《操作系统》申优答辩

Lab6-Challenge

18375354 董翰元

目录

准备工作

Part 1.

$ shell_

Before a Better Shell

- 扩展文件系统
- 简化输出信息
- 实现光标控制
- 丰富库函数

扩展文件系统

1. 创建文件
2. 创建目录
3. 实现 lseek

创建文件
Create File

- 用户态添加 `create()`
- `fs_serv` 添加 `serv_create`
- 使 `open` 支持 `O_CREAT`
  - 尝试打开不存在的文件时自动创建

创建目录
Create Directory

- 向 create 方法提供 f_type 参数
- 自动创建多级目录
  - 在 f_type 参数中引入控制位 MKDIR_P

这一部分与 lab5-2 课上测试内容相似

lseek

user/fd.c 中已经实现好一个 seek:

```
  int seek(int fd, u_int offset);
```

不够用?

现有的 seek 过于简单，只支持绝对偏移量

history: 如何方便地跳转到文件的**结尾**?

```
int lseek(int fd, u_int offset, int where);
```

```
SEEK_SET
SEEK_CUR
SEEK_END <-
```

相关代码展示

fs/serv.c
创建文件

```c
void
serve_create(u_int envid, struct Fsreq_create *rq)
{
    char path[MAXPATHLEN];
    strcpy(path, rq->req_path);
    int ftype = rq->req_ftype;
    struct File *f;
    int r;
    if ((ftype & MKDIR_P)) {
        ftype &= (~MKDIR_P);
        char *p = path;
        if (*p == '/') p++;
        p++;
        while (*p) {
            while (*p && *p != '/') p++;
            if (!*p) break;
            *p = 0; // temporary set
            r = file_create(path, &f, FTYPE_DIR);
            if (r < 0 && r != -E_FILE_EXISTS) {
                ipc_send(envid, r, 0, 0);
                return ;
            }
            *p = '/';
            p++;
        }
        r = file_create(path, &f, ftype);
        ipc_send(envid, r, 0, 0);
    } else {
        r = file_create(path, &f, ftype);
        ipc_send(envid, r, 0, 0);
    }
}
```

fs/fs.c
创建文件

```c
int
file_create(char *path, struct File **file, int ftype)
{
>---char name[MAXNAMELEN];
>---int r;
    struct File *dir, *f;

>---if ((r = walk_path(path, &dir, &f, name)) == 0) {
>--->---return -E_FILE_EXISTS;
>---}

    if (r == -E_NOT_FOUND && dir == 0) {
        return -E_DIR_NOT_EXISTS;
    }

>---if (r != -E_NOT_FOUND || dir == 0) {
>--->---return r;
>---}

>---if (dir_alloc_file(dir, &f) < 0) {
>--->---return r;
>---}

>---strcpy((char *)f->f_name, name);
    f->f_type = ftype;
>---*file = f;
>---return 0;
}
```

fs/serv.c
serve_open
支持 O_CREAT 选项

```c
if ((r = file_open((char *)path, &f)) < 0) {
    // MyLab6: add support for O_CREAT
    if (( (rq->req_omode) & O_CREAT )) {
        r = file_create((char *)path, &f, FTYPE_REG);
        if (r < 0) {
            ipc_send(envid, r, 0, 0);
            return ;
        }
        r = file_open((char *)path, &f);
        if (r < 0) {
            ipc_send(envid, r, 0, 0);
            return ;
        }

    } else {
>---     // user_panic("file_open failed: %d, invalid path: %s", r, path);
>---     ipc_send(envid, r, 0, 0);
>---     return ;
    }
}
```

user/lib.h
自动创建目录

```c
/* File create modes */
#define MKDIR_P     0x0100      /* create dir if dir not exist */
```

相关代码展示

user/fd.c
lseek 方法

```c
int
lseek(int fdnum, u_int offset, int where) {
    int r;
    struct Fd *fd;
    if ((r = fd_lookup(fdnum, &fd)) < 0) {
        return -1; // failed!
    }
    // need to know the size of file
    int fsize, fpeof;
    struct Filefd *ffd = (struct Filefd *)fd;
    fsize = ffd->f_file.f_size;
    fpeof = fsize;

    int off = fd->fd_offset;

    switch (where) {
        case LSEEK_SET:
            off = 0 + offset;
            break;
        case LSEEK_CUR:
            off = off + offset;
            break;
        case LSEEK_END:
            off = fpeof + offset;
            break;
        default:;
    }
    if (off > fpeof) off = fpeof;
    if (off < 0) off = 0;
    fd->fd_offset = off;
    return fd->fd_offset;
}
```

user/lib.h
lseek 第三个参数

```c
/* File lseek marks */
#define LSEEK_SET   0
#define LSEEK_CUR   1
#define LSEEK_END   2
```

简化输出信息

- 输出换行多出的空行
- serve_open
- env free and env destroy
- pageout
- spawn
- ……

简化输出信息

```
$ ls.b

[00002803] SPAWN: ls.b

serve_open 00002803 ffff000 0x0

pageout:        @@@___0x7f3fcc84___@@@  ins a page



::::::::::spawn size : 20  sp : 7f3fdfe8::::::::

serve_open 00003004 ffff000 0x0

serve_open 00003004 ffff000 0x0

motd newmotd testarg.b init.b num.b echo.b ls.b sh.b cat.b testptelibrary.b [00003004] destroying 00003004

[00003004] free env 00003004

i am killed ...

[00002803] destroying 00002803

[00002803] free env 00002803

i am killed ...
```

```
mos $ ls
motd newmotd testarg.b init.b num.b echo.b ls.b sh.b cat.b

mos $ exit
```


简化后

简化前

光标控制

封装了若干简单实用的光标控制方法:

- 清除屏幕　"\033[2J"
- 光标的移动与定位
- 行内容清除
- 彩色输出控制

光标控制

```
void cur_clear();
void cur_move(int, int);
void cur_set(int, int);

void cur_clear_lright();
void cur_backto_lhead();

void cur_color_front(int);
void cur_color_back(int);
void cur_color(int, int);
void cur_color_restore();
```

光标控制函数接口

```
#define CUR_MOVE_UP          0x01
#define CUR_MOVE_DOWN        0x02
#define CUR_MOVE_LEFT        0x03
#define CUR_MOVE_RIGHT       0x04

#define COLOR_FRONT_BLACK        30
#define COLOR_FRONT_RED          31
#define COLOR_FRONT_GREEN        32
#define COLOR_FRONT_YELLOW       33
#define COLOR_FRONT_BLUE         34
#define COLOR_FRONT_PURPLE       35
#define COLOR_FRONT_DEEP_GREEN   36
#define COLOR_FRONT_WHITE        37

#define COLOR_BACK_BLACK         40
#define COLOR_BACK_RED           41
#define COLOR_BACK_GREEN         42
#define COLOR_BACK_YELLOW        43
#define COLOR_BACK_BLUE          44
#define COLOR_BACK_PURPLE        45
#define COLOR_BACK_DEEP_GREEN    46
#define COLOR_BACK_WHITE         47
```

颜色、移动方向宏定义

库函数扩展

- 在 user/printf.c 中实现 swritef 函数
  - 相当于 sprintf
- 扩展字符串库 string.c
  - 添加 strcat, strncmp
  - 复制一份到内核态
- 实现 memset 函数
  - 用 {} 初始化局部数组会用到

Shell 总体设计

Part 2.

$ shell_

Shell 代码结构

1. 欢迎信息与命令提示符
2. 命令输入缓冲区，光标与显示
3. 特殊按键的检测及事件
4. 控制台输入主体逻辑
5. 内部命令支持
6. 历史命令相关
7. 命令解析器

Feature: 增强型光标控制

更为流畅的用户体验：

- 支持 `Backspace` 与 `Delete` 键删除字符
- 左右键移动光标位置

Feature: 增强型光标控制

重构了原有的 `readline` 逻辑

- 维护输入缓冲区数组和光标位置
- "状态机"结构，根据输入的字符进行状态转移
  换行或回车：进入命令解释
  其他字符：
  - 插入缓冲区：在数组中插入字符
  - 特殊按键序列检测：移动光标或删除字符
  - 根据缓冲区数组的内容刷新屏幕显示

相关代码实现

```c
void mainloop() {
    char ch;
    while (1) {
        ch = syscall_noblock_getc();
        if (ch == 0) {
            syscall_yield();
            continue;
        }
        if (ch == '\r' || ch == '\n') {
            syscall_putchar('\n');
            inputenter();
        } else {
            if (ch >= 32 && ch <= 126)
                syscall_putchar(ch);
            else
                syscall_putchar('\?');
            insertChar(ch);
            special_key_detect();
        }
    }
}
```

接收输入字符并响应

```c
#define SPECIAL_KEY_BACKSPACE 1
#define SPECIAL_KEY_UP      2
#define SPECIAL_KEY_DOWN    3
#define SPECIAL_KEY_LEFT    4
#define SPECIAL_KEY_RIGHT   5
#define SPECIAL_KEY_DELETE  6
typedef struct {
    u_char len;
    char chr[0];
} SpecialKey;
char special_key_data[] = {1, 0x7f, 3, 0x1b, 0x5b, 0x41, 3, 0x1b,
SpecialKey *spec_key[] = {
    0,
    (SpecialKey*)(special_key_data + 0), // BACKSPACE
    (SpecialKey*)(special_key_data + 2), // UP
    (SpecialKey*)(special_key_data + 6), // DOWN
    (SpecialKey*)(special_key_data + 10), // LEFT
    (SpecialKey*)(special_key_data + 14), // RIGHT
    (SpecialKey*)(special_key_data + 18) // DELETE
};

static void key_event_backspace();
static void key_event_delete();
static void key_event_up();
static void key_event_down();
static void key_event_left();
static void key_event_right();

void (*key_event[])(void) = {0, key_event_backspace, key_event_up,
```

定义特殊按键序列和事件

```c
static void special_key_detect() {
    int numkey = ARRAY_SIZE(spec_key);
    SpecialKey *sk;
    int i;
    for (i = 0; i < numkey; i++) {
        if (spec_key[i] == 0) continue;
        sk = spec_key[i];
        // input_buf[cursor_pos - sk->len : cursor_pos] equals sk-> chr
        if (cursor_pos < sk->len) continue;
        int ppos = cursor_pos - sk->len;
        int eqflag = 1;
        int j;
        for (j = 0; j < sk->len; j++) {
            if (input_buf[ppos + j] == sk->chr[j]) continue;
            else { eqflag = 0; break; }
        }
        if (eqflag) {
            // Hit!
            text_cur_move(ppos - cursor_pos);
            for (j = 0; j < sk->len; j++) {
                deleteChar();
            }
            if (key_event[i] != 0)
                (key_event[i])(); // run callback
            break;
        }
    }
}
```

特殊按键序列检测

Shell 整体实现

由于代码过长，故无法在 PPT 中贴下截图

需打开代码编辑器展示

Easy Task

Part 3.

- 实现清屏
- 分号：一行多命令
- &：后台运行
- 引号支持
- 新增命令
- 彩色输出(在命令输出中体现)

$ shell_

Easy - clear

shell 内部命令机制 + 封装好的光标库

```
/* ^^^^^^ Part 5. Inner commands ^^^^^^^^^ */
static void cmd_clear(int argc, char **argv) {
    cur_clear();
}
```

```
typedef struct {
    char *cmd;
    void (*run)(int , char **);
} InnerCommand;

InnerCommand inner_cmd[] = {
    {"clear", cmd_clear},
    {"exit", cmd_halt},
    {"quit", cmd_halt},
    {"halt", cmd_halt},
    {"set", cmd_set},
    {"unset", cmd_unset},
    {"export", cmd_export}
};
```

Easy – 分号多命令

在 runcmd 中进行相应的处理

```
case ';':
    if ((r = fork()) == 0) {
        fdredirect[0] = 0;
        fdredirect[1] = 1;
        goto again;
    } else {
        goto runit;
    }
    break;
```

展示效果：

```
mos $ echo.b 1234 ; cat.b newmotd
1234
This is the NEW message of the day!


mos $
```

Easy – 后台运行

修改读取字符时的内核态忙等机制
识别到 & 符号后：
父进程不再等待子进程执行完

```
case '&':
    isbackground = 1;
    break;
```

```
if (r >= 0) {
    if (debug_) writef("[%08x] WAIT %s %08x\n", env->env_id, argv[0], r);
    if (!isbackground)
        wait(r);
}
```

为了展示后台运行效果，
编写了一个运行时间较长的程序 slow.b

展示效果：

```
mos $ slow.b &
@@ A slow program @@

mos $ @@@ i = 10000 @@@
@@@ i = 20000 @@@


mos $ ca@@@ i = 30000 @@@
t.b newmotd
This is the NEW message of the day!


mos $ @@@ i = 40000 @@@
@@@ i = 50000 @@@
@@@ i = 60000 @@@
@@@ i = 70000 @@@


mos $ echo.b 1122
1122

mos $ @@@ i = 80000 @@@
```

Easy – "引号支持"

修改 gettoken，遇到引号则将整体当做一个 WORD

```
#define WHITESPACE " \t\r\n"
#define SYMBOLS "<|>&;()\""
```

```
static char* extractQuote(char *dst, const char *src) {
    // src: the first character **after** the beginning \"
    // returns: the position of the **ending** \", or the end of string
    char *s = src, *t = dst;
    while (*s && *s != '\"') {
        if (t) { *t = *s; }
        s++;
    }
    return s;
}
```

```
if(strchr(SYMBOLS, *s)){
    if (*s == '\"') {
        s++;
        *p1 = s;
        s = extractQuote(NULL, s);
        if (*s == 0)
            writef("@@@ Quotation not end, unexpected EOL\n");
        *s = 0;
        s++;
        *p2 = s;
        return 'w';
    }
    else {
        t = *s;
        *p1 = s;
        *s++ = 0;
        *p2 = s;
          if (debug_ > 1) writef("TOK %c\n", t);
        return t;
    }
```

展示效果：

```
mos $ echo.b "111 | 222"
111 | 222
```

Easy – 新增命令

分别编写用户态程序（USERAPP）

- `touch, mkdir`: 借助实现好的文件/目录创建接口
- `tree`: 模仿 `ls` 中的遍历目录逻辑+彩色输出

mkdir / touch

```c
void mmakedir(const char *path, int recursive) {
    int r;
    int mode = recursive ? (FTYPE_DIR | MKDIR_P) : (FTYPE_DIR);
    r = create(path, mode);
    if (r < 0) {
        writef("mkdir failed with code %d\n", r);
        return;
    }

}

void umain(int argc, char **argv) {
    // writef("@ mkdir: argc = %d\n", argc);
    if (argc == 1) {
        writef("usage: mkdir [-p] dirname\n");
        return;
    }
    if (strcmp(argv[1], "-p") == 0) {
        if (argc >= 2) {
            mmakedir(argv[2], 1);
        } else {
            writef("error: need dirname");
        }
    } else {
        if (argv[1][0] == '-') {
            writef("error: invalid option %s\n", argv[1]);
        } else {
            mmakedir(argv[1], 0);
        }
    }
}
```

左：mkdir
右：touch

```c
#include "lib.h"
#include <args.h>

void umain(int argc, char **argv)
{
    // writef("@ touch: argc = %d\n", argc);
    if (argc < 2) {
        writef("usage: touch filename");
    }
    int r;
    r = create(argv[1], FTYPE_REG | MKDIR_P);
    if (r < 0) {
        writef("touch %s failed with err code %d\n", argv[1], r);
    }
}
```

tree

```
void tree_print(char *name, int depth, int ftype) {
    int i;
    int len;
    for (i = 0; i < depth; i++) {
        writef("--");
    }
    len = strlen(name);
    if (ftype == FTYPE_DIR)
        cur_color_front(COLOR_FRONT_BLUE);
    else {
        if (len > 2 && name[len - 1] == 'b' && name[len - 2] == '.')
            cur_color_front(COLOR_FRONT_GREEN);
        else
            cur_color_front(COLOR_FRONT_WHITE);
    }
    writef("%s", name);
    cur_color_restore();
    writef("\n");
}
```

```
void tree_dir1(char *path, int depth) {
    int fd, n;
    struct File f;
    if ((fd = open(path, O_RDONLY)) < 0) {
        write("error on tree: open %s returns %d\n", path, fd);
        exit();
    }
    while ((n = readn(fd, &f, sizeof(f))) == sizeof(f)) {
        if (f.f_name[0]) {
            tree_print(f.f_name, depth, f.f_type);
        }
        if (f.f_type == FTYPE_DIR) {
            char nextpath[MAXPATHLEN];
            strcpy(nextpath, path);
            strcat(nextpath, "/");
            strcat(nextpath, f.f_name);
            tree_dir(nextpath, depth + 1);
        }
    }
}
```

```
void tree_dir(char *path, int depth) {
    int r;
    struct Stat st;
    if ((r = stat(path, &st)) < 0) {
        writef("error on tree: stat %s returns %d\n", path, r);
        exit();
    }
    if (st.st_isdir) {
        tree_dir1(path, depth);
    }

}

void umain(int argc, char **argv) {
    // writef("@ tree: argc = %d\n", argc);
    if (argc == 1) {
        tree_dir("/", 0);
    } else {
        tree_dir(argv[1], 0);
    }
}
```

Easy – 彩色输出

通过光标库或直接输出相应的 ANSI 控制序列

在 tree.b 和 ls.b 以及 shell 的命令提示符中已经体现

命令运行效果

```
mos $ mkdir dir1; mkdir dir2; mkdir dir1/dir12; mkdir -p dir3/dir31/dir311;

mos $ touch file1.txt ; touch dir1/file2.txt ; touch dir3/dir31/file3.txt;

mos $ tree
motd
newmotd
testarg.b
init.b
num.b
echo.b
ls.b
sh.b
cat.b
fstest.b
testcursor.b
testinput.b
extsh.b
slow.b
mkdir.b
touch.b
tree.b
history.b
testptelibrary.b
.history
dir1
--dir12
--file2.txt
dir2
dir3
--dir31
----dir311
----file3.txt
file1.txt
```

Medium - History

Part 4.

- 历史命令的 `userlib`
- 展示历史的 `userapp history.b`
- `shell` 对历史命令的支持

用户库: 历史命令存取

位于 user/lib.h 中的声明

```c
// historylib.c

int history_getcount();
void history_store(const char *command);
void history_load(int index, char *dst);
void history_clear();
```

部分方法的实现

```c
void history_load(int index, char *dst) {
    // index starts at zero
    char ch = 10;
    char *p = dst;
    int countlf = 0;
    int fd = open(HISTORY_FILENAME, O_RDONLY);
    lseek(fd, 0, LSEEK_SET);
    if (fd < 0) {
        // writef("error history_load\n");
        return;
    }
    while (countlf < index) {
        ch = mhfgetc(fd);
        if (ch == -1) break;
        if (ch == 10) countlf++;
    }

    if (ch != -1 && ch) {
        while (1) {
            ch = mhfgetc(fd);
            if (!ch || ch == -1 || ch == '\n')
                break;

            *p = ch;
            p++;
        }
        *p = 0;
    }
    close(fd);
}
```

history.b

```
void umain(int argc, char **argv) {
    // writef("@ history.b : argc = %d\n", argc);

    // testhistory();
    int flagclr = 0;
    int hiscount = 0;
    char buf[1024];
    int i;

    if (argc >= 2) {
        if (strcmp(argv[1], "-c") == 0) {
            flagclr = 1;
        }
    }

    if (flagclr) {
        history_clear();
        exit();
    }

    hiscount = history_getcount();
    for (i = 0; i < hiscount; i++) {
        history_load(i, buf);
        writef("\033[31m%d\033[0m: \033[34m%s\033[0m\n", i + 1, buf);
    }

}
```

shell 历史命令支持

```
void key_event_up() {
    load_prev_history();
}

void key_event_down() {
    load_next_history();
}
```

```
void inputenter() {
    if (input_size > 0) {
        input_buf[input_size] = 0;
        int r;
        // record history
        store_history();
```

```
void load_next_history() {
    int c = history_getcount();
    history_index++;
    if (history_index >= c) {
        update_history_index();
        clearbuffer();
        // reprint();
        return;
    }
    history_load(history_index, input_buf);
    input_size = 0;
    for (; input_buf[input_size]; input_size++);
    text_cur_move(-cursor_pos);
    reprint();
    text_cur_move(input_size);
}

void store_history() {
    history_store(input_buf);
    update_history_index();
}
```

展示效果

```
mos $ history
1: clear
2: ls.b
3: cat.b newmotd
4: echo.b 123456 ; echo.b abcdef;
5: history
```

Challenge – 环境变量

Part 5.

$ shell_

环境变量

实现原理

- 名称 + 值
- 存储在内核态
- 通过系统调用来访问
- 用户态程序共享读取

实现步骤

1. 在内核态实现相应数据结构
2. 封装好相应的读写接口
3. 新增系统调用沟通内核态与用户态
4. 在 shell 中增加相应命令
5. 对 $ 开头命令参数的处理

环境变量的存储

lib/environment_var.c

```c
#define MAX_ENVVAR_NUM 16
#define MAX_ENVVAR_LEN 256
int var_count = 0;

char var_names[MAX_ENVVAR_NUM][MAX_ENVVAR_LEN];
char var_values[MAX_ENVVAR_NUM][MAX_ENVVAR_LEN];
u_char var_isro[MAX_ENVVAR_NUM];

static int name2index(const char *name) {
    int i;
    for (i = 0; i < var_count; i++) {
        if (strcmp(var_names[i], name) == 0) {
            return i;
        }
    }
    return -1;
}
```

字符数组存储名称和值
通过下标 index 一一对应

include/var.h

```c
int envvar_count();
void envvar_name(int, char *dst);
int envvar_set(const char *name, const char *val, u_char ro);
int envvar_get(const char *name, char *dst);
void envvar_rm(const char *name);
int envvar_isro(const char *name);

#define ENV_VAR_GET 0
#define ENV_VAR_SET 1
#define ENV_VAR_UNSET 2
#define ENV_VAR_COUNT 3
#define ENV_VAR_NAME 4
#define ENV_VAR_SETRO 5
#define ENV_VAR_ISRO 6
```

内核态的访问接口
操作种类的定义

环境变量系统调用

```c
int sys_environment_var(int sysno, int op, char *name, char *value, int index) {
    switch (op) {
    case ENV_VAR_GET:
        return envvar_get(name, value);
    case ENV_VAR_SET:
        return envvar_set(name, value, 0);
    case ENV_VAR_UNSET:
        envvar_rm(name);
        return 0;
    case ENV_VAR_COUNT:
        return envvar_count();
    case ENV_VAR_NAME:
        envvar_name(index, name);
        return 0;
    case ENV_VAR_SETRO:
        return envvar_set(name, value, 1);
    case ENV_VAR_ISRO:
        return envvar_isro(name);
    default:
        return -1;
    }
}
```

借鉴 fsipc 的实现，只增加一个系统调用
通过 operation 参数做到增删改查

用户态访问接口

user/lib.h

```c
// envvar.c

int user_envvar_count();
void user_envvar_name(int, char *dst);
int user_envvar_set(const char *name, const char *val, u_int ro);
int user_envvar_get(const char *name, char *dst);
void user_envvar_rm(const char *name);
int user_envvar_isro(const char *name);
```

环境变量用户态接口：
封装一下系统调用

user/envvar.c

```c
int user_envvar_count() {
    return syscall_environment_var(ENV_VAR_COUNT, 0, 0, 0);
}

void user_envvar_name(int index, char *dst) {
    syscall_environment_var(ENV_VAR_NAME, dst, 0, index);
}

int user_envvar_set(const char *name, const char *val, u_int ro) {
    if (ro)
        return syscall_environment_var(ENV_VAR_SETRO, name, val, 0);
    return syscall_environment_var(ENV_VAR_SET, name, val, 0);
}

int user_envvar_get(const char *name, char *dst) {
    return syscall_environment_var(ENV_VAR_GET, name, dst, 0);
}

void user_envvar_rm(const char *name) {
    return syscall_environment_var(ENV_VAR_UNSET, name, 0, 0);
}

int user_envvar_isro(const char *name) {
    return syscall_environment_var(ENV_VAR_ISRO, name, 0, 0);
}
```

Shell 环境变量命令

set：设置环境变量

```
static void cmd_set(int argc, char **argv) {
    char *name = 0, *value = 0;
    int i;
    u_char flagro = 0;
    for (i = 1; i < argc; i++) {
        if (argv[i][0] == '-') {
            if (strcmp(argv[i], "-r") == 0) {
                flagro = 1;
            } else {
                writef("warning: unrecognized option %s\n", argv[i]);
            }
        } else {
            if (name == 0)
                name = argv[i];
            else if (value == 0)
                value = argv[i];
        }
    }
    if (name && value) {
        user_envvar_set(name, value, flagro);
    } else {
        writef("usage: set [-r] varname varvalue\n");
    }

}
```

unset：删除环境变量

```
static void cmd_unset(int argc, char **argv) {
    char *name = 0;
    if (argc >= 2) {
        name = argv[1];
        user_envvar_rm(name);
    }
    else {
        writef("usage: unset varname\n");
    }
}
```

Shell 环境变量命令

export：导出所有环境变量

```
static void cmd_export(int argc, char **argv) {
    static const char *ROMark = " RO";
    static const char *ROMark_col = " \033[32mRO\033[0m";
    u_char argFlag[128] = "";
    int i, j;
    char *foutname = 0;
    int fd = 1;
    char name[256], value[256];
    int n;
    int ro;
    for (i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-o") == 0) {
            if (i + 1 >= argc) {
                writef("error: no output file.\n");
            } else {
                foutname = argv[i + 1];
            }
        }
    }
    if (foutname) {
        fd = open(foutname, O_WRONLY | O_CREAT);
        if (fd < 0) {
            writef("error: cannot open file %s\n", foutname);
            return;
        }
    }
    n = user_envvar_count();
    // default: list all environment variables
    for (i = 0; i < n; i++) {
        user_envvar_name(i, name);
        user_envvar_get(name, value);
        ro = user_envvar_isro(name);
        if (fd > 1)
            fwritef(fd, "%s=%s%s\n", name, value, ro ? ROMark : "");
        else
            writef("\033[37m%s\033[0m=\033[35m%s\033[0m%s\n", name, value, ro ? ROMark_col : "");
    }
    if (fd > 1) {
        close(fd);
    }
}
```

- 彩色输出名称和值
- 显示某个变量是否只读
- 通过 "-o" 选项导出到文件

echo $var

对 $ 开头的命令参数进行判断，
提取相应环境变量的值

- 目前仅实现了 echo 命令显示环境变量
  - 对 $ 的判断在 echo 程序内部进行

```
char var_value[256];

void
umain(int argc, char **argv)
{
    int i, nflag;
    int r;

    nflag = 0;
    if (argc > 1 && strcmp(argv[1], "-n") == 0) {
        nflag = 1;
        argc--;
        argv++;
    }
    for (i = 1; i < argc; i++) {
        if (i > 1)
            write(1, " ", 1);
        if (argv[i][0] == '$') {
            r = user_envvar_get((char*)(argv[i] + 1), var_value);
            if (r == 0)
                write(1, var_value, strlen(var_value));
        }
        else
            write(1, argv[i], strlen(argv[i]));
    }
    if (!nflag)
        write(1, "\n", 1);
}
```

user/echo.c

环境变量展示

Part 1

```
mos $ set exam 10

mos $ set Extra 7

mos $ set -r school BUAA

mos $ set challenge Lab6

mos $ echo $exam $Extra $school $challenge
10 7 BUAA Lab6

mos $ export
exam=10
Extra=7
school=BUAA RO
challenge=Lab6
```

赋值、读取与导出

Part 2

```
mos $ unset exam

mos $ set school THU
error: school is read-only

mos $ echo $school
BUAA

mos $ export
Extra=7
school=BUAA RO
challenge=Lab6
```

删除变量、只读变量

Part 3

```
mos $ export -o variables.conf

mos $ cat variables.conf
Extra=7
school=BUAA RO
challenge=Lab6
```

导出到文件

其他
Part 6.

$ shell_

ASID

- 修改了 ASID 的分配算法
- 采用位图法管理 ASID 资源的分配和回收
- 修复了 shell 不能执行超过 31 条命令的 bug

重定向

- 在 runcmd 中解析到重定向符号时不再直接 dup
- dup 延后至 runit 阶段
  - 用变量记录输入输出是否有重定向
- 解决了分号后的命令受到分号前的重定向影响的 bug.

- 例: echo.b 123 > a.txt ; cat.b newmotd

.b 补全

- 键入命令时经常忘记打上 ".b"
- 在 runcmd 进入 spawn 前判断 argv[0] 结尾
  - 如缺失 ".b" 则采用 strcat 拼接补上
- 这样就无需打 ".b" 也能正常使用命令了

- 例: echo 123; cat motd; ls

syscall_halt

- 在 shell 中输入内部命令 halt 或 exit 退出 gxemul

综合展示