  of  the file.
        if (access(fullPathFile,F_OK)==0){//文件存在 access 函数返回 0

            break;
        }else{
            continue;}
    }
    if(c==k){//可执行文件，路径为当前目录
        sprintf(fullPathFile, "%s", command->name);
    //  printf("executed:%s\n",fullPathFile);
    }

}

int main(){
    char
*infile,*outfile,*lineBuf,*lineBuf2,*tempBuf,*path,fullpathfilename[6
4],
    fullpathfilename2[64],*lineBuf3,**lines;
    int pid1,pid2;
    int i=0,j;
    int f_in,f_out;
    int filein,fileout,ispipe,isbkg,mypipe[2];
    lineBuf=(char *)malloc(LS+1);
    lineBuf2=(char *)malloc(LS+1);
    lineBuf3=(char *)malloc(LS+1);
    lines=(char *)malloc(sizeof(lines));
    int t;
    for(t=0;t<5;t++)
        lines[t]=malloc(LS);

    command=(struct command_struct *)malloc(command);
```

```
command2=(struct command_struct *)malloc(command2);


fullpath=malloc(sizeof(fullpath));
path=getenv("PATH");
printf("path:%s\n",path);
fullpath[k]=malloc(LS);
fullpath[k++]=strtok(path,":");
while(tempBuf=strtok(NULL,":")){//以“:”分割字符串
//  printf("ok:");
//  printf(tempBuf);
    fullpath[k]=malloc(LS);
    sprintf(fullpath[k++],"%s",tempBuf);
}

if((prompt=getenv(PROMPT_ENVIRONMENT_VARIABLE))==NULL)
    prompt=SHELL_NAME;


while(1){
    t=0;
    j=0;
    filein=0;
    fileout=0;
    ispipe=0;
    isbkg=0;
    bzero(fullpathfilename,64);//置字节字符串 fullpathfile 为零
    bzero(lineBuf,LS+1);
    printf(prompt);
    gets(lineBuf3);
    if(strlen(lineBuf3)==0)
        continue;
    if(strstr(lineBuf3,"&")){//有后台执行符“&”
        isbkg=1;
        lineBuf3=strtok(lineBuf3,"&");
        }
        lines[t++]=strtok(lineBuf3,";");
        printf("lines[0]:%s\n",lines[0]);
        while(lines[t]=strtok(NULL,";"))
            t++;
    int q;

    for(q=0;q<t;q++){
        lineBuf=lines[q];
```

```
        if(strstr(lineBuf,">")){//输出重定向
            filein=1;
            lineBuf=strtok(lineBuf,">");
            infile=strtok(NULL,">");
        }
        if(strstr(lineBuf,"<")){//输入重定向
            fileout=1;
            lineBuf=strtok(lineBuf,"<");
            outfile=strtok(NULL,"<");
        }
        if(strstr(lineBuf,"|")){//管道
            ispipe=1;
            lineBuf=strtok(lineBuf,"|");
            lineBuf2=strtok(NULL,"|");
            //printf("lineBuf2%s\n",lineBuf2);
            if (pipe(mypipe) < 0) {// 创建管道， mypipe[0]用来读，
mypipe[1]用来写
                /*mypipe[0] Read
                 mypipe[1] Write*/
                printf("pipe create error\n");
                }

        }
        //printf("lineBuf=%s\n",lineBuf);
        splitCmd(lineBuf,command);
        getfullPath(fullpathfilename,command);


        if(strcasecmp(command->name,"exit")==0)
            exit(0);
        if((pid1=fork())==0){//child process

        //printf("fullpathfilename%s\n",fullpathfilename);
            if (filein) {//输出重定向
                    //只写打开（若不存在自动创建）
                    //参数mode（第三个参数）只有在建立新文件时才会生效
                    //S_IRWXU 00700 权限，代表该文件所有者具有可读、可
写及可执行的权限。
                    //S_IRWXG 00070权限，代表该文件用户组具有可读、可写
及可执行的权限。
                    //S_IRWXO 00007权限，代表其他用户具有可读、可写及可
执行的权限。
                f_in = open(infile, O_WRONLY | O_CREAT, S_IRWXU |
```

```
S_IRWXG
                    | S_IRWXO);
            close(1);//1为标准输出设备
            dup(f_in);//These system calls create a copy of the
file descriptor oldfd.
            close(f_in);
        }
        if (fileout) {//输入重定向
            f_out = open(outfile, O_RDONLY | O_CREAT, S_IRWXU |
S_IRWXG
                    | S_IRWXO);
            close(0);//0为标准输入设备
            //由dup返回的新文件描述符一定是当前可用文件描述中的最
小数值
            dup(f_out);
            close(f_out);
        }
        if(ispipe){//管道
            close(mypipe[0]); //关闭read
            close(1);
            dup2(mypipe[1], 1); //int dup2(int oldfd, int newfd)
makes newfd be the copy of
                                    //oldfd, closing newfd first if
neces - sary
            close(mypipe[1]);
        }
        execvp(fullpathfilename,command->argv);
        printf("EXEC %s failed:\n",command->name);
        exit(1);
    }
    else {//parent process

        if(ispipe){
            close(mypipe[1]); //关闭“write”一端
            if ((pid2 = fork())==0) {//second child process

                splitCmd(lineBuf2,command2);
                getfullPath(fullpathfilename2,command2);

                close(mypipe[1]); //关闭“write”的的一端
                close(0);
                dup2(mypipe[0], 0); //在执行系统调用exec后，会释放
0,1,2之外由父进程打开的文件，
                close(mypipe[0]); //所以要把myPipe[0]复制到stdin对
```

```
应的文件句柄0
    //printf("fullpathfilename2:%s\n",fullpathfilename2);

                    execvp(fullpathfilename2,command2->argv);
                    printf("EXEC %s failed:%s\n",command2->name);
                    exit(1);
                }//end of second child process

            }//end of if(ispipe)

            if(!isbkg){//没有后台执行
                wait(&pid1);
                if(ispipe){
                    close(mypipe[0]);
                    wait(&pid2);
                }
            }

        /*if(WIFEXITED(pid1)){
            if(WEXITSTATUS(pid1)){
                fprintf(stderr,"child      exited      with      status
%d.\n",WEXITSTATUS(pid1));
        }
        }else {
        fprintf(stderr,"child exited unexpectedly\n");
        }*/
        } }
    }
    int rel;
    for(rel=1;rel<=command->argc;rel++){
                free(command->argv[rel]);
                command->argv[rel] == NULL;
            }
    free(command);
    free(fullpath);
    free(lineBuf);
    free(command2);
    free(lineBuf2);
}
```

参考材料

linux内核设计与实现