《操作系统》课第 09 次实验报告

学院:	软件学院
姓名:	郁万祥
学号:	2013852
邮箱:	yuwanxiang0114@163.com
时间:	2022.11.11

0. 开篇感言

经过了多进程和多线程的实验，感觉此次实验还是比较容易的，主要分成了三个部分：

第一部分是对一些和程序执行无关的处理问题：包括了信息（姓名学号）的展示，对于命令的分割（包括多命令的分号分隔和单命令的的空格键及 Tab 键的分割），这些都属于处理层面上的操作。

第二部分是对于自定义的外部命令的定义，此次实现的是 cd 外部命令的重新定义，包括 cd：结束本轮循环，cd~：回到用户根目录，cd-：回到上一次的位置。

第三部分就是内部命令的执行，对于输入的内部命令，我们可以通过直接调用系统，切换线程，通过 exec 族的函数进行系统执行输入的命令从而达到目的。

1. 实验题目

Mini Shell

2. 实验目标

在 Linux 平台上，采用 C 语言编写一个 Mini Shell 命令解释环境（即类似 Bash
Shell 环境）。该环境可以循环接受用户（从标准输入中）输入的（外部和内部）命令以及若干参数，然后能对上述命令进行解析和执行，最后将用户输入的命令的执行结果显示在标准输出上。

- 支持用户输入一行命令及其多个参数，并解析执行，并输出结果；
- 支持 cd 命令，若无参数则回到当前用户的登录目录；
- 支持以“当前路径”和“用户名”为提示符；
- 支持对命令行中空格的自动忽略处理；
- 支持对命令行中 tab 键的自动忽略处理；
- 支持一行中以“；”（为标志）分隔的多个命令及多个参数的**顺序**执行

3. 原理方法

3.1、Linux Shell：Shell 是系统的用户界面，提供了用户与内核交互的一种接口。他接收用户输入的命令并把它送入内核去执行。实际上 Shell 是一个命令解释器，它解释由用户输入的命令并且把它们送到内核。不仅如此，Shell 有自己的编程语言用于对命令的编辑，它允许用户编写由 shell 命令组成的程序。Shell 编程语言具有普通编程语言的很多特点，比如它也有循环结构和分支控制结构等，用这种编程语言编写的 Shell 程序与其他应用程序具有同样的效果。

3.2、Shel 运行过程原理:

1.读取用户由键盘输入的命令:

2.对命令进行分析，以命令名为文件名，并将其他参数改造为系统调用 ecv()参数处理所要求的格式;

3.终端进程(shell)调用 fork(或者 vfork()建立一个子进程;

4.子进程根据文件名（命令名）到目录中查找有关文件，将他调入内存，并创建新的文本段，并根据写时拷贝的方式创建相应的数据段、堆栈段;

5.当子进程完成处理或者出现异常后，通过 eit()或 eit()函数向父进程报告;

6.终端进程调用 wait 函数来等待子进程完成，并对子进程进行回收;

3.3、Shell 对于命令的分析过程:

1.首先，检查用户输入的命令是否是内部命令，如果不是在检查是否是一个应用程序:

2.shell 在搜索路径或者环境变量中寻找这些应用程序;

3.如果键入命令不是一个内部命令并且没有在搜索路径中查找到可执行文件，那么将会显示一条错误信息;

4.如果能够成功找到可执行文件，那么该内部命令或者应用程序将会被分解为系统调用传给 Liux 内核，然后内核在完成相应的工作;

4. 具体步骤

1、使用 fgets 函数从命令行中读取输入的命令。

2、将获得的命令进行分割，忽略空格、tab、判断命令个数等处理。

3、通过 strtok 函数，将命令根据”；“进行分割，然后将每一个命令继续循环进行操作。

4、根据命令第一个词语判断是否为内部命令（此处实现的内部命令 cd 和 exit）。

5、如果是系统调用内部命令，直接使用 fork+exec 进行处理。

6、进行循环。

执行结果:

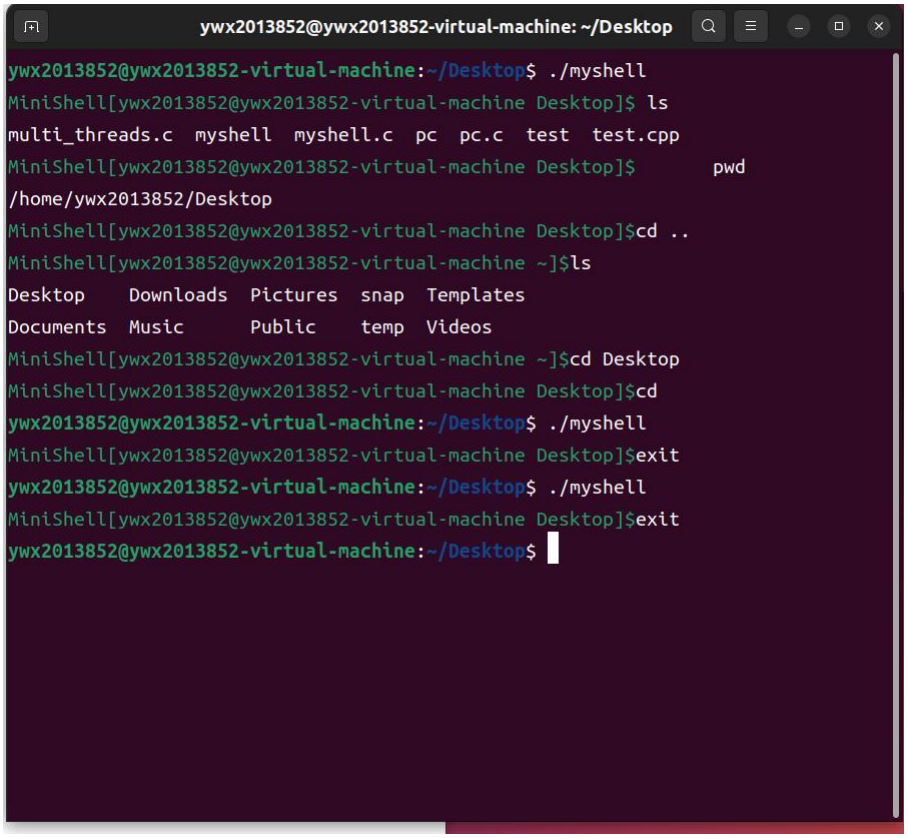
支持用户输入一行命令及其多个参数，并解析执行，并输出结果；

支持 cd 命令，若无参数则回到当前用户的登录目录；

支持以“当前路径”和“用户名”为提示符；

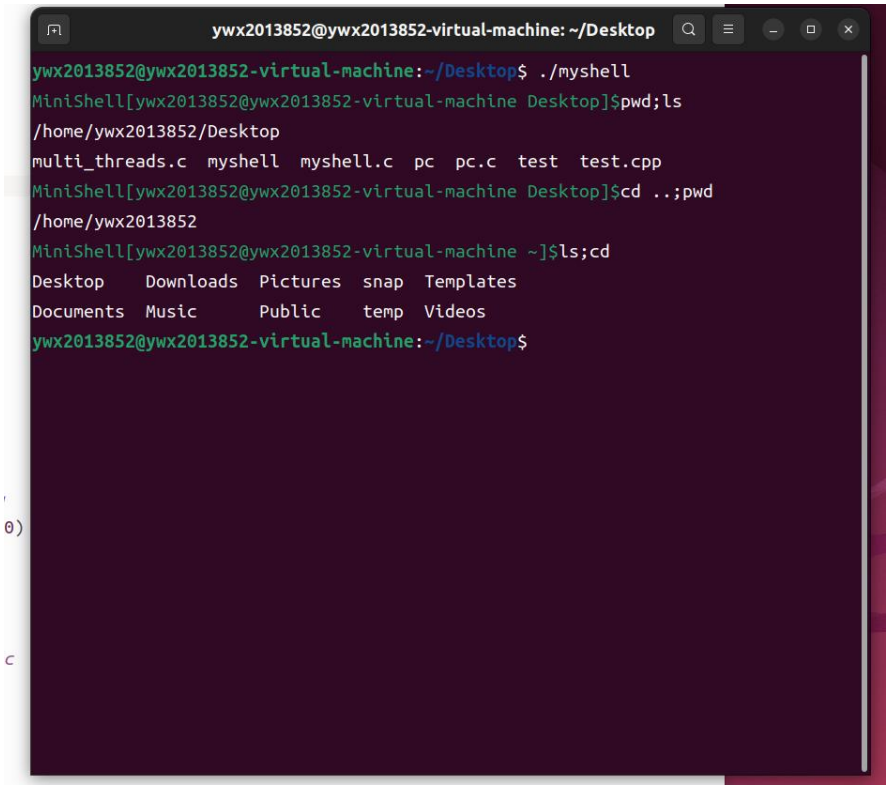
支持对命令行中空格的自动忽略处理；（示例中 ls 命令前面输入空格）

支持对命令行中 tab 键的自动忽略处理（示例中 pwd 命令前面输入 tab 键）；



支持一行中以“；”（为标志）分隔的多个命令及多个参数的**顺序**执行

对多种命令进行了结合验证，都支持。



5. 总结心得

虽然操作系统仍然是进行代码编写工作，但是可以发现，更多的是对于硬件资源的调配工作的代码编写，通过了解操作系统层面上的工具以及封装好的对于硬件资源调配的 API，我们可以通过软件的编写之间调配硬件资源。

6. 参考资料

源码：

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
```

```
#include <string.h>
#include <pwd.h>
#include <sys/utsname.h>
#include <sys/types.h>
#include <unistd.h>
#define MAX 10
#define STRLEN 128
#define PATH "/bin/"   //系统 bin 路径位置
char OLDPWD[STRLEN]={0};      //记录上一次的命令，为了 cd -这条命令
//输出当前所在用户信息
void Printf_Info()
{
    char flag='$';
    struct passwd *pw=getpwuid(getuid());
    assert(pw!=NULL);

    //uid 为 0 则为 root 用户
    if(pw->pw_uid==0)
    {
        flag='#';
    }
    struct utsname hostname;   //主机名
    uname(&hostname);
    char node[STRLEN]={0};
    strcpy(node,hostname.nodename); //获取网络上的名称
    char* name=strtok(node,".");
    //获取绝对路径
    char path[STRLEN]={0};
    getcwd(path,STRLEN-1);
```

```
    char*p=path+strlen(path);  //p 指向绝对路径的末尾
    while(*p!='/')
    {
        p--;
    }
    //p 指向路径末尾往前的第一个'/'位置处
    if(strlen(path)!=1)
    {
        p++;
    }
    if(strcmp(path,pw->pw_dir)==0)
    {
        p="~";
    }
    printf("\033[;32mMiniShell[%s@%s %s]%c\033[0m",pw->pw_name,name,p,flag);
    fflush(stdout);

}

void Mycd(char*path)
{
    //第一个字符串为 cd 而第二为空  如：cd  则结束本轮循环
    if(path==NULL)
    {
        exit(0);
    }
    //cd ~  回到用户根目录
    if(strcmp(path,"~")==0)
    {
```

```
            struct passwd*pw=getpwuid(getuid());
            path=pw->pw_dir;
        }
        //cd - 回到上一次的位置
        if(strcmp(path,"-")==0)
        {
            //若是第一次输入命令，则 cd -命令不存在!
            if(strlen(OLDPWD)==0)
            {
                printf("\033[;31mMyBash:cd:OLDPWD not set\n\033[0m");
                return ;
            }
            //否则把上一次的命令给 path
            path=OLDPWD;
        }
        //getpwd 记录当前工作目录的绝对路径
        char oldpwd[STRLEN]={0};
        getcwd(oldpwd,STRLEN-1);

        if(-1==chdir(path))//反之则不是空，则通过 chdir 系统调用进入到该目录中
        {
            char err[128]="\033[;31mMybash: cd \033[0m";
            strcat(err,path);
            perror(err);
        }
        //每次执行完 cd 命令后，把工作路径赋给 OLDPWD
        strcpy(OLDPWD,oldpwd);
}
//命令分割函数
```

```
void Strtok_cmd(char*buff,char*myargv[])
{
    char *s=strtok(buff," "); //分割输入的字符串
    if(s==NULL)          //如果 s 为空，则进入下一轮循环
    {
        exit(0);
    }
    myargv[0]=s; //把分割出来的第一个字符串放在 myargv[0]中
    int i=1;
    while((s=strtok(NULL,""))!=NULL)      //把后续分割出来的字符串依次存放在数组中
    {
        myargv[i++]=s;
    }
}
int main()
{
    while(1)
    {
        char buff[128]={0};

        Printf_Info();
        char *p=NULL;
        //从终端获取命令存入 buff 中
        fgets(buff,128,stdin);
        buff[strlen(buff)-1]=0;
        p = strtok(buff,";");   // 将命令根据；进行分割
        while(p)   // 循环执行每条命令
        {
            char* temp = p;
```

```
p = strtok(NULL,";");
char *myargv[MAX]={0};

//分割输入的命令
Strtok_cmd(temp,myargv);

//如果输入 exit 则退出循环
if(strcmp(myargv[0],"exit")==0)
{
    exit(0);
}
//如果分割出来的第一个字符串为 cd
else if(strcmp(myargv[0],"cd")==0)
{
    Mycd(myargv[1]);
    continue;
}
//若是系统调用，直接替换 fork+exec
pid_t pid=fork();
assert(pid!=-1);
if(pid==0)
{
    char path[256]={0};
    if(strncmp(myargv[0],"./",2)!=0 && strncmp(myargv[0],"/",1)!=0)
    {
        //先把路径放入 path 中
        strcpy(path,PATH);
    }
    //进行命令拼接，路径+名称
```

```
                strcat(path,myargv[0]);
                //替换进程    例如：/bin/ls
                execv(path,myargv);
                perror("\033[;31mexecv error\033[0m");
            }
            //处理僵死进程
            else
            {
                wait(NULL);
            }
        }
    }
}
```