实验 01：具有历史功能的 UNIX 外壳

截止日期

请参考实验室作业要求。

目标

本项目的目标是：（1）对操作系统外壳和系统调用有良好的理解；（2）练习创建进程并协调父进程和子进程的运行。

细节

该项目包括设计一个 C 程序，作为外壳界面，接受用户命令，然后在单独的进程中执行每个命令。该项目可以在任何 Linux、UNIX 或 Mac OS X 系统上完成。shell 界面会给用户一个提示符，之后输入下一条命令。下面的示例展示了诸如“osh>”这样的提示符以及用户的下一条命令：cat prog.c 。（这条命令使用 UNIX 的 cat 命令在终端上显示文件 prog.c 。）

osh> 猫 prog.c

实现 shell 界面的一种技术是，让父进程首先读取用户在命令行输入的内容（在本例中，为 cat prog.c ），然后创建一个单独的子进程来执行该命令。除非另有指定，父进程会在子进程退出后继续。然而，UNIX 外壳通常还允许子进程在后台运行，或者并发运行。为了实现这一点，我们在命令末尾添加一个 ampersand（&）。因此，如果我们重写上述命令为

osh> 猫 prog.c 并

父进程和子进程将并发运行。

单独的子进程是通过 fork（） 系统调用创建的，而用户的命令是通过 exec（） 系列中的一个系统调用执行的。

以下代码片段提供了一个 C 程序，该程序实现了命令行外壳的一般操作。main（） 函数呈现提示符 osh-> 并概述了读取用户输入后应采取的步骤。只要 should run 等于 1，main（） 函数就会持续循环；当用户在提示符处输入 exit 时，您的程序会将 should run 设置为 0 并终止。

```
#include <stdio.h>
#include <unistd.h>
```

```c
#define MAX_LINE 80 /* The maximum length command */
int main(void)
{
    char *args[MAX_LINE / 2 + 1]; /* command line arguments */
    int should_run = 1; /* flag to determine when to exit program */
    while (should_run)
    {
        printf("osh>");
        fflush(stdout);
        /**
        * After reading user input, the steps are:
        * (1) fork a child process using fork()
        * (2) the child process will invoke execvp()
        * (3) if command included &, parent will invoke wait()
        */
    }
    return 0;
}
```

该项目分为两部分：（1）创建子进程并在子进程中执行命令；（2）修改 shell 以实现历史功能。

第一部分 - 创建子进程

第一个任务是修改上述代码中的 main（） 函数，以便分叉一个子进程并执行用户指定的命令。这将需要把用户输入的内容解析为单独的标记，并将这些标记存储在一个字符串数组中（上述代码中的 args）。例如，如果用户在 osh> 提示符下输入命令 ps -ael，存储在 args 数组中的值为：

```
args[0] = "ps"
args[1] = "-ael"
args[2] = NULL
```

这个 args 数组将被传递给 execvp（） 函数，该函数具有以下原型：execvp（char *command, char *params[]） ；
在这里，command 表示要执行的命令，params 存储该命令的参数。对于这个项目，execvp（） 函数应该以 execvp（args[0], args） 的形式调用。一定要检查用户是否包含了“&”符号，以确定父进程是否要等待子进程退出。

第二部分 - 创建历史专题

接下来的任务是修改 shell 界面程序，使其提供历史记录功能，允许用户访问最近输入的命令。用户将能够

通过使用该功能，最多可以访问 10 条命令。这些命令将从 1 开始连续编号，编号超过 10 会继续。例如，如果用户输入了 35 条命令，最近的 10 条命令将编号为 26 至 35。
用户将能够通过输入命令来列出命令历史记录。

历史

在 osh> 提示符下。例如，假设历史记录包含以下命令（从最近到最旧）：

```
ps, ls -l, top, cal, who, date
```

命令历史记录将会输出：

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

您的程序应该支持两种从命令历史记录中检索命令的技术：

1. 当用户输入“！！”时，会执行历史记录中的最近一条命令。
2. 当用户输入一个单独的“！”后跟一个整数 N 时，会执行历史记录中的第 N 条命令。

继续我们上面的例子，如果用户输入！！，则执行ps命令；如果用户输入！3，则执行cal命令。以这种方式执行的任何命令都应在用户的屏幕上回显。该命令还应作为下一个命令放入历史缓冲区。

该程序还应处理基本的错误处理。如果历史记录中没有命令，输入！！应显示一条消息“历史记录中没有命令。”如果输入单个！后输入的数字没有对应的命令，程序应输出“历史记录中没有此类命令。”

提交

您的提交内容应包括代码、一份简要描述您的设计的自述文件、如何编译/使用您的代码以及在结对编程的情况下所做的贡献，还有一份报告，报告应包括以下部分（但不限于）：

- ☒ 总结 Linux（或 Windows，如适用）提供的流程控制方法，并描述如何使用它们。
- ☒ 您对该程序的设计
- ☒ 实验结果（统计数据）的截图及分析
- ☒ 遇到的问题及您的解决方案
- ☒ 参考资料
- ☒ 您的建议和意见

环境

Linux（推荐使用，2.6 之后的任何内核版本均可）以及 C/C++ 。

参考文献

N/A