

Report: Lab6-challenge

郭晓燕 20373543

Press Space for next page →

目录

Content of this report.

- 任务列表
- 前置工作
- 特殊字符特判
- 外部命令扩展
- 内部命令扩展
- 测试工作 & 问题及解决方案



任务列表

In lab6-challenge, I have finished the following tasks.

1. 特殊字符特判 [basic]

- `&`: 通过 `<command> &` 实现后台运行
- `;`: 通过 `<command1>; <command2>` 实现一行多命令
- `“”`: 通过 `“<String>”` 实现引号支持，使得双引号内的内容被看作一个简单字符串

2. 外部命令扩展 [basic]

- `tree`: 通过 `tree` 递归地列出当前目录下文件的树形结构
- `mkdir`: 通过 `mkdir [[PATH] DIRNAME]` 创建新的目录文件
- `touch`: 通过 `touch [[PATH] FILENAME]` 创建新的常规文件

任务列表

In lab6-challenge, I have finished the following tasks.

3. 内部命令扩展 [Normal & Extra]

history: 历史命令功能

- 实现将 shell 中输入过的指令保存到 `~/.history` 文件
- 通过 `history` 命令输出所有的历史指令
- 通过上下键回溯历史指令

unset NAME: 解除变量定义

- 解除对 NAME 变量的定义：如果 NAME 不是只读变量，删除 NAME，否则给出相应的报错

任务列表

In lab6-challenge, I have finished the following tasks.

3. 内部命令扩展 [Normal & Extra]

`declare [-xr] [NAME [=VALUE]]`

- `-x`: 表示将变量 NAME 导出为环境变量，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` 启动一个子 shell 后，可以读取 NAME 的值
 - 局部变量对子 shell 不可见，也就是说在 shell 中输入 `sh` 启动一个子 shell 后，没有该局部变量，会给出相应报错
- `-r`: 表示将变量 NAME 设为只读。只读变量不能被 `declare` 重新定义或被 `unset` 删除
- 如果没有 `[-xr]` 及 `[NAME [=VALUE]]` 部分，则输出当前 shell 的所有变量，包括局部变量 和环境变量
- 支持并在执行诸如 `echo \$variable` 指令时能显示正确的值

前置工作

Pre work of the lab.

- 完善文件系统功能

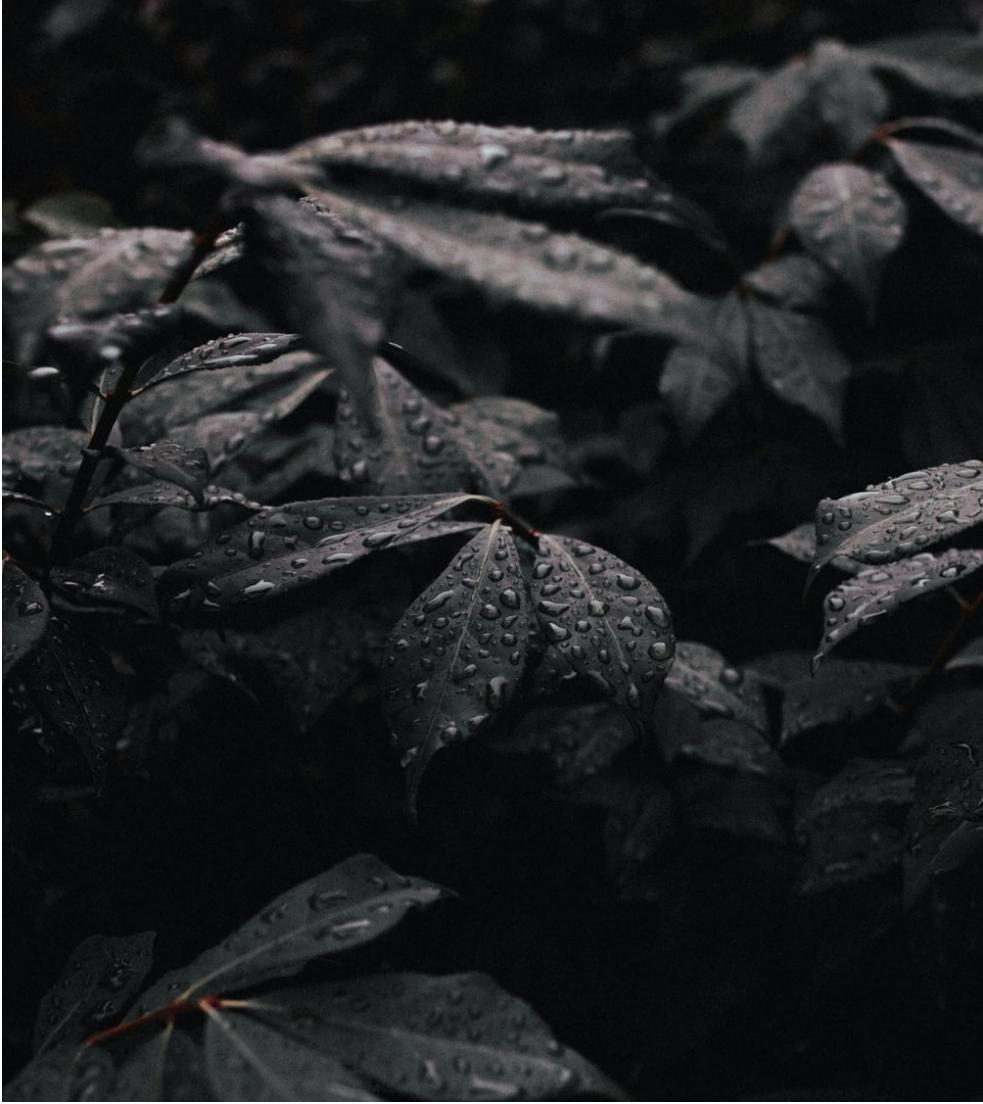
- 实现创建常规文件和目录的功能
- 实现 O_APPEND

- 清除多余输出

- 关于进程和系统服务的调试信息
- `fwritef` 和 `writef` 的多余空行

- 扩充 `string.c` 库以支持更多字符串操作

- `strlen`, `strcat`, `strhash` 等
- 用户态和内核态都需要扩充



前置工作

完善文件系统功能

- 实现创建常规文件和目录的功能 - for `mkdir` && `touch` && `history`
 - 添加新的文件系统服务 `file_create`，为用户态支持 `create` 操作
- 创建文件需要：路径 (path + name) + 类型 (FTYPE_REG or FTYPE_DIR)
- 实现 O_APPEND 将内容写到文件末尾 -for `history`

前置工作

完善文件系统功能

- 实现创建常规文件和目录的功能

```
1  /* include/fs.h */  
2  #define FSREQ_OPEN  1  
3  #define FSREQ_MAP   2  
4  #define FSREQ_SET_SIZE 3  
5  #define FSREQ_CLOSE  4  
6  #define FSREQ_DIRTY  5  
7  #define FSREQ_REMOVE 6  
8  #define FSREQ_SYNC   7  
9  #define FSREQ_CREATE 8  
10  
11  struct Fsreq_create {  
12      u_char req_path[MAXPATHLEN];  
13      u_int req_type;  
14  };
```

```
1  /* user/fsipc.c */  
2  int fsipc_create(const char *path, u_int type) {  
3      struct Fsreq_create *req;  
4      req = (struct Fsreq_create*) fsipcbuf;  
5  
6      if (strlen(path) >= MAXPATHLEN) {  
7          return -E_BAD_PATH;  
8      }  
9  
10     strcpy((char*)req->req_path, path);  
11     req->req_type = type;  
12     return fsipc(FSREQ_CREATE, req, 0, 0);  
13  }  
14  
15  /* user/lib.h */  
16  int fsipc_create(const char*, u_int);
```

前置工作

完善文件系统功能

- 实现创建常规文件和目录的功能

```
1  /* include/fs.h */  
2  #define FSREQ_OPEN  1  
3  #define FSREQ_MAP   2  
4  #define FSREQ_SET_SIZE 3  
5  #define FSREQ_CLOSE  4  
6  #define FSREQ_DIRTY 5  
7  #define FSREQ_REMOVE 6  
8  #define FSREQ_SYNC   7  
9  #define FSREQ_CREATE 8  
10  
11 struct Fsreq_create {  
12     u_char req_path[MAXPATHLEN];  
13     u_int req_type;  
14 };
```

```
1  /* user/fsipc.c */  
2  int fsipc_create(const char *path, u_int type) {  
3      struct Fsreq_create *req;  
4      req = (struct Fsreq_create*) fsipcbuf;  
5  
6      if (strlen(path) >= MAXPATHLEN) {  
7          return -E_BAD_PATH;  
8      }  
9  
10     strcpy((char*)req->req_path, path);  
11     req->req_type = type;  
12     return fsipc(FSREQ_CREATE, req, 0, 0);  
13 }  
14  
15 /* user/lib.h */  
16 int fsipc_create(const char*, u_int);
```

前置工作

完善文件系统功能

- 实现创建常规文件和目录的功能

```
1  /* include/fs.h */  
2  #define FSREQ_OPEN  1  
3  #define FSREQ_MAP   2  
4  #define FSREQ_SET_SIZE 3  
5  #define FSREQ_CLOSE  4  
6  #define FSREQ_DIRTY 5  
7  #define FSREQ_REMOVE 6  
8  #define FSREQ_SYNC   7  
9  #define FSREQ_CREATE 8  
10  
11 struct Fsreq_create {  
12     u_char req_path[MAXPATHLEN];  
13     u_int req_type;  
14 };
```

```
1  /* user/fsipc.c */  
2  int fsipc_create(const char *path, u_int type) {  
3      struct Fsreq_create *req;  
4      req = (struct Fsreq_create*) fsipcbuf;  
5  
6      if (strlen(path) >= MAXPATHLEN) {  
7          return -E_BAD_PATH;  
8      }  
9  
10     strcpy((char*)req->req_path, path);  
11     req->req_type = type;  
12     return fsipc(FSREQ_CREATE, req, 0, 0);  
13 }  
14  
15 /* user/lib.h */  
16 int fsipc_create(const char*, u_int);
```

前置工作

完善文件系统功能

- 实现创建常规文件和目录的功能

```
1  /* include/fs.h */  
2  #define FSREQ_OPEN 1  
3  #define FSREQ_MAP 2  
4  #define FSREQ_SET_SIZE 3  
5  #define FSREQ_CLOSE 4  
6  #define FSREQ_DIRTY 5  
7  #define FSREQ_REMOVE 6  
8  #define FSREQ_SYNC 7  
9  #define FSREQ_CREATE 8  
10  
11 struct Fsreq_create {  
12     u_char req_path[MAXPATHLEN];  
13     u_int req_type;  
14 };
```

```
1  /* user/fsipc.c */  
2  int fsipc_create(const char *path, u_int type) {  
3      struct Fsreq_create *req;  
4      req = (struct Fsreq_create*) fsipcbuf;  
5  
6      if (strlen(path) >= MAXPATHLEN) {  
7          return -E_BAD_PATH;  
8      }  
9  
10     strcpy((char*)req->req_path, path);  
11     req->req_type = type;  
12     return fsipc(FSREQ_CREATE, req, 0, 0);  
13 }  
14  
15 /* user/lib.h */  
16 int fsipc_create(const char*, u_int);
```

前置工作

完善文件系统功能

- 实现创建常规文件和目录的功能

```
1  /* include/fs.h */  
2  #define FSREQ_OPEN  1  
3  #define FSREQ_MAP   2  
4  #define FSREQ_SET_SIZE 3  
5  #define FSREQ_CLOSE  4  
6  #define FSREQ_DIRTY 5  
7  #define FSREQ_REMOVE 6  
8  #define FSREQ_SYNC   7  
9  #define FSREQ_CREATE 8  
10  
11 struct Fsreq_create {  
12     u_char req_path[MAXPATHLEN];  
13     u_int req_type;  
14 };
```

```
1  /* user/fsipc.c */  
2  int fsipc_create(const char *path, u_int type) {  
3      struct Fsreq_create *req;  
4      req = (struct Fsreq_create*) fsipcbuf;  
5  
6      if (strlen(path) >= MAXPATHLEN) {  
7          return -E_BAD_PATH;  
8      }  
9  
10     strcpy((char*)req->req_path, path);  
11     req->req_type = type;  
12     return fsipc(FSREQ_CREATE, req, 0, 0);  
13 }  
14  
15 /* user/lib.h */  
16 int fsipc_create(const char*, u_int);
```

```
1  /* fs/fs.c */
2  int file_create(char *path, struct File **file) {
3      int r;
4      /* check if it is a directory */
5      char *p1 = (path[0] == '/') ? (path + 1) : 0;
6      while (p1 && *p1) {
7          while (*p1 && *p1 != '/') {
8              p1++;
9          }
10         if (!*p1 || !*(p1 + 1)) {
11             break;
12         }
13         *p1 = '\0';
14
15         struct File *dir;
16         r = self_file_create(path, &dir);
17         if (r < 0 && r != -E_FILE_EXISTS) {
18             return r;
19         }
20
21         *p1++ = '/';
22         dir->f_type = FTYPE_DIR;
23     }
24     return self_file_create(path, file);
25 }
```

```
1  /* fs/serve.c */
2  void
3  serve_create(u_int envid, struct Fsreq_create *rq) {
4      u_char path[MAXPATHLEN];
5      struct File *file;
6      int r;
7
8      user_bcopy(rq->req_path, path, MAXPATHLEN);
9      path[MAXPATHLEN - 1] = 0;
10
11     r = file_create(path, &file);
12     file->f_size = 0;
13     file->f_type = rq->req_type;
14     ipc_send(envid, r, 0, 0);
15 }
16
17 void
18 serve(void)
19 {
20     /* .... */
21     case FSREQ_CREATE:
22         serve_create(which, (struct Fsreq_create *)
23         break;
24 }
```

```
1  /* fs/fs.c */
2  int file_create(char *path, struct File **file) {
3      int r;
4      /* check if it is a directory */
5      char *p1 = (path[0] == '/') ? (path + 1) : 0;
6      while (p1 && *p1) {
7          while (*p1 && *p1 != '/') {
8              p1++;
9          }
10         if (!*p1 || !*(p1 + 1)) {
11             break;
12         }
13         *p1 = '\0';
14
15         struct File *dir;
16         r = self_file_create(path, &dir);
17         if (r < 0 && r != -E_FILE_EXISTS) {
18             return r;
19         }
20
21         *p1++ = '/';
22         dir->f_type = FTYPE_DIR;
23     }
24     return self_file_create(path, file);
25 }
```

```
1  /* fs/serve.c */
2  void
3  serve_create(u_int envid, struct Fsreq_create *rq) {
4      u_char path[MAXPATHLEN];
5      struct File *file;
6      int r;
7
8      user_bcopy(rq->req_path, path, MAXPATHLEN);
9      path[MAXPATHLEN - 1] = 0;
10
11     r = file_create(path, &file);
12     file->f_size = 0;
13     file->f_type = rq->req_type;
14     ipc_send(envid, r, 0, 0);
15 }
16
17 void
18 serve(void)
19 {
20     /* ..... */
21     case FSREQ_CREATE:
22         serve_create(whom, (struct Fsreq_create *)
23         break;
24 }
```

```
1  /* fs/fs.c */
2  int file_create(char *path, struct File **file) {
3      int r;
4      /* check if it is a directory */
5      char *p1 = (path[0] == '/') ? (path + 1) : 0;
6      while (p1 && *p1) {
7          while (*p1 && *p1 != '/') {
8              p1++;
9          }
10         if (!*p1 || !*(p1 + 1)) {
11             break;
12         }
13         *p1 = '\0';
14
15         struct File *dir;
16         r = self_file_create(path, &dir);
17         if (r < 0 && r != -E_FILE_EXISTS) {
18             return r;
19         }
20
21         *p1++ = '/';
22         dir->f_type = FTYPE_DIR;
23     }
24     return self_file_create(path, file);
25 }
```

```
1  /* fs/serv.c */
2  void
3  serve_create(u_int envid, struct Fsreq_create *rq) {
4      u_char path[MAXPATHLEN];
5      struct File *file;
6      int r;
7
8      user_bcopy(rq->req_path, path, MAXPATHLEN);
9      path[MAXPATHLEN - 1] = 0;
10
11     r = file_create(path, &file);
12     file->f_size = 0;
13     file->f_type = rq->req_type;
14     ipc_send(envid, r, 0, 0);
15 }
16
17 void
18 serve(void)
19 {
20     /* ..... */
21     case FSREQ_CREATE:
22         serve_create(whom, (struct Fsreq_create *)
23         break;
24 }
```

```
1  /* fs/fs.c */
2  int file_create(char *path, struct File **file) {
3      int r;
4      /* check if it is a directory */
5      char *p1 = (path[0] == '/') ? (path + 1) : 0;
6      while (p1 && *p1) {
7          while (*p1 && *p1 != '/') {
8              p1++;
9          }
10         if (!*p1 || !*(p1 + 1)) {
11             break;
12         }
13         *p1 = '\0';
14
15         struct File *dir;
16         r = self_file_create(path, &dir);
17         if (r < 0 && r != -E_FILE_EXISTS) {
18             return r;
19         }
20
21         *p1++ = '/';
22         dir->f_type = FTYPE_DIR;
23     }
24     return self_file_create(path, file);
25 }
```

```
1  /* fs/serve.c */
2  void
3  serve_create(u_int envid, struct Fsreq_create *rq) {
4      u_char path[MAXPATHLEN];
5      struct File *file;
6      int r;
7
8      user_bcopy(rq->req_path, path, MAXPATHLEN);
9      path[MAXPATHLEN - 1] = 0;
10
11     r = file_create(path, &file);
12     file->f_size = 0;
13     file->f_type = rq->req_type;
14     ipc_send(envid, r, 0, 0);
15 }
16
17 void
18 serve(void)
19 {
20     /* .... */
21     case FSREQ_CREATE:
22         serve_create(whom, (struct Fsreq_create *)
23         break;
24 }
```

至此，我们可以实现常规文件和目录文件的创建功能，只需要限制文件的 type 即可。

```
1  /* user/file.c */
2  int
3  create(const char *path, int type)
4  {
5      int r;
6      if ((r = fsipc_create(path, type)) < 0) {
7          return r;
8      }
9      return 0;
10 }
```

前置工作

完善文件系统功能

- 实现 O_APPEND

```
1  /* user/lib.h */
2  #define O_APPEND      0x1000
3
4  /* user/file.c */
5  int
6  open(const char *path, int mode)
7  {
8      /* ..... */
9      va = fd2data(fd);
10     ffd = (struct Filefd *)fd;
11     size = ffd->f_file.f_size;
12     /* ..... */
13     if (mode & O_APPEND) {
14         seek(fdnum, size);
15     }
16     return fdnum;
17 }
```

前置工作

完善文件系统功能

- 实现 O_APPEND

```
1  /* user/lib.h */  
2  #define O_APPEND      0x1000  
3  
4  /* user/file.c */  
5  int  
6  open(const char *path, int mode)  
7  {  
8      /* ..... */  
9      va = fd2data(fd);  
10     ffd = (struct Filefd *)fd;  
11     size = ffd->f_file.f_size;  
12     /* ..... */  
13     if (mode & O_APPEND) {  
14         seek(fdnum, size);  
15     }  
16     return fdnum;  
17 }
```

前置工作

完善文件系统功能

- 实现 O_APPEND

```
1  /* user/lib.h */
2  #define O_APPEND      0x1000
3
4  /* user/file.c */
5  int
6  open(const char *path, int mode)
7  {
8      /* ..... */
9      va = fd2data(fd);
10     ffd = (struct Filefd *)fd;
11     size = ffd->f_file.f_size;
12     /* ..... */
13     if (mode & O_APPEND) {
14         seek(fdnum, size);
15     }
16     return fdnum;
17 }
```

前置工作

完善文件系统功能

- 实现 O_APPEND

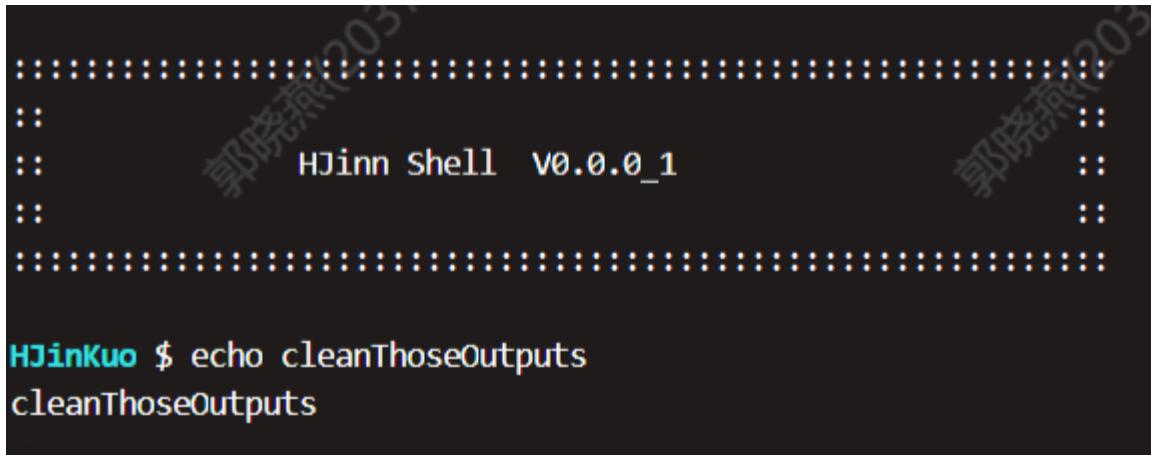
```
1  /* user/lib.h */
2  #define O_APPEND      0x1000
3
4  /* user/file.c */
5  int
6  open(const char *path, int mode)
7  {
8      /* ..... */
9      va = fd2data(fd);
10     ffd = (struct Filefd *)fd;
11     size = ffd->f_file.f_size;
12     /* ..... */
13     if (mode & O_APPEND) {
14         seek(fdnum, size);
15     }
16     return fdnum;
17 }
```

清除多余输出

Make the shell prettier.

- 关于进程和系统服务的调试信息

- 删除 spawn, serve_open, env_* 中的输出调试信息



The screenshot shows a terminal window with a dark background. At the top, there is a decorative header consisting of a grid of dots and two vertical colons on either side. In the center of the grid, the text "HJinn Shell v0.0.0_1" is displayed. Below this, there is more decorative text and a series of dots. At the bottom of the window, the command "HJinKuo \$ echo cleanThoseOutputs" is typed, followed by the output "cleanThoseOutputs".

- 清除 `fwritef` 和 `writef` 输出的多余空行

扩充 string.c 库

支持更丰富的字符串操作.

- 添加了 `strcmp`、`strcat`、`strlen`，
`strhash` 等函数
- 由于后续为实现环境变量添加的系统调用也有
相应的需求，因此需要在**内核**和**用户态**都进行
扩充
 - `user/string.c`
 - `lib/env.c`



特殊字符特判

`&` + ` ;` + `` ` `

- 核心问题：解析输入的字符串时的特殊处理
- 需要关注的核心文件：`user/sh.c`

```
HJinKuo $ testhang &
--- Start to test a slow nohup program ---

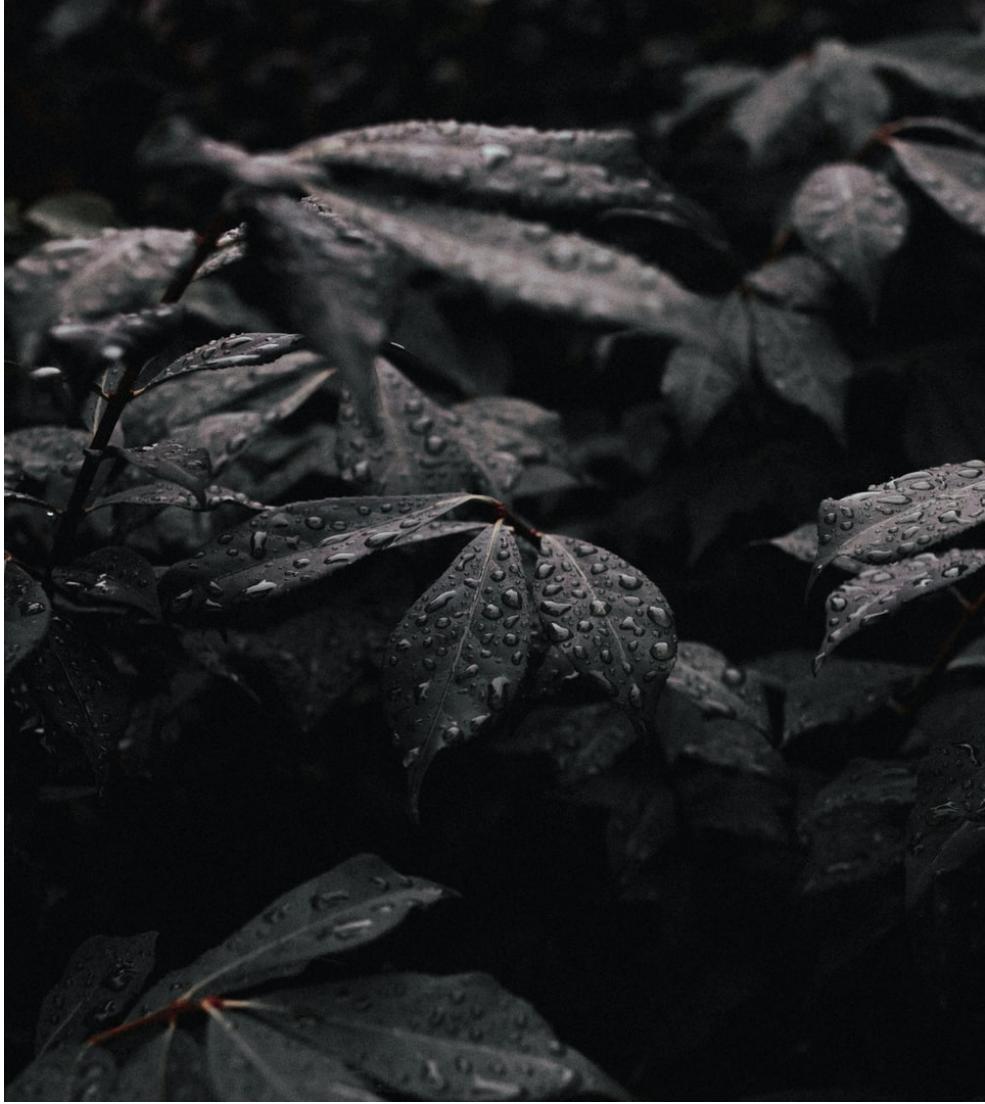
HJinKuo $ 000 Hey it is hanging 000
echo hey
hey

HJinKuo $ 000 Hey it is hanging 000
000 Hey it is hanging 000
000 Hey it is hanging 000
echo hey
hey

HJinKuo $ 000 Hey it is hanging 000
000 Hey it is hanging 000
000 Hey it is hanging 000
--- The nohup progame has done! ---
```

```
HJinKuo $ echo test1; cat newmotto
test1
This is a different massage of the day!
```

```
HJinKuo $ echo "test1 | test2"
test1 | test2
```



特殊字符特判

`&` + ` ; ` + ``"``

■ &后台运行

- 为了处理后台运行，设置一个标记变量 flag_hang，若解析到 & 字符则标记其为 1
- 修改核心：**原有的内核忙等机制。现在父进程不再等待子进程执行完，直接 exit 返回

```
1  /* user/sh.c */
2  void runcmd(char *s) {
3      /* ..... */
4      int fdnum, flag_hang = 0, pid;
5      again:
6      argc = 0;
7      for (; ;) {
8          /* ..... */
9          case '&':
10              flag_hang = 1;
11              break;
12      }
13  }
14  runit:
15  /* ..... */
16  close_all();
17  if (r >= 0) {
18      if (debug_) writef("[%08x] WAIT %s %08x\n", en
19      if (!flag_hang) wait(r);
20  }
21  /* ..... */
22 }
```

特殊字符特判

`&` + ` ` ; ` + ``"``

- ; 一行多命令
 - 解析到分号时，在执行完当前指令后再 goto again 执行下一条指令

```
1 case ';' :  
2     if ((pid = fork()) == 0) {  
3         fd_redirect[0] = fd_redirect[1] = -1;  
4         goto again;  
5     } else {  
6         goto runit;  
7     }  
8     break;
```

特殊字符特判

`&` + ` ;` + `|||`

■ "" 引号支持

- 在解析命令之前需要在参数中解析好，因此关注函数 `gettoken`
- 解析到引号时，需要利用一个 while 循环继续向下读，直到读到与之匹配的引号，将引号包括起来的部分当成一个字符串来解析
 - 注意转义字符

```
1  /* user/sh.c */
2  if (*s == '\"') {
3      *p1 = ++s;
4      while (*s && !( *s == '\"' && *(s - 1) != '\\')) {
5          ++s;
6      }
7      *s++ = 0;
8      *p2 = s;
9      return 'w';
10 }
```

外部命令扩展

tree, mkdir, touch

- 此时需要针对三个命令分别编写用户态程序

```
`user/~/c`
```

- tree: 模仿 ls 命令的逻辑遍历当前目录
- mkdir, touch: 利用先前完善的文件系统功能, 调用相应的接口创建文件
- Extra: 为了强化层次感与优化交互体验, 提供彩色输出



外部命令扩展

tree, mkdir, touch

- tree命令

- 核心功能：递归地遍历目录下的文件
 - 每读到一个文件都将其文件名按照一定格式打印出来，并根据文件此时位处的深度修改打印的缩进值
 - 如果当前文件为目录文件，则递归地调用遍历函数，将该目录下的文件进行打印

```
HJinKuo $ tree
. /
|----motd
|----newmotd
|----testarg.b
|----init.b
|----num.b
|----echo.b
|----ls.b
|----sh.b
|----cat.b
|----tree.b
|----mkdir.b
|----touch.b
|----history.b
|----declare.b
|----unset.b
|----testhang.b
|----testptelibrary.b
|----.history
```

外部命令扩展

tree, mkdir, touch

- tree命令

- 核心功能：递归地遍历目录下的文件

- 每读到一个文件都将其文件名按照一定格式打印出来，并根据文件此时位处的深度修改打印的缩进值
 - 如果当前文件为目录文件，则递归地调用遍历函数，将该目录下的文件进行打印

```
1  /* user/tree.c */
2  void walk_dir(char *path, int level) {
3      int fd, n;
4      struct File f;
5      char nextDir[MAXPATHLEN] = {0};
6
7      if ((fd = open(path, O_RDONLY)) < 0)
8          user_panic("Fail to open %s: %e", path, fd);
9      while ((n = readn(fd, &f, sizeof f)) == sizeof
10         if (f.f_name[0]) {
11             tree_print(f.f_name, level);
12             if (f.f_type == FTYPE_DIR) {
13                 strcpy(nextDir, path);
14                 strcat(nextDir, "/");
15                 strcat(nextDir, f.f_name);
16                 walk_dir(nextDir, level + 1);
17             }
18         }
19     }
20 }
```

外部命令扩展

tree, mkdir, touch

- **tree命令**

- **核心功能：**递归地遍历目录下的文件

- 每读到一个文件都将其文件名按照一定格式打印出来，并根据文件此时位处的深度修改打印的缩进值
 - 如果当前文件为目录文件，则递归地调用遍历函数，将该目录下的文件进行打印

```
1  /* user/tree.c */
2  void walk_dir(char *path, int level) {
3      int fd, n;
4      struct File f;
5      char nextDir[MAXPATHLEN] = {0};
6
7      if ((fd = open(path, O_RDONLY)) < 0)
8          user_panic("Fail to open %s: %e", path, fd);
9      while ((n = readn(fd, &f, sizeof f)) == sizeof
10         if (f.f_name[0]) {
11             tree_print(f.f_name, level);
12             if (f.f_type == FTYPE_DIR) {
13                 strcpy(nextDir, path);
14                 strcat(nextDir, "/");
15                 strcat(nextDir, f.f_name);
16                 walk_dir(nextDir, level + 1);
17             }
18         }
19     }
20 }
```

外部命令扩展

tree, mkdir, touch

- **tree命令**

- **核心功能：**递归地遍历目录下的文件

- 每读到一个文件都将其文件名按照一定格式打印出来，并根据文件此时位处的深度修改打印的缩进值
 - 如果当前文件为目录文件，则递归地调用遍历函数，将该目录下的文件进行打印

```
1  /* user/tree.c */
2  void walk_dir(char *path, int level) {
3      int fd, n;
4      struct File f;
5      char nextDir[MAXPATHLEN] = {0};
6
7      if ((fd = open(path, O_RDONLY)) < 0)
8          user_panic("Fail to open %s: %e", path, fd);
9      while ((n = readn(fd, &f, sizeof f)) == sizeof f) {
10         if (f.f_name[0]) {
11             tree_print(f.f_name, level);
12             if (f.f_type == FTYPE_DIR) {
13                 strcpy(nextDir, path);
14                 strcat(nextDir, "/");
15                 strcat(nextDir, f.f_name);
16                 walk_dir(nextDir, level + 1);
17             }
18         }
19     }
20 }
```

外部命令扩展

tree, mkdir, touch

- **tree命令**

- **核心功能：**递归地遍历目录下的文件

- 每读到一个文件都将其文件名按照一定格式打印出来，并根据文件此时位处的深度修改打印的缩进值
 - 如果当前文件为目录文件，则递归地调用遍历函数，将该目录下的文件进行打印

```
1  /* user/tree.c */
2  void walk_dir(char *path, int level) {
3      int fd, n;
4      struct File f;
5      char nextDir[MAXPATHLEN] = {0};
6
7      if ((fd = open(path, O_RDONLY)) < 0)
8          user_panic("Fail to open %s: %e", path, fd);
9      while ((n = readn(fd, &f, sizeof f)) == sizeof f) {
10         if (f.f_name[0]) {
11             tree_print(f.f_name, level);
12             if (f.f_type == FTYPE_DIR) {
13                 strcpy(nextDir, path);
14                 strcat(nextDir, "/");
15                 strcat(nextDir, f.f_name);
16                 walk_dir(nextDir, level + 1);
17             }
18         }
19     }
20 }
```

外部命令扩展

tree, mkdir, touch

- mkdir & touch

- 直接调用 前置工作中已经完成的文件创建服务接口即可，但需要注意路径解析问题
- mkdir: a, a/b, a/c/d, a
- touch: test1.c, a/test2.c, f/test3.c, test1.c

```
1  /* user/touch.c */  
2  void touch(char *path, char *prefix) {  
3      /* ..... */  
4      if((r = create(curpath, FTYPE_REG)) < 0) {  
5          fprintf(1, "\033[0m\033[1;31mFile %s Already  
6              exists\n");  
7          return;  
8      }  
9      fprintf(1, "\033[0m\033[1;33mFile %s created :  
10     ");  
11 }
```

```
1  /*user/mkdir.c */  
2  void mkdir(char *path, char *prefix) {  
3      /* ..... */  
4      if((r = create(curpath, FTYPE_DIR)) < 0) {  
5          fprintf(1, "\033[0m\033[1;31mDirectory %s  
6              already exists\n");  
7          return;  
8      }  
9      fprintf(1, "\033[0m\033[1;33mCreated Directory %s  
10     ");  
11 }
```

外部命令扩展

tree, mkdir, touch

- mkdir 效果展示

```
HJinKuo $ mkdir a
Created Directory /a successfully!

HJinKuo $ mkdir a/b
Created Directory /a/b successfully!

HJinKuo $ mkdir a/c/d
Created Directory /a/c/d successfully!

HJinKuo $ mkdir a
Directory /a Already Exists!
```

```
HJinKuo $ tree
.
├── motd
├── newmotd
├── testarg.b
├── init.b
├── num.b
├── echo.b
├── ls.b
├── sh.b
├── cat.b
├── tree.b
├── mkdir.b
├── touch.b
├── history.b
├── declare.b
├── unset.b
├── testhang.b
├── testptelibrary.b
├── .history
└── a
    ├── b
    ├── c
    └── d
```

外部命令扩展

tree, mkdir, touch

- touch 效果展示

```
HJinKuo $ touch test1.c
File /test1.c created successfully!

HJinKuo $ touch a/test2.c
File /a/test2.c created successfully!

HJinKuo $ touch f/test3.c
File /f/test3.c created successfully!

HJinKuo $ touch test1.c
File /test1.c Already Exists!
```

```
HJinKuo $ tree
.
├── motd
├── newmotd
├── testarg.b
├── init.b
├── num.b
├── echo.b
├── ls.b
├── sh.b
├── cat.b
├── tree.b
├── mkdir.b
├── touch.b
├── history.b
├── declare.b
├── unset.b
├── testhang.b
├── testptelibrary.b
└── .history
    └── a
        ├── b
        ├── c
        └── d
            └── test2.c
└── test1.c
```

内部命令扩展

history & declare & unset

history

- 核心功能：历史输入命令的记录、导出与按键回溯

```
HJinKuo $ history  
  
---- History Commands ----  
  
mkdir a  
mkdir a/b  
mkdir a/c/d  
mkdir a  
tree  
touch test1.c  
touch a/test2.c  
touch f/test3.c  
touch test1.c  
tree  
history
```



内部命令扩展

history & declare & unset

history

- **记录：**将在 shell 中输入的每步指令，在解析前保存进一个专用文件 `/.history` 中，每行一条指令
 - **history_init：**在解析指令前进行 history_init 时自动创建该文件用以保存历史命令
 - 支持：文件系统中的创建文件服务

```
1  /* user/sh.c*/
2  void u_main(int argc, char **argv) {
3      history_init();
4      /* ..... */
5  }
```

```
1  /* user/sh_history.c*/
2  void history_init() {
3      int r;
4      if((r = create("./.history", FTYPE_REG)) < 0)
5          user_panic("Fail to initialize /.history:");
6  }
```

内部命令扩展

history & declare & unset

history

- 按行保存

- 在指令解析之前及时将指令内容写入到 .history 文件的末尾
 - 支持: O_APPEND

```
1  /* user/sh.c */  
2  void runcmd(char *s) {  
3      history_save(s);  
4      /* ..... */  
5 }
```

```
1  /* user/sh_history.c */  
2  void history_store(char *s) {  
3      int r;  
4      if ((r = open("./.history", O_RDWR | O_CREAT |  
5                  user_panic("Fail to open /.history.");  
6                  fwritef(r, "%s\n", s);  
7                  close(r);  
8 }
```

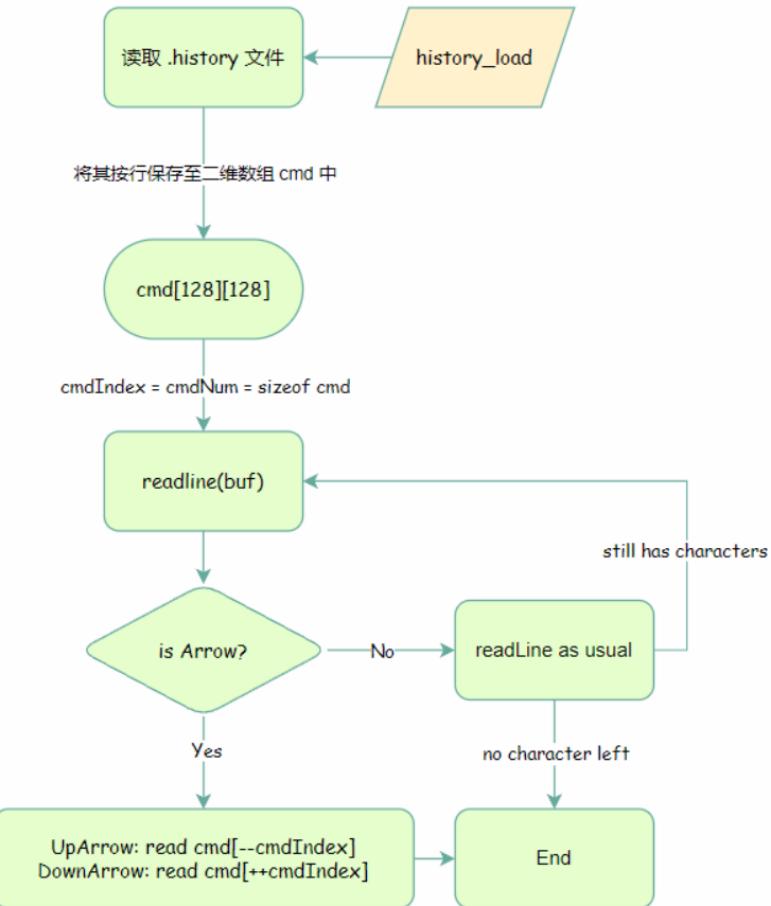
内部命令扩展

history & declare & unset

history

- 按键回溯

- 按键对应的指令: `esc[A` , `esc[B`
- 修改核心: `readline`
 - 将 `.history` 记录的命令保存到一个二维数组里, 以实现按行移动



内部命令扩展

history & declare & unset

history

▪ 按键回溯

- 按键对应的指令: `esc[A` , `esc[B`
- 修改核心: `readline`
 - 将 `history` 记录的命令保存到一个二维数组里, 以实现按行移动

```
1  /* user/sh.c_readline */  
2  if (i >= 2 && buf[i - 2] == 27 && buf[i - 1] ==  
3      // Clean all the words in the current line  
4      int cleanLen = strlen(buf) - 3;  
5      writef("%c%c%c", 27, 91, 66);    // Keep the  
6      flush(cleanLen);  
7  
8      if (cmdi)  
9          strcpy(buf, cmds[--cmdi]);  
10     else  
11         strcpy(buf, cmds[cmdi]);  
12     writef("%s", buf);  
13     i = strlen(buf) - 1;    // goto end  
14 } else if (i >= 2 && buf[i - 2] == 27 && buf[i - 1] ==  
15     int cleanLen = strlen(buf) - 3;  
16     flush(cleanLen);  
17  
18     if (cmdi < cmdn - 1) {  
19         strcpy(buf, cmds[++cmdi]);  
20         writef("%s", buf);  
21     }  
22     i = strlen(buf) - 1;    //goto end  
23 }
```

内部命令扩展

history & declare & unset

Challenge: 环境变量

■ 原理

- NAME + VALUE: 构造哈希表进行映射对照
- 应当存储在**内核态**, 供给用户态程序共享读取
 - 构造对应的系统调用 syscall_envar 进行调用

■ 步骤

- 构建环境变量系统调用 syscall_envar, 沟通用户态和内核态
- 添加对应的命令文件 `declare.c`, `unset.c`
- 设置一个数组 `definedAge[]` 记录当前环境, 以区分局部变量与环境变量
- echo \${VARIABLE}: 在 echo.c 中解析 `\$\$` 开头的字符串进行替换

内部命令扩展

history & declare & unset

Challenge: 环境变量

- `syscall_envar`
 - `user/syscall_lib.c`: int syscall_envar(char *name, char *value, u_int op, u_int isEnvvar)
 - `lib/syscall_all.c`: int sys_envar(int sysno, char *name, char *value, u_int op, u_int isEnvvar)
 - `lib/syscall.S`:.word sys_envar
 - `include/unistd.h`: #define SYS_envar

内部命令扩展

history & declare & unset

Challenge: 环境变量

- syscall_envar

```
int sys_envar(int sysno, char *name, char *value,
u_int op, u_int isEnvar)
```

```
1  int sys_envar(int sysno, char *name, char *value, u_
2      const int MOD = 256;
3      static char name_table[256][64];
4      static char value_table[256][256];
5      static int is_READONLY[256];
6      static int definedAge[256]; /* When the variab
7      static int currentAge;
8      /* If definedAge == -1, then it is an environmen
9      /* Else if defineAge == currentAge, then it is a
10
11     if (strcmp(name, "curpath") == 0) {
12         currentAge++;
13         /* Every time we invoke the command 'sh' to (
14     }
15
16     /* [1] Find the position of the NAME in our hash
17     /* [2] Operate according to the value of op. */
18
19     return 0;
20 }
```

```
1 -----[op]-----
2
3 0 创建变量 NAME-VALUE
4
5 1 获得 NAME 的 VALUE 值
6
7 2 重新定义变量 NAME
8
9 3 删除非只读的变量 NAME
10
11 4 输出当前进程的所有变量
12
13 5 设置只读变量 NAME-VALUE
14
15 -----[isEnvar]-----
16
17 1 environment variable
18
19 0 local variable
```

```
1 /* [2] */
2 if (op == 0) {
3     // Create a new variable
4     strcpy(name_table[pos], n);
5     strcpy(value_table[pos], `);
6     if (isEnvar == 1) {
7         definedAge[pos] = -1;
8     } else {
9         definedAge[pos] = cur;
10    }
11 } else if (op == 1) {
12     // Get the value of NAME
13     if (definedAge[pos] != cur) {
14         // Not local & Not Env
15         printf("\033[0m\033[1");
16         return -E_ENVAR_NOT_F;
17     }
18     if (strcmp(name_table[pos], `) == 0) {
19         printf("\033[0m\033[1");
20         return -E_ENVAR_NOT_F;
21     }
22     strcpy(value, value_table[pos]);
23 }
24 }
```

```
1 else if (op == 2) {
2     // Set NAME-VALUE
3     if (strcmp(name_table[pos], `) == 0) {
4         return -E_ENVAR_NOT_F;
5     }
6     if (is_READONLY[pos]) {
7         return -E_ENVAR_READONLY;
8     }
9     if (definedAge[pos] != -1) {
10        // Not environment or local
11        return -E_ENVAR_NOT_F;
12    }
13     // Set variable
14     strcpy(value_table[pos], `);
15     if (isEnvar) {
16         definedAge[pos] = -1;
17     }
18     else if (!isEnvar) {
19         definedAge[pos] = cur;
20     }
21 }
```

```
1 -----[op]-----
2
3 0 创建变量 NAME-VALUE
4
5 1 获得 NAME 的 VALUE 值
6
7 2 重新定义变量 NAME
8
9 3 删除非只读的变量 NAME
10
11 4 输出当前进程的所有变量
12
13 5 设置只读变量 NAME-VALUE
14
15 -----[isEnvar]-----
16
17 1 environment variable
18
19 0 local variable
```

```
1 /* [2] */
2 if (op == 0) {
3     // Create a new variable
4     strcpy(name_table[pos], n);
5     strcpy(value_table[pos], `);
6     if (isEnvar == 1) {
7         definedAge[pos] = -1;
8     } else {
9         definedAge[pos] = cur;
10    }
11 } else if (op == 1) {
12     // Get the value of NAME
13     if (definedAge[pos] != cur) {
14         // Not local & Not Env
15         printf("\033[0m\033[1");
16         return -E_ENVAR_NOT_F;
17     }
18     if (strcmp(name_table[pos], `) == 0) {
19         printf("\033[0m\033[1");
20         return -E_ENVAR_NOT_F;
21     }
22     strcpy(value, value_table[pos]);
23 }
24 }
```

```
1 else if (op == 2) {
2     // Set NAME-VALUE
3     if (strcmp(name_table[pos], `) == 0) {
4         return -E_ENVAR_NOT_F;
5     }
6     if (is_READONLY[pos]) {
7         return -E_ENVAR_READONLY;
8     }
9     if (definedAge[pos] != -1) {
10        // Not environment or local
11        return -E_ENVAR_NOT_F;
12    }
13     // Set variable
14     strcpy(value_table[pos], `);
15     if (isEnvar) {
16         definedAge[pos] = -1;
17     }
18     else if (!isEnvar) {
19         definedAge[pos] = cur;
20     }
21 }
```

```
1 else if (op == 3) {
2     // unset a non-read-only variable
3     if (strcmp(name_table[pos], name))
4         return -E_ENVAR_NOT_FOUND;
5     if (is_READONLY[pos]) {
6         return -E_ENVAR_READONLY;
7     } else {
8         definedAge[pos] = 0;
9         int p = 0;
10        while (p < 64 && name_table[pos][p]) {
11            name_table[pos][p] = 0;
12            p++;
13        }
14        p = 0;
15        while (p < 256 && value_table[pos][p]) {
16            value_table[pos][p] = 0;
17            p++;
18        }
19    }
20}
```

```
1 else if (op == 4) {
2     // List all the variable
3     int pos = 0, i, j, cnt = 0;
4     for (i = 0; i < 256; ++i)
5         if (name_table[i][0]) {
6             if (strcmp(name_table[i], "curpath"))
7                 continue;
8         }
9         printf("NAME: %s", name_table[i]);
10        for (j = 0; j < (16 - strlen(name_table[i])); j++)
11            printf("%c", name_table[i][j]);
12        for (j = 0; j < (16 - strlen(value_table[i])); j++)
13            if (definedAge[i] == -1) {
14                printf("\033[0m\033[1;32m--[");
15            } else {
16                printf("\033[0m\033[1;32m--[%s]", value_table[i][j]);
17                if (is_READONLY[i]) {
18                    printf("\033[0m\033[1;33mHave");
19                    cnt++;
20                }
21        }
22        if (cnt == 0) {
23            printf("\033[0m\033[1;33mHave");
24        }
25    }
26}
```

测试工作

Test is important.

- 命令组合检查 shell 输出现象
- 编写测试程序测试后台命令

```
1  /* user/testhang.c */
2  void testHang(int n) {
3      int i = 2;
4      int checkpoint = n / 10;
5      for (; i <= n; i++) {
6          if (i % checkpoint == 0) {
7              printf("\033[0m\033[1;32m@@@ Hey it is h:");
8          }
9          if (isprime(i)) {
10      }
11  }
12
13 void umain() {
14     printf("\033[0m\033[1;31m--- Start to test a slow program ---");
15     testHang(100000);
16     printf("\033[0m\033[1;31m--- The nohup progame has been run ---");
17     return;
18 }
```

问题及解决方案

Problems & solutions.

- 方向键对应的指令

- 命令行窗口调试

- 上下键回溯命令时如何使得 heading 不被错误覆盖

- 弄明白 buf 存储的内容和 `readline` 的流程，以正确计算退格清空字符的个数

- 区别局部变量和环境变量

- 定义 `definedAge[]` 和 `currentAge`

- 每次调用 `sh` 时调用一次 `syscall_envvar` 增加当前 `currentAge`，据此区分父 shell 和子 shell

- `echo` 调用 `sys_envvar` 替换变量值时比较每个变量的 `definedAge` 和 `currentAge`

Extra

Extra works that I have done.

- `.` 补全
 - runcmd 进入 `spawn` 前为 argv[0] 补全 `.` 字符
 - 部分语句的彩色输出

- ```
1 printf("\033[0m\033[1;%dm%s\033[0m");
2 // %d - Indicate the color number
3 // %s - The string that you want to print
```

# Complete Code and Report

[Github](#)