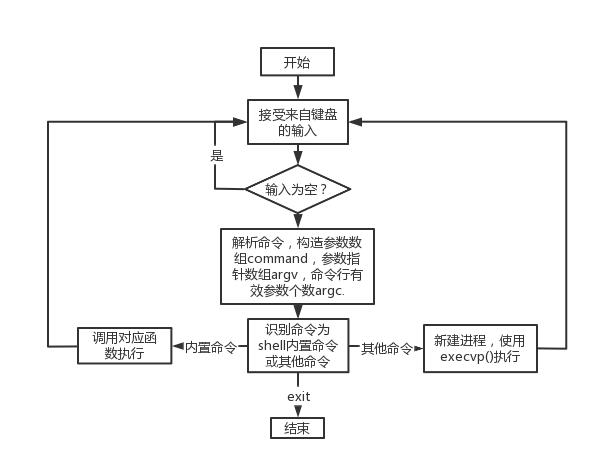
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实习题目：shell 程序 | | | | |
| 完成人 | 姓名： | 课序号： | 班级： | 学号： |
| 实习内容及要求简要描述 | 本实验要实现一个简单的命令解释器，也就是 Linux 中的 shell 程序。实验程序起名为 ysh，要求其设计类似于目前流行的 shell 解释程序，如 bash、csh、tcsh，但不需要具备那么复杂的功能。ysh 程序应当具有如下一些重要的特征：  （1）能够执行外部程序命令，命令可以带参数。  （2）能够执行 fg、bg、cd、history、exit 等内部命令。  （3）使用管道和输入输出重定向。  （4）支持前后台作业，提供作业控制功能，包括打印作业的清单，改变当前运行作业的前台/后台状态，以及控制作业的挂起、中止和继续运行。  除此之外，在这个实验中还须做到：  （1）使用 make 工具建立工程。  （2）使用调试器 gdb 来调试程序。  （3）提供清晰、详细的设计文档和解决方案。 | | | |
| 主要代码结构  （附注释） | （1）头文件  #include<stdio.h>  #include<string.h>  #include<ctype.h>  #include<unistd.h>  #include<stdlib.h>  #include <sys/types.h>  #include <sys/wait.h>  （2）宏定义  #define MAX\_CMD 10    // 最大命令数量  #define BUFFSIZE 255   // 输入最大命令字符数  #define MAX\_CMD\_LEN 100   // 每条命令的最大长度  #define IN 1  #define OUT 0  （3）全局变量  int argc;      // 命令行的有效参数个数  char\* argv[MAX\_CMD];      // 参数指针数组  char command[MAX\_CMD][MAX\_CMD\_LEN]; //参数二维数组  char buf[BUFFSIZE];    // 接受键盘输入的参数数组  int commandNum;   // 已经输入指令数目  char history[MAX\_CMD][BUFFSIZE];    // 存放历史命令   1. 函数声明   1.shell主要架构  **input函数**  @brief 接受输入的字符  @param buf 存放命令输入的数组  @return int 即输入字符的数量  int input(char buf[]);  **analysis函数**  @brief 解析输入的buf字符串  @param buf  @return void  void analysis(char\* buf);  **work函数**  @brief 判断并执行命令  @param argc  @param argv  void work(int argc, char\* argv[]);  2.shell内置命令实现  **printHistory函数**  @brief 打印历史命令  @param char command[MAX\_CMD][MAX\_CMD\_LEN] 存放历史命令的二维数组  @return void  int printHistory(char command[MAX\_CMD][MAX\_CMD\_LEN]);  **helpCd函数**  @brief 执行cd命令  @param args 命令参数数量  @return void  int helpCd(int argc); | | | |
| 结果分析（或错误原因分析） | 该程序可以执行cd,history等shell内置命令，以及一些常见的linux命令。对于输入不合法的指令或路径不存在等错误能准确报错。不足之处在于没有实现重定向命令、后台运行命令、管道命令。  测试了ls、pwd、cd、vi、history等命令，均运行成功。 | | | |

附录一、流程图

****

附录二、源程序

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#include<unistd.h>

#include<stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#define IN 1

#define OUT 0

#define MAX\_CMD 10    // 最大命令数量

#define BUFFSIZE 255   // 输入最大命令字符数

#define MAX\_CMD\_LEN 100   // 每条命令的最大长度

/\* 全局变量 \*/

int argc;      // 命令行的有效参数个数

char\* argv[MAX\_CMD];      // 参数指针数组

char command[MAX\_CMD][MAX\_CMD\_LEN]; // 参数二维数组

char buf[BUFFSIZE];    // 接受键盘输入的参数数组

int commandNum;         // 已经输入指令数目

char history[MAX\_CMD][BUFFSIZE];        // 存放历史命令

/\* 函数声明 \*/

int input(char buf[]);

void analysis(char\* buf);

void work(int argc, char\* argv[]);

int printHistory(char command[MAX\_CMD][MAX\_CMD\_LEN]);

int helpCd(int argc);

/\* 函数实现 \*/

int printHistory(char command[MAX\_CMD][MAX\_CMD\_LEN]) {

    int n = atoi(command[1]);   // 将数目从字符串转为int，即打印前n条指令

    for (int i = n; i > 0 && commandNum - i >= 0; i--) {

        printf("%d\t%s\n", n - i + 1, history[commandNum - i]);

    }

    return 0;

}

int input(char buf[]) {

    // buf和backBuf数组初始化

    memset(buf, 0x00, BUFFSIZE);

    fgets(buf, BUFFSIZE, stdin);

    // 去除fgets带来的末尾\n字符，使其以空字符结尾，便于后续指令的执行

buf[strlen(buf) - 1] = '\0';

//返回字符数量

    return strlen(buf);

}

void analysis(char\* buf) {

    // 初始化argv和command数组

    for (int i = 0; i < MAX\_CMD; i++) {

        argv[i] = NULL;

        for (int j = 0; j < MAX\_CMD\_LEN; j++)

            command[i][j] = '\0';

    }

    argc = 0;

    // 构建command数组

    int len = strlen(buf);

    int i,j;

    for ( i = 0, j = 0; i < len; ++i) {

        if (buf[i] != ' ') {

            command[argc][j++] = buf[i];

        } else {

            if (j != 0) {

                command[argc][j] = '\0';

                ++argc;

                j = 0;

            }

        }

    }

    if (j != 0) {

        command[argc][j] = '\0';

    }

    argc = 0;

    int flg = OUT;

    for (i = 0; buf[i] != '\0'; i++) {

        if (flg == OUT && !isspace(buf[i])) {

            flg = IN;

            argv[argc++] = buf + i;

        } else if (flg == IN && isspace(buf[i])) {

            flg = OUT;

            buf[i] = '\0';

        }

    }

    argv[argc] = NULL;

}

void work(int argc, char\* argv[]) {

    pid\_t pid;

    /\* 识别shell内置命令 \*/

    if (strcmp(command[0], "cd") == 0) {

        int res = helpCd(argc);

        if (!res)

            printf("cd指令输入错误!\n");

    } else if (strcmp(command[0], "history") == 0) {

        printHistory(command);

    } else if (strcmp(command[0], "exit") == 0) {

        exit(0);

    } else {

        switch(pid = fork()) {

            // fork子进程失败

            case -1:

                printf("创建子进程未成功");

                return;

            // 处理子进程

            case 0:

                    execvp(argv[0], argv);

                  // 如果命令成功执行则不会执行以下两条语句，如果子进程未被成功执行, 则报错

                    printf("%s: 命令输入错误\n", argv[0]);

                    exit(1);

                }

//主进程

            default: {

                    int status=0;

                    waitpid(pid, &status, 0);    // 等待子进程返回

                    int err = WEXITSTATUS(status); // 读取子进程的返回码

                    if (err) {

                        printf("Error: %s\n", strerror(err));

                    }

            }

        }

    }

}

int helpCd(int argc) {

    int result = 1;    // 1代表执行成功, 为0代表执行失败

    if (argc != 2) {

        printf("指令数目错误!");

    } else {

        int ret = chdir(command[1]);

        if (ret) return 0;

    }

    if (result) {

        char Path[100];

        char\* res = getcwd(Path, sizeof(Path));//将当前的工作目录改变成以参数path 所指的目录

        if (res == NULL) {

            printf("文件路径不存在!");

        }

        return result;

    }

    return 0;

}

int main() {

    while(1) {

        printf("ysh>");

        // 接受来自键盘的输入, 如果输入字符(不包括回车)为0, 则跳过开始下一次循环

        if (input(buf) == 0)

            continue;

//存入历史命令数组

        strcpy(history[commandNum++], buf);

//分析命令，构建指令数组，指令参数数组

        analysis(buf);

//执行指令

        work(argc, argv);

        argc = 0；

    }

    return 0;

}

附录三、运行结果截屏

