





学号：202200130223		姓名：李洋永	班级：4 班
实验题目：实验 3 进程综合实验（Shell）			
实验学时：2		实验日期：2024/10/23	
实验目的：掌握操作系统 shell 的工作机制与实现过程，练习 Linux 系统中进程创建与控制有关的编程和调试技术。			
实验环境：Ubuntu			
源程序清单：			
			
a	a.c	b.txt	input.txt
编译及运行结果：			
<pre>q@q-virtual-machine:~/Desktop/lab3\$ gcc a.c -o a -lreadline q@q-virtual-machine:~/Desktop/lab3\$ ./a lab3&gt;ls -la 总计 40 drwxrwxr-x 2 q q 4096 10月 23 20:26 . drwxr-xr-x 8 q q 4096 10月 23 20:11 .. -rwxrwxr-x 1 q q 16816 10月 23 20:26 a -rw-rw-r-- 1 q q 3575 10月 23 20:15 a.c -rw-r--r-- 1 q q 6 10月 23 20:23 b.txt -rw-rw-r-- 1 q q 3 10月 23 19:55 input.txt lab3&gt;ps -la F S  UID      PID      PPID  C  PRI  NI ADDR  SZ  WCHAN  TTY          TIME CMD 0 S  1000      1155      1147  0   80   0  - 56575 do_pol tty2      00:00:00 gnome-session-b 0 S  1000      8199      8178  0   80   0  - 3035 do_wai pts/0      00:00:00 a 4 R  1000      8204      8199  0   80   0  - 3945 -      pts/0      00:00:00 ps lab3&gt;echo hello hello lab3&gt;echo hello &gt; b.txt lab3&gt;cat b.txt hello lab3&gt;cat &lt; output.txt Input redirection failed: No such file or directory lab3&gt;cat &lt; input.txt in lab3&gt;cat b.txt   wc       1      1      6 lab3&gt;&gt;false_command Command execution failed: No such file or directory lab3&gt;cat b.txt &amp; Running command in background with PID: 8233 lab3&gt;hello  lab3&gt;exit q@q-virtual-machine:~/Desktop/lab3\$</pre>			

```

int main() {
    char *cmd;

    signal(SIGINT, sigint_handler); // 捕获Ctrl+C信号

    while (running) {
        cmd = readline("lab3>"); // 使用readline获取输入
        if (cmd == NULL) break; // EOF, 退出

        if (strcmp(cmd, "exit") == 0) { // 退出
            free(cmd);
            break;
        }

        add_history(cmd); // 将命令添加到历史中,直接调用readline中的函数
        execute_command(cmd);
        free(cmd); // 释放命令字符串
    }

    return 0;
}

```

While 循环中 readline 将输出放入 cmd，如果输入“exit”则退出，调用 add\_history 函数，可直接保存执行过的命令，上下方向键选择也调用库中的函数。

```

8 #include <readline/readline.h>
9 #include <readline/history.h>
-

```

```

void execute_command(char *cmd) {
    char *args[MAX_ARGS];
    char *input_file = NULL, *output_file = NULL;
    int background = 0;

    // 处理管道
    char *pipe_pos = strchr(cmd, '|');
    if (pipe_pos) {
        *pipe_pos = '\0'; // 分割命令

        // 创建管道
        int pipefd[2];
        if (pipe(pipefd) == -1) {
            perror("Pipe creation failed");
            return;
        }

        // 执行左侧命令
        if (fork() == 0) {
            dup2(pipefd[1], STDOUT_FILENO); // 输出到管道
            close(pipefd[0]);
            close(pipefd[1]);
            execute_command(cmd); // 递归执行左侧命令
            exit(0);
        }

        // 执行右侧命令
        if (fork() == 0) {
            dup2(pipefd[0], STDIN_FILENO); // 输入来自管道
            close(pipefd[1]);
            close(pipefd[0]);
            execute_command(pipe_pos + 1); // 递归执行右侧命令
            exit(0);
        }

        close(pipefd[0]);
        close(pipefd[1]);
        wait(NULL); // 等待左侧进程结束
        wait(NULL); // 等待右侧进程结束
        return;
    }
}

```

执行函数，如果有管道命令则分割命令字符串，创建管道，将数据传入管道。

```

// 处理重定向和背景运行
char *token = strtok(cmd, " \n");
int i = 0;
while (token != NULL) {
    if (strcmp(token, "&") == 0) {
        background = 1; // 设置后台标志
        break;
    } else if (strcmp(token, ">") == 0) {
        output_file = strtok(NULL, " \n");
    } else if (strcmp(token, "<") == 0) {
        input_file = strtok(NULL, " \n");
    } else {
        args[i++] = token;
    }
    token = strtok(NULL, " \n");
}
args[i] = NULL;

```

如果查找到&，则设置后台标志，background 设为 1，如果查找到重定向标识则更新对应的 input\_file 和 output\_file。

```

pid_t pid;
if ((pid = fork()) == 0) { // 子进程
    if (input_file) {
        int fd_in = open(input_file, O_RDONLY);
        if (fd_in < 0) {
            perror("Input redirection failed");
            exit(EXIT_FAILURE);
        }
        dup2(fd_in, STDIN_FILENO);
        close(fd_in);
    }
    if (output_file) {
        int fd_out = open(output_file, O_WRONLY | O_CREAT | O_TRUNC, 0644);
        if (fd_out < 0) {
            perror("Output redirection failed");
            exit(EXIT_FAILURE);
        }
        dup2(fd_out, STDOUT_FILENO);
        close(fd_out);
    }
    execvp(args[0], args);
    perror("Command execution failed");
    exit(EXIT_FAILURE);
}

// 在后台运行
if (background) {
    printf("Running command in background with PID: %d\n", pid);
} else {
    wait(NULL); // 等待前台进程结束
}

```

重定向及后台执行过程，如果在后台运行则输出提示及其 pid。

问题及收获：

对比一个真实的 Linux Shell，我实现的 Shell 只支持简单命令和单一管道，只支持基本的重定向和简单的历史管理，错误处理也比较单一，如果要改进的话可以解析输入命令，将内置命令（如 cd, exit）直接在 Shell 中处理，而不是直接调用外部程序，要实现多个管道的话需要使用递归调用且要创建多个进程，对于错误处理，可以针对不同的错误提供详细的错误提示，为实现更加完善的历史搜索，可以使用 Ctrl+R，并允许用户编辑历史命令。