

函钩 2025.7.8

在计算机科学中，Shell 是指为用户提供操作界面的软件，即为 `command interpreter`（命令解析器），类似于 `win` 系统中的 `cmd` 或者 `powershell`。在本实验中，我们需要补全实验提供的函数来实现一个简单的 `shell` 模拟。Shell 重复地打印提示符，等待 `stdin` 上输入的命令行，然后通过函数对命令行进行解析。

先来看命令行的结构，命令行通常由几个单词和空格组成。根据 `README.md` 中的描述：

命令行是一系列由空白分隔的 ASCII 文本单词。命令行中的第一个单词要么是内置命令的名称，要么是可执行文件的路径名。其余的单词是命令行参数。如果第一个单词是内置命令，Shell 立即在当前进程中执行该命令。否则，该单词被假定为可执行程序的路径名。在这种情况下，Shell 创建一个子进程，然后在子进程的上下文中加载并运行程序。作为解释单个命令行的结果创建的子进程总称为作业。一般来说，一个作业可以由通过 Unix 管道连接的多个子进程组成。

对于一个需要解析命令行的程序，其 `main` 函数需要包含两个参数：`int argc`，`char *argv[]`，其中 `argc` 为命令行中的单词数量，`argv` 的每个指针指向的是每个单词。使用 `getopt` 来解析命令行参数。Main 函数在实验开始时已是完整形式，不再赘述。

```
/bin/ls -l -d
```

```
...
```

在前台运行 `ls` 程序。shell 应确保当程序开始执行其主函数时，`argc` 和 `argv` 参数具有以下值：

```
...
```

```
int main(int argc, char *argv[])
```

```
...
```

```
* argc == 3
```

```
* argv[0] == "/bin/ls"
```

```
* argv[1] == "-l"
```

```
* argv[2] == "-d"
```

实验提供的 `tsh.c` 是一个实现 `shell` 程序的框架，并声明了基本的函数，要求我们补全以下这些函数：

```
/* Here are the functions that you will implement */
```

```
void eval(char *cmdline);
```

```
int builtin_cmd(char **argv);
```

```
void do_bgfg(char **argv);
```

```
void waitfg(pid_t pid);
```

```
void sigchld_handler(int sig);
```

```
void sigtstp_handler(int sig);
void sigint_handler(int sig);
```

实验提供了 16 个测试案例，我们可以根据测试案例的内容对函数进行补全。

首先阅读 `tsh.c` 的代码，大致了解该程序的运行原理。头文件引用部分之后是常量的定义

```
/* Misc manifest constants */
#define MAXLINE    1024 /* max line size */
#define MAXARGS    128 /* max args on a command line */
#define MAXJOBS    16  /* max jobs at any point in time */
#define MAXJID     1<<16 /* max job ID */
```

这些常量或许在后续程序中对数组等的定义起到一个最大值的作用。

```
/* Job states */
#define UNDEF 0 /* undefined */
#define FG 1   /* running in foreground */
#define BG 2   /* running in background */
#define ST 3   /* stopped */

/*
 * Jobs states: FG (foreground), BG (background), ST (stopped)
 * Job state transitions and enabling actions:
 *
 *   FG -> ST : ctrl-z
 *   ST -> FG : fg command
 *   ST -> BG : bg command
 *   BG -> FG : fg command
 *
 * At most 1 job can be in the FG state.
 */
```

此处是对几种工作状态的定义以及不同指令的对应工作状态的变化。

然后是一个重要全局变量的定义 `job_t` 类型的 `jobs`，包含了 `pid`，`jid`，`state` 和 `cmdline` 四种类型。

```
struct job_t { /* The job struct */
    pid_t pid; /* job PID */
    int jid; /* job ID [1, 2, ...] */
    int state; /* UNDEF, BG, FG, or ST */
    char cmdline[MAXLINE]; /* command line */
};
struct job_t jobs[MAXJOBS]; /* The job list */
```

然后是一些帮助我们实现 `shell` 的函数，可以通过 VScode 的“转到定义”功能来查看

函数的功能

```
/* Here are helper routines that we've provided for you */
int parseline(const char *cmdline, char **argv);
void sigquit_handler(int sig);

void clearjob(struct job_t *job);
void initjobs(struct job_t *jobs);
int maxjid(struct job_t *jobs);
int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline);
int deletejob(struct job_t *jobs, pid_t pid);
pid_t fgpid(struct job_t *jobs);
struct job_t *getjobpid(struct job_t *jobs, pid_t pid);
struct job_t *getjobjid(struct job_t *jobs, int jid);
int pid2jid(pid_t pid);
void listjobs(struct job_t *jobs);

void usage(void);
void unix_error(char *msg);
void app_error(char *msg);
typedef void handler_t(int);
handler_t *Signal(int signum, handler_t *handler);
```

在七个需要补全的函数中，三个信号处理函数是比较简单的，优先处理这个。

“一个信号就是一条小消息，它通知进程系统中发生了一个某种类型的事件。”在 CSAPP 课本中如此说道。在本实验中，主要涉及到以下几个信号：SIGINT(ctrl-c), SIGSTP(ctrl-z), SIGCHLD, SIGQUIT。

1.sigint_handler

sigint_handler 函数的作用为当用户在键盘上按下 ctrl 和 c 时，内核向 shell 发送 SIGINT 信号，函数捕获该信号并将其发送给前台作业。在系统内核中，每个进程都有自己的 pid，阅读 tsh.c 中已有的函数，发现函数 fgpid 能够获取前台作业的 pid。

```
/* fgpid - Return PID of current foreground job, 0 if no such job */
pid_t fgpid(struct job_t *jobs)
{
    int i;

    for (i = 0; i < MAXJOBS; i++)
        if (jobs[i].state == FG)
            return jobs[i].pid;
```

```
    return 0;
}
```

当前台没作业的时候，fgpid 返回 0，所以在 sigint_handler 中，只需要考虑 pid>0 的情况

```
void sigint_handler(int sig){
    pid_t pid = fgpid(jobs);

    if(pid>0){
        kill(-pid, sig); // 负数表示将信号发送给进程组，而不是单个的进程
        //进程组的 ID 是 pid 的绝对值
    }
    return;
}
```

在 kill 函数中使用 -pid，是为了向 pid 对应进程以及所有产生的子进程都发送信号。kill 函数的作用并非“杀死”进程，而是向进程发送信号。kill 函数有两个参数，第一个是 pid_t 类型的 pid，第二个是 int 类型的 sig。当 pid>0，发送信号给指定的 pid 进程，当 pid=0，发送信号给当前进程组的所有进程，当 pid=-1，发送信号给所有有权限发送的进程，当 pid<-1，发送信号给进程组 ID 为 |pid| 的所有进程（进程组的 ID 是 pid 的绝对值）。第二个参数 sig 则是信号的类型，在上面提到过。返回值，若发送成功则返回 0，若失败则返回 -1 并设置 errno（即为错误码）。

2.sigstsp_handler

由于 sigint_handler 和 sigstsp_handler 都是简单的接受键盘输入并发送信号的指令，所以代码实现是完全一样的

```
void sigstsp_handler(int sig)
{
    pid_t pid = fgpid(jobs);
    if(pid>0){
        kill(-pid, sig);
    }
    return;
}
```

经过测试，如果将 main 函数中的 sigstsp_handler 换成 sigint_handler 是完全不影响运行结果的

```
/* These are the ones you will need to implement */
Signal(SIGINT, sigint_handler); /* ctrl-c */
Signal(SIGTSTP, sigint_handler); /* ctrl-z */
Signal(SIGCHLD, sigchld_handler); /* Terminated or stopped child */
```

3.sigchld_handler

该函数的核心功能为处理 SIGCHLD 信号，主要负责 1.回收子进程 2.状态同步：根据子进程状态变化（终止/暂停/恢复）更新作业列表 3. 资源清理：删除已终止作业的条目。代码实现如下：

```
void sigchld_handler(int sig) {
    int oldErrno = errno;
    int status;
    pid_t pid;
    // 恢复while循环以处理所有终止的子进程
    while ((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0)
    {
        if (WIFSTOPPED(status))
        {
            struct job_t *job = getjobpid(jobs, pid);
            if(job){
                job->state = ST;
                int jid = pid2jid(pid);
                printf("Job [%d] (%d) Stopped by signal %d\n", jid, pid, WSTOPSIG(status));
            }
        }
        else if (WIFSIGNALED(status))
        {
            int jid = pid2jid(pid);
            printf("Job [%d] (%d) Stopped by signal %d\n", jid, pid, WTERMSIG(status));
            deletejob(jobs, pid);
        }
        else if (WIFEXITED(status))
        {
            deletejob(jobs, pid);
        }
    }
    if (errno != ECHILD && errno != EINTR && errno != 0) //竟然给我改对了
    {
        unix_error("waitpid error");
    }
    errno = oldErrno;
}
```

该函数涉及了一个重要函数 waitpid。Waitpid 的定义为

```
pid_t waitpid (pid_t pid, int *statusp, int options);
```

当一个进程终止时，它会保持在一种已终止的状态直到被它的父进程回收。当父进程回收已终止的子进程时，内核将子进程的退出状态传递给父进程，然后抛弃已终止的进程。一个进程可以调用 waitpid 函数来等待它的子进程的终止或停止。等待集合的成员是由参数 pid 决定的，若 pid>0 则等待集合是一个单独的子进程，若 pid==-1 则等待集合是父进程的所有子进程。Options 有三个常量及其组合可选，WNOHANG，WUNTRACED，WCONTINUED。WNOHANG：如果等待集合中等任何子进程都还没有终止，则立即返回 0；WUNTRACED 是挂起调用进程的执行，直到等待集合中的一个进程变成终止或已停止。

在非阻塞循环内，函数需要有三种状态处理分支：WIFSTOPPED，WIFSIGNALED，WIFEXITED，分别处理暂停，终止，正常退出三种情况的子进程。在 while 的条件

内，waitpid 函数会同时更改 status 的状态，因此在下面可以通过 status 来判断情况。在非正常退出情况下，使用实验已给出的辅助函数 pid2jid，将 pid 转换为 jid (job id)。暂停情况下将结构体 job->state 设置为 ST(暂停)，终止和退出情况下需要删除僵死进程。

对于错误处理，首先要使用 int oldErrno = errno 来保存原始的错误码，

在过程中可能会修改 errno 值，最后忽略 errno==ECHLD 和 errno==EINTR 两种正常的错误情况，同时其他异常情况需要打印错误提示。

4.waitfg

```
/*
 * waitfg - Block until process pid is no longer the foreground process
 * 阻塞直到进程 pid 不再是前台进程
 */
void waitfg(pid_t pid)//use a busy loop around the sleep function
{
    struct job_t *job=getjobpid(jobs,pid);
    if(job==NULL){
        return;
    }
    while(fgpid(jobs)==pid){//spin
        sleep(1);
    }
    return;
}
```

创建一个 job_t 类型的 job 指针并用 getjobpid 函数初始化。若 job 为空指针，则直接返回。然后通过 fgpid(jobs) 获取前台中正在运行程序的 pid。在 while 循环中，通过 spin(不断旋转)的方式不停检测是否前台 jobs 的 pid 与当前 pid 相同，否则不予以返回，直到前台作业完成。从而达到等待（阻塞 shell）的效果。

例如当执行前台命令（如 ./myspin 3）时，shell 通过该函数保持等待状态，直到 3 秒后子进程退出才会显示新的提示符。

5.builtin_cmd

builtin_cmd 函数的主要功能是识别并处理 shell 的内置命令。以下是函数的代码实现：

```
int builtin_cmd(char **argv){
    if (strcmp(argv[0], "quit") == 0){
        /*strcmp 比较两个字符串相等则返回 0*/
        exit(0);
    }
}
```

```

    else if (strcmp(argv[0], "jobs") == 0){
        listjobs(jobs); /*listjobs 函数在后面已经定义了，直接用*/
        return 1;
    }
    else if (strcmp(argv[0], "&") == 0){
        return 1;
    }
    else if (strcmp(argv[0], "bg") == 0 || strcmp(argv[0], "fg") ==
0){
        do_bgfg(argv);
        return 1;
    }

    return 0; /* not a builtin command */
}

```

builtin_cmd 传入的参数为 char **argv，即指向字符数组的指针，也就是字符串组成的数组。该函数的任务是识别和处理四条内置命令：quit，&，fg，bg 和 jobs。通过 strcmp 函数来比较输入命令行的单词与指定命令是否相同。

****strcmp 的运行逻辑为：int strcmp(const char *str1, const char *str2);**

str1: 指向第一个要比较的字符串的指针。

str2: 指向第二个要比较的字符串的指针。

如果返回值小于 0，表示 str1 小于 str2。如果返回值等于 0，表示 str1 和 str2 相等。如果返回值大于 0，表示 str1 大于 str2。比较是逐个字符进行的，从每个字符串的第一个字符开始比较。比较字符的 ASCII 值大小。例如，字符 'a' 的 ASCII 值是 97，字符 'b' 的 ASCII 值是 98，所以 "apple" 小于 "banana"。

quit 的实现方式为通过调用 exit(0)直接终止 shell 程序。在前面提到的辅助函数中，包含了 listjobs 函数，通过转到定义可以看到其功能为

```

/* listjobs - Print the job list */
void listjobs(struct job_t *jobs)
{

```

也就是打印出作业的列

表，与 jobs 命令的功能相符，可以使用这个函数。&命令的含义为直接返回。bg/fg 命令的含义为作业的前后台切换：

```

*
* Jobs states: FG (foreground), BG (background), ST (stopped)
* Job state transitions and enabling actions:
*   FG -> ST : ctrl-z
*   ST -> FG : fg command
*   ST -> BG : bg command
*   BG -> FG : fg command
* At most 1 job can be in the FG state.
*/

```

实现前后台切换功能的函数需要我们自己在 do_bgfg 函数中实现。

当命令不是指定命令，则返回 0

6.do_bgfg

do_bgfg 函数的主要功能是实现 bg 和 fg 内置命令，用于控制作业在后台和前台之间的切换。当 argv 的第一个单词为 bg 或者 fg 时，程序将会调用 do_bgfg 函数。do_bgfg 的传入参数也是 char **argv，要求能够解析两种参数格式（第二个单词）：%<jid>或 pid。

```
void do_bgfg(char **argv){
    struct job_t *job = NULL; // 统一初始化
    char *tmp = NULL;
    int jid = 0;
    pid_t pid = 0;

    tmp = argv[1];
    // 如果参数不存在，输出错误信息并返回
    if (tmp == NULL){
        printf("%s command requires PID or %%jobid argument\n", argv[0]);
        return;
    }

    // 如果是工作 id
    if (tmp[0] == '%'){
        jid = atoi(&tmp[1]);
        // 获取作业
        job = getjobjid(jobs, jid);
        if (job == NULL){
            printf("%s: No such job\n", tmp);
            return; // 添加返回语句，避免后续错误
        }
        // 获取有效的作业 pid，供后续使用
        pid = job->pid;
    }
    else if (isdigit(tmp[0])){
        // 获取 pid
        pid = atoi(tmp);
        // 获取作业
        job = getjobpid(jobs, pid);
        if (job == NULL){
            printf("(%d): No such process\n", pid);
            return;
        }
    }
    else{
        printf("%s: argument must be a PID or %%jobid\n", argv[0]);
        return;
    }

    // 发送 SIGCONT 信号继续执行作业
    if (kill(-pid, SIGCONT) < 0) {
        unix_error("kill error");
        return;
    }

    if (!strcmp("fg", argv[0])){
        // 将作业状态设置为前台运行
        job->state = FG;
        // 等待前台作业完成
        waitfg(job->pid);
    }
    else {
        // 将作业状态设置为后台运行并打印信息
        job->state = BG;
        printf("[%d] (%d) %s\n", job->jid, job->pid, job->cmdline); // 添加\n 换行符
    }
}
```

首先初始化变量

```
struct job_t *job = NULL; // 统一初始化
char *tmp = NULL;
int jid = 0;
pid_t pid = 0;
```

为了方便检查命令行参数，我们可以定义一个 tmp=argv[1]，即为命令行的第二个单词。若 tmp 为 NULL，即参数不存在，则打印错误提示并返回；

如果 `tmp[0]!='%'` 则表示输入的是工作 id (工作 id 由%和数字组成, jid 是数字部分), 使用 `atoi` 函数将命令行中字符串形式的数字转成整形数字(jid), 通过 `getjobjid` 函数, 在作业表中获取指定的作业, 若作业为空, 则输出错误提示并直接返回, 否则获取有效的作业 pid 使: `pid=job->pid`;

若 `tmp` 中不含'%', 则说明命令输入的单词是 pid, 使用 `atoi` 直接获取 pid, 使用 `getjobpid` 获取对应进程, 若进程为空, 则输出错误提示, 并直接返回。

若 `tmp` 中的参数错误, 直接输出错误提示。

上方所有步骤都是为了获取进程 (pid) 和指定的作业 (job) 并做好错误处理。

由于 `do_bgfg` 的作用是将已暂停的进程进行前后台切换或者将正在后台运行的作业进行切换, 所以若原来是暂停的, 则需要先发送 `SIGCONT` 信号来继续执行作业:

```
374 // 发送SIGCONT信号继续执行作业
375 if (kill(-pid, SIGCONT) < 0) {
376     unix_error("kill error");
377     return;
378 }
```

该函数 `kill(-pid, SIGCONT)` 成功

执行返回 0, 失败时返回 -1. 若失败则执行错误处理代码并返回, 若成功则继续执行。

```
if (!strcmp("fg", argv[0])){
    // 将作业状态设置为前台运行
    job->state = FG;
    // 等待前台作业完成
    waitfg(job->pid);
}
```

`strcmp` 若两字符串相等则返回 0, 若 `argv[0]`

为 "fg" 则将状态设置为 FG, 先阻塞当前进程, 并等待直到当前的前台作业完成。如果命令为 bg (即 else 情况), 则将状态设置为 BG, 并输出作业信息, 包括作业 ID、进程 ID 和命令行。

7.eval

```
void eval(char *cmdline)
{
    char *argv[MAXARGS];
    //int to record for bg
    int bg;
    pid_t pid;
    sigset_t mask;

    // parse the line
    bg = parseline(cmdline, argv);
    //check if valid builtin_cmd
    if(!builtin_cmd(argv)) {

        // blocking first
        sigemptyset(&mask);
        sigaddset(&mask, SIGCHLD);
        sigprocmask(SIG_BLOCK, &mask, NULL);
        // forking
        if((pid = fork()) < 0){
            unix_error("forking error");
        }
        else if(pid == 0) { // child
            sigprocmask(SIG_UNBLOCK, &mask, NULL);
```

```

    setpgid(0, 0);
    //check if command is there
    if (execvp(argv[0], argv) < 0) {
        printf("%s: Command not found\n", argv[0]);
        exit(1);
    }
}
else { // parent
    if (!bg) {
        addjob(jobs, pid, FG, cmdline);
    } else {
        addjob(jobs, pid, BG, cmdline);
    }
    sigprocmask(SIG_UNBLOCK, &mask, NULL);

    // if bg/fg
    if (!bg) {
        // wait for fg
        waitfg(pid);
    }
    else{
        //print for bg
        // 原 cmdline 包含换行符，此处去除换行符
        char fixed_cmdline[MAXLINE];
        strcpy(fixed_cmdline, cmdline);
        fixed_cmdline[strcspn(fixed_cmdline, "\n")] = 0;
        printf("[%d] (%d) %s\n", pid2jid(pid), pid, fixed_cmdline);
    }
}
}
}
}

```

`eval` 函数是迷你 shell 程序中最重要函数，负责评估用户刚刚输入的命令行，如果用户请求的是内置命令（`quit`、`jobs`、`bg` 或 `fg`），则立即执行该命令。否则，创建一个子进程，并在子进程的上下文中运行该作业。如果作业在前台运行，则等待其终止后再返回。注意每个子进程必须有一个唯一的进程组 ID，这样当我们在键盘上输入 `ctrl-c`（`ctrl-z`）时，后台子进程不会从内核接收 `SIGINT`（`SIGTSTP`）信号。

```

char *argv[MAXARGS];
//int to record for bg
int bg;
pid_t pid;
sigset_t mask;

```

统一初始化，声明 `*argv`，`bg`，`pid` 和 `mask`。`Sigset_t mask` 用于存储一组信号，通过 `sigemptyset` 初始化空信号集，`sigaddset` 将 `SIGCHLD` 信号添加到集合中。`bg` 用于标识用户输入的命令是否需要在后台运行。`bg` 的值由 `parseline` 函数返回。

`parseline` - 解析命令行并构建 `argv` 数组。用单引号括起来的字符被视为一个单独的参数。如果用户请求一个后台作业，则返回 `true`；如果用户请求一个前台作业，则返回 `false`。（来自源代码注释的翻译）

然后处理内置命令：函数要求若用户请求为内置命令则直接在 `builtin_cmd` 中执行，否则 `builtin_cmd` 不执行并返回 0，运行后续代码：创建子进程并在子进程的上下文中运行作业。

接下来需要阻塞 `SIGCHLD` 信号，这样是为了防止在创建子进程期间父进程收到子进程

```
// 先阻塞 SIGCHLD 信号
sigemptyset(&mask);
sigaddset(&mask, SIGCHLD);
sigprocmask(SIG_BLOCK, &mask, NULL);
```

终止的信号而提前处理。`sigemptyset` 是首先清空信号集 `mask`, `sigaddset` 是将 `SIGCHLD` 添加到信号集中, 然后 `sigprocmask` 用于阻塞 `SIGCHLD` 信号。

调用 `fork` 函数创建子进程, 如果 `fork` 失败 (`fork()` 返回 -1), 调用 `unix_error` 函数输出错误信息并退出; 若 `fork` 返回 0 (当前代码正在子进程里执行), 则 `SIG_UNBLOCK` 解除 `SIGCHLD` 信号的阻塞,

```
else if(pid == 0) { // child
    sigprocmask(SIG_UNBLOCK, &mask, NULL);
    setpgid(0, 0);
    //check if command is there
    if (execvp(argv[0], argv) < 0) {
        printf("%s: Command not found\n", argv[0]);
        exit(1);
    }
}
```

创建子进程。创建子进程组使用了

`setpgid` (把进程编进进程组里)。`setpgid(0,0)` 意为让子进程脱离父进程原来的进程组, 创建一个新的进程组, 并将子进程设置为一个新进程组的组长, 确保子进程有一个唯一的进程组 ID, 避免在键盘输入 `ctrl+c` 或者 `ctrl+z` 时后台子进程接受到 `SIGINT` 或者 `SIGTSTP` 信号。`execvp(argv[0], argv)`; (用于以指定的参数列表启动一个新程序) 尝试执行用户输入的命令。如果执行失败, 输出错误信息并退出子进程。第一个参数 `argv[0]` 为可执行文件名或其路径, 第二个参数 `argv` 是以 `NULL` 结尾的字符串数组, 表示新程序的命令行参数。若运行失败则输出错误指令, 退出子进程。

若 `fork` 返回值大于 0, 则代码正在父进程中执行, 需要进行父进程处理:

```
else { // parent
    if (!bg) {
        addjob(jobs, pid, FG, cmdline);
    } else {
        addjob(jobs, pid, BG, cmdline);
    }
    sigprocmask(SIG_UNBLOCK, &mask, NULL);

    // if bg/fg
    if (!bg) {
        // wait for fg
        waitfg(pid);
    }
    else{
        //print for bg
        // 原 cmdline 包含换行符, 此处去除换行符
        char fixed_cmdline[MAXLINE];
        strcpy(fixed_cmdline, cmdline);
        fixed_cmdline[strcspn(fixed_cmdline, "\n")] = 0;
        printf("[%d] (%d) %s\n", pid2jid(pid), pid, fixed_cmdline);
    }
}
```

根据 `bg` 的值 (后合作业为 1 前合作业为 0) 调用 `addjob` 函数将子进程添加到作业列表中, 并标记为前台或后合作业。解除阻塞信号, 根据情况, 等待前合作业完成, 若为后合作业则去除原命令行 (用户输入) 中的换行符, 并输出作业信息。

`eval` 函数的要点在于，创建子进程前后对 `SIGCHLD` 信号的阻塞和解除阻塞。

在多线程编程中，当多个进程或线程同时访问共享资源时，可能会出现竞态条件。在这个 shell 程序里，共享资源就是作业列表 `jobs`。当父进程创建子进程后，需要将子进程添加到作业列表中。而 `SIGCHLD` 信号是在子进程终止或停止时由内核发送给父进程的。如果在父进程创建子进程后，还没来得及将子进程添加到作业列表中时，子进程就已经终止并发送了 `SIGCHLD` 信号，那么 `sigchld_handler` 信号处理函数会尝试从作业列表中删除这个还未添加进去的子进程，这就会导致作业列表的不一致，引发竞态条件。

综上所述，`eval` 函数以解析命令行，判断是否为内置命令，阻塞信号，创建子进程，判断进程情况，解除阻塞信号，处理作业等步骤，完成了对输入命令行的评估。

至此实验要求的七个函数都已经完成，接下来需要进行案例测试。

接下来看测试案例，测试案例涉及了四类命令：

内置命令 `trace02/trace05/trace09/trace10`

信号处理 `trace06/trace08/trace11/trace16`

后台作业 `trace04/trace05/trace07`

错误处理 `trace14/trace15`

没有包含在内的 `trace01` 为测试 EOF 文件结束符的案例，`trace11,12,13` 为作业的状态查看，由命令 `/bin/ps a` 触发。

以下为测试案例输出：

```
root@localhost:/home/shelllab# make test01&&make rtest01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
./sdriver.pl -t trace01.txt -s ./tshref -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
test01
```

```
root@localhost:/home/shelllab# make test02&&make rtest02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
./sdriver.pl -t trace02.txt -s ./tshref -a "-p"
#
# trace02.txt - Process builtin quit command.
#
test02
```

```
root@localhost:/home/shelllab# make test03 && make rtest03
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
./sdriver.pl -t trace03.txt -s ./tshref -a "-p"
#
# trace03.txt - Run a foreground job.
#
test03 tsh> quit
```

test04

```
root@localhost:/home/shelllab# make test04 && make rtest04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (228962) ./myspin 1 &
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (228968) ./myspin 1 &
```

test05

```
root@localhost:/home/shelllab# make test05 && make rtest05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (228974) ./myspin 2 &
tsh> ./myspin 3 &
[2] (228976) ./myspin 3 &
tsh> jobs
[1] (228974) Running ./myspin 2 &
[2] (228976) Running ./myspin 3 &
./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (228983) ./myspin 2 &
tsh> ./myspin 3 &
[2] (228985) ./myspin 3 &
tsh> jobs
[1] (228983) Running ./myspin 2 &
[2] (228985) Running ./myspin 3 &
```

test06

```
root@localhost:/home/shelllab# make test06 && make rtest06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (228992) terminated by signal 2
./sdriver.pl -t trace06.txt -s ./tshref -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (228998) terminated by signal 2
```

test07

```
root@localhost:/home/shelllab# make test07 && make rtest07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (229004) ./myspin 4 &
tsh> ./myspin 5
Job [2] (229006) terminated by signal 2
tsh> jobs
[1] (229004) Running ./myspin 4 &
./sdriver.pl -t trace07.txt -s ./tshref -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (229013) ./myspin 4 &
tsh> ./myspin 5
Job [2] (229015) terminated by signal 2
tsh> jobs
[1] (229013) Running ./myspin 4 &
```

```

root@localhost:/home/shelllab# make test08 && make rtest08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (252088) ./myspin 4 &
tsh> ./myspin 5
Job [2] (252090) stopped by signal 20
tsh> jobs
[1] (252088) Running ./myspin 4 &
[2] (252090) Stopped ./myspin 5
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (252097) ./myspin 4 &
tsh> ./myspin 5
Job [2] (252099) stopped by signal 20
tsh> jobs
[1] (252097) Running ./myspin 4 &
[2] (252099) Stopped ./myspin 5

```

test08

```

./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (252132) ./myspin 4 &
tsh> ./myspin 5
Job [2] (252134) stopped by signal 20
tsh> jobs
[1] (252132) Running ./myspin 4 &
[2] (252134) Stopped ./myspin 5
tsh> bg %2
[2] (252134) ./myspin 5
tsh> jobs
[1] (252132) Running ./myspin 4 &
[2] (252134) Running ./myspin 5

./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (237845) ./myspin 4 &
tsh> ./myspin 5
Job [2] (237847) stopped by signal 20
tsh> jobs
[1] (237845) Running ./myspin 4 &
[2] (237847) Stopped ./myspin 5
tsh> bg %2
[2] (237847) ./myspin 5
tsh> jobs
[1] (237845) Running ./myspin 4 &
[2] (237847) Running ./myspin 5

```

test09

```

root@localhost:/home/shelllab# make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (252268) ./myspin 4 &
tsh> fg %1
Job [1] (252268) stopped by signal 20
tsh> jobs
[1] (252268) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs

root@localhost:/home/shelllab# make rtest10
./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (252282) ./myspin 4 &
tsh> fg %1
Job [1] (252282) stopped by signal 20
tsh> jobs
[1] (252282) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs

```

test10

```

root@localhost:/home/shelllab# make test11
./sdriver.pl -t trace11.txt -s ./tsh -a "-p"
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (240085) terminated by signal 2
tsh> /bin/ps a
  PID TTY          STAT       TIME COMMAND
  445 hvc0      Ss+        0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
  449 tty1      Ss+        0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
  622 pts/0      Ss         0:00 -bash
  623 pts/1      Ss         0:00 /bin/login -f
  760 pts/1      S+         0:00 -bash
 240080 pts/0      S+         0:00 make test11
 240081 pts/0      S+         0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -a "-p"
 240082 pts/0      S+         0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tsh -a -p
 240083 pts/0      S+         0:00 ./tsh -p
 240088 pts/0      R          0:00 /bin/ps a

```

test11

```

root@localhost:/home/shelllab# make rtest11
./sdriver.pl -t trace11.txt -s ./tshref -a "-p"
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (240094) terminated by signal 2
tsh> /bin/ps a
  PID TTY          STAT TIME  COMMAND
    445 hvc0      Ss+   0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
    449 tty1      Ss+   0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
    622 pts/0      Ss    0:00 -bash
    623 pts/1      Ss    0:00 /bin/login -f
    760 pts/1      S+    0:00 -bash
  240089 pts/0      S+    0:00 make rtest11
  240090 pts/0      S+    0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tshref -a "-p"
  240091 pts/0      S+    0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tshref -a -p
  240092 pts/0      S+    0:00 ./tshref -p
  240097 pts/0      R     0:00 /bin/ps a

```

test12

```

root@localhost:/home/shelllab# make test12
./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (252398) stopped by signal 20
tsh> jobs
[1] (252398) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY          STAT TIME  COMMAND
    445 hvc0      Ss+   0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
    449 tty1      Ss+   0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
    622 pts/0      Ss+   0:00 -bash
    623 pts/1      Ss    0:00 /bin/login -f
    760 pts/1      S+    0:00 -bash
  252038 pts/4      Ss    0:00 -bash
  252393 pts/4      S+    0:00 make test12
  252394 pts/4      S+    0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
  252395 pts/4      S+    0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p
  252396 pts/4      S+    0:00 ./tsh -p
  252398 pts/4      T     0:00 ./mysplit 4
  252399 pts/4      T     0:00 ./mysplit 4
  252402 pts/4      R     0:00 /bin/ps a

```

```

root@localhost:/home/shelllab# make rtest12
./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (252414) stopped by signal 20
tsh> jobs
[1] (252414) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY          STAT TIME  COMMAND
    445 hvc0      Ss+   0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
    449 tty1      Ss+   0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
    622 pts/0      Ss+   0:00 -bash
    623 pts/1      Ss    0:00 /bin/login -f
    760 pts/1      S+    0:00 -bash
  252038 pts/4      Ss    0:00 -bash
  252409 pts/4      S+    0:00 make rtest12
  252410 pts/4      S+    0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
  252411 pts/4      S+    0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tshref -a -p
  252412 pts/4      S+    0:00 ./tshref -p
  252414 pts/4      T     0:00 ./mysplit 4
  252415 pts/4      T     0:00 ./mysplit 4
  252418 pts/4      R     0:00 /bin/ps a

```

test13

```

root@localhost:/home/shelllab# make test13
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (252435) stopped by signal 20
tsh> jobs
[1] (252435) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY          STAT       TIME COMMAND
    445 hvc0      Ss+        0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
    449 tty1      Ss+        0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
    622 pts/0    Ss+        0:00 -bash
    623 pts/1     Ss         0:00 /bin/login -f
    760 pts/1     S+         0:00 -bash
  252038 pts/4     Ss         0:00 -bash
  252430 pts/4     S+         0:00 make test13
  252431 pts/4     S+         0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
  252432 pts/4     S+         0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
  252433 pts/4     S+         0:00 ./tsh -p
  252435 pts/4     T          0:00 ./mysplit 4
  252436 pts/4     T          0:00 ./mysplit 4
  252439 pts/4     R          0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY          STAT       TIME COMMAND
    445 hvc0      Ss+        0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
    449 tty1      Ss+        0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
    622 pts/0    Ss+        0:00 -bash
    623 pts/1     Ss         0:00 /bin/login -f
    760 pts/1     S+         0:00 -bash
  252038 pts/4     Ss         0:00 -bash
  252430 pts/4     S+         0:00 make test13
  252431 pts/4     S+         0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
  252432 pts/4     S+         0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
  252433 pts/4     S+         0:00 ./tsh -p
  252442 pts/4     R          0:00 /bin/ps a

```

```

root@localhost:/home/shelllab# make rtest13
./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (252448) stopped by signal 20
tsh> jobs
[1] (252448) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY          STAT       TIME COMMAND
    445 hvc0      Ss+        0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
    449 tty1      Ss+        0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
    622 pts/0    Ss+        0:00 -bash
    623 pts/1     Ss         0:00 /bin/login -f
    760 pts/1     S+         0:00 -bash
  252038 pts/4     Ss         0:00 -bash
  252443 pts/4     S+         0:00 make rtest13
  252444 pts/4     S+         0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
  252445 pts/4     S+         0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
  252446 pts/4     S+         0:00 ./tshref -p
  252448 pts/4     T          0:00 ./mysplit 4
  252449 pts/4     T          0:00 ./mysplit 4
  252452 pts/4     R          0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY          STAT       TIME COMMAND
    445 hvc0      Ss+        0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
    449 tty1      Ss+        0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
    622 pts/0    Ss+        0:00 -bash
    623 pts/1     Ss         0:00 /bin/login -f
    760 pts/1     S+         0:00 -bash
  252038 pts/4     Ss         0:00 -bash
  252443 pts/4     S+         0:00 make rtest13
  252444 pts/4     S+         0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
  252445 pts/4     S+         0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
  252446 pts/4     S+         0:00 ./tshref -p
  252455 pts/4     R          0:00 /bin/ps a

```


<pre> root@localhost:/home/shelllab# make test14 ./sdriver.pl -t trace14.txt -s ./tsh -a "-p" # # trace14.txt - Simple error handling # tsh> ./bogus ./bogus: Command not found tsh> ./myspin 4 & [1] (252463) ./myspin 4 & tsh> fg fg command requires PID or %jobid argument tsh> bg bg command requires PID or %jobid argument tsh> fg a fg: argument must be a PID or %jobid tsh> bg a bg: argument must be a PID or %jobid tsh> fg 9999999 (9999999): No such process tsh> bg 9999999 (9999999): No such process tsh> fg %2 %2: No such job tsh> fg %1 Job [1] (252463) stopped by signal 20 tsh> bg %2 %2: No such job tsh> bg %1 [1] (252463) ./myspin 4 & tsh> jobs [1] (252463) Running ./myspin 4 & </pre>	<pre> root@localhost:/home/shelllab# make rtest14 ./sdriver.pl -t trace14.txt -s ./tshref -a "-p" # # trace14.txt - Simple error handling # tsh> ./bogus ./bogus: Command not found tsh> ./myspin 4 & [1] (240183) ./myspin 4 & tsh> fg fg command requires PID or %jobid argument tsh> bg bg command requires PID or %jobid argument tsh> fg a fg: argument must be a PID or %jobid tsh> bg a bg: argument must be a PID or %jobid tsh> fg 9999999 (9999999): No such process tsh> bg 9999999 (9999999): No such process tsh> fg %2 %2: No such job tsh> fg %1 Job [1] (240183) stopped by signal 20 tsh> bg %2 %2: No such job tsh> bg %1 [1] (240183) ./myspin 4 & tsh> jobs [1] (240183) Running ./myspin 4 & </pre>
---	---

此处好像把 stopped 小写写成大写了

<pre> root@localhost:/home/shelllab# make test15 ./sdriver.pl -t trace15.txt -s ./tsh -a "-p" # # trace15.txt - Putting it all together # tsh> ./bogus ./bogus: Command not found tsh> ./myspin 10 Job [1] (252482) terminated by signal 2 tsh> ./myspin 3 & [1] (252484) ./myspin 3 & tsh> ./myspin 4 & [2] (252486) ./myspin 4 & tsh> jobs [1] (252484) Running ./myspin 3 & [2] (252486) Running ./myspin 4 & tsh> fg %1 Job [1] (252484) stopped by signal 20 tsh> jobs [1] (252484) Stopped ./myspin 3 & [2] (252486) Running ./myspin 4 & tsh> bg %3 %3: No such job tsh> bg %1 [1] (252484) ./myspin 3 & tsh> jobs [1] (252484) Running ./myspin 3 & [2] (252486) Running ./myspin 4 & tsh> fg %1 tsh> quit </pre>	<pre> root@localhost:/home/shelllab# make rtest15 ./sdriver.pl -t trace15.txt -s ./tshref -a "-p" # # trace15.txt - Putting it all together # tsh> ./bogus ./bogus: Command not found tsh> ./myspin 10 Job [1] (243074) terminated by signal 2 tsh> ./myspin 3 & [1] (243100) ./myspin 3 & tsh> ./myspin 4 & [2] (243102) ./myspin 4 & tsh> jobs [1] (243100) Running ./myspin 3 & [2] (243102) Running ./myspin 4 & tsh> fg %1 Job [1] (243100) stopped by signal 20 tsh> jobs [1] (243100) Stopped ./myspin 3 & [2] (243102) Running ./myspin 4 & tsh> bg %3 %3: No such job tsh> bg %1 [1] (243100) ./myspin 3 & tsh> jobs [1] (243100) Running ./myspin 3 & [2] (243102) Running ./myspin 4 & tsh> fg %1 tsh> quit </pre>
---	---

```

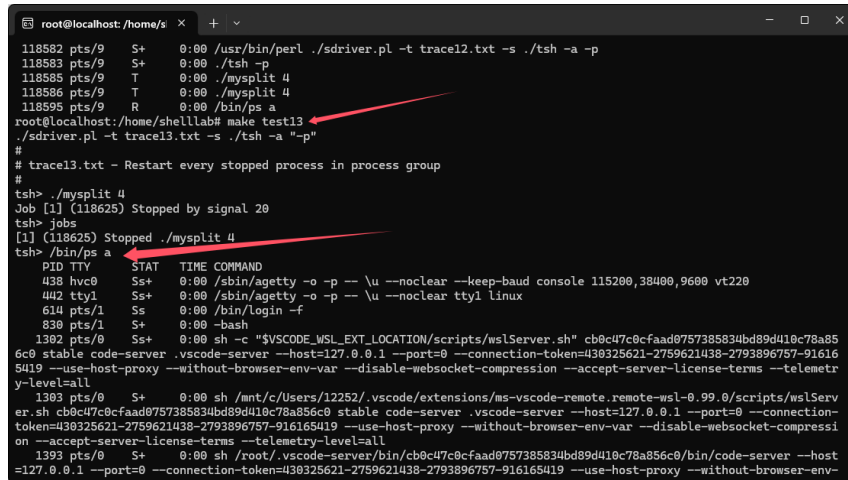
root@localhost:/home/shelllab# make test16
./sdriver.pl -t trace16.txt -s ./tsh -a "-p"
#
# trace16.txt - Tests whether the shell can handle SIGTSTP and SIGINT
# signals that come from other processes instead of the terminal.
#
tsh> ./mystop 2
Job [1] (246249) stopped by signal 20
tsh> jobs
[1] (246249) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (246325) terminated by signal 2
root@localhost:/home/shelllab# make rtest16
./sdriver.pl -t trace16.txt -s ./tshref -a "-p"
#
# trace16.txt - Tests whether the shell can handle SIGTSTP and SIGINT
# signals that come from other processes instead of the terminal.
#
tsh> ./mystop 2
Job [1] (246514) stopped by signal 20
tsh> jobs
[1] (246514) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (246557) terminated by signal 2

```

test16

但是经过比对就会发现当 windows 系统 ubuntu 系统通过开发环境（VS Code 或者 TRAE 等）远程连接到 VScode 的时候，不论是自己编写的 tsh，还是原项目给出的参考程序 tshref，使用命令/bin/ps a 输出的内容会显示远程连接插件服务的进程（看起来很吓人，但其实就是一些进程，是正常的）。当关闭了远程连接时，输出内容则变少。下图为示例：

未关闭开发环境远程连接时：



```

root@localhost:/home/si # ps a
118582 pts/9 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p
118583 pts/9 S+ 0:00 ./tsh -p
118585 pts/9 T 0:00 ./mysplit 4
118586 pts/9 T 0:00 ./mysplit 4
118595 pts/9 R 0:00 /bin/ps a
root@localhost:/home/shelllab# make test13
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (118625) Stopped by signal 20
tsh> jobs
[1] (118625) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY          STAT TIME COMMAND
 438 hvc0      Ss+  0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
 442 tty1      Ss+  0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
 614 pts/1      Ss  0:00 /bin/login -f
 830 pts/1      S+   0:00 -bash
1382 pts/0      Ss+  0:00 sh -c "$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh" cb0c47c8cfaad0757385834bd89d410c78a856c0
6c0 stable code-server .vscode-server --host=127.0.0.1 --port=0 --connection-token=430325621-2759621438-2793896757-916165419 --use-host-proxy --without-browser-env-var --disable-websocket-compression --accept-server-license-terms --telemetry-level=all
1393 pts/0      S+   0:00 sh /mnt/c/Users/12252/.vscode/extensions/ms-vscode-remote.remote-wsl-0.99.0/scripts/wslServer.sh cb0c47c8cfaad0757385834bd89d410c78a856c0 stable code-server .vscode-server --host=127.0.0.1 --port=0 --connection-token=430325621-2759621438-2793896757-916165419 --use-host-proxy --without-browser-env-var --disable-websocket-compression --accept-server-license-terms --telemetry-level=all
1393 pts/0      S+   0:00 sh /root/.vscode-server/bin/cb0c47c8cfaad0757385834bd89d410c78a856c0/bin/code-server --host=127.0.0.1 --port=0 --connection-token=430325621-2759621438-2793896757-916165419 --use-host-proxy --without-browser-env-

```

```
root@localhost:/home/s x + v
root@localhost:/home/shelllab# make rtest13
./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (120957) stopped by signal 20
tsh> jobs
[1] (120957) Stopped ./mysplit 4
tsh> /bin/ps a
PID TTY STAT TIME COMMAND
438 hvc0 Ss+ 0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
442 tty1 Ss+ 0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
614 pts/1 Ss 0:00 /bin/login -f
830 pts/1 S+ 0:00 -bash
1302 pts/0 Ss+ 0:00 sh -c "$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh" cb0c47c0cfad0757385834bd89d410c78a85
6c0 stable code-server .vscode-server --host=127.0.0.1 --port=0 --connection-token=438325621-2759621438-2793896757-91616
5419 --use-host-proxy --without-browser-env-var --disable-websocket-compression --accept-server-license-terms --telemetr
y-level=all
1303 pts/0 S+ 0:00 sh /mnt/c/Users/12252/.vscode/extensions/ms-vscode-remote.remote-wsl-0.99.0/scripts/wslServ
er.sh cb0c47c0cfad0757385834bd89d410c78a856c0 stable code-server .vscode-server --host=127.0.0.1 --port=0 --connection-
token=438325621-2759621438-2793896757-916165419 --use-host-proxy --without-browser-env-var --disable-websocket-compressi
on --accept-server-license-terms --telemetry-level=all
1393 pts/0 S+ 0:00 sh /root/.vscode-server/bin/cb0c47c0cfad0757385834bd89d410c78a856c0/bin/code-server --host
=127.0.0.1 --port=0 --connection-token=438325621-2759621438-2793896757-916165419 --use-host-proxy --without-browser-env-
var --disable-websocket-compression --accept-server-license-terms --telemetry-level=all
1397 pts/0 Sl+ 0:00 /root/.vscode-server/bin/cb0c47c0cfad0757385834bd89d410c78a856c0/node /root/.vscode-server
/bin/cb0c47c0cfad0757385834bd89d410c78a856c0/out/server-main.js --host=127.0.0.1 --port=0 --connection-token=438325621-
2759621438-2793896757-916165419 --use-host-proxy --without-browser-env-var --disable-websocket-compression --accept-serv
er-license-terms --telemetry-level=all
```

```
root@localhost:/home/s x + v
root@localhost:/home/shelllab# make test13
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (121475) Stopped by signal 20
tsh> jobs
[1] (121475) Stopped ./mysplit 4
tsh> /bin/ps a
PID TTY STAT TIME COMMAND
438 hvc0 Ss+ 0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
442 tty1 Ss+ 0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
614 pts/1 Ss 0:00 /bin/login -f
830 pts/1 S+ 0:00 -bash
117402 pts/9 Ss 0:00 -bash
121470 pts/9 S+ 0:00 make test13
121471 pts/9 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
121472 pts/9 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
121473 pts/9 S+ 0:00 ./tsh -p
121475 pts/9 T 0:00 ./mysplit 4
121476 pts/9 T 0:00 ./mysplit 4
121479 pts/9 R 0:00 /bin/ps a
tsh> fg %1
```

完全正常

关闭了远程连接后：

```
root@localhost:/home/s x + v
121470 pts/9 S+ 0:00 make test13
121471 pts/9 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
121472 pts/9 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
121473 pts/9 S+ 0:00 ./tsh -p
121482 pts/9 R 0:00 /bin/ps a
root@localhost:/home/shelllab# make rtest13
./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (121495) stopped by signal 20
tsh> jobs
[1] (121495) Stopped ./mysplit 4
tsh> /bin/ps a
PID TTY STAT TIME COMMAND
438 hvc0 Ss+ 0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
442 tty1 Ss+ 0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
614 pts/1 Ss 0:00 /bin/login -f
830 pts/1 S+ 0:00 -bash
117492 pts/9 Ss 0:00 -bash
121490 pts/9 S+ 0:00 make rtest13
121491 pts/9 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
121492 pts/9 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
121493 pts/9 S+ 0:00 ./tshref -p
121495 pts/9 T 0:00 ./mysplit 4
121496 pts/9 T 0:00 ./mysplit 4
121499 pts/9 R 0:00 /bin/ps a
tsh> fg %1
```

ref程序也完全正常

实验心得：

通过仔细研读 CSAPP 课本的第 8 章以及参考多方网络资料，我认真完成了该实验。在 shelllab 中，我能够充分学习到系统中异常控制流的运作原理，例如进程的上下文切换，进程 id，父进程和子进程的创建，对子进程的回收，以及不同信号的发送，接收，阻塞和解除阻塞以及信号的用途。受益匪浅。

后记：做完实验之后没仔细看参照输出就上传了测试平台，结果就得了 40 分，又费了老大功夫重新排查了一遍，平行眼都用上了，结果才发现我把 `stopped by signal %d\n` 的 `stopped` 写成大写 `S` 了。后来统计了一下，`Stopped` 出错的刚好占了 40 分（笑哭）