SHELL 挑战性任务实验报告

实验要求

本任务需要你基于MOS LAB6中的Shell进行增强，完成任务内容一节中的要求

思路分析

注释功能

问题描述

你需要使用 `#` 实现注释功能，例如 `ls | cat # this is a comment meow`，`ls | cat` 会被正确执行，而后面的注释则会被抛弃

解决方案

在处理指令的时候，对于 `#` 之后的字符串直接置 `\0`

```
        int i;
        for(i = 0; buf[i]; i += 1) {
            if(buf[i] == '#') {
                buf[i] = '\0';
                break;
            }
        }
```

不带 `.b` 后缀指令

问题描述

你需要实现不带 `.b` 后缀的指令，但仍需兼容带有 `.b` 后缀的指令，如 `ls` 与 `ls.b` 都应能够正确列出当前目录下的文件

解决方案

对于 `spawn.c` 文件中处理单个指令时，统一加上 `.b` 即可

```
    char ptmp[512] = {0};
    strcpy(ptmp, prog);
    int plen = strlen(prog);
    ptmp[plen ++] = '.';
    ptmp[plen ++] = 'b';
```

一行多指令

问题描述

你需要支持使用`;`将指令隔开，并按顺序执行，比如:

```
ls;ls;ls;ls
```

解决方案

处理指令时按照`;`进行分割存储，按照管道进行处理即可

```
int parsecmdlist(char **cmds, char *s) {
    while (*s && strchr(WHITESPACE SYMBOLS, *s)) {
        s++;
    }
    if (*s == 0) {
        return 0;
    }
    int cnt = 1, flag = 1;
    cmds[0] = s;
    while (*s)
    {
        if (*s == ';') {
            *s = 0;
            s++;
            while (*s && strchr(WHITESPACE SYMBOLS, *s)) {
                s++;
            }
            if (*s == 0) {
                return cnt;
            } else {
                cmds[cnt++] = s;
            }
        } else {
            s++;
        }
    }
    return cnt;
}
```

```
            int child = spawn(argv[0], argv);
            if(lst) close_all();
            if (child >= 0) {
                wait(child);
            } else {
                debugf("spawn %s: %d\n", argv[0], child);
            }
            if (rightpipe) {
                wait(rightpipe);
            }
            if(lst) exit();
```

`cd` 指令

问题描述

输入	行为	输出	返回值
`cd`	切换工作目录到 `/`	无	0
`cd <abspath>`	若绝对路径 `<abspath>` 存在且为目录，切换到该目录	无	0
`cd <relpath>`	根据当前工作路径，对相对路径 `<relpath>`，若存在且为目录，切换到该目录	无	0
`cd <noexist_path>`	路径不存在	`cd: The directory '原始输入' does not exist\n`	1
`cd <filepath>`	路径存在但不是目录	`cd: '原始输入' is not a directory\n`	1
`cd <arg1>.. <argn>`	输入多于 1 个参数	`Too many args for cd command\n`	1
`cd .`	解析为当前目录 `/dir1/dir2`，无变化	无	0

输入	行为	输出	返回值
`cd ..`	解析为 `/dir1`，若存在且为目录，切换到该目录	无	0

解决方案

通过系统调用存储当前进程的绝对路径，之后进行字符串路径处理即可

```
    [SYS_chdir] = sys_chdir,
    [SYS_getcwd] = sys_getcwd,
```

```
int command_cd(int argc, char *argv[]) {
    char *path = argv[0];
    if (argc == 1) {
        syscall_chdir("/");
        syscall_set_value(0);
        return 0;
    }
    if (argc > 2) {
        syscall_set_value(1);
        printf("Too many args for cd command\n");
        return 1;
    }
    int fd;
    if ((fd = open(path, O_RDONLY)) < 0) {
        syscall_set_value(1);
        printf("cd: The directory '%s' does not exist\n", path);
        return 1;
    }
    struct Filefd *ffd = (struct Filefd *)num2fd(fd);
    if (ffd -> f_file.f_type != FTYPE_DIR) {
        syscall_set_value(1);
        printf("cd: '%s' is not a directory\n", path);
        return 1;
    }
    if(path[0] == '/') {
        syscall_chdir(path);
    } else {
        char buf[MAXPATHLEN], buf2[MAXPATHLEN];
        syscall_getcwd(buf);
        int len = strlen(buf);
```

```
        if(buf[len - 1] != '/') strcat(buf, "/");
        strcat(buf, path);
        solve_relative_path(buf, buf2);
        syscall_chdir(buf2);
    }
    syscall_set_value(0);
    return 0;
}
```

`pwd` 指令

问题描述

输入	行为	输出	返回值
`pwd`	输出当前工作目录	`/dir1/dir2\n`	0
`pwd arg1 ... argn`	参数数量错误	`pwd: expected 0 arguments; got n\n`	2

解决方案

通过系统调用直接输出

```
void sys_getcwd(char *buf) {
    strcpy(buf, curenv->full_path);
    return;
}
```

`touch` 指令

问题描述

- `touch <file>`：创建空文件 `file`，若文件存在则放弃创建，正常退出无输出。 若创建文件的父目录不存在则输出 `touch: cannot touch '<file>': No such file or directory`。 例如 `touch nonexistent/dir/a.txt` 时应输出 `touch: cannot touch 'nonexistent/dir/a.txt': No such file or directory`。

解决方案

由于不是内建指令，在 `new.mk` 中加入相应指令，并建立可执行文件进行处理

```
#include <lib.h>

int main(int argc, char **argv) {
    char *filename = argv[1];
    int fd;
    if ((fd = open(filename, O_RDONLY)) >= 0) {
        close(fd);
        syscall_set_value(0);
        return 0;
    }
    fd = open(filename, O_CREAT);
    if (fd == -10) {
        syscall_set_value(1);
        printf("touch: cannot touch '%s': No such file or
directory\n", filename);
        return 1;
    } else if (fd >= 0) {
        close(fd);
    }
    syscall_set_value(0);
    return 0;
}
```

`mkdir` 指令

问题描述

- `mkdir <dir>`：若目录已存在则输出 `mkdir: cannot create directory '<dir>': File exists`，若创建目录的父目录不存在则输出 `mkdir: cannot create directory '<dir>': No such file or directory`，否则正常创建目录。
- `mkdir -p <dir>`：当使用 `-p` 选项时忽略错误，若目录已存在则直接退出，若创建目录的父目录不存在则递归创建目录。

解决方案

```
#include <lib.h>

int main(int argc, char **argv) {
    int fd;
```

```
    if (argc == 2) {
        char *dir;
        dir = argv[1];
        if ((fd = open(dir, O_RDONLY)) >= 0) {
            close(fd);
            syscall_set_value(1);
            printf("mkdir: cannot create directory '%s': File
exists\n", dir);
            return 1;
        }
        fd = open(dir, O_MKDIR);
        if (fd < 0) {
            syscall_set_value(1);
            printf("mkdir: cannot create directory '%s': No such
file or directory\n", dir);
            return 1;
        }
        close(fd);
        syscall_set_value(0);
        debugf("created %s\n", dir);
        return 0;
    } else {
        char *dir = argv[2];
        if ((fd = open(dir, O_RDONLY)) >= 0) {
            close(fd);
            syscall_set_value(0);
            return 0;
        }
        char buf[MAXPATHLEN] = "";
        int i;
        int len = strlen(dir);
        for (i = 0; i < len; i++) {
            if (dir[i] == '/') {
                buf[i] = 0;
                if ((fd = open(buf, O_RDONLY)) >= 0) {
                    close(fd);
                } else {
                    break;
                }
            }
            buf[i] = dir[i];
        }
        for (; i < len; i++) {
            if (dir[i] == '/') {
                buf[i] = 0;
```

```
                fd = open(buf, O_MKDIR);
                if (fd >= 0) {
                    close(fd);
                }
            }
            buf[i] = dir[i];
        }
        buf[i] = 0;
        fd = open(buf, O_MKDIR);
        if (fd >= 0) {
            close(fd);
        }
        syscall_set_value(0);
        return 0;
    }
}
```

`rm` 指令

问题描述

- `rm <file>`：若文件存在则删除 `<file>`，否则输出 `rm: cannot remove '<file>': No such file or directory`。
- `rm <dir>`：命令行输出: `rm: cannot remove '<dir>': Is a directory`。
- `rm -r <dir>|<file>`：若文件或文件夹存在则删除，否则输出 `rm: cannot remove '<dir>|<file>': No such file or directory`。
- `rm -rf <dir>|<file>`：如果对应文件或文件夹存在则删除，否则直接退出。

解决方案

```
#include <lib.h>

int main(int argc, char **argv) {
    char *path;
    int flag_r = 0, flag_f = 0;
    if (argc == 2) {
        path = argv[1];
    } else {
        path = argv[2];
        if (strcmp("-r", argv[1]) == 0) {
            flag_r = 1;
        } else {
            flag_r = 1, flag_f = 1;
```

```
        }
    }
    int fd;
    if ((fd = open(path, O_RDONLY)) < 0) {
        if (!flag_f) {
            syscall_set_value(1);
            printf("rm: cannot remove '%s': No such file or
directory\n", path);
            return 1;
        }
        syscall_set_value(0);
        return 0;
    }
    close(fd);
    struct Stat st;
    stat(path, &st);
    if (st.st_isdir && !flag_r) {
        syscall_set_value(1);
        printf("rm: cannot remove '%s': Is a directory\n", path);
        return 1;
    }
    remove(path);
    syscall_set_value(0);
    return 0;
}
```

`exit` 指令

问题描述

执行后退出当前shell

解决方案

通过系统调用关闭所有的进程

```
else if(strcmp(argv[0], "exit") == 0) {
                close_all();
                syscall_env_destroy(0);
```

自由输入

问题描述

现有的 shell 不支持在输入命令时移动光标。你需要实现：键入命令时，可以使用 Left 和 Right 移动光标位置，并可以在当前光标位置进行字符的增加与删除。要求每次在不同位置键入后，可以完整回显修改后的命令，并且键入回车后可以正常运行修改后的命令。

解决方案

```
        左箭头：ESC [ D（对应 ASCII 码：27 91 68）
        右箭头：ESC [ C（对应 ASCII 码：27 91 67）
            初始状态：esc_mode = 0
            ESC 键触发：当读取到 ESC（ASCII 27）时，进入第一阶段转义模式（esc_mode = 1）
            检测 [ 字符：在 esc_mode = 1 状态下，若下一个字符是 [（ASCII 91），则进入第二阶段转义模式（esc_mode = 2）
            识别方向键：在 esc_mode = 2 状态下，根据后续字符判断具体方向：
            D（ASCII 68）表示左箭头
            C（ASCII 67）表示右箭头
```

通过字符判断设计状态机，通过系统调用 `write` 向屏幕进行写入，注意 `\b` 的使用

```
        if(mode == 2) {
            if(c == 'D') {
                if(pos > 0) pos -= 1;
                else printf("\033[C");
            } else if(c == 'C') {
                if(pos , len) pos += 1;
                else printf("\033[D");
            }
            mode = 0;
            continue;
        } else if(mode == 1) {
            mode = ((c == '[') ? 2 : 0);
            continue;
        }
```

```
        if(c == 27) mode = 1;
        else if(c == '\r' || c == '\n') {
            buf[len] = '\0';
            return;
        } else if(c == 0x7f || c == '\b') {
            int i;
```

```
            if(pos > 0) {
                // 移动
                for(i = pos - 1; i < len - 1; i++)
                    buf[i] = buf[i + 1];
                len -= 1;
                pos -= 1;
                buf[len] = '\0';

                printf("\b");
                if(pos < len) {
                    write(1, buf + pos, len - pos);
                    write(1, " ", 1);
                    for(i = len; i >= pos; i--)
                        write(1, "\b", 1);
                } else printf(" \b");
            }
        } else if(c <= 126 && c >= 32) {
            int i;
            if(pos < len)
                for(i = len; i > pos; i -= 1)
                    buf[i] = buf[i - 1];

            buf[pos] = c;
            len += 1, pos += 1;
            buf[len] = '\0';

            if(pos != len) {
                write(1, buf + pos, len - pos);
                for(i = pos; i < len; i += 1)
                    write(1, "\033[D", 3);
            }
        }
```

环境变量管理

问题描述

MOS 中的Shell目前并不支持环境变量,你需要在shell中增加对环境变量的支持。

规定环境变量在命令中以`$`开头，名称与C语言变量命名要求，且长度不超过16，环境变量的值长度同样不超过16。环境变量可分为局部变量与非局部变量，仅非局部变量可传入子Shell中，并且只有非只读变量可被修改。

你需要实现以下内容:

1. 支持 内建指令`declare [-xr] [NAME [=VALUE]]`，其中:

- `-x` 表示变量 `NAME` 为环境变量，否则为局部变量。
  - 环境变量对子 shell 可见，也就是说在 shell 中输入 `sh.b` 启动一个子 shell 后，可以读取并修改 `NAME` 的值，即支持环境变量的继承。
  - 局部变量对子 shell **不可见**，也就是说在 shell 中输入 `sh.b` 启动一个子 shell 后，没有该局部变量。
- `-r` 表示将变量 `NAME` 设为只读。只读变量不能被 `declare` 重新赋值或被 `unset` 删除。
- 如果变量 `NAME` 不存在，需要创建该环境变量；如果变量 `NAME` 存在，将该变量赋值为 `VALUE`。
- 其中 `VALUE` 为可选参数，缺省时将该变量赋值为空字符串即可。
- 如果没有 `[-xr]` 及 `[NAME [=VALUE]]` 部分，即只输入 `declare`，则输出当前 shell 的所有变量，包括局部变量和环境变量。

2. 支持内建指令 `unset NAME` 命令，若变量 `NAME` 不是只读变量，则删除变量 `NAME`。
3. 支持在命令中展开变量的值，如使用 `echo.b $NAME` 打印变量 `NAME` 的值。

解决方案

新增三个系统调用

```
    SYS_declare,
    SYS_unset,
    SYS_list,
```

增加进程存储的环境变量

```
// 环境变量
struct Variable {
    int type, only; // 局部/全局, 只读/可编辑
    char name[17], val[17];
};

struct Env {
    ...
    struct Variable var[32];
    int var_cnt;
}
```

完成系统调用

```
int sys_declare(char *key, char *val, int type, int readonly) {
```

```
    struct Env *par;
    envid2env(curenv -> env_parent_id, &par, 0);
    int pos = -1, i;
    for (i = 0; i < par -> var_cnt; i++) {
        if (strcmp(key, par -> var[i].name) == 0) {
            pos = i;
            break;
        }
    }
    if (pos != -1) {
        if (par -> var[pos].only) {
            return -E_INVAL;
        }
        strcpy(par -> var[pos].val, val);
        strcpy(par -> var[pos].name, key);
        par -> var[pos].type = type;
        par -> var[pos].only = readonly;
        return 0;
    } else {
        pos = par -> var_cnt++;
        strcpy(par -> var[pos].val, val);
        strcpy(par -> var[pos].name, key);
        par -> var[pos].type = type;
        par -> var[pos].only = readonly;
        return 0;
    }
}

int sys_list(int is_parent, struct Variable *buf) {
    struct Env *par;
    if(!is_parent) {
        par = curenv;
    } else envid2env(curenv -> env_parent_id, &par, 0);
    memcpy(buf, par -> var, (sizeof (struct Variable)) * (par ->
var_cnt));
    return par -> var_cnt;
}

int sys_unset(char *key) {
    struct Env *par;
    envid2env(curenv -> env_parent_id, &par, 0);
    int pos = -1, i;
    for (i = 0; i < par -> var_cnt; i++) {
        if (strcmp(key, par -> var[i].name) == 0) {
            pos = i;
```

```
            break;
        }
    }
    if(pos == -1) return 0;
    if(par -> var[pos].only)
        return -E_INVAL;
    for (i = pos; i < par -> var_cnt - 1; i++) {
        par -> var[i] = par -> var[i + 1];
    }
    par -> var_cnt--;
    return 0;
}
```

在指令中进行处理

```
int command_declare(int argc, char *argv[]) {
    int i, r;
    int only = (argc == 3 && (strchr(argv[1], 'r') != NULL)), type
= (argc == 3 && (strchr(argv[1], 'x') != NULL));

    if(argc == 1) {
        struct Variable buf[32];
        int cnt = syscall_list(1, buf);
        for(i = 0; i < cnt; i += 1)
            printf("%s=%s\n", buf[i].name, buf[i].val);
        return 0;
    }

    char *name = argv[argc - 1], *val = name;
    while((*val) != '\0') {
        if((*val) == '=') {
            *val = 0;
            val += 1;
            break;
        }
        val += 1;
    }

    if((r = syscall_declare(name, val, type, only)) != 0) {
        debugf("declare: '%s' is readonly\n", name);
    }
    return r;
}
```

历史指令

问题描述

你需要实现 shell 中保存历史指令的功能，可以通过 Up 和 Down 选择所保存的指令并执行。你需要将历史指令保存到根目录的 `.mos_history` 文件中（一条指令一行），为了评测的方便，我们设定 `$HISTFILESIZE=20`（bash 中默认为 500），即在 `.mos_history` 中至多保存最近的 20 条指令。你还需要支持通过 `history` 命令输出 `.mos_history` 文件中的内容。

> 注：在 bash 中，`history` 为 shell built-in command，我们规定需要将 `history` 实现为 built-in command。

你需要将当前执行的指令先存入 `.mos_history` 中，例如：

```
echo `ls | cat`
echo meow # comment
history
history | cat
```

当历史指令为空时，依次执行上述四条指令后，后两条指令会分别输出

```
echo `ls | cat`
echo meow # comment
history
```

与

```
echo `ls | cat`
echo meow # comment
history
history | cat
```

使用 Up 能够切换到上一条指令（如果上一条指令存在），使用 Down 能够切换到下一条指令（如果下一条指令存在）。能够选择的指令范围为：用户当前输入的指令与 `.mos_history` 文件中保存的所有指令。例如在执行了上述四条指令后，用户输入了 `echo`，此时 Up 应该将指令切换至 `history | cat`，再次进行三次 Up 后切换至 `echo`ls | cat`，此时再次 Up 应保留在该指令（因为已经不存在上一条指令）；再进行四次 Down 后将切换回`echo`，此时再次 Down 应保留在该指令（因为不存在下一条指令）。

解决方案

添加 `static` 变量管理文件中的指令历史

```
static int his_cnt = 0, his_id = -1;
static char his[HISMAX][1024];
```

加载与管理历史

```
// 记录历史
void his_save() {
    int fd, i;
    if((fd = open("/.mos_history", O_WRONLY | O_CREAT | O_TRUNC))
< 0) {
        debugf("failed to save history\n");
        return;
    }

    for(i = 0; i < his_cnt; i += 1) {
        int len = strlen(his[i]);
        write(fd, his[i], len);
        write(fd, "\n", 1);
    }

    close(fd);
}

// 加载历史
void his_load() {
    int fd, r, i;
    char buf[1024];
    if((fd = open("/.mos_history", O_RDONLY)) < 0) {
        fd = open("/.mos_history", O_RDONLY | O_CREAT);
    }

    his_cnt = 0;
    while(r = read(fd, buf, sizeof(buf))) {
        if(r < 0) {
            debugf("read history error\n");
            close(fd);
            return;
        }

        // 从头开始加载
        char *line = buf;
```

```
        while((*line) != '\0') {
            char *end = strchr(line, '\n');
            if(end == NULL) break;
            *end = '\0';

            if(his_cnt < HISMAX) {
                strcpy(his[his_cnt++], line);
            } else {
                for(i = 0; i < HISMAX; i += 1)
                    strcpy(his[i], his[i + 1]);
                strcpy(his[HISMAX - 1], line);
            }
            line = end + 1;
        }
    }
    close(fd);
}
```

对于输入的上箭头与下箭头进行处理

```
            } else if(c == 'A') {
                printf("\033[B");
                if(his_cnt > 0) {
                    if(his_id == -1 && len > 0) {
                        strcpy(cur_line, buf);
                        save_cur = 1;
                    }

                    his_id += (his_cnt - 1 > his_id);
                    int i;
                    for(i = 0; i < len; i += 1)
                        printf("\b \b");

                    strcpy(buf, his[his_id]);
                    len = strlen(buf);
                    pos = len;

                    printf("%s", buf);
                }
            } else if(c == 'B') {
                if(his_id >= 0) {
                    his_id -= 1;

                    int i;
                    for(i = 0; i < len; i += 1)
```

```
                    printf("\b \b");

                if(save_cur && his_id == -1) {
                    strcpy(buf, cur_line);
                    pos = strlen(buf);
                    printf("%s", buf);
                } else if(his_id >= 0) {
                    strcpy(buf, his[his_id]);
                    pos = strlen(buf);
                    printf("%s", buf);
                } else buf[0] = '\0', len = pos = 0;
            }
        }
```

指令条件执行

问题描述

你需要实现 Linux shell 中的 `&&` 与 `||`。 对于 `command1 && command2`，`command2` 被执行当且仅当 `command1` 返回 0；对于 `command1 || command2`，`command2` 被执行当且仅当 `command1` 返回非 0 值

解决方案

新增系统调用，对于指令的执行结果进行存储和管理

```
    SYS_set_value,
    SYS_push_value,
    SYS_get_lst_code,
```

通过指令解析拆分，并分别处理每一个部分，存储每一部分的返回值，之后单独进行处理

```
int logiccmd(char **cmds, char *s, char *opt) {
    int flag = 1, cnt = 1;
    while(strchr(WHITESPACE SYMBOLS, *s)) s += 1;
    if(!(*s)) return 0;
    cmds[0] = s;
    while(*s) {
        char c = (*s);
        if(c == '|' && *(s + 1) == '|' || c == '&' && *(s + 1) ==
'&') {
            *s = 0;
            s += 1;
            *s = 0;
```

```
            s += 1;
            while(strchr(WHITESPACE SYMBOLS, *s)) s += 1;
            if(*s != 0) {
                opt[cnt - 1] = ((c == '|') ? '|' : '&');
                cmds[cnt ++] = s;
            } else return cnt;
        } else s += 1;
    }
    return cnt;
}
```

追加重定向

问题描述

你需要实现 shell 中 `>>` 追加重定向的功能，例如：

```
ls >> file1
ls >> file1
```

最后文件 `file1` 中将会有两次 `ls` 指令的输出。

解决方案

```
        case 'a':
            if (gettoken(0, &t) != 'w') {
                debugf("syntax error: > not followed by word\n");
                exit();
            }
            fd = open(t, O_WRONLY | O_CREAT | O_APPEND);
            if(fd < 0) {
                debugf("failed to open '%s'\n", t);
                exit();
            }
            dup(fd, 1);
            close(fd);
            break;
```

反引号

问题描述

你需要使用反引号实现指令替换。你需要将反引号内指令执行的所有标准输出代替原有指令中的反引号内容。例如：

```
echo `ls | cat | cat | cat`
```

解决方案

解析指令并进行提前处理

```
            for(k = 0; k < argc; k += 1) {
                if(argv[k][0] != '`') continue;
                argv[k][0] = ' ';
                int p[2];
                if(pipe(p) < 0) {
                    debugf("failed to create pipe\n");
                    exit();
                }
                int id = fork();
                if(id < 0) {
                    debugf("failed to fork in sh.c\n");
                    exit();
                }
                int offset = 0, num = 0;
                if(id != 0) {
                    close(p[1]);
                    memset(rev_buf, 0, sizeof(rev_buf));
                    while((num = read(p[0], rev_buf + offset,
114)) > 0)
                        offset += num;
                    close(p[0]);
                    int len = strlen(rev_buf);
                    while(len > 0 && strchr(WHITESPACE,
rev_buf[len - 1]))
                        rev_buf[len - 1] = 0, len -= 1;
                    argv[k] = rev_buf;
                } else {
                    close(p[0]);
                    dup(p[1], 1);
                    close(p[1]);

                    char tp[1024] = {0};
                    strcpy(tp, argv[k]);
                    runcmd(tp);
                    exit();
                }
```

体会与感想

与操作系统相处的日子也是告一段落、完结撒花了，但与操作系统的羁绊并未结束，学到的算法和系统思想将一直留存~~~