UNIX Shell with History Feature

Name：程维森

Number:21231264

一：实验介绍

这个项目的任务是设计一个 C 程序，作为一个具有历史功能的 Shell 界面，它接受用户命令，然后在一个独立的进程中执行每个命令。

项目实现分为两个部分：

第一部分为实现 shell 功能，第二部分则为展示历史命令。

二：实验梗概

2.1 实现 shell 命令：

思路：

首先实现接受一个长的字符串，采用回车结束，同时需要分割字符串来达成“命令+参数”的模式，具体的代码可以见如下形式：

```
#include<bits/stdc++.h>
using namespace std;

int main() {
    char input[1000];
    char* args[100];
    int i = 0;
    fgets(input, sizeof(input), stdin);
    input[strcspn(input, "\n")] = '\0';

    char *p = strtok(input, " ");
    while (p) {
        args[i] = p;
        p = strtok(NULL, " ");
        i++;
    }
    args[i] = NULL;

    for (int j = 1; args[j] != NULL; j++) {
        printf("%s\n", args[j]);
    }

    return 0;
}

```

然后使用无限循环（while）创建一个 Shell，通过 fork()系统调用来生成子进程。在 fork()中，如果 pid 等于 0，则代表当前正在执行的代码是子进程，然后使用 execvp()函数来执行用户输入的命令。execvp()函数用于加载并执行新的程序。

在 fork()系统调用中，父进程会创建一个子进程，而子进程将复制父进程的状态。如果 fork()返回 0，那么当前的执行上下文是在子进程中。在子进程中，可以使用 execvp()函数加载并执行用户指定的命令。execvp()会覆盖当前进程的内存映像，将其替换为新的程序，从而执行用户命令。

这个过程使得 Shell 能够同时创建子进程来执行用户输入的命令，而不会影响 Shell 本身的执行。

由于篇幅过长，代码放在最后的部分展示。

2.2 实现 history 功能

命令历史记录数据结构：首先，需要创建一个数据结构来存储命令历史记录。通常，这可以通过一个数组或循环队列来实现。在这个示例代码中，使用了一个固定大小为 10 的数组 command_history 来存储历史记录。

记录用户输入：每当用户输入一个命令，Shell 需要将该命令记录到命令历史中。这是在用户输入命令后，解析并执行命令之前完成的。在示例代码中，记录是通过以下行完成的：

```
memcpy(command_history[history_count % MAX_HISTORY_SIZE], input_command, MAX_COMMAND_LENGTH + 1);
```

这将用户输入的命令复制到历史记录数组的下一个位置。

显示历史记录：当用户输入"history"命令时，Shell 需要显示命令历史记录。在示例代码中，这是通过以下行完成的：

display_history(command_history, history_count);

display_history

函数遍历历史记录数组并打印出存储的命令。

重新执行历史命令：Shell 还需要支持重新执行历史命令的功能。在示例代码中，如果用户输入"!!"，则 Shell 会从历史记录中获取最后一次执行的命令，将其放回输入中，并再次执行。如果用户输入"!N"（N 为命令编号），则 Shell 会找到历史记录中的相应命令，将其放回输入，并再次执行。

这一部分是通过解析用户输入来实现的，然后从历史记录中检索相应的命令，并将其放回输入缓冲区中以进行执行。

三．操作示例与代码运行

可以观察到实现了 shell 命令与 history 命令

示例代码附在附件中，增加了一些美化与判断格式并且增加了输入输出的详细指示，由于篇幅问题不予展示