操作系统(D)

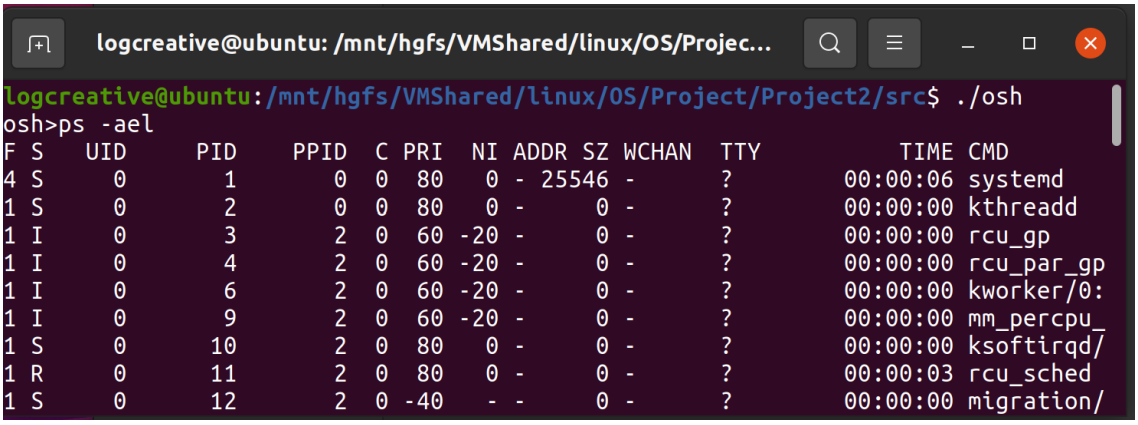
项目 2

李子龙 518070910095

2021 年 3 月 14 日

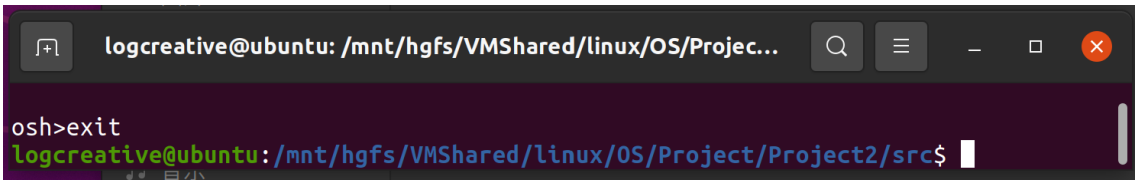
一 Unix 外壳程序

1. 在子进程中执行命令



同时执行命令：

当输入 `exit` 时，会退出外壳程序：



可以看到外壳主进程在子进程尚未打印完成前就已经开始等待用户输入了，也就是两个进程同时进行。

该部分需要获取用户输入，这里采用了 `fgets` 函数获取用户的整行输入。

```
fgets(rline, MAX_LINE, stdin);
```

第二参数用于说明最大读入字符数，将会限定最大的范围防止越界。该函数也支援存储换行符 \n。

在遍历输入字符时，如果遇到 & 字符，将会对 should_wait 标志置为 0，

```
        case '&':
            should_wait = 0;
```

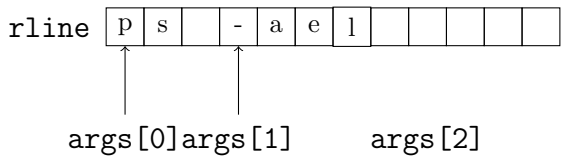
在之后的子进程创建函数中将会根据这个标志判定是否等待：

```
        pid_t pid;
        pid = fork();

        if (pid < 0) fprintf(stderr, "Fork Failed");
        else if (pid == 0) return execvp(args[0],args);
        else if (should_wait) wait(NULL);
```

对于 args 数组的存储，将会将每个字符串数组的起始位置指向 rline 用户输入数组的对应位置（该过程通过判定前一个字符是否是空格实现），以实现参数存储

```
        default:
            if (prev == ' ') args[argc++] = rptr;
            prev = *rptr;
            break;
```



当遇到空格、换行、&符号时，都会被替换为 \0，从而限制每一个字符串数组的终止位置

```
        case '&':
            should_wait = 0;
        case '\n':
            flag = 1;
        case ' ':
            prev = *rptr;
            *rptr = '\0';
            break;
```

最后一个参数根据规定，必须被赋值为空指针

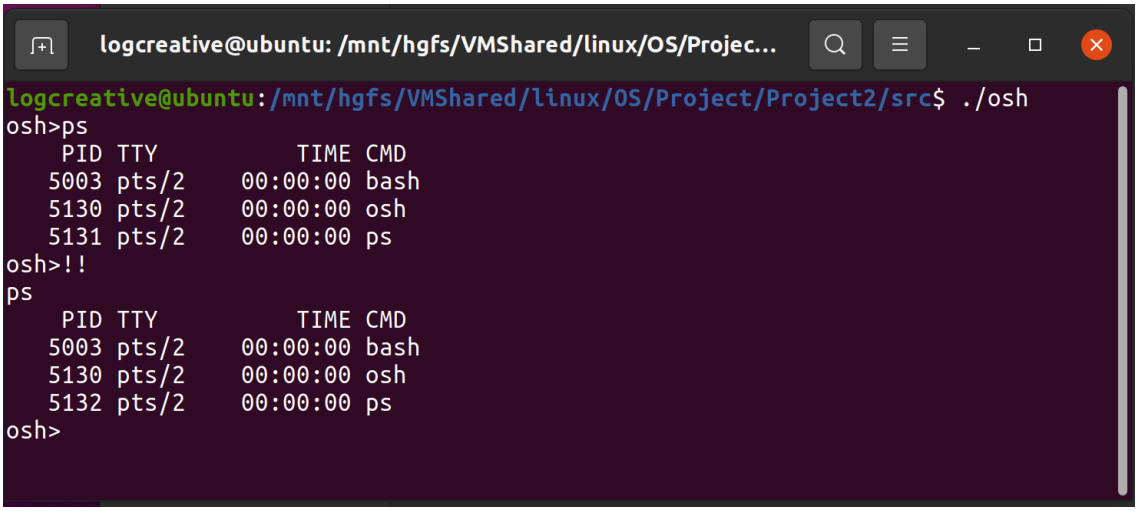
```
        args[argc] = NULL;
```

退出的判定是通过 strcmp 函数实现的

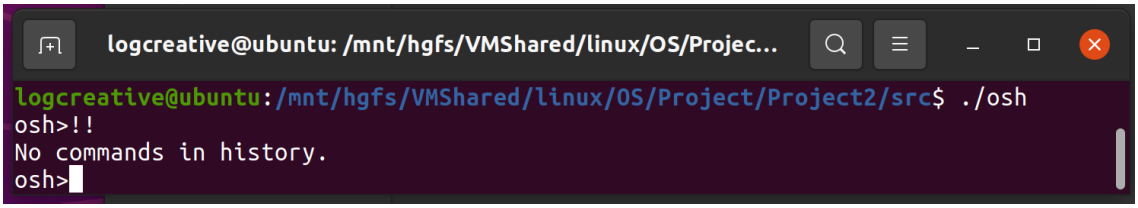
```
        should_run = strcmp(args[0],"exit");
```

2. 历史记录存储

当输入 !! 时将会回显并运行上一条指令



当没有上一条指令时，将会返回提示



这一部分需要将上一条指令存储在 buf 中，开始时置为空

```
char buf[MAX_LINE] = "\0";
```

当连续输入两个 ！时，将会首先检查 buf[0] 是否是空字符，否则将通过 memcpy 将缓存复制到当前的阅读行数组中，注意这里不能使用 strcpy，因为其的复制到 \0 就会立刻停止，导致后面的部分没有被有效复制。循环条件也增加了 flag 判定。

```
for(char* rptr = rline; rptr < rline + MAX_LINE && !flag; ++rptr){
    //...
    case '!':
        if (prev=='!' && argc == 0){
            if(!buf[0]){
                fprintf(stdout, "%s\n", "No commands in history.");
                flag = -1;
            } else {
                memcpy(rline, buf, MAX_LINE*sizeof(char));
                fprintf(stdout, "%s\n", rline);
                flag = 2;
            }
        }
        prev = *rptr;
        break;
    //...
```

如果是正常指令，就会将当前指令复制到缓冲区中去。如果不是正常指令，将不会执行任何指令。

```
switch(flag){
    case 1:
```

```
                args[argc] = NULL;
            case 2:
                if (!args[0]) break;
                memcpy(buf, rline, MAX_LINE*sizeof(char));
                should_run = strcmp(args[0],"exit");

                // run the fork part
        }
```

3. 重定向输入输出

4. 使用下面的 Makefile 编译：

Listing 1: src/Makefile

```
all:
    gcc -g osh.c -o osh
clean:
    rm *.o -f
```