《操作系统》
实 验 报 告

学 生 姓 名　夏雨轩

学　　号　21009201006

班　　级　2118021

20 22 — 20 23 学年 第 2 学期

《操作系统》实验报告			
实验名称	多进程编程	实验序号	2
实验日期	2023.3.28	实验人	夏雨轩

一、实验题目

Project 1-UNIX Shell and History Feature

This project consists of designing a C program to serve as a shell interface that accepts user commands and then executes each command in a separate process. This project can be completed on any Linux, UNIX, or Mac OSX system.

A shell interface gives the user a prompt, after which the next command is entered. The example below illustrates the prompt osh> and the user's next command: cat prog. c. (This command displays the file prog. c on the terminal using the UNIX cat command.)

osh> cat prog.C

user enters the command ps -ael at the osh> prompt, the values stored in the

args array are:

args[0]

args[1]

args [2]

This args array will be passed to the execvp () function, which has the following prototype:

execvp (char *command, char *params ([]);

Here, command represents the command to be performed and params stores the parameters to this command. For this project, the execvp () function should be invoked as execvp (args[0], args). Be sure to check whether the user included an & to determine whether or not the parent process is to wait for the child to exit.

Part Il-Creating a History Feature

The next task is to modify the shell interface program so that it provides a history feature that allows the user to access the most recently entered commands. The user will be able to access up to 10 commands by using the feature. The commands will be consecutively numbered starting at 1, and the numbering will continue past 10. For example, if the user has entered 35 commands, the 10 most recent commands will be numbered 26 to 35.

The user will be able to list the command history by entering the command history.

The program should also manage basic error handling. If there are no commands in the history, entering !! should result in a message "NO commands in history." If there is no command corresponding to the number entered with the single !, the program should output "No such command in history.”

1.设计一个 C 程序作为一个 shell 接口，它接受用户命令，然后在单独的进程中执行每个命令。

2.下一个任务是修改 shell 接口程序，使其提供 history 功能，允许用户访问最近输入的命令。通过使用该功能，用户最多可以访问 10 个命令。

3.该程序还应该管理基本的错误处理。

二、相关原理与知识

1. osh>Cat prog.c(采用 cat 在终端上显示文件 prog.c)

用 C 语言实现简单的命令行解析器，支持用户输入命令行并运行在其他的进程中。该命令行解析器可以运行在任何 Linux 或 Mac 系统，支持&修饰命令行，使命令进入后台运行。本质上就是使命令行在子进程运行于后台，或者子进程和父进程同时运行。这个 project 主要分成两部分，一部分是解析用户命令并使其在子进程运行，另一部分是支持我们自己命令行终端特有的 history 功能。即：当用户输入 history 命令，系统将展示最近使用的 10 条命令。如当前 history 里面存储了 6 条输入过的命令（从最近使用的到最远使用的顺序排列）：ps -l, ls -l, top, date, cal, whoami

当输入 history 命令后，系统将会输出

6 ps -l
5 ls -l
4 top
3 date
2 cal
1 whoami

当用户输入!!命令，就运行最近运行的命令，在当前环境下也就是 ps -l，

当用户输入!N 命令，N 代表数字，就运行第 N 个命令在 history 中，!2 就运行 cal.

如输入!!命令时没有命令在 history 中，就输出 No command in history.

如输入!N 命令没有对应第 N 个命令在 history 中，就输出 No such command in history.

用户输入 exit 将退出当前命令行解析器。

osh>cat prog.c &（父进程和子进程可以并发执行）

2. 多进程编程相关理论知识

进程：进程是一个正在执行的程序，是向 CPU 申请资源的，进程之间数据相互独立，一个进程至少有一个线程。

线程：线程是进程中的单一的顺序控制流程也可以叫做最小控制单元，线程是进程中执行单元，开启一个线程比开启一个进程更加节省资源。

多线程：多线程是多任务处理的一种特殊形式，多任务处理允许让电脑同时运行两个或两个以上的程序。一般情况下，两种类型的多任务处理：基于进程和基于线程。

多线程程序包含可以同时运行的两个或多个部分。这样的程序中的每个部分称为一个线程，每个线程定义了一个单独的执行路径。

基于进程的多任务处理是程序的并发执行。基于线程的多任务处理是同一程序的片段的并发执行。

3. 进程的操作

创建进程有两种方式，一是由操作系统创建；二是由父进程创建。操作系统创建的进程，它们之间是平等的，一般不存在资源继承关系。而由父进程创建的进程（子进程），它们和父进程存在隶属关系。子进程又可以创建进程，形成一个进程家族。

fork（）函数调用后有 2 个返回值，调用一次，返回两次。成功调用 fork 函数后，当前进程实际上已经分裂为两个进程，一个是原来的父进程，另一个是刚刚创建的子进程。fork（）函数的 2 个返回值，一个是父进程调用 fork 函数后的返回值，该返回值是刚刚创建的子进程的 ID；另一个是子进程中 fork 函数的返回值，该返回值是 0。这样可以用返回

值来区分父、子进程。

4. **进程的三种基本状态：**

（1） 就绪状态：进程已获得除 CPU 外的所有必要资源，只等待 CPU 时的状态。一个系统会将多个处于就绪状态的进程排成一个就绪队列。

（2） 执行状态：进程已获 CPU，正在执行。单处理机系统中，处于执行状态的进程只一个；多处理机系统中，有多个处于执行状态的进程。

（3） 阻塞状态：正在执行的进程由于某种原因而暂时无法继续执行，便放弃处理机而处于暂停状态，即进程执行受阻。（这种状态又称等待状态或封锁状态）

5. **Fork（） （创建单独的子进程）**

一个进程，包括代码、数据和分配给进程的资源。fork（）函数通过系统调用创建一个与原来进程几乎完全相同的进程，也就是两个进程可以做完全相同的事，但如果初始参数或者传入的变量不同，两个进程也可以做不同的事。

一个进程调用 fork（）函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程的所有值都复制到新的新进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。

6. **Exec（） （执行用户命令）**

用 fork 函数创建子进程后，子进程往往要调用一种 exec 函数以执行另一个程序。当进程调用一种 exec 函数时，该进程执行的程序完全替换为新程序，而新程序则从其 main 函数开始执行。因为调用 exec 并不创建新进程，所以前后的进程 ID 并未改变。exec 只是用一个全新的程序替换了当前进程的正文、数据、堆和栈段。

三、实验过程

```
ımoon@cent 2]$ vi Command.c
ımoon@cent 2]$ gcc -g -o Command Command.c
ımoon@cent 2]$ ./Command
```

用 vi 编辑器编辑代码后进行编译并运行

```
osh>Please enter the command : (Quit please enter q)
pwd
/home/windamoon/Desktop/operation sysytem/2
```

弹出提示符，输入命令即可调用系统命令进行运行

```
osh>Please enter the command : (Quit please enter q)
history
6 ps -l
5 ds -l
4 whoami
3 pwd
2 date
1 df
```

输入 history 即可查看历史命令

```
>Please enter the command : (Quit please enter q)

!!
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   3441   3430  0  80   0 - 29230 do_wai pts/0    00:00:00 bas
0 T  1000   4886   3441  0  80   0 -  1088 do_sig pts/0    00:00:00 Com
0 T  1000   4943   3441  0  80   0 -  1088 do_sig pts/0    00:00:00 Com
0 S  1000   7619   3441  0  80   0 -  1089 do_wai pts/0    00:00:00 Com
0 R  1000   7635   7619  0  80   0 - 38331 -      pts/0    00:00:00 ps
!3
/home/windamoon/Desktop/operation sysytem/2
```

输入！！即可执行最近一条命令

输入！N 即可执行第 N 条指令

```
>Please enter the command : (Quit please enter q)

q
```

输入 q 即可退出程序

四、实验结果与分析

```
moon@cent 2]$ ./Command
  osh>Please enter the command : (Quit please enter q)
df
文件系统                    1K-块    已用    可用 已用% 挂载点
devtmpfs                  915524       0  915524    0% /dev
tmpfs                     932640       0  932640    0% /dev/shm
tmpfs                     932640   10672  921968    2% /run
tmpfs                     932640       0  932640    0% /sys/fs/cgroup
/dev/mapper/centos-root 10475520 7618796 2856724   73% /
/dev/sda1                 201380  170060   31320   85% /boot
tmpfs                     186532      44  186488    1% /run/user/1000
/dev/sr0                 4364408 4364408       0  100% /run/media/windamoon/Cent
OS 7 x86_64
tmpfs                     186532       0  186532    0% /run/user/0
  osh>Please enter the command : (Quit please enter q)
date
2023年 03月 31日 星期五 21:35:16 CST
  osh>Please enter the command : (Quit please enter q)
pwd
/home/windamoon/Desktop/operation sysytem/2
```

运行程序后，弹出类终端用户的提示，即 osh>。

根据输入提示，依次输入几个命令模拟命令行窗口输入，都成功执行

```
  osh>Please enter the command : (Quit please enter q)
ds -l
sh: ds: 未找到命令
  osh>Please enter the command : (Quit please enter q)
ps -l
F S   UID    PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   3441   3430  0  80   0 - 29230 do_wai pts/0    00:00:00 bas
0 T  1000   4886   3441  0  80   0 -  1088 do_sig pts/0    00:00:00 Com
0 T  1000   4943   3441  0  80   0 -  1088 do_sig pts/0    00:00:00 Com
0 S  1000   5298   3441  0  80   0 -  1089 do_wai pts/0    00:00:00 Com
0 R  1000   5335   5298  0  80   0 - 38331 -      pts/0    00:00:00 ps
  osh>Please enter the command : (Quit please enter q)
history
6 ps -l
5 ds -l
4 whoami
3 pwd
2 date
1 df
```

当输入不合法命令时，系统给出提示，未找到命令

当输入 histroy 时，按照时间线由近及远的顺序打印历史指令

```
  osh>Please enter the command : (Quit please enter q)
!!
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   3441   3430  0  80   0 - 29230 do_wai pts/0    00:00:00 bas
0 T  1000   4886   3441  0  80   0 -  1088 do_sig pts/0    00:00:00 Com
0 T  1000   4943   3441  0  80   0 -  1088 do_sig pts/0    00:00:00 Com
0 S  1000   7619   3441  0  80   0 -  1089 do_wai pts/0    00:00:00 Com
0 R  1000   7635   7619  0  80   0 - 38331 -      pts/0    00:00:00 ps
  osh>Please enter the command : (Quit please enter q)
!3
/home/windamoon/Desktop/operation sysytem/2

[windamoon@cent 2]$ ./Command
  osh>Please enter the command : (Quit please enter q)
!!
No command in history
  osh>Please enter the command : (Quit please enter q)
date
2023年 03月 31日 星期五 22:28:52 CST
  osh>Please enter the command : (Quit please enter q)
history
1 date
  osh>Please enter the command : (Quit please enter q)
!3
No such command in history
  osh>Please enter the command : (Quit please enter q)
q
```

当输入！！时，执行最近一条指令，由上 history 指令运行结果可以发现是 ps -l 指令，如图，成功运行 ds -l；当 history 中无最近指令时，输出 history 中无历史命令的提示

当输入！N 时，执行第 N 条指令，若第 N 条指令不存在，则输出无此命令的提示

输入 q 即中止程序，退出界面

1.进程的创建

（1） 首先将用户输入的命令保存在特定数组中。定义一个函数 read，实现将用户输入分解为多个令牌，并将这些令牌保存到字符串数组中。最后，该函数返回 count 命令字数的值。

（2） 首先将数组的内容传递到 exe 函数中执行。

再定义一个 command 函数，使其实现创立一个子进程并执行用户指定的命令。最后检查检查用户输入，以此查看父进程和子进程终止的先后顺序是否正确。

2.提供创建 history 功能

首先编写一个函数来保存用户历史输入的命令。该程序需要考虑到当输入各种错误字符时候的应对方式。在主函数中，当用户输入 history 命令时，数组 history 中的命令能够按照由近及远依次输出。当用户输入！！时，执行 history-read 函数，读取打印最近的用户命令。当在用户输入时执行！n 时，读取打印第 n 个用户命令。

五、问题总结

1. 没有区别！N 指令与 history 的不同识别方式

识别“！N”指令时，需要先识别第一个字符是否为“！”,N 只是一个名称，不能直接进行整个字符串的识别。

2. 线程的优点：

1）创建一个新线程的代价要比创建一个新进程小得多
2）与进程之间的切换相比，线程之间的切换需要操作系统做的工作要少很多
3）线程占用的资源要比进程少很多
4）能充分利用多处理器的可并行数量
5）在等待慢速 I/O 操作结束的同时，程序可执行其他的计算任务
6）计算密集型应用，为了能在多处理器系统上运行，将计算分解到多个线程中实现
7）I/O 密集型应用，为了提高性能，将 I/O 操作重叠。线程可以同时等待不同的 I/O 操作。

3. 线程的缺点

1）性能损失

一个很少被外部事件阻塞的计算密集型线程往往无法与共它线程共享同一个处理器。如果计算密集型线程的数量比可用的处理器多，那么可能会有较大的性能损失，这里的性能损失指的是增加了额外的同步和调度开销，而可用的资源不变。

2）健壮性降低

编写多线程需要更全面更深入的考虑，在一个多线程程序里，因时间分配上的细微偏差或者因共享了不该共享的变量而造成不良影响的可能性是很大的，换句话说线程之间是缺乏保护的。

3）缺乏访问控制

进程是访问控制的基本粒度，在一个线程中调用某些 OS 函数会对整个进程造成影响。

4）编程难度提高

编写与调试一个多线程程序比单线程程序困难得多。

4. 父子进程终止的先后顺序不同会产生不同的结果。

在子进程退出前父进程先退出，则系统会让 init 进程接管子进程。

当子进程先于父进程终止，而父进程又没有调用 wait 函数等待子进程结束，子进程进入僵尸状态，并且会一直保持下去除非系统重启。子进程处于僵尸状态时，内核只保存该进程的一些必要信息以备父进程所需。此时子进程始终占用着资源，同时也减少了系统可以创建的最大进程数。如果子进程先于父进程终止，且父进程调用了 wait 或 waitpid 函数，则父进程会等待子进程结束。

在 Linux 下，可以简单地将 SIGCHLD 信号的操作设为 SIG_IGN，这样当子进程结束

时就不会变为僵尸进程。

六、源代码

```
#include<stdio.h>
#include <unistd.h>
#include<string.h>
#include<stdlib.h>
#include <wait.h>

#define MAX_LINE 80
int input(char* str);
int str_to_args(char* str,char* args[],int str_num);
void display_history(int history_front,int history_rear);
void add_history(char* str,int* front,int* rear);

char history[10][MAX_LINE];//history queue

int input(char* str){
    char c;
    int i = 0;
    while((c = getchar())!='\n' && i<MAX_LINE){
        str[i] = c;
        i++;
    }
    if(i == MAX_LINE && c != '\n'){        printf("over maximum length!");
        return 0;
    }
    else{
        str[i] = 0;
        return i;
    }
}

int str_to_args(char* str,char* args[],int str_num){
    const char s[2] = " ";
    int i = 0;
    char* temp;

    temp = strtok(str,s);
    while(temp != NULL){
        args[i] = (char*)malloc(strlen(temp));
```

```
        strcpy(args[i],temp);
        i++;
        temp = strtok(NULL,s);
    }
    args[i] = 0;
    return i;
}
void display_history(int history_front,int history_rear){
    int i;
    for(i = history_front;i < history_rear;i++){
        printf("%d\t%s\n",history_rear - i,history[i%10]);
    }
}
void add_history(char* str,int* front,int* rear){
    strcpy(history[*rear % 10],str);
        *rear = *rear+1;
    if(*rear - *front > 10)
        *front++;
}

int main(void){
    char* args[MAX_LINE/2+1];//execvp's argslist
    int should_run = 1;
    int args_num = 0;

    char str[MAX_LINE];// user input
    int str_num;

    int history_front = 0;
    int history_rear = 0;

    int i = 0;
    int background = 0;
    int pid_status;

    while(should_run){
        printf(" osh>Please enter the command : (Quit please enter q)\n");
        fflush(stdout);

        str_num = input(str);

        if(str_num == 0){
            continue;
        }
```

```
        if(strcmp(str,"q")== 0){
            should_run = 0;
            continue;
        }
        if(strcmp(str,"history") == 0){
            display_history(history_front,history_rear);
        }
        if(strcmp(str,"!!") == 0){
            if(history_rear != 0){
                strcpy(str,history[(history_rear -1) % 10]);
                str_num = strlen(str);
            }
            else{
                printf("No command in history.\n");
            }
        }
        else if(str[0] == '!'){
            if(str[1] <= '0' || str[1] > '9' || (str[1]-'0') >
history_rear-history_front){
                printf("No such command in history.\n");
            }
            else if(str[1] == '1'){
                if(str[2] == '0')
                    strcpy(str,history[history_front %10]);
                else if(str[2] == 0)
                    strcpy(str,history[(history_rear-1) %10]);
                else
                    printf("No such command in history.\n");
            }
            else
                strcpy(str,history[history_rear - str[1] + '0' -1]);
        }

        add_history(str,&history_front,&history_rear);

        args_num = str_to_args(str,args,str_num);

        if(strcmp(args[args_num-1],"&") == 0){
            background = 1;
            args_num--;
            args[args_num] = NULL;
        }
```

```
        pid_t pid = fork();
        if(pid == 0){
            pid_status = execvp(args[0], args);
        }
        else{
            if(background == 1){
                printf("%d is running in background %s \n",pid,str);
            }
            else{
                wait(&pid_status);
            }
        }
        background  = 0;
    }
    return 0;
```