lab4

实验名称

Writing Your Own Shell（构建一个简单的类Unix/Linux Shell）

实验目的

本实验旨在帮助我们深入理解进程控制和信号处理的概念。

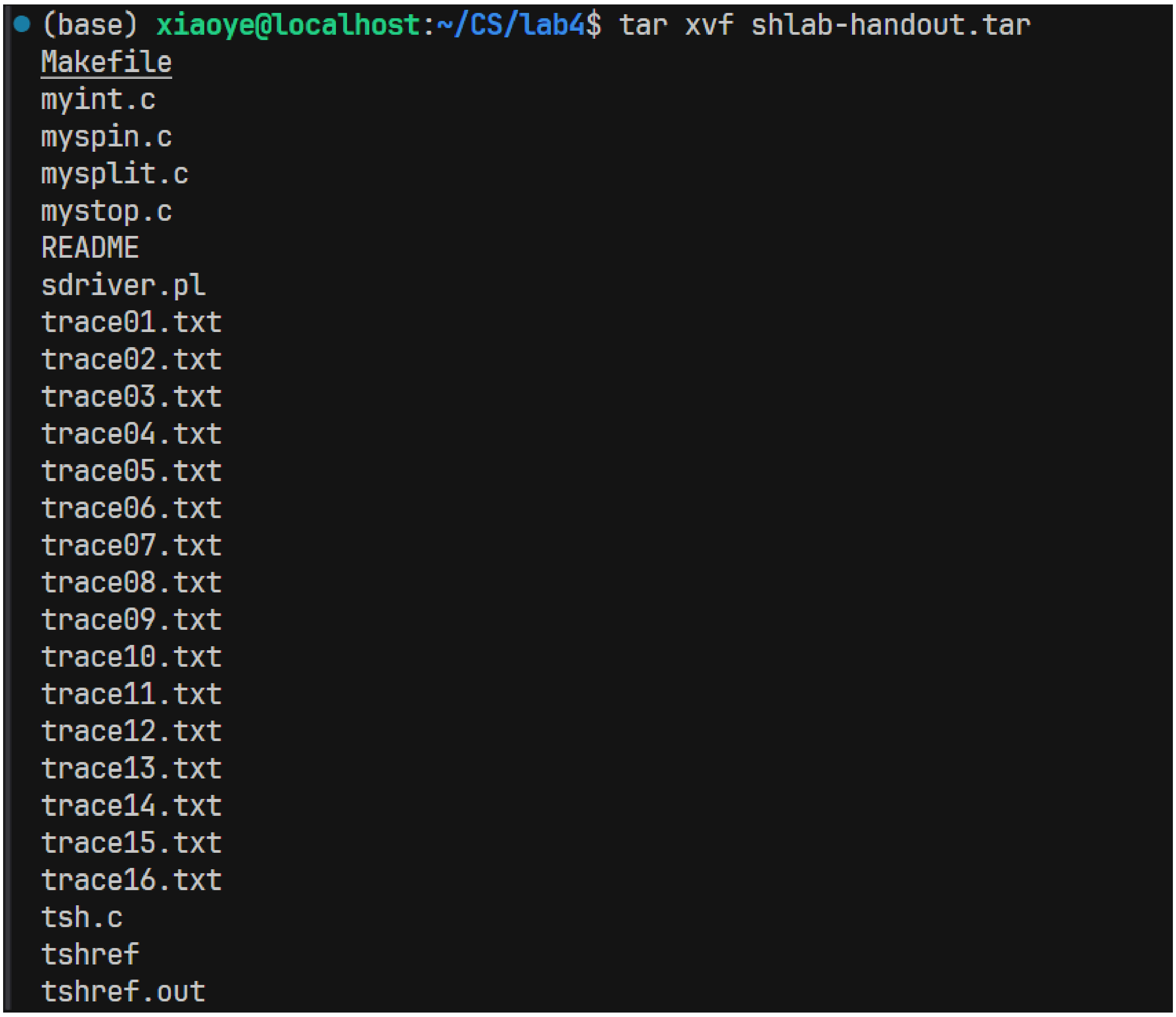
实验任务

实验准备

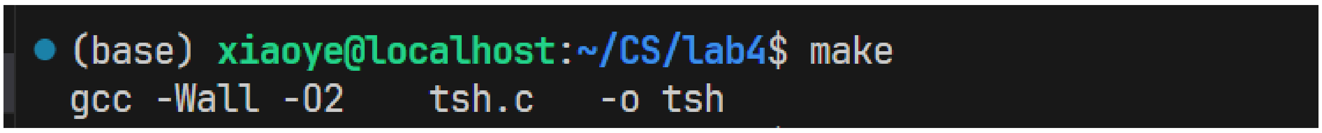
先将文件`shlab-handout.tar`复制到受保护的工作目录，之后：

```
tar xvf shlab-handout.tar //执行解压命令
make                      //编译测试程序
//在`tsh.c`文件头部注释中添加团队成员姓名和Andrew ID
```

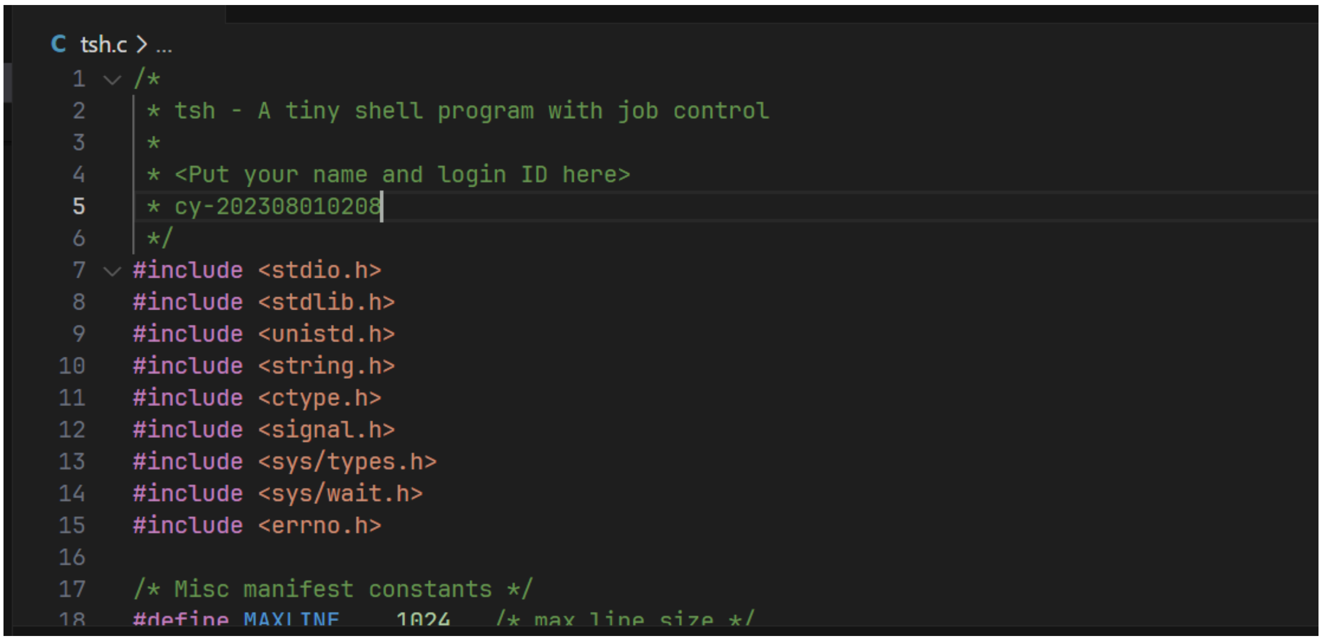
结果如下：



```
(base) xiaoye@localhost:~/CS/lab4$ tar xvf shlab-handout.tar
Makefile
myint.c
myspin.c
mysplit.c
mystop.c
README
sdriver.pl
trace01.txt
trace02.txt
trace03.txt
trace04.txt
trace05.txt
trace06.txt
trace07.txt
trace08.txt
trace09.txt
trace10.txt
trace11.txt
trace12.txt
trace13.txt
trace14.txt
trace15.txt
trace16.txt
tsh.c
tshref
tshref.out
```



```
(base) xiaoye@localhost:~/CS/lab4$ make
gcc -Wall -O2    tsh.c    -o tsh
```



tsh.c > ...

```
/*
 * tsh - A tiny shell program with job control
 *
 * <Put your name and login ID here>
 * cy-202308010208
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

/* Misc manifest constants */
#define MAXLINE    1024   /* max line size */
```

函数实现

代码框架

已提供的 `tsh.c` （微型Shell）包含基础框架结构。待完成的函数包括：

函数	功能描述
`eval`	主命令解析执行逻辑
`builtin_cmd`	处理内置命令(quit/fg/bg/jobs)
`do_bgfg`	实现bg和fg命令功能
`waitfg`	等待前台作业完成
`sigchld_handler`	SIGCHLD信号处理程序
`sigint_handler`	SIGINT(ctrl-c)信号处理程序
`sigtstp_handler`	SIGTSTP(ctrl-z)信号处理程序

修改代码后使用 `make` 重新编译，执行 `./tsh` 启动Shell。

全局变量等与main函数

这是tsh.c已经提供的部分。
我们对其进行一个解释。

头文件

```
// [字符处理] 提供字符分类及大小写转换函数（如isalpha, tolower等）

// [错误处理] 定义错误码errno及错误处理宏（如EDOM, ERANGE）
#include <errno.h>

// [信号处理] 包含信号处理函数和宏（如signal, SIGINT等）
#include <signal.h>

// [标准I/O] 输入输出核心库（printf/scanf/文件操作）
#include <stdio.h>

// [通用工具] 内存管理/随机数/程序终止等（malloc/exit/atoi）
#include <stdlib.h>

// [字符串操作] 字符串处理函数（strlen/strcpy/strcmp等）
#include <string.h>
```

```
// [系统类型] 定义基本系统数据类型（pid_t, size_t等）[^sys]
#include <sys/types.h>

// [进程控制] 包含waitpid等进程等待相关函数声明[^sys]
#include <sys/wait.h>

// [POSIX接口] 提供POSIX操作系统API（fork/exec/pipe等）[^sys]
#include <unistd.h>
```

宏定义

```
/* 杂项常量定义 */
#define MAXLINE 1024   /* 最大行长度 */
#define MAXARGS 128    /* 命令行参数最大数量 */
#define MAXJOBS 16     /* 同时存在的最大作业数 */
#define MAXJID 1 << 16 /* 最大作业ID值 */

/* 作业状态定义 */
#define UNDEF 0 /* 未定义状态 */
#define FG 1    /* 前台运行状态 */
#define BG 2    /* 后台运行状态 */
#define ST 3    /* 已停止状态 */

/*
 * 作业状态说明:
 * FG（前台运行）、BG（后台运行）、ST（已停止）
 *
 * 作业状态转换规则及触发操作:
 *     FG -> ST  : 按Ctrl+z
 *     ST -> FG  : 执行fg命令
 *     ST -> BG  : 执行bg命令
 *     BG -> FG  : 执行fg命令
 * 注意：最多只能有1个作业处于FG状态
 */

/* 全局变量 */
extern char **environ;   /* 定义在C标准库中（环境变量） */
char prompt[] = "tsh> "; /* 命令行提示符（请勿修改） */
int verbose = 0;         /* 详细模式开关（1为启用） */
int nextjid = 1;         /* 下一个待分配的作业ID */
char sbuf[MAXLINE];      /* 用于格式化sprintf消息的缓冲区 */

/* 作业结构体定义 */
```

```
struct job_t {
    pid_t pid;              /* 进程ID */
    int jid;                /* 作业ID [1,2,...] */
    int state;              /* 状态：UNDEF/BG/FG/ST */
    char cmdline[MAXLINE]; /* 原始命令行内容 */
};
struct job_t jobs[MAXJOBS]; /* 作业列表容器 */
/* 全局变量定义结束 */
```

函数声明

```
/* 用户需实现的函数原型 */

/* 核心命令解析执行函数：处理输入的命令字符串 */
void eval(char *cmdline);

/* 内置命令判断函数：检查是否为shell内置命令(返回1)或外部程序(返回0) */
int builtin_cmd(char **argv);

/* 前后台切换函数：处理fg/bg命令修改作业状态 */
void do_bgfg(char **argv);

/* 前台作业等待函数：阻塞直到前台进程完成 */
void waitfg(pid_t pid);

/* 信号处理函数 */

/* SIGCHLD处理器：处理子进程终止/停止状态变化 */
void sigchld_handler(int sig);
/* SIGTSTP处理器：处理Ctrl+Z停止信号 */
void sigtstp_handler(int sig);
/* SIGINT处理器：处理Ctrl+C中断信号 */
void sigint_handler(int sig);

/* 提供的辅助函数原型 */

/* 命令行解析器：拆分命令行参数到argv数组，返回是否为后台作业 */
int parseline(const char *cmdline, char **argv);

/* SIGQUIT处理器：处理退出信号（开发者调试用） */
void sigquit_handler(int sig);

/* 作业管理函数 */
/* 作业结构体清零：初始化作业结构体成员 */
void clearjob(struct job_t *job);

/* 作业列表初始化：重置所有作业条目为未定义状态 */
void initjobs(struct job_t *jobs);

/* 获取最大作业ID：遍历作业列表查找当前最大JID */
int maxjid(struct job_t *jobs);

/* 添加新作业：将进程PID加入作业列表，记录状态和命令行 */
int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline);

/* 删除作业：通过PID从作业列表中移除指定作业 */
int deletejob(struct job_t *jobs, pid_t pid);

/* 获取前台进程PID：查找当前前台作业的进程ID */
pid_t fgpid(struct job_t *jobs);

/* 作业查询函数 */

/* 通过PID获取作业：在作业列表中查找指定进程ID的作业 */
struct job_t *getjobpid(struct job_t *jobs, pid_t pid);

/* 通过JID获取作业：在作业列表中查找指定作业ID的作业 */
struct job_t *getjobjid(struct job_t *jobs, int jid);
/* PID转JID：将进程ID转换为对应的作业ID */
int pid2jid(pid_t pid);

/* 列出所有作业：打印当前作业列表的状态信息 */
void listjobs(struct job_t *jobs);

/* 工具函数 */

/* 用法说明：打印shell的使用帮助信息 */
void usage(void);

/* 系统错误处理：报告Unix API调用错误并终止 */
void unix_error(char *msg);

/* 应用错误处理：报告程序逻辑错误并终止 */
void app_error(char *msg);
```

```
/* 信号处理类型定义：用于声明信号处理器函数签名 */
typedef void handler_t(int);

/* 安全信号处理封装：包装signal()系统调用，确保可靠安装处理器 */
handler_t *Signal(int signum, handler_t *handler);
```

> [补充说明]
> 前台进程（Foreground Process）是直接与用户交互的进程，独占终端输入/输出设备，需用户主动操作或等待其完成才能执行其他命令。
>
> - 可通过 `Ctrl+C` 发送 `SIGINT` 终止进程，或 `Ctrl+Z` 发送 `SIGTSTP` 暂停进程。
> - 启动：默认执行方式（如 `./tsh`）;终止：自然结束、用户主动终止或异常退出。
>   后台进程（Background Process）独立于终端运行（一个终端可同时运行多个后台进程），不占用用户输入，适合执行无需即时交互的任务。
> - 启动：命令后加 `&` （如 `./tsh &`），或通过 `Ctrl+Z` + `bg` 将暂停的前台进程转后台。
> - `Ctrl+C` 不影响后台进程，需通过 `kill` 命令或信号终止。
>   作业（Job）是用户提交给操作系统的一组关联任务，可包含单个或多个进程，是资源调度的基本单元。

```
/* 主函数：程序入口点，处理命令行参数和主逻辑 */
int main(int argc, char **argv)
{
    char c;                         // 存储命令行选项字符
    char cmdline[MAXLINE];          // 存储用户输入的命令行（最大长度MAXLINE）
    int emit_prompt = 1;            // 是否显示提示符的标志

    /* 将标准错误重定向到标准输出（使所有输出通过管道传输）*/
    dup2(1, 2);                    // Unix系统调用，让stderr(2)指向stdout(1)

    /* 解析命令行参数 */
    while ((c = getopt(argc, argv, "hvp")) != EOF) // 使用getopt解析选项
    {
        switch (c)
        {
        case 'h':                  // -h 显示帮助信息
            usage();               // 调用帮助函数
            break;
        case 'v':                  // -v 启用详细模式
            verbose = 1;           // 设置全局详细标志
            break;
        case 'p':                  // -p 禁用提示符显示
            emit_prompt = 0;       // 用于自动化测试场景
            break;
        default:                    // 无效选项处理
            usage();                // 显示用法说明后退出
        }
    }

    /* 注册信号处理函数 */
    Signal(SIGINT, sigint_handler);   // Ctrl+C 中断信号处理
    Signal(SIGTSTP, sigtstp_handler); // Ctrl+Z 停止信号处理
    Signal(SIGCHLD, sigchld_handler); // 子进程状态变化处理
    Signal(SIGQUIT, sigquit_handler); // 退出信号处理（如Ctrl+\）

    /* 初始化作业列表（用于管理后台进程）*/
    initjobs(jobs);                 // 初始化作业管理结构

    /* Shell的主循环：读取->解析->执行 */
    while (1)                       // 无限循环保持shell运行
    {
        /* 读取命令行输入 */
        if (emit_prompt) {          // 根据标志决定是否显示提示符
            printf("%s", prompt); // 显示提示符（如"$ "）
            fflush(stdout);         // 强制刷新输出缓冲区
        }
        if ((fgets(cmdline, MAXLINE, stdin) == NULL) && ferror(stdin))
            app_error("fgets error"); // 处理输入错误
        if (feof(stdin)) {           // 检测到文件结束符（Ctrl+D）
            fflush(stdout);          // 确保所有输出完成
            exit(0);                 // 正常退出shell
        }

        /* 执行命令 */
        eval(cmdline);              // 核心逻辑：解析并执行命令
        fflush(stdout);             // 两次刷新确保输出完整性
        fflush(stdout);
    }

    exit(0); // 理论上不会执行到此处（因有无限循环）
}
```

关键结构说明：

1. **参数处理**：通过 `getopt` 解析 `-hvp` 选项，实现帮助、详细模式和提示符控制；
2. **信号处理**：注册了4个Unix信号处理器，实现中断、停止和子进程管理；

3. **I/O重定向**：`dup2(1,2)`让错误输出与标准输出合并，便于管道处理；
4. **主循环设计**：采用经典的Read-Eval-Print Loop模式，支持交互式和脚本执行；
5. **作业管理**：`initjobs()`初始化后台进程管理结构，用于实现任务控制功能。

`eval` 函数

```
/*

* eval - 分析命令，并派生子进程执行 主要功能是解析cmdline并运行

* 首先调用 parseline 解析输入的命令行字符串，分割为参数列表 argv 并判断前台/后台作业状态。

* 如果用户请求了一个内置命令（quit、jobs、bg或fg），通过 builtin_cmd 直接执行。

* 否则，创建子进程并通过 execve 执行外部程序。

* 如果作业正在前台运行，请等待它终止，然后返回。

*/

void eval(char *cmdline)

{

    char *argv[MAXARGS]; // execve()函数的参数

    int state = UNDEF;   // 工作状态，FG或BG

    sigset_t set;

    pid_t pid; // 进程id



    // 处理输入的数据

    if (parseline(cmdline, argv) == 1) // 解析命令行，返回给argv数组，返回是否为后台作业（`&`结尾判断）

        state = BG;

    else
```

```
        state = FG;

    if (argv[0] == NULL) // 若命令行为空，parseline也会返回1，但对argv[0]判定后，eval在这里直接返回

        return;



    // 如果不是内置命令

    if (!builtin_cmd(argv)) // 若是内置命令，builtin_cmd(argv)会执行，若该函数返回0，则表示非内置命令

    {

        // 初始化信号集set并把SIGINT SIGTSTP SIGCHLD三个信号放入信号集中，方便管理

        if (sigemptyset(&set) < 0)

            unix_error("sigemptyset error");

        if (sigaddset(&set, SIGINT) < 0 || sigaddset(&set, SIGTSTP) < 0 || sigaddset(&set, SIGCHLD) < 0)

            unix_error("sigaddset error");



        if (sigprocmask(SIG_BLOCK, &set, NULL) < 0)

            unix_error("sigprocmask error");



        if ((pid = fork()) < 0) // fork创建子进程失败

            unix_error("fork error");

        else if (pid == 0) // fork创建子进程

        {
            // 子进程控制流从这里开始



            if (sigprocmask(SIG_UNBLOCK, &set, NULL) < 0) // 解除阻塞

                unix_error("sigprocmask error");


            //函数原型int setpgid(pid_t pid, pid_t pgid);此函数用于设置指定进程
的进程组ID（PGID）。
            //在<unistd.h>下定义。

            if (setpgid(0, 0) < 0) // 设置子进程id（实际上并没有分进程组，因为一人
一组）


                unix_error("setpgid error");




            if (execve(argv[0], argv, environ) < 0)

            {

                printf("%s: Command not found\n", argv[0]);

                exit(0);

            }

        }




        // 父进程控制流从这里开始

        addjob(jobs, pid, state, cmdline); // 将当前进程添加进jobs中，参数为当前
进程pid,state,cmdline
```

```
        // 恢复受阻塞的信号 SIGINT SIGTSTP SIGCHLD

        if (sigprocmask(SIG_UNBLOCK, &set, NULL) < 0)

            unix_error("sigprocmask error");



        // 判断子进程类型并做处理

        if (state == FG)

            waitfg(pid); // 等待子进程的前台作业完成

        else

            printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline); // 将进程id映
射到job id

    }

    return;

}
```

<signal.h>定义的数据类型和函数解释

`sigset_t` **类型**是POSIX标准定义的特殊结构体类型，用于表示信号集合。
int sigemptyset(sigset_t *set); 初始化并清空信号集，使其不包含任何信号。*
int sigaddset(sigset_t set, int signum);将指定信号 `signum` 添加到信号集合中。
int sigprocmask(int how, const sigset_t *restrict set, sigset_t* restrict oldset);修改进程的信号屏蔽字，控制信号的阻塞状态。成功返回0，失败返回-1并设置 `errno`
在eval（）函数中，如果不是内置命令，那么我们会创建子进程。在创建子进程之前，先初始化信号集合，然后将信号**SIGINT**、**SIGTSTP**、**SIGCHLD**加入到信号集合中。

竞态条件的防护

SIGCHLD：通知父进程子进程的状态变化。触发条件是子进程终止（正常退出或异常终止）或状态改变（如被暂停或恢复）。
如果子进程快速地完成（或者外部中断干扰作业进程），那么发送信号 SIGCHLD给父进程，父进程快速响应，将子进程从job列表中删除。而后续父进程还是会将子进程加入job列表。这就导致产生的数据不一致。

那我们的防护工作：在创建子进程之前，阻塞信号SIGCHLD。通过这行代码实现 `sigprocmask(SIG_BLOCK, &set, NULL)`。同时我们要注意父子进程的资源继承和独立。父进程中屏蔽了信号。子进程继承被屏蔽的信号。子进程在解除信号屏蔽。但父进程信号仍然被屏蔽。当addjob（）函数执行之后，父进程中的信号屏蔽才会解除。防止父进程添加一个不存在的子进程。

独立进程组

我们的代码（将当前进程设为进程组的组长，见注释）：每个进程自己一个独立的组。
原因：如果不这么做，子进程默认和父进程同一个进程组。那所有进程都能接收Ctrl+C/Ctrl+Z，导致后台进程也会意外响应。这样前台进程和后台进程就没有办法很好地区分了。
因此，我们需要给每个进程分配独立的进程组PID。

`builtin_cmd` 函数

```
/*

 * builtin_cmd：解析和执行bulidin命令，包括 quit, fg, bg, jobs

 */

int builtin_cmd(char **argv)

{

    if (!strcmp(argv[0], "quit")) // 如果命令是quit，退出

        exit(0);

    else if (!strcmp(argv[0], "bg") || !strcmp(argv[0], "fg")) // 如果是bg或者fg命令，执行do_fgbg函数

        do_bgfg(argv);

    else if (!strcmp(argv[0], "jobs")) // 如果命令是jobs，列出正在运行和停止的后台作业

        listjobs(jobs);

    else
```

```
        return 0; /* 不是内置命令，以0返回eval */

    return 1;

}
```

核心功能

此函数用于识别并执行 Shell 的**内置命令**（如 `quit`、`fg`、`bg`、`jobs`），若输入命令是内置的，则直接处理并返回 `1`；否则返回 `0`，通知 `eval` 函数执行外部程序。

`waitfg` 函数

```
/*

 * waitfg - Block until process pid is no longer the foreground process

 */

void waitfg(pid_t pid)

{

    // 通过pid获取该pid对应的job本体

    struct job_t *job = getjobpid(jobs, pid);

    if (!job)

        return;

    // 新设立一个wait信号集

    sigset_t wait;

    if (sigemptyset(&wait) < 0) // 清空wait信号集，该信号集为空

        unix_error("sigemptyset &wait error");

    //   如果当前子进程的状态没有发生改变，则tsh继续休眠

    while (job->state == FG)
```

```
        // 一旦有信号改变，就判定是否

        sigsuspend(&wait); // 使用空信号集替换信号掩码，即信号掩码为空，此时任何信号
都会唤醒该进程



    return;

}
```

`sigsuspend` **函数**

头文件：int sigsuspend(const sigset_t * mask);
函数原型：#include <signal.h>
`mask` 是一个信号集指针，表示在等待信号期间**临时替换进程的信号屏蔽字**（即阻塞哪些信号）。
`sigsuspend` 用于**挂起进程执行，直到接收到特定信号**，同时确保操作的原子性。具体过程如下：

- 临时将进程的信号屏蔽字设置为 `mask`，并挂起进程
- 仅当接收到 `mask` 之外的信号（即未被阻塞的信号）时，进程才会被唤醒。
  在我们的代码中，用之前初始化并且清理的空wait信号集替换信号掩码。那么空信号集取反对应的全集（所有信号）都会唤醒该进程。也就是说，当前子进程是FG前台进程那就挂起。当信号改变。这个进程就会被唤醒。

核心功能

此函数用于**阻塞当前进程（Shell）**，直到指定进程（`pid`）不再是前台进程（Foreground Job）。其核心应用场景是：当用户启动一个前台作业时，Shell 需等待该作业执行完毕或被挂起（如用户按下 `Ctrl+Z`）后，才能继续接收新命令。

`sigchld_handler` 函数

```
/*

 * sigchld_handler- 处理僵尸子进程例程

 每当子作业终止（变成僵尸），或者因为接收到SIGSTOP或SIGTSTP信号而停止时，内核都会向
shell发送sigchld。处理程序获取所有可用的僵尸子进程，但不等待任何其他当前正在运行的子进
程终止。
```

```
 */

void sigchld_handler(int sig)

{

    int status, jid;//status：存储子进程终止状态

    pid_t pid;

    struct job_t *job;



    if (verbose)

        puts("sigchld_handler: entering"); // 输出额外信息



    while ((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0) // 以非阻塞方式循环等待所有子进程，若成功回收了子进程，则返回这个子进程的PID，&status中返回其状态

    {



        // 从全局 `jobs` 数组中通过 PID 查找作业。
        //如果当前这个子进程的job已经删除了，则表示有错误发生

        if ((job = getjobpid(jobs, pid)) == NULL)

        {

            printf("Lost track of (%d)\n", pid);

            return;

        }
        jid = job->jid;

        // 接下来判断三种状态

        //  如果这个子进程收到了一个暂停信号（如Ctrl+Z）

        if (WIFSTOPPED(status))
        //WIFSTOPPED：判断是否因信号暂停（如 `SIGTSTP`）

        {

            printf("Job [%d] (%d) stopped by signal %d\n", jid, job->pid,
WSTOPSIG(status));

            // 使用WSTOPSIG（status） 提取导致暂停的信号编号

            job->state = ST; // 状态设为挂起

        }

        // 如果子进程通过调用 exit 或者一个返回 (return) 正常终止

        else if (WIFEXITED(status))
        //WIFEXITED(status)判断是否正常退出

        {

            if (deletejob(jobs, pid))

                if (verbose)

                {

                    printf("sigchld_handler: Job [%d] (%d) deleted\n", jid,
pid);

                    printf("sigchld_handler: Job [%d] (%d) terminates OK
(status %d)\n", jid, pid, WEXITSTATUS(status));

                    // 用WEXITSTATUS(status) 获取子进程退出状态码（exit的参数）

                }
        }

        // 如果子进程是因为一个未被捕获的信号终止的，例如SIGKILL

        else

        {

            if (deletejob(jobs, pid))

            { // 清除进程

                if (verbose)

                    printf("sigchld_handler: Job [%d] (%d) deleted\n", jid,
pid);

            }

            printf("Job [%d] (%d) terminated by signal %d\n", jid, pid,
WTERMSIG(status));

            // 使用WTERMSIG (status) 得到使子进程退出得信号编号

        }

    }



    if (verbose)

        puts("sigchld_handler: exiting");



    return;

}
```

waitpid函数

头文件：#include <sys/types.h>、#include <sys/wait.h>
函数原型：pid_t waitpid(pid_t pid, int *status, int options);*
参数解析：pid_t pid：指定监控的子进程范围、 int status：子进程状态存储（保存子进程的终止/暂停状态，需通过宏解析具体原因）、int options：控制等待行为的选项

参数值	说明
`pid > 0`	只等待进程ID等于pid的子进程，只要该子进程不结束，就会一直等待下去
`pid = -1`	等待任何一个子进程的退出，此时作用和wait相同
`pid = 0`	等待与当前进程同进程组的任何子进程。
`pid < -1`	等待一个指定进程组中的任何子进程，这个进程组的ID等于pid的绝对值

关键宏	作用
`WIFEXITED(status)`	若子进程正常退出（通过 `exit()` 或 `return`），返回非零值。
`WEXITSTATUS(status)`	若 `WIFEXITED` 为真，获取子进程的退出码（`exit(3)` 返回 3）。
`WIFSIGNALED(status)`	若子进程因未捕获的信号终止（如 `SIGKILL`），返回非零值。
`WTERMSIG(status)`	若 `WIFSIGNALED` 为真，获取导致终止的信号编号。
`WIFSTOPPED(status)`	若子进程因信号暂停（如 `SIGTSTP`），返回非零值。

options可以通过位或（ | ）组合以下选项：

选项	说明
`WNOHANG`	**非阻塞模式**：若没有子进程结束/暂停，立即返回 `0`（不等待）。
`WUNTRACED`	**监控暂停状态**：报告因信号（如 `SIGTSTP`）暂停的子进程状态。
`WCONTINUED`	（部分系统支持）报告因 `SIGCONT` 恢复执行的子进程状态。

那么在我们的代码中，采用非阻塞模式。它的返回值是这样的。如果有子进程结束/暂停，返回子进程PID（成功回收子进程，需解析 `status` 判断终止原因）；如果没有子进程结束/暂停，立即返回0，父进程继续执行其他任务；当没有匹配的子进程/调用被信号中断，返回-1。而且我们使用了while循环。这样可以等待所有子进程。

具体的执行逻辑，注释已经很清楚啦。

核心功能

这是用于处理 `SIGCHLD` 信号的函数，负责回收终止或暂停的子进程，并更新作业（`job`）状态。

其核心任务是：它可以通过 `waitpid` 回收子进程资源，并且在作业列表中更新作业状态。我们会根据子进程的三种状态（正常退出、被信号终止、被暂停）来进行不同的处理。

`sigint_handler` 函数

前面我们已经提到过可通过 `Ctrl+C` 发送 `SIGINT` 终止前台作业，但它不影响后台作业。

```
/* 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;

}
```

所以我们获取pid，调用的函数是fgpid。只取前台作业的pid。

```
/*

 * sigint_handler - 当用户在键盘上键入ctrl-c时，内核会向shell发送一个SIGINT。抓住它并将其发送到前台作业。

 */

void sigint_handler(int sig)

{

    if (verbose)

        puts("sigint_handler: entering");
    pid_t pid = fgpid(jobs); // 获取前台进程的pid



    if (pid)

    {

        // 发送SIGINT给前台进程组里的所有进程

        // 需要注意的是，前台进程组内的进程除了当前前台进程以外，还包括前台进程的子进
程。

        // 最多只能存在一个前台进程，但前台进程组内可以存在多个进程

        if (kill(-pid, SIGINT) < 0) // 采用负数发送信号到进程组，使该进程终止

            unix_error("kill (sigint) error");

        if (verbose)

        {

            printf("sigint_handler: Job (%d) killed\n", pid);

        }

    }

    if (verbose)

        puts("sigint_handler: exiting");

    return;

}
```

kill函数

头文件：#include <sys/types.h>、#include <signal.h>
函数原型：int kill(pid_t pid, int sig);
参数：pid_t pid：目标进程/进程组标识；int sig：发送的信号类型

参数值	说明
`pid > 0`	向 PID 为 `pid` 的特定进程发送信号（如终止进程 1234）
`pid = 0`	向 当前进程组的所有进程（调用进程所在的进程组）发送信号
`pid = -1`	向 系统中所有有权限的进程（除 `init` 进程外）发送信号，需超级用户权限
`pid < -1`	向 进程组ID等于pid绝对值的所有进程发送信号

我们这段代码，整体就是：

- 从全局作业列表中查找当前前台作业的PID。将pid改成负数作为参数传入kill函数，向该前台作业的整个进程组发送信号SIGINT。
- （因为前台作业可能包括多个进程如父子进程，所以必须确保整个进程组都被终止。）

`sigtstp_handler`函数

```
/*

 * sigtstp_handler - 每当用户在键盘上键入ctrl-z时，内核都会向shell发送一个sigtstp。捕获它并通过向它发送SIGTSTP来挂起前台作业。

 */

void sigtstp_handler(int sig)

{

    if (verbose)

        puts("sigstp_handler: entering");



    pid_t pid = fgpid(jobs);                    // 获取前台进程的pid号

    struct job_t *job = getjobpid(jobs, pid); // 获取前台进程的job本体
    if (pid)


        if (kill(-pid, SIGTSTP) < 0) // 采用负数发送信号到进程组，使该进程挂起
            unix_error("kill (tstp) error");
    if (verbose)
    {
        printf("sigstp_handler: Job [%d] (%d) stopped\n", job->jid, pid);
    }


    if (verbose)
        puts("sigstp_handler: exiting");
    return;
}
```

核心功能

它和 `sigint_handler` 最大的区别在于处理的信号不同。当用户按下Ctrl-Z时，内核会向Shell发送 `SIGTSTP` 信号。该函数的作用是捕获该信号，并向前台作业的进程组发送 `SIGTSTP` 信号，使其挂起（暂停执行），但不终止进程。整体逻辑完全一样。只是更改了kill函数的第二个参数（发送信号的类型）。 `sigint_handler` 中发送的是SIGINT，这里发送的是SIGTSTP。

挂起和终止

值得一提的是，这段代码中获取了前台作业的本体，并且打印出了用户可见的作业编号jid。而 `sigint_handler` 函数中没有。
究其原因在于挂起并不像终止，会把作业从后台列表中删除。

- `SIGTSTP` 会将作业状态从 `FG` （前台运行）改为 `ST` （停止），但作业仍存在于作业列表（ `jobs` ）中，用户可通过 `fg` 或 `bg` 恢复。

- `SIGINT` 终止作业会直接删除作业（`deletejob`），获取前台作业的本体没有意义。

验证

Makefile文件

```
TEAM = NOBODY              # 提交者标识
VERSION = 1                # 版本号
HANDINDIR = /afs/cs/...    # 作业提交路径
DRIVER = ./sdriver.pl       # 测试驱动脚本
TSH = ./tsh                # 学生实现的 Shell 程序
TSHREF = ./tshref           # 参考实现的 Shell 程序
TSHARGS = "-p"             # 传递给 Shell 的参数（如启用进程组）
CC = gcc                   # 编译器
CFLAGS = -Wall -O2         # 编译选项（严格警告 + 优化）
FILES = $(TSH) ./myspin... # 需要构建的目标文件列表

all: $(FILES)  # 默认构建所有文件（学生 Shell 及测试工具）

handin:
    cp tsh.c $(HANDINDIR)/$(TEAM)-$(VERSION)-tsh.c
# 用于提交作业，将学生的 `tsh.c` 文件复制到指定目录，命名格式为 `TEAM-VERSION-
tsh.c`


##################

# Regression tests

##################



# Run tests using the student's shell program

test01:

    $(DRIVER) -t trace01.txt -s $(TSH) -a $(TSHARGS)

test02:

    $(DRIVER) -t trace02.txt -s $(TSH) -a $(TSHARGS)
```

```
test03:

    $(DRIVER) -t trace03.txt -s $(TSH) -a $(TSHARGS)

test04:

    $(DRIVER) -t trace04.txt -s $(TSH) -a $(TSHARGS)

test05:

    $(DRIVER) -t trace05.txt -s $(TSH) -a $(TSHARGS)

test06:

    $(DRIVER) -t trace06.txt -s $(TSH) -a $(TSHARGS)

test07:

    $(DRIVER) -t trace07.txt -s $(TSH) -a $(TSHARGS)

test08:

    $(DRIVER) -t trace08.txt -s $(TSH) -a $(TSHARGS)

test09:

    $(DRIVER) -t trace09.txt -s $(TSH) -a $(TSHARGS)

test10:

    $(DRIVER) -t trace10.txt -s $(TSH) -a $(TSHARGS)

test11:

    $(DRIVER) -t trace11.txt -s $(TSH) -a $(TSHARGS)

test12:

    $(DRIVER) -t trace12.txt -s $(TSH) -a $(TSHARGS)

test13:

    $(DRIVER) -t trace13.txt -s $(TSH) -a $(TSHARGS)
```

```
test14:

    $(DRIVER) -t trace14.txt -s $(TSH) -a $(TSHARGS)

test15:

    $(DRIVER) -t trace15.txt -s $(TSH) -a $(TSHARGS)

test16:

    $(DRIVER) -t trace16.txt -s $(TSH) -a $(TSHARGS)



# Run the tests using the reference shell program

rtest01:

    $(DRIVER) -t trace01.txt -s $(TSHREF) -a $(TSHARGS)

rtest02:

    $(DRIVER) -t trace02.txt -s $(TSHREF) -a $(TSHARGS)

rtest03:

    $(DRIVER) -t trace03.txt -s $(TSHREF) -a $(TSHARGS)

rtest04:

    $(DRIVER) -t trace04.txt -s $(TSHREF) -a $(TSHARGS)

rtest05:

    $(DRIVER) -t trace05.txt -s $(TSHREF) -a $(TSHARGS)

rtest06:

    $(DRIVER) -t trace06.txt -s $(TSHREF) -a $(TSHARGS)

rtest07:
	$(DRIVER) -t trace07.txt -s $(TSHREF) -a $(TSHARGS)

rtest08:

	$(DRIVER) -t trace08.txt -s $(TSHREF) -a $(TSHARGS)

rtest09:

	$(DRIVER) -t trace09.txt -s $(TSHREF) -a $(TSHARGS)

rtest10:

	$(DRIVER) -t trace10.txt -s $(TSHREF) -a $(TSHARGS)

rtest11:

	$(DRIVER) -t trace11.txt -s $(TSHREF) -a $(TSHARGS)

rtest12:

	$(DRIVER) -t trace12.txt -s $(TSHREF) -a $(TSHARGS)

rtest13:

	$(DRIVER) -t trace13.txt -s $(TSHREF) -a $(TSHARGS)

rtest14:

	$(DRIVER) -t trace14.txt -s $(TSHREF) -a $(TSHARGS)

rtest15:

	$(DRIVER) -t trace15.txt -s $(TSHREF) -a $(TSHARGS)

rtest16:

	$(DRIVER) -t trace16.txt -s $(TSHREF) -a $(TSHARGS)



# clean up
```

```
clean:
    rm -f $(FILES) *.o *~   # 删除所有编译生成的文件和临时文件
```

从test01-test16:

```
test01:
  $(DRIVER) -t trace01.txt -s $(TSH) -a $(TSHARGS)
```

每个 `testXX` 目标通过 `sdriver.pl` 驱动脚本执行对应的 `traceXX.txt` 测试用例，使用学生实现的 Shell (`tsh`)。

- `-t traceXX.txt`：指定测试用例文件（包含一系列 Shell 命令和预期行为）。
- `-s $(TSH)`：指定被测试的 Shell 程序（学生实现）。
- `-a $(TSHARGS)`：传递给 Shell 的参数（如 `-p` 启用进程组控制）。

从rtest01-rtest16:

```
rtest01:
  $(DRIVER) -t trace01.txt -s $(TSHREF) -a $(TSHARGS)
```

与 `testXX` 类似，但使用参考实现的 Shell (`tshref`)。

故此，我们的单个验证方式为：

```
make testXX   #这是运行我们自己的tinyshell
make rtestXX  #这是使用参考的tinyshell
#如果两者输出一致，可以认为这个测试点通过了
```

16个测试点通过后，我们的tinyshell可以被认为基本搭建正确。

整体验证

我们可以修改makefile，自己给它新增一个功能，使它能够一次性打印所有的测试点数据。

```
//在原代码下面添加
testall:
    $(DRIVER) -t trace01.txt -s $(TSH) -a $(TSHARGS)
    $(DRIVER) -t trace02.txt -s $(TSH) -a $(TSHARGS)
    $(DRIVER) -t trace03.txt -s $(TSH) -a $(TSHARGS)
    $(DRIVER) -t trace04.txt -s $(TSH) -a $(TSHARGS)
    $(DRIVER) -t trace05.txt -s $(TSH) -a $(TSHARGS)
    $(DRIVER) -t trace06.txt -s $(TSH) -a $(TSHARGS)
    $(DRIVER) -t trace07.txt -s $(TSH) -a $(TSHARGS)
```

```
	$(DRIVER) -t trace08.txt -s $(TSH) -a $(TSHARGS)
	$(DRIVER) -t trace09.txt -s $(TSH) -a $(TSHARGS)
	$(DRIVER) -t trace10.txt -s $(TSH) -a $(TSHARGS)
	$(DRIVER) -t trace11.txt -s $(TSH) -a $(TSHARGS)
	$(DRIVER) -t trace12.txt -s $(TSH) -a $(TSHARGS)
	$(DRIVER) -t trace13.txt -s $(TSH) -a $(TSHARGS)
	$(DRIVER) -t trace14.txt -s $(TSH) -a $(TSHARGS)
	$(DRIVER) -t trace15.txt -s $(TSH) -a $(TSHARGS)
	$(DRIVER) -t trace16.txt -s $(TSH) -a $(TSHARGS)

rtestall:
	$(DRIVER) -t trace01.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace02.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace03.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace04.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace05.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace06.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace07.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace08.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace09.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace10.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace11.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace12.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace13.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace14.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace15.txt -s $(TSHREF) -a $(TSHARGS)
	$(DRIVER) -t trace16.txt -s $(TSHREF) -a $(TSHARGS)
```

测试所有测试集并使用重定向符将其导出

```
make testall > my_output.txt
make rtestall > reference_output.txt
```

通过比较这两个文件，获取验证结果。

test01-正确地终止EOF

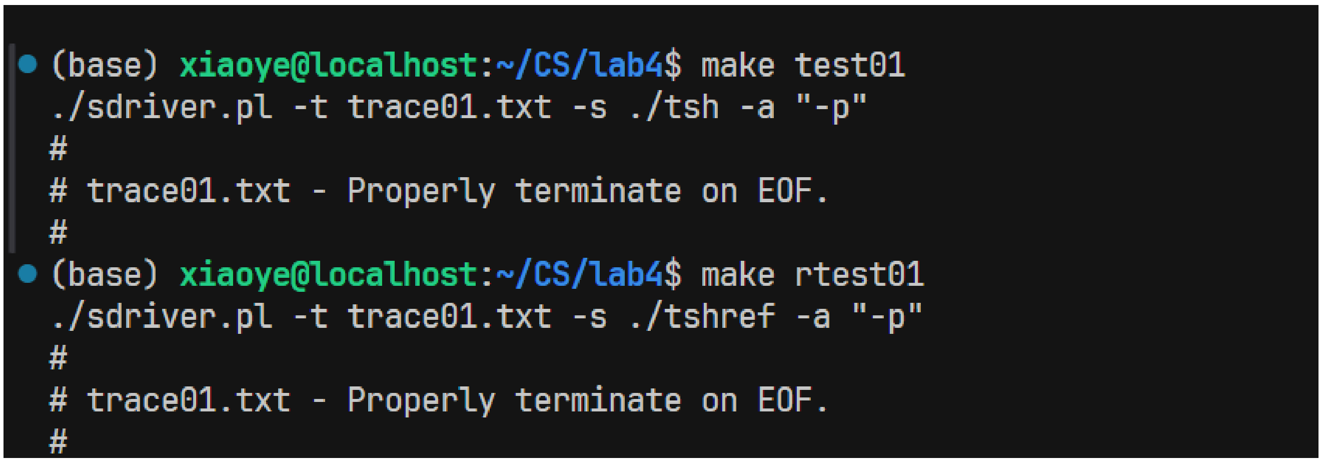
```
#
# trace01.txt - Properly terminate on EOF.
#
```

```
CLOSE
WAIT
```

trace01.txt:

调用linux命令close关闭文件并wait等待，在EOF上正常终止。

验证



```
(base) xiaoye@localhost:~/CS/lab4$ make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
(base) xiaoye@localhost:~/CS/lab4$ make rtest01
./sdriver.pl -t trace01.txt -s ./tshref -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
```

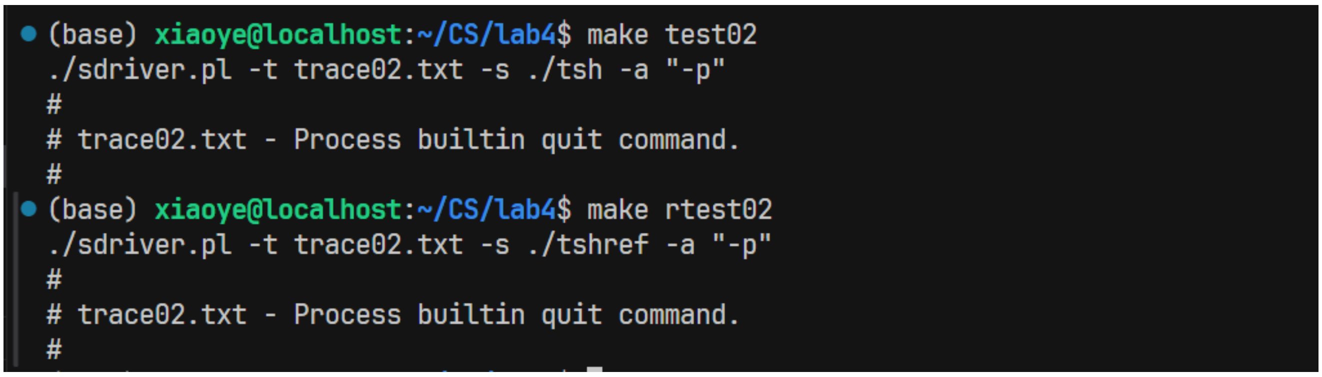
test02-进程内置quit命令

```
#
# trace02.txt - Process builtin quit command.
#
quit
WAIT
```

trace02.txt:

quit命令退出shell进程。
首先我们需要调用eval（）函数，解析命令行参数，将其返回给argv数组；
其次调用builtin_cmd（）函数，进行字符串的比对，如果是内置命令“quit”，那就exit（0）退出。

验证



```
(base) xiaoye@localhost:~/CS/lab4$ make test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
(base) xiaoye@localhost:~/CS/lab4$ make rtest02
./sdriver.pl -t trace02.txt -s ./tshref -a "-p"
#
# trace02.txt - Process builtin quit command.
#
```

test03-运行一个前台job

```
#
# trace03.txt - Run a foreground job.
#
/bin/echo tsh> quit
quit
```

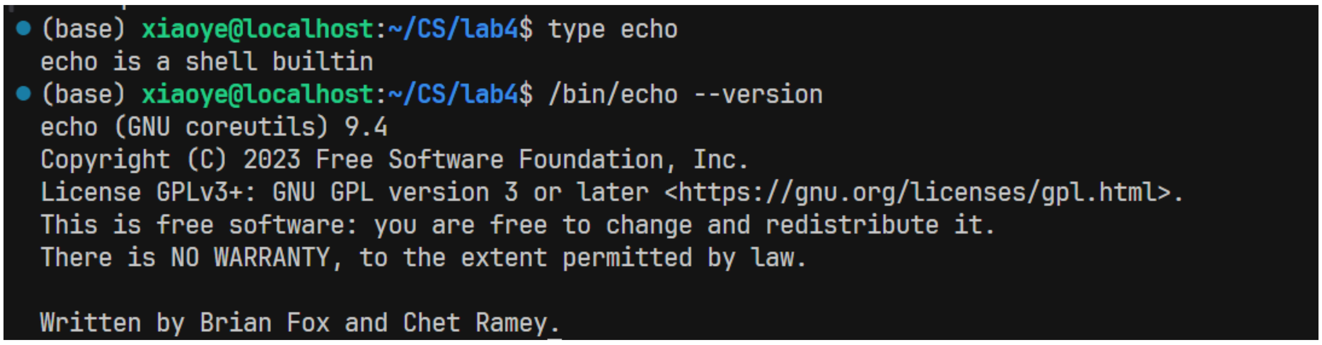
trace03.txt:

- 首先我们可以看到eval（）函数解析命令行参数返回为argv数组。分别有内容：/bin/echo、tsh>、quit;
- 其次，我们会调用builtin_cmd（）函数，比较argv[0]是不是内置命令的字符串。我们可以看到，/bin/echo不是内置命令（quit、bg、fg、jobs）。
- 接着，我们会创建子进程，并且调用execve（）来创建新的程序映像。
- 调用 execve()函数通过 argv[0]来寻找路径，并在子进程中运行路径中的可执行文件，如果找不到则说明命令为无效命令，输出命令无效，并用 exit(0)结束该子进程。
- /bin/echo就是打开bin目录下的echo文件。echo会将后面的内容作为字符串输出。所以，正常情况下，会输出tsh> quit。

echo

echo：将 `echo` 作为shell的内置命令实现。
/bin/echo：外部命令是独立可执行文件，遵循标准化行为，适合跨环境使用。
功能：将其后面的内容作为字符串输出

```
# 检查是否为内置命令
type echo           # 输出 "echo is a shell builtin"
# 查看外部版本
/bin/echo --version  # 显示 GNU coreutils 版本
```

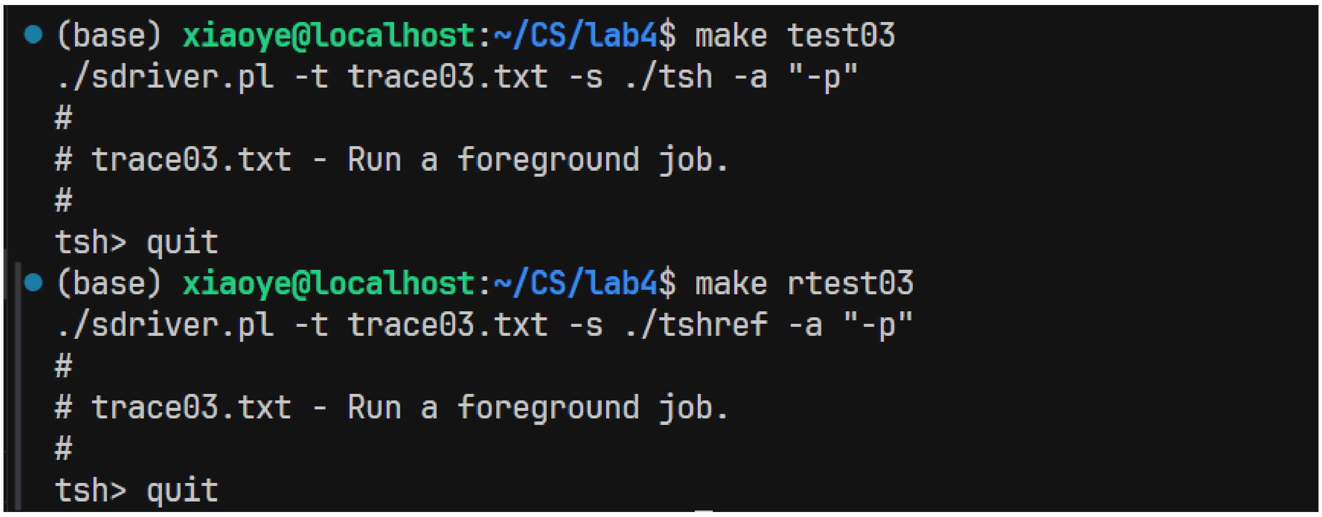


```
(base) xiaoye@localhost:~/CS/lab4$ type echo
echo is a shell builtin
(base) xiaoye@localhost:~/CS/lab4$ /bin/echo --version
echo (GNU coreutils) 9.4
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Brian Fox and Chet Ramey.
```

execve（）函数介绍

函数原型：int execve(const char *pathname, char* const argv[], char * const envp[]);
pathname：要执行的程序文件的完整路径（如 `/bin/ls`）；
argv[]：传递给新程序的命令行参数数组，以 `NULL` 结尾；
envp[]：传递给新程序的环境变量数组，格式为 `NAME=value`，以 `NULL` 结尾。
成功时，不返回，原进程映像被新程序替换；失败时，返回-1。

验证



```
● (base) xiaoye@localhost:~/CS/lab4$ make test03
  ./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
  #
  # trace03.txt - Run a foreground job.
  #
  tsh> quit
● (base) xiaoye@localhost:~/CS/lab4$ make rtest03
  ./sdriver.pl -t trace03.txt -s ./tshref -a "-p"
  #
  # trace03.txt - Run a foreground job.
  #
  tsh> quit
```

test04-运行后台job

```
#
# trace04.txt - Run a background job.
#
/bin/echo -e tsh> ./myspin 1 \046
./myspin 1 &
```

trace04.txt:

- 首先我们看到这里有两行命令。
- 第一行，和trace03.txt很像。使用/bin/echo将后面的内容“tsh> ./myspin 1"作为字符串输出。其中-e参数会启用转义字符解析。\046是 ASCII 字符 & 的八进制转义形式。在 Shell 中，反斜杠 \ 用于转义特殊字符，避免其被解析为语法符号。所以实际的输出应该是tsh> ./myspin 1 &。（因为第一行的命令，没有使用&符号，所以这个进程是前台进程。）
- 第二行，运行程序myspin，传入参数1，&表示将命令置于后台运行。

&后台作业

在 Shell 中，&表示将命令置于后台运行。此操作会：

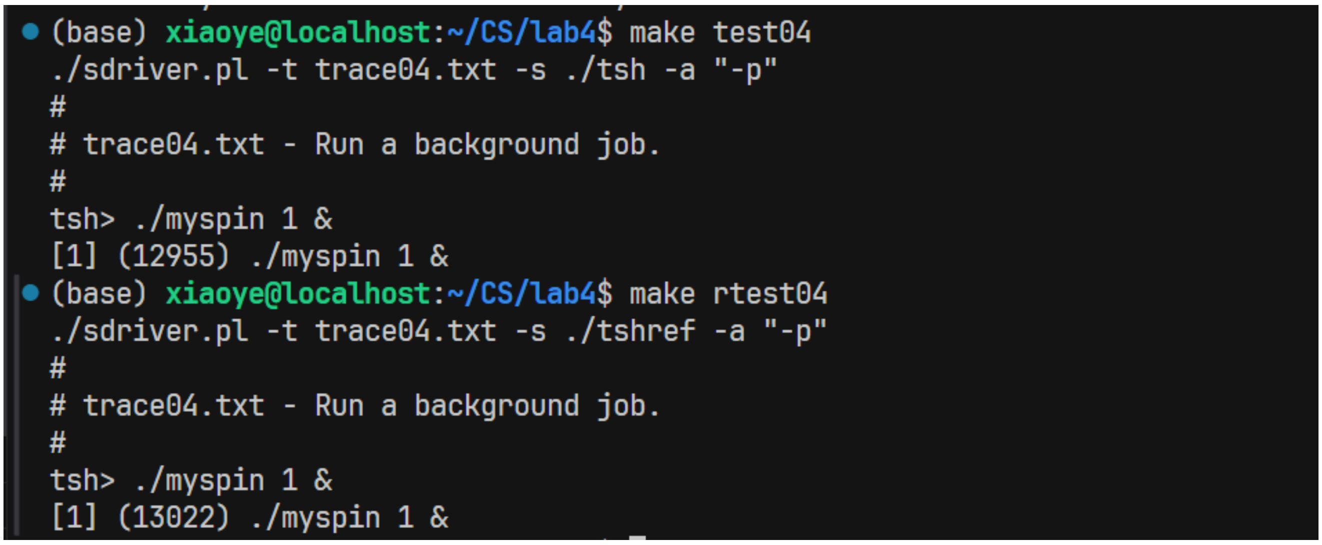
- 立即释放终端控制权，允许用户继续输入其他命令。

- 返回后台作业的进程 ID（PID）和作业编号。

所以，执行的过程应该是：先在前台执行echo命令，等待程序执行完毕回收子进程。&代表是一个后台程序，myspin睡眠1秒，然后停止。实现了后台程序的运行，输出后台程序的运行信息。
myspin的程序如下，传入参数是1，那就循环休眠1次，即睡眠1秒：

```
/*
 * myspin.c - A handy program for testing your tiny shell
 *
 * usage: myspin <n>
 * Sleeps for <n> seconds in 1-second chunks.
 */
#include <stdio.h>
#include <unistd.h>    // 提供 sleep 函数（POSIX 标准）
#include <stdlib.h>
int main(int argc, char **argv)
{
    int i, secs;       // 声明循环计数器 i 和休眠总秒数 secs

    // 检查命令行参数数量是否合法（程序名 + 1 个参数）
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <n>\n", argv[0]);
        exit(0);                                     // 非零退出码表示异常终止
    }

    secs = atoi(argv[1]);  // 将参数转换为整数

    // 循环休眠 n 次，每次 1 秒
    for (i=0; i < secs; i++) {
        sleep(1);  // POSIX 标准休眠函数，单位为秒
    }

    exit(0);  // 正常退出，返回码 0 表示成功
}
```

验证



```
(base) xiaoye@localhost:~/CS/lab4$ make test04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (12955) ./myspin 1 &
(base) xiaoye@localhost:~/CS/lab4$ make rtest04
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (13022) ./myspin 1 &
```

每次通过 `fork()` 或 `exec()` 系统调用创建新进程时，内核会从可用 PID 池中分配一个未被使用的最小数值作为新进程的 PID。进程标识符（PID）是操作系统内核为每个进程动态分配的唯一编号。我们观察到的 `12955` 和 `13022` 差异是正常现象。每次运行 `./myspin` 都会创建一个全新的进程，其 PID 必然不同。

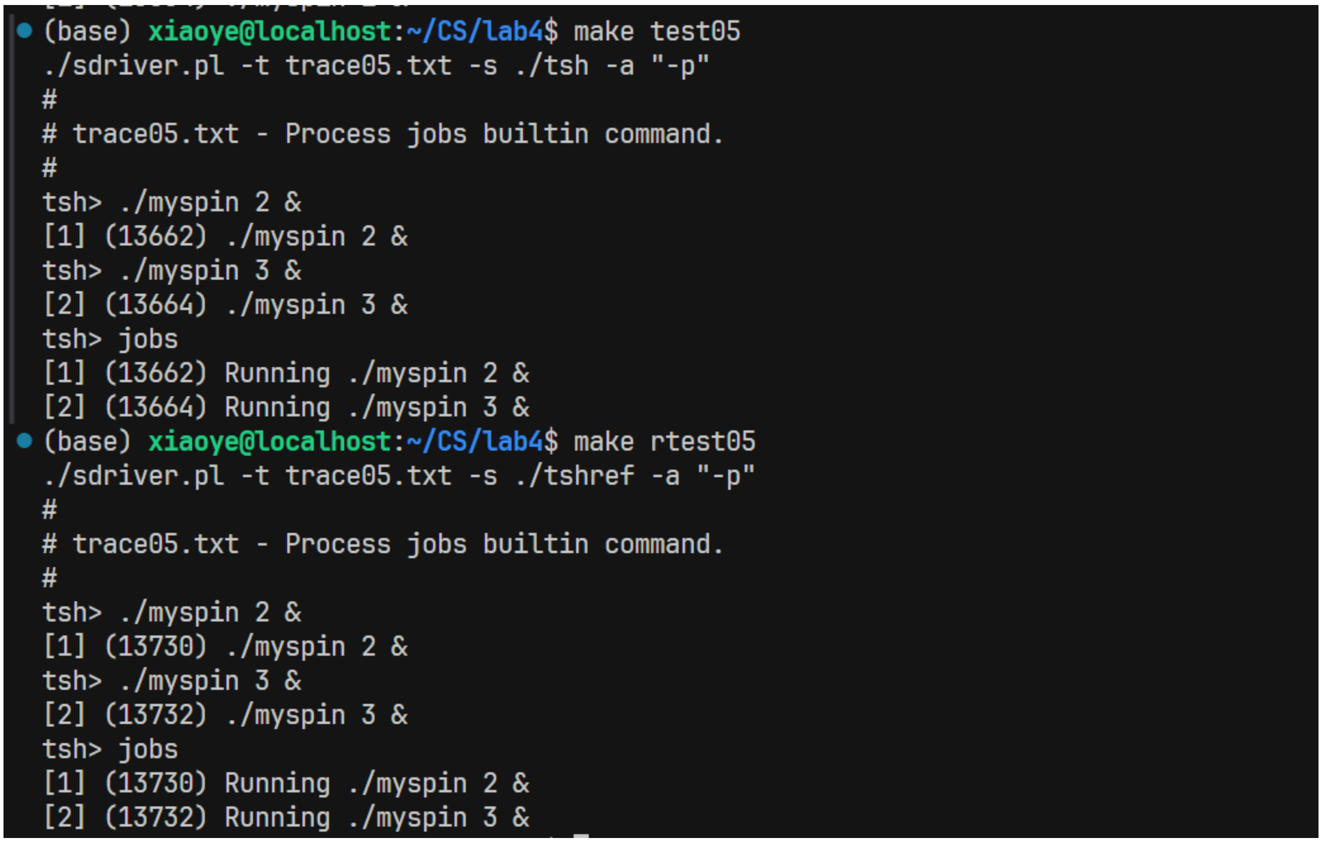
test05-处理jobs内置命令

```
#
# trace05.txt - Process jobs builtin command.
#
/bin/echo -e tsh> ./myspin 2 \046
./myspin 2 &

/bin/echo -e tsh> ./myspin 3 \046
./myspin 3 &

/bin/echo tsh> jobs
jobs
```

trace05.txt

- 首先我们看到这里有六行命令。前面四行，和test04一样。前台运行echo，打印”tsh> ./myspin 2 &“，后台运行myspin；前台运行echo，打印”tsh> ./myspin 3 &“，后台运行myspin；
- 其次，前台运行echo，打印”tsh> jobs“；
- 接着给出内置命令jobs。首先调用eval（）函数解析参数，再调用builtin_cmd（）判断是否是内置命令”jobs“，如果是，调用listjobs(jobs)，显示目前任务列表中的所有任务的所有属性。

验证


```
● (base) xiaoye@localhost:~/CS/lab4$ make test05
  ./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
  #
  # trace05.txt - Process jobs builtin command.
  #
  tsh> ./myspin 2 &
  [1] (13662) ./myspin 2 &
  tsh> ./myspin 3 &
  [2] (13664) ./myspin 3 &
  tsh> jobs
  [1] (13662) Running ./myspin 2 &
  [2] (13664) Running ./myspin 3 &
● (base) xiaoye@localhost:~/CS/lab4$ make rtest05
  ./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
  #
  # trace05.txt - Process jobs builtin command.
  #
  tsh> ./myspin 2 &
  [1] (13730) ./myspin 2 &
  tsh> ./myspin 3 &
  [2] (13732) ./myspin 3 &
  tsh> jobs
  [1] (13730) Running ./myspin 2 &
  [2] (13732) Running ./myspin 3 &
```

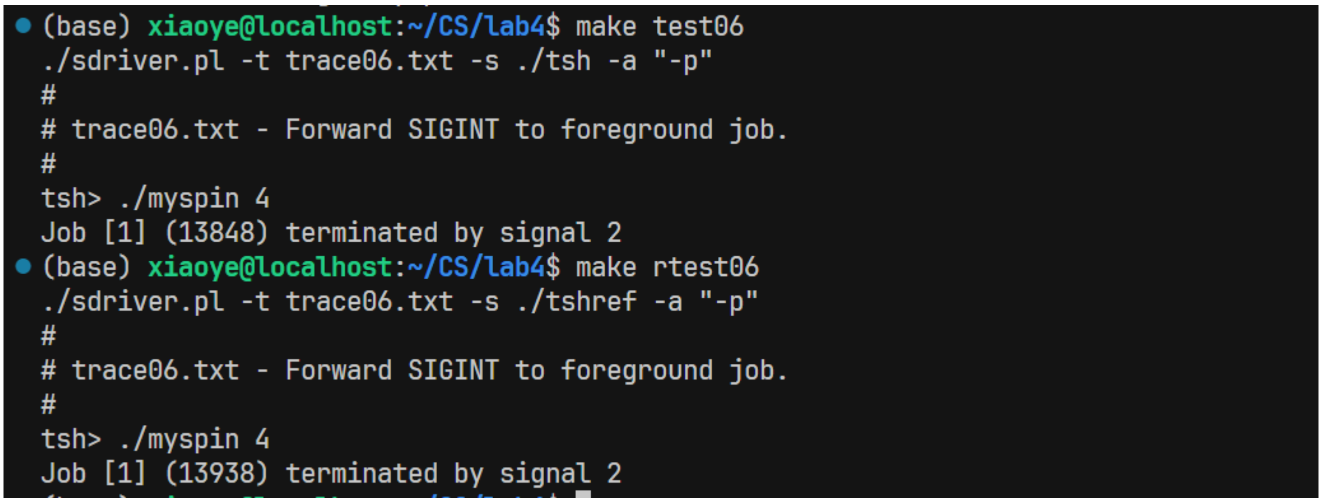
test06-将SIGINT转发到前台作业

```
#
# trace06.txt - Forward SIGINT to foreground job.
#
/bin/echo -e tsh> ./myspin 4
./myspin 4

SLEEP 2
INT
```

trace06.txt

- 前台运行echo,打印"tsh> ./myspin 4“，睡眠4s，
- 前台运行myspin，睡眠4s;
- shell执行sleep 2，shell进程休眠，但前台进程组仍然存在活跃进程（如 `./myspin 4`）;
- 当用户发送INT(模拟Ctrl+C)，内核会中断 Shell 的 sleep()系统调用（无论该调用是否处于前台进程组），并唤醒 Shell 进程。随后，Shell 根据信号处理逻辑，将 SIGINT转发给前台进程组中的所有成员（包括 `./myspin 4`）。SIGINT的转发优先级高于Shell自身的恢复逻辑，因此，myspin 4会先被终止，随后shell继续执行命令。

验证



```
● (base) xiaoye@localhost:~/CS/lab4$ make test06
  ./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
  #
  # trace06.txt - Forward SIGINT to foreground job.
  #
  tsh> ./myspin 4
  Job [1] (13848) terminated by signal 2
● (base) xiaoye@localhost:~/CS/lab4$ make rtest06
  ./sdriver.pl -t trace06.txt -s ./tshref -a "-p"
  #
  # trace06.txt - Forward SIGINT to foreground job.
  #
  tsh> ./myspin 4
  Job [1] (13938) terminated by signal 2
```

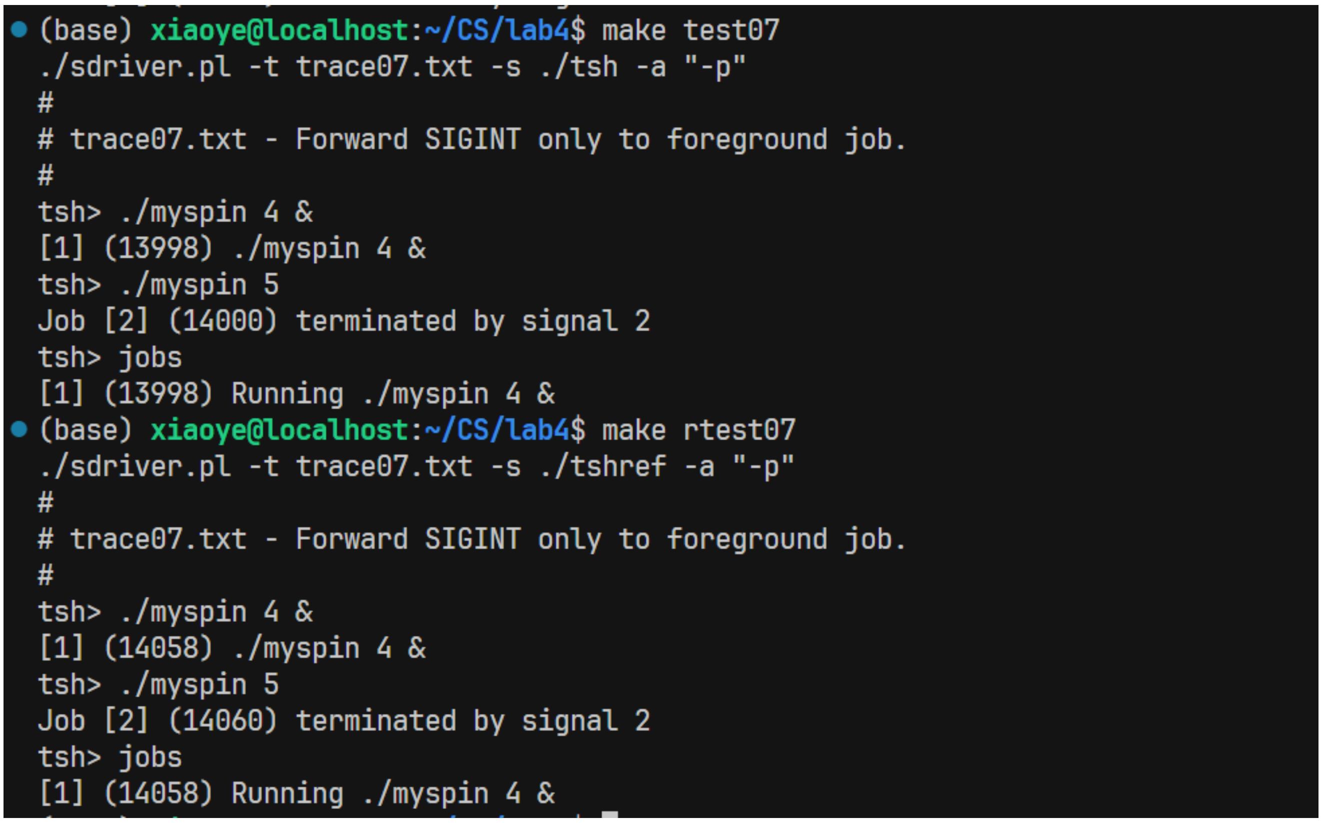
test07-仅将SIGINT转发到前台作业

```
#
# trace07.txt - Forward SIGINT only to foreground job.
#
/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

/bin/echo -e tsh> ./myspin 5
./myspin 5

SLEEP 2
INT

/bin/echo tsh> jobs
jobs
```

trace07.txt

- 顾名思义，我们要检查SIGINT只转发到前台作业不转发到后台作业。
- 所以，我们首先有两个作业，一个fg，一个bg。
- 然后传递SIGINT指令。
- 调用内置命令jobs，来查看此时的工作信息。（如果前台作业收到SIGINT，那作业列表会将这个作业删除）。

验证


```
(base) xiaoye@localhost:~/CS/lab4$ make test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (13998) ./myspin 4 &
tsh> ./myspin 5
Job [2] (14000) terminated by signal 2
tsh> jobs
[1] (13998) Running ./myspin 4 &
(base) xiaoye@localhost:~/CS/lab4$ make rtest07
./sdriver.pl -t trace07.txt -s ./tshref -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (14058) ./myspin 4 &
tsh> ./myspin 5
Job [2] (14060) terminated by signal 2
tsh> jobs
[1] (14058) Running ./myspin 4 &
```

我们发现列表只剩下后台作业，验证完成。

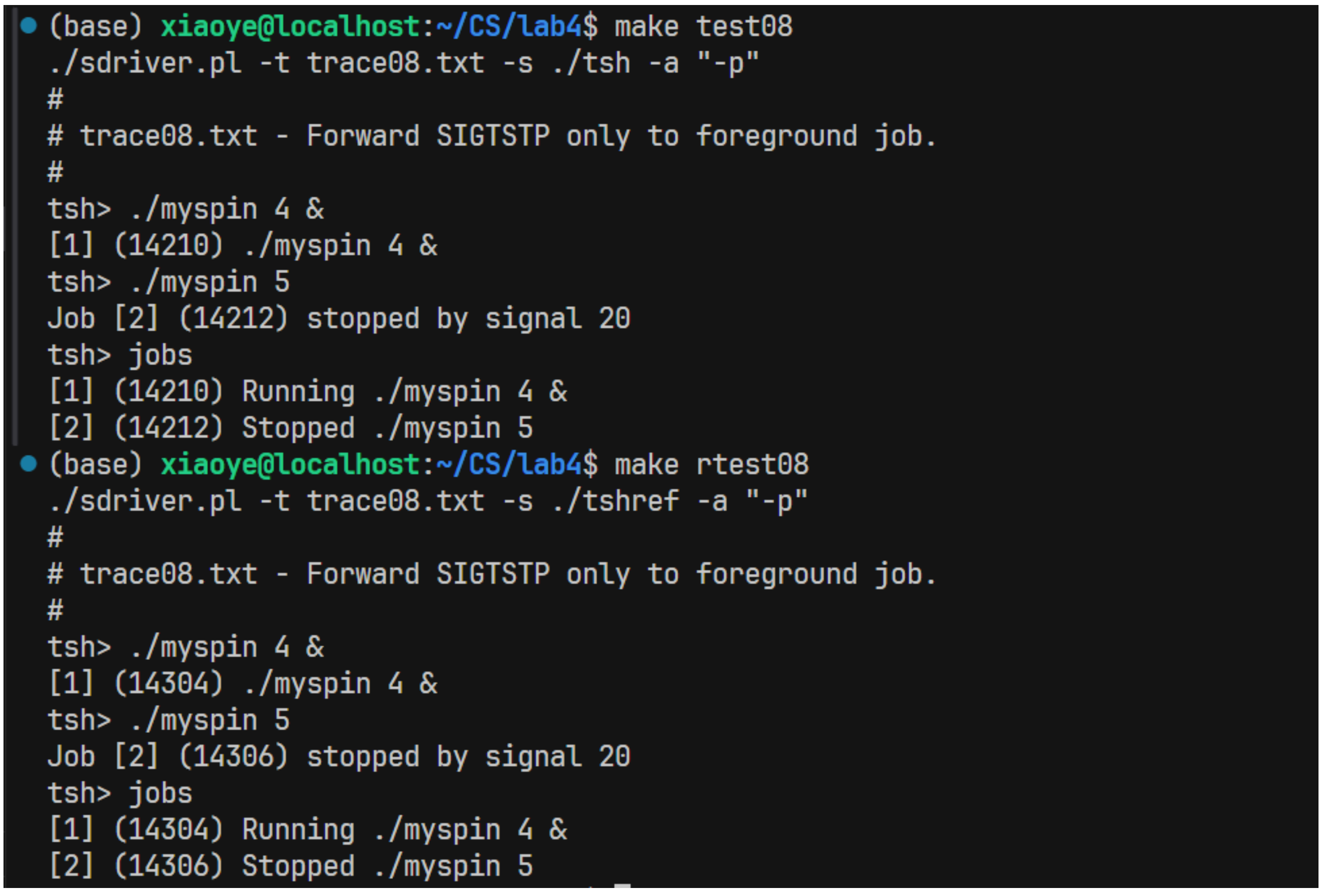
test08-仅将SIGTSTP转发到前台作业

```
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

/bin/echo -e tsh> ./myspin 5
./myspin 5

SLEEP 2
TSTP

/bin/echo tsh> jobs
jobs
```

trace08.txt

- 这里和上面最大的区别就是转发的信号发生了变化。上面是SIGINT，这里是SIGTSTP（模拟crtl+z）。它是使得向前台作业的进程组挂起（暂停执行），但不终止进程。并不会从进程列表中删去它，只是将进程的状态改成ST（暂停）。
- 同样我们运行一个前台进程，一个后台进程，发送信号SIGSTP，使用jobs打印出进程的信息。

验证



```
(base) xiaoye@localhost:~/CS/lab4$ make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (14210) ./myspin 4 &
tsh> ./myspin 5
Job [2] (14212) stopped by signal 20
tsh> jobs
[1] (14210) Running ./myspin 4 &
[2] (14212) Stopped ./myspin 5
(base) xiaoye@localhost:~/CS/lab4$ make rtest08
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (14304) ./myspin 4 &
tsh> ./myspin 5
Job [2] (14306) stopped by signal 20
tsh> jobs
[1] (14304) Running ./myspin 4 &
[2] (14306) Stopped ./myspin 5
```

我们发现此时仍然可以在作业列表中看到前台作业。并且它很明显被标注了”Stopped“。

test09-进程bg内置命令

```
#
# trace09.txt - Process bg builtin command
#
/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

/bin/echo -e tsh> ./myspin 5
./myspin 5

SLEEP 2
TSTP

/bin/echo tsh> jobs
jobs

/bin/echo tsh> bg %2
bg %2
```

```
/bin/echo tsh> jobs
jobs
```

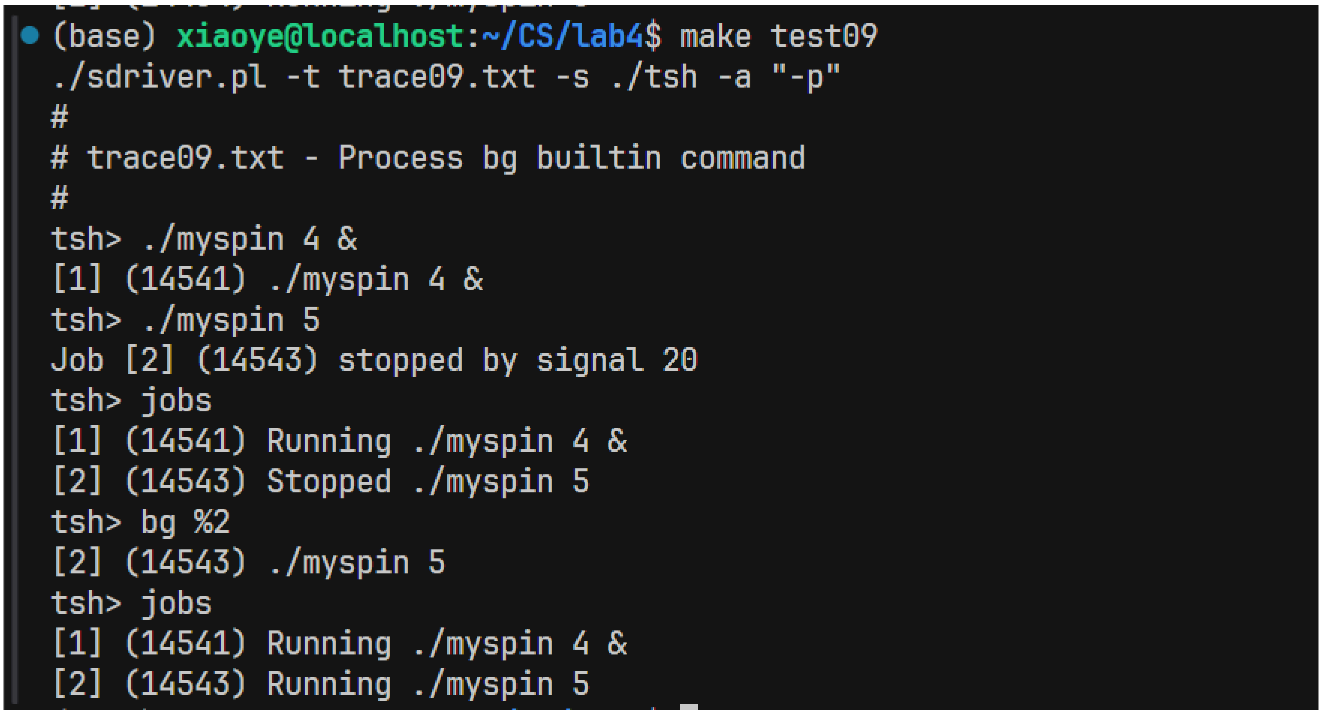
trace09.txt

- 作业1是后台作业，作业2是前台作业。
- 发送信号SIGTSTP到前台进程组的所有进程，挂起它们。（那么myspin 5就被挂起了）
- jobs查看作业列表。
- bg %2,这一行命令，首先调用eval（）解析参数，然后调用builtin_cmd（）判断是不是内置命令。bg是内置命令，调用do_bgfg（），切换/恢复到前台/后台运行。我们这里将作业2恢复到后台运行。
- 然后再次查看作业列表。

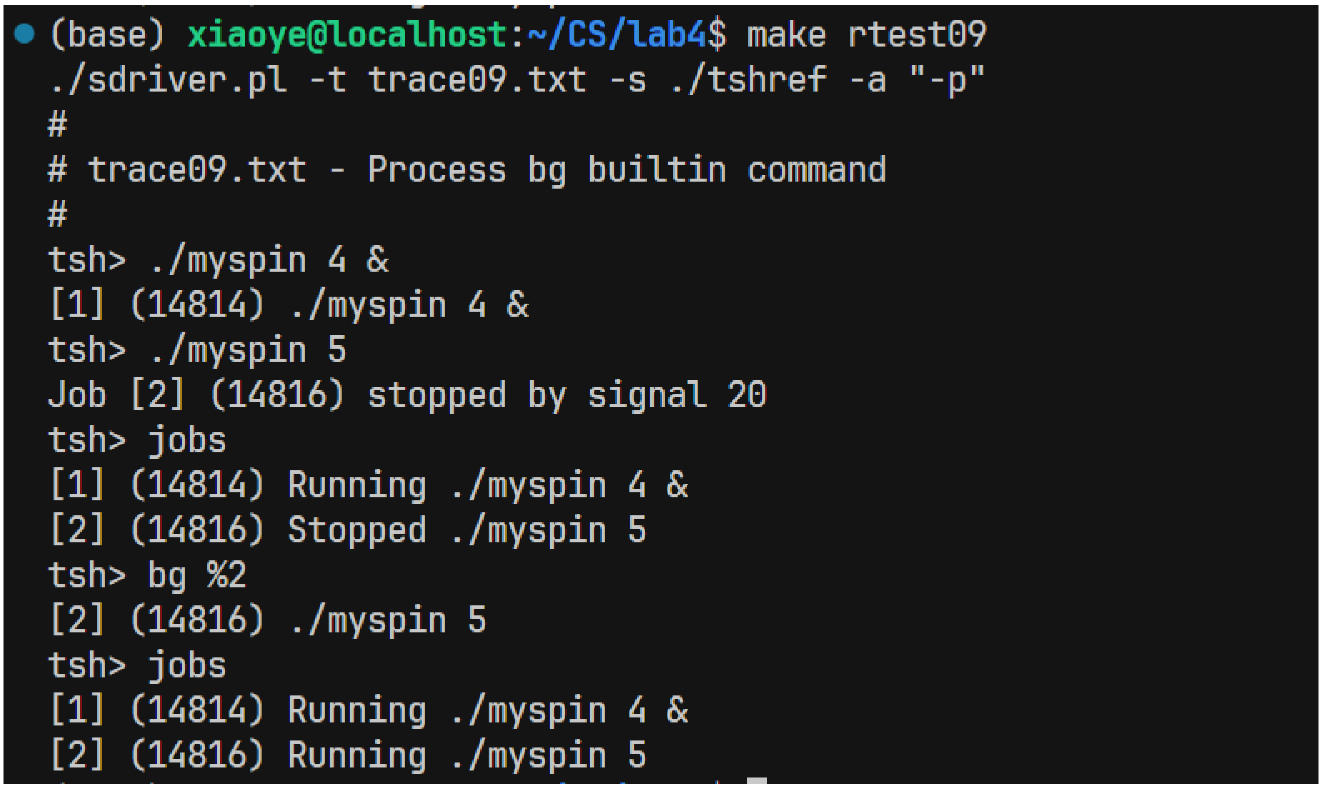
```
/*

 * do_bgfg - 执行bg和fg命令

 */

void do_bgfg(char **argv)

{

    int num;

    struct job_t *job;

    // 没有参数的fg/bg不符合规定

    if (!argv[1])

    { // 命令行为空

        printf("%s command requires PID or %%jobid argument\n", argv[0]);

        return;

    }

    // 检测fg/bg参数，其中%开头的数字是JobID，纯数字的是PID

    // 找到jobID或PID后通过这个找出job
    if (argv[1][0] == '%')

    {                                                   // 解析jid

        if ((num = strtol(&argv[1][1], NULL, 10)) <= 0) // 获取jid

        {

            printf("%s: argument must be a PID or %%jobid\n", argv[0]); //
失败,打印错误消息

            return;

        }

        if ((job = getjobjid(jobs, num)) == NULL) // 根据jid获取job

        {

            printf("%%%d: No such job\n", num); // 没找到对应的job

            return;

        }

    }

    // strtol函数原型：long int strtol(const char *nptr, char **endptr, int
base);

    // strtol函数会将参数nptr字符串根据参数base来转换成长整型数，参数base范围从2至
36。

    else

    {                                               // 解析PID

        if ((num = strtol(argv[1], NULL, 10)) <= 0) // 获取PID

        {

            printf("%s: argument must be a PID or %%jobid\n", argv[0]); //
失败,打印错误消息
```

```
            return;

        }

        if ((job = getjobpid(jobs, num)) == NULL) // 根据PID获取job

        {

            printf("(%d): No such process\n", num); // 没找到对应的进程

            return;

        }

    }

    //SIGCONT 是Linux/Unix系统中用于恢复被暂停进程执行的信号，其名称源于
"Continue"（继续）



    // kill函数原型： int kill(pid_t pid,int signo)

    // pid > 0：将信号发送给进程 ID 为 pid 的进程。

    // pid ==0：将信号发送给与发送进程属于同一进程组的所有进程。

    // pid < 0：将信号发送给进程组 ID 等于 pid 的绝对值的所有进程。

    // pid ==-1：将信号发送给系统中所有进程。

    if (!strcmp(argv[0], "bg")) // 该进程需要在后台运行

    {

        // bg会启动子进程，并将其放置于后台执行

        job->state = BG;                  // 设置状态BG

        if (kill(-job->pid, SIGCONT) < 0) // 采用负数发送信号到进程组

            unix_error("kill error");
        printf("[%d] (%d) %s", job->jid, job->pid, job->cmdline);

    }

    else if (!strcmp(argv[0], "fg")) // 该进程需要在前台运行

    {

        job->state = FG;                  // 设置状态FG

        if (kill(-job->pid, SIGCONT) < 0) // 采用负数发送信号到进程组

            unix_error("kill error");

        // 当一个进程被设置为前台执行时，当前tsh应该等待该子进程结束

        waitfg(job->pid);

    }

    else // 指令出现异常

    {

        puts("do_bgfg: Internal error");

        exit(0);

    }

    return;

}
```

验证



```
(base) xiaoye@localhost:~/CS/lab4$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (14541) ./myspin 4 &
tsh> ./myspin 5
Job [2] (14543) stopped by signal 20
tsh> jobs
[1] (14541) Running ./myspin 4 &
[2] (14543) Stopped ./myspin 5
tsh> bg %2
[2] (14543) ./myspin 5
tsh> jobs
[1] (14541) Running ./myspin 4 &
[2] (14543) Running ./myspin 5
```



```
(base) xiaoye@localhost:~/CS/lab4$ make rtest09
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (14814) ./myspin 4 &
tsh> ./myspin 5
Job [2] (14816) stopped by signal 20
tsh> jobs
[1] (14814) Running ./myspin 4 &
[2] (14816) Stopped ./myspin 5
tsh> bg %2
[2] (14816) ./myspin 5
tsh> jobs
[1] (14814) Running ./myspin 4 &
[2] (14816) Running ./myspin 5
```

myspin 5的状态确实从stopped转换成了running

test10-进程fg内置命令

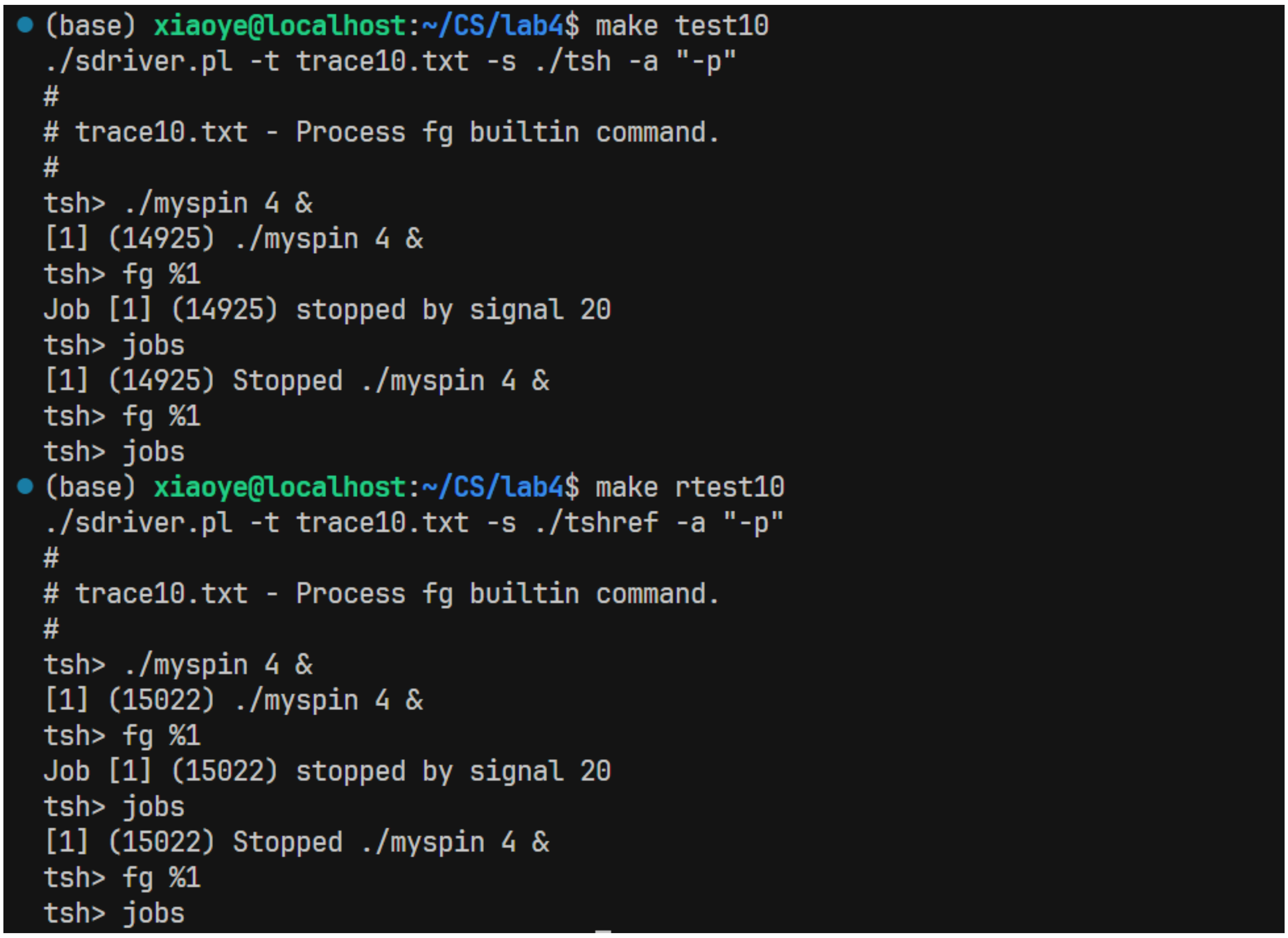
```
#
# trace10.txt - Process fg builtin command.
#
/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &
```

```
SLEEP 1
/bin/echo tsh> fg %1
fg %1

SLEEP 1
TSTP

/bin/echo tsh> jobs
jobs

/bin/echo tsh> fg %1
fg %1

/bin/echo tsh> jobs
jobs
```

trace10.txt

- 首先我们是建立一个后台作业myspin 4 &;
- 然后我们调用内置命令fg，将它切换到前台运行;
- 发送信号SIGTSTP到前台进程组的所有进程，挂起它;
- jobs查看作业列表;
- 再次调用内置命令fg，将被暂停的进程恢复到前台运行;
- jobs查看作业列表。

验证



```
● (base) xiaoye@localhost:~/CS/lab4$ make test10
  ./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
  #
  # trace10.txt - Process fg builtin command.
  #
  tsh> ./myspin 4 &
  [1] (14925) ./myspin 4 &
  tsh> fg %1
  Job [1] (14925) stopped by signal 20
  tsh> jobs
  [1] (14925) Stopped ./myspin 4 &
  tsh> fg %1
  tsh> jobs
● (base) xiaoye@localhost:~/CS/lab4$ make rtest10
  ./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
  #
  # trace10.txt - Process fg builtin command.
  #
  tsh> ./myspin 4 &
  [1] (15022) ./myspin 4 &
  tsh> fg %1
  Job [1] (15022) stopped by signal 20
  tsh> jobs
  [1] (15022) Stopped ./myspin 4 &
  tsh> fg %1
  tsh> jobs
```

test11-将SIGINT转发给前台进程组中的每个进程

```
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
/bin/echo -e tsh> ./mysplit 4
./mysplit 4

SLEEP 2
INT

/bin/echo tsh> /bin/ps a
/bin/ps a
```

trace11.txt

- 首先我们看一下mysplit：创建子进程

```
/*
 * mysplit.c - 用于测试微型shell的进程控制工具
 *
 * 用法: mysplit <n>
 * 功能: 创建一个子进程，该子进程以1秒为间隔休眠n次（总耗时n秒）
 */
#include <stdio.h>      // 标准输入输出
#include <unistd.h>     // fork(), sleep()
#include <stdlib.h>     // exit(), atoi()
#include <sys/types.h>  // pid_t类型定义
#include <sys/wait.h>   // wait()函数
#include <signal.h>

int main(int argc, char **argv)
{
    int i, secs;  // 定义循环计数器和休眠总秒数

    // 检查参数数量是否正确（程序名 + 1个数字参数）
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <n>\n", argv[0]);  // 输出错误提示到标准错误
        exit(0);  // 非零退出码表示错误
    }

    secs = atoi(argv[1]);  // 将输入参数转换为整数（注意：atoi不安全，建议用
strtol）

    // 创建子进程
    if (fork() == 0) { /* 子进程 */
        for (i=0; i < secs; i++)  // 循环休眠n次
            sleep(1);  // 每次休眠1秒
        exit(0);  // 子进程正常退出
    }

    /* 父进程 */
    wait(NULL);  // 等待任意子进程结束（阻塞直到子进程终止）
    exit(0);  // 父进程退出
}
```

- 我们将SIGINT发送给前台进程组的所有进程
- ps -a显示所有进程，用它来查看是不是进程组中的每个进程都停止了

```
(base) xiaoye@localhost:~/CS/lab4$ 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] (15249) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 hvc0     Sl+    0:00 /init
    6 hvc0     Sl+    0:00 plan9 --control-socket 6 --log-level 4 --server-
fd 7 --pipe-fd 9 --log-truncate
   15 pts/0    Ss+    0:00 sh -c
"$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh"
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   16 pts/0    S+     0:00 sh /mnt/c/Users/23844/.vscode/extensions/ms-
vscode-remote.remote-wsl-0.99.0/scripts/wslServer.sh
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   21 pts/0    S+     0:00 sh /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/bin/code-server --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   25 pts/0    Sl+    0:25 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/server-main.js --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   83 pts/0    Sl+    0:03 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=fileWatcher
  142 pts/0    Sl+    2:12 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node --dns-result-
order=ipv4first /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=extensionHost --transformURIs --useHostProxy=true
  181 pts/0    Sl+    0:16 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
```

```
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=ptyHost --logsPath /home/xiaoye/.vscode-
server/data/logs/20250519T144148
  208 pts/4    Ss     0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
  228 pts/0    Sl+    0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/extensions/json-
language-features/server/dist/node/jsonServerMain --node-ipc --
clientProcessId=142
  237 pts/0    Sl+    0:06 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools
  345 pts/0    Sl+    0:02 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools-srv 237 {498D9CA4-BC79-4335-
AFA1-2124C17844AC}
14176 pts/1    Ssl+   0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
14185 pts/2    Ssl+   0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
15114 pts/0    Sl+    0:00 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools-srv 237 {21439A5D-76ED-4F3F-
83A4-246A98973DDC}
15152 pts/3    Ss+    0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
15244 pts/4    S+     0:00 make test11
15245 pts/4    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -
a "-p"
```

```
15246 pts/4    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s
./tsh -a -p
15247 pts/4    S+     0:00 ./tsh -p
15255 pts/0    S+     0:00 /bin/sh -c "/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh" 208 15244 15245 15246 15247 15249 15250
15256 pts/0    R+     0:00 /bin/bash /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh 208 15244 15245 15246 15247 15249 15250
15271 pts/4    R      0:00 /bin/ps a
```

```
(base) xiaoye@localhost:~/CS/lab4$ 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] (15483) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 hvc0     Sl+    0:00 /init
    6 hvc0     Sl+    0:00 plan9 --control-socket 6 --log-level 4 --server-
fd 7 --pipe-fd 9 --log-truncate
   15 pts/0    Ss+    0:00 sh -c
"$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh"
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   16 pts/0    S+     0:00 sh /mnt/c/Users/23844/.vscode/extensions/ms-
vscode-remote.remote-wsl-0.99.0/scripts/wslServer.sh
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   21 pts/0    S+     0:00 sh /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/bin/code-server --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   25 pts/0    Sl+    0:26 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
```

```
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/server-main.js --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   83 pts/0    Sl+    0:03 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=fileWatcher
  142 pts/0    Sl+    2:13 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node --dns-result-
order=ipv4first /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=extensionHost --transformURIs --useHostProxy=true
  181 pts/0    Sl+    0:17 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=ptyHost --logsPath /home/xiaoye/.vscode-
server/data/logs/20250519T144148
  208 pts/4    Ss     0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
  228 pts/0    Sl+    0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/extensions/json-
language-features/server/dist/node/jsonServerMain --node-ipc --
clientProcessId=142
  237 pts/0    Sl+    0:06 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools
  345 pts/0    Sl+    0:02 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools-srv 237 {498D9CA4-BC79-4335-
AFA1-2124C17844AC}
14176 pts/1    Ssl+   0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
14185 pts/2    Ssl+   0:01 /home/xiaoye/.vscode-
```

```
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
15114 pts/0    Sl+    0:00 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools-srv 237 {21439A5D-76ED-4F3F-
83A4-246A98973DDC}
15152 pts/3    Ss+    0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
15478 pts/4    S+     0:00 make rtest11
15479 pts/4    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s
./tshref -a "-p"
15480 pts/4    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s
./tshref -a -p
15481 pts/4    S+     0:00 ./tshref -p
15489 pts/0    S+     0:00 /bin/sh -c "/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh" 208 15478 15479 15480 15481 15483 15484
15490 pts/0    R+     0:00 /bin/bash /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh 208 15478 15479 15480 15481 15483 15484
15501 pts/4    R      0:00 /bin/ps a
```

自己编写的和参考的tinyshell输出是一致的。

test12-将SIGTSTP转发到前台进程组中的每个进程

```
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
/bin/echo -e tsh> ./mysplit 4
./mysplit 4

SLEEP 2
TSTP

/bin/echo tsh> jobs
jobs
```

```
/bin/echo tsh> /bin/ps a
/bin/ps a
```

trace12.txt

- 测试将SIGTSTP转发给前台进程组中的每个进程，和test11相近

验证

```
(base) xiaoye@localhost:~/CS/lab4$ 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] (15798) stopped by signal 20
tsh> jobs
[1] (15798) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 hvc0     Sl+    0:00 /init
    6 hvc0     Sl+    0:00 plan9 --control-socket 6 --log-level 4 --server-
fd 7 --pipe-fd 9 --log-truncate
   15 pts/0    Ss+    0:00 sh -c
"$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh"
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   16 pts/0    S+     0:00 sh /mnt/c/Users/23844/.vscode/extensions/ms-
vscode-remote.remote-wsl-0.99.0/scripts/wslServer.sh
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   21 pts/0    S+     0:00 sh /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/bin/code-server --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   25 pts/0    Sl+    0:28 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
```

```
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/server-main.js --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   83 pts/0    Sl+    0:03 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=fileWatcher
  142 pts/0    Sl+    2:16 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node --dns-result-
order=ipv4first /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=extensionHost --transformURIs --useHostProxy=true
  181 pts/0    Sl+    0:18 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=ptyHost --logsPath /home/xiaoye/.vscode-
server/data/logs/20250519T144148
  208 pts/4    Ss     0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
  228 pts/0    Sl+    0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/extensions/json-
language-features/server/dist/node/jsonServerMain --node-ipc --
clientProcessId=142
  237 pts/0    Sl+    0:07 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools
  345 pts/0    Sl+    0:02 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools-srv 237 {498D9CA4-BC79-4335-
AFA1-2124C17844AC}
14176 pts/1    Ssl+   0:02 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
14185 pts/2    Ssl+   0:01 /home/xiaoye/.vscode-
```

```
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
15152 pts/3    Ss+    0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
15793 pts/4    S+     0:00 make test12
15794 pts/4    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -
a "-p"
15795 pts/4    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s
./tsh -a -p
15796 pts/4    S+     0:00 ./tsh -p
15798 pts/4    T      0:00 ./mysplit 4
15799 pts/4    T      0:00 ./mysplit 4
15804 pts/0    S+     0:00 /bin/sh -c "/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh" 208 15793 15794 15795 15796 15798 15799
15805 pts/0    R+     0:00 /bin/bash /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh 208 15793 15794 15795 15796 15798 15799
15822 pts/4    R      0:00 /bin/ps a
```

```
(base) xiaoye@localhost:~/CS/lab4$ 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] (15859) stopped by signal 20
tsh> jobs
[1] (15859) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 hvc0     Sl+    0:00 /init
    6 hvc0     Sl+    0:00 plan9 --control-socket 6 --log-level 4 --server-
fd 7 --pipe-fd 9 --log-truncate
   15 pts/0    Ss+    0:00 sh -c
"$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh"
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
```

```
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   16 pts/0    S+     0:00 sh /mnt/c/Users/23844/.vscode/extensions/ms-
vscode-remote.remote-wsl-0.99.0/scripts/wslServer.sh
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   21 pts/0    S+     0:00 sh /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/bin/code-server --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   25 pts/0    Rl+    0:28 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/server-main.js --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   83 pts/0    Sl+    0:03 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=fileWatcher
  142 pts/0    Sl+    2:16 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node --dns-result-
order=ipv4first /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=extensionHost --transformURIs --useHostProxy=true
  181 pts/0    Sl+    0:18 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=ptyHost --logsPath /home/xiaoye/.vscode-
server/data/logs/20250519T144148
  208 pts/4    Ss     0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
  228 pts/0    Sl+    0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
```

```
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/extensions/json-
language-features/server/dist/node/jsonServerMain --node-ipc --
clientProcessId=142
  237 pts/0    Sl+    0:07 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools
  345 pts/0    Sl+    0:02 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools-srv 237 {498D9CA4-BC79-4335-
AFA1-2124C17844AC}
14176 pts/1    Ssl+   0:02 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
14185 pts/2    Ssl+   0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
15152 pts/3    Ss+    0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
15854 pts/4    S+     0:00 make rtest12
15855 pts/4    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s
./tshref -a "-p"
15856 pts/4    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s
./tshref -a -p
15857 pts/4    S+     0:00 ./tshref -p
15859 pts/4    T      0:00 ./mysplit 4
15860 pts/4    T      0:00 ./mysplit 4
15868 pts/0    S+     0:00 /bin/sh -c "/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh" 208 15854 15855 15856 15857 15859 15860
15869 pts/0    R+     0:00 /bin/bash /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh 208 15854 15855 15856 15857 15859 15860
15885 pts/4    R      0:00 /bin/ps a
```

自己编写的和参考的tinyshell输出是一致的。

test13-重新启动进程组中的每个已停止的进程

```
#
# trace13.txt - Restart every stopped process in process group
#
/bin/echo -e tsh> ./mysplit 4
./mysplit 4

SLEEP 2
TSTP

/bin/echo tsh> jobs
jobs

/bin/echo tsh> /bin/ps a
/bin/ps a

/bin/echo tsh> fg %1
fg %1

/bin/echo tsh> /bin/ps a
/bin/ps a
```

trace13.txt

- 这里和test10最大的区别是：使用/bin/ps a测试fg是否可以唤醒进程组中的每个进程

验证

```
(base) xiaoye@localhost:~/CS/lab4$ 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] (15980) stopped by signal 20
tsh> jobs
[1] (15980) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 hvc0     Sl+    0:00 /init
    6 hvc0     Sl+    0:00 plan9 --control-socket 6 --log-level 4 --server-
fd 7 --pipe-fd 9 --log-truncate
   15 pts/0    Ss+    0:00 sh -c
"$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh"
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   16 pts/0    S+     0:00 sh /mnt/c/Users/23844/.vscode/extensions/ms-
vscode-remote.remote-wsl-0.99.0/scripts/wslServer.sh
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   21 pts/0    S+     0:00 sh /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/bin/code-server --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   25 pts/0    Sl+    0:28 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/server-main.js --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   83 pts/0    Sl+    0:03 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=fileWatcher
  142 pts/0    Sl+    2:17 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node --dns-result-
order=ipv4first /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=extensionHost --transformURIs --useHostProxy=true
  181 pts/0    Sl+    0:19 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=ptyHost --logsPath /home/xiaoye/.vscode-
server/data/logs/20250519T144148
  208 pts/4    Ss     0:00 /bin/bash --init-file /home/xiaoye/.vscode-
```

```
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
  228 pts/0    Sl+    0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/extensions/json-
language-features/server/dist/node/jsonServerMain --node-ipc --
clientProcessId=142
  237 pts/0    Sl+    0:07 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools
  345 pts/0    Sl+    0:02 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools-srv 237 {498D9CA4-BC79-4335-
AFA1-2124C17844AC}
14176 pts/1    Ssl+   0:02 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
14185 pts/2    Ssl+   0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
15152 pts/3    Ss+    0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
15975 pts/4    S+     0:00 make test13
15976 pts/4    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -
a "-p"
15977 pts/4    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s
./tsh -a -p
15978 pts/4    S+     0:00 ./tsh -p
15980 pts/4    T      0:00 ./mysplit 4
15981 pts/4    T      0:00 ./mysplit 4
15986 pts/0    S+     0:00 /bin/sh -c "/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh" 208 15975 15976 15977 15978 15980 15981
```

```
15987 pts/0    S+     0:00 /bin/bash /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh 208 15975 15976 15977 15978 15980 15981
15996 pts/0    S+     0:00 sleep 1
15999 pts/4    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 hvc0     Sl+    0:00 /init
    6 hvc0     Sl+    0:00 plan9 --control-socket 6 --log-level 4 --server-
fd 7 --pipe-fd 9 --log-truncate
   15 pts/0    Ss+    0:00 sh -c
"$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh"
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   16 pts/0    S+     0:00 sh /mnt/c/Users/23844/.vscode/extensions/ms-
vscode-remote.remote-wsl-0.99.0/scripts/wslServer.sh
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   21 pts/0    S+     0:00 sh /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/bin/code-server --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   25 pts/0    Sl+    0:28 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/server-main.js --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   83 pts/0    Sl+    0:03 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=fileWatcher
  142 pts/0    Sl+    2:17 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node --dns-result-
order=ipv4first /home/xiaoye/.vscode-
```

```
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=extensionHost --transformURIs --useHostProxy=true
  181 pts/0    Sl+    0:19 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=ptyHost --logsPath /home/xiaoye/.vscode-
server/data/logs/20250519T144148
  208 pts/4    Ss     0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
  228 pts/0    Sl+    0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/extensions/json-
language-features/server/dist/node/jsonServerMain --node-ipc --
clientProcessId=142
  237 pts/0    Sl+    0:07 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools
  345 pts/0    Sl+    0:02 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools-srv 237 {498D9CA4-BC79-4335-
AFA1-2124C17844AC}
14176 pts/1    Ssl+   0:02 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
14185 pts/2    Ssl+   0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
15152 pts/3    Ss+    0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
15975 pts/4    S+     0:00 make test13
15976 pts/4    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -
```

```
a "-p"
15977 pts/4    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s
./tsh -a -p
15978 pts/4    S+     0:00 ./tsh -p
16020 pts/0    S+     0:00 /bin/sh -c "/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh" 208 15975 15976 15977 15978 15980 15981
16021 pts/0    S+     0:00 /bin/bash /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh 208 15975 15976 15977 15978 15980 15981
16030 pts/0    S+     0:00 sleep 1
16032 pts/4    R      0:00 /bin/ps a
```

```
(base) xiaoye@localhost:~/CS/lab4$ 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] (16073) stopped by signal 20
tsh> jobs
[1] (16073) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 hvc0     Sl+    0:00 /init
    6 hvc0     Sl+    0:00 plan9 --control-socket 6 --log-level 4 --server-
fd 7 --pipe-fd 9 --log-truncate
   15 pts/0    Ss+    0:00 sh -c
"$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh"
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   16 pts/0    S+     0:00 sh /mnt/c/Users/23844/.vscode/extensions/ms-
vscode-remote.remote-wsl-0.99.0/scripts/wslServer.sh
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   21 pts/0    S+     0:00 sh /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/bin/code-server --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
```

```
compression --accept-server-license-terms --telemetry-level=all
   25 pts/0    Sl+    0:28 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/server-main.js --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   83 pts/0    Sl+    0:03 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=fileWatcher
  142 pts/0    Sl+    2:17 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node --dns-result-
order=ipv4first /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=extensionHost --transformURIs --useHostProxy=true
  181 pts/0    Sl+    0:19 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=ptyHost --logsPath /home/xiaoye/.vscode-
server/data/logs/20250519T144148
  208 pts/4    Ss     0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
  228 pts/0    Sl+    0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/extensions/json-
language-features/server/dist/node/jsonServerMain --node-ipc --
clientProcessId=142
  237 pts/0    Sl+    0:07 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools
  345 pts/0    Sl+    0:02 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools-srv 237 {498D9CA4-BC79-4335-
AFA1-2124C17844AC}
14176 pts/1    Ssl+   0:02 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
```

```
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
14185 pts/2    Ssl+   0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
15152 pts/3    Ss+    0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
16068 pts/4    S+     0:00 make rtest13
16069 pts/4    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s
./tshref -a "-p"
16070 pts/4    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s
./tshref -a -p
16071 pts/4    S+     0:00 ./tshref -p
16073 pts/4    T      0:00 ./mysplit 4
16074 pts/4    T      0:00 ./mysplit 4
16079 pts/0    S+     0:00 /bin/sh -c "/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh" 208 16068 16069 16070 16071 16073 16074
16080 pts/0    S+     0:00 /bin/bash /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh 208 16068 16069 16070 16071 16073 16074
16094 pts/4    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 hvc0     Sl+    0:00 /init
    6 hvc0     Sl+    0:00 plan9 --control-socket 6 --log-level 4 --server-
fd 7 --pipe-fd 9 --log-truncate
   15 pts/0    Ss+    0:00 sh -c
"$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh"
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   16 pts/0    S+     0:00 sh /mnt/c/Users/23844/.vscode/extensions/ms-
vscode-remote.remote-wsl-0.99.0/scripts/wslServer.sh
```

```
848b80aeb52026648a8ff9f7c45a9b0a80641e2e stable code-server .vscode-server -
-host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   21 pts/0    S+     0:00 sh /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/bin/code-server --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   25 pts/0    Sl+    0:28 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/server-main.js --
host=127.0.0.1 --port=0 --connection-token=2367371195-3526608509-835105246-
3545820565 --use-host-proxy --without-browser-env-var --disable-websocket-
compression --accept-server-license-terms --telemetry-level=all
   83 pts/0    Sl+    0:03 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=fileWatcher
  142 pts/0    Sl+    2:17 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node --dns-result-
order=ipv4first /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=extensionHost --transformURIs --useHostProxy=true
  181 pts/0    Sl+    0:19 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/bootstrap-fork --
type=ptyHost --logsPath /home/xiaoye/.vscode-
server/data/logs/20250519T144148
  208 pts/4    Ss     0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
  228 pts/0    Sl+    0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node
/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/extensions/json-
language-features/server/dist/node/jsonServerMain --node-ipc --
clientProcessId=142
  237 pts/0    Sl+    0:07 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools
```

```
  345 pts/0    Sl+    0:02 /home/xiaoye/.vscode-server/extensions/ms-
vscode.cpptools-1.25.3-linux-x64/bin/cpptools-srv 237 {498D9CA4-BC79-4335-
AFA1-2124C17844AC}
14176 pts/1    Ssl+   0:02 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
14185 pts/2    Ssl+   0:01 /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/node -e const net =
require('net'); process.stdin.pause(); const client = net.createConnection({
host: '127.0.0.1', port: 40587 }, () => { client.pipe(process.stdout);
process.stdin.pipe(client); }); client.on('close', function (hadError) {
console.error(hadError ? 'Remote close with error' : 'Remote close');
process.exit(hadError ? 1 : 0); }); client.on('error', function (err) {
process.stderr.write(err && (err.stack || err.message) || String(err)); });
15152 pts/3    Ss+    0:00 /bin/bash --init-file /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/workbench/contrib
/terminal/common/scripts/shellIntegration-bash.sh
16068 pts/4    S+     0:00 make rtest13
16069 pts/4    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s
./tshref -a "-p"
16070 pts/4    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s
./tshref -a -p
16071 pts/4    S+     0:00 ./tshref -p
16113 pts/0    S+     0:00 /bin/sh -c "/home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh" 15152
16114 pts/0    S+     0:00 /bin/bash /home/xiaoye/.vscode-
server/bin/848b80aeb52026648a8ff9f7c45a9b0a80641e2e/out/vs/base/node/cpuUsag
e.sh 15152
16117 pts/0    S+     0:00 sleep 1
16119 pts/4    R      0:00 /bin/ps a
```

可以看到自己编写和参考的tinyshell输出是一致的。

test14-简单的错误处理

```
#
# trace14.txt - Simple error handling
```

```
#
/bin/echo tsh> ./bogus
./bogus

/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

/bin/echo tsh> fg
fg

/bin/echo tsh> bg
bg

/bin/echo tsh> fg a
fg a

/bin/echo tsh> bg a
bg a

/bin/echo tsh> fg 9999999
fg 9999999

/bin/echo tsh> bg 9999999
bg 9999999

/bin/echo tsh> fg %2
fg %2

/bin/echo tsh> fg %1
fg %1

SLEEP 2
TSTP

/bin/echo tsh> bg %2
bg %2

/bin/echo tsh> bg %1
bg %1

/bin/echo tsh> jobs
jobs
```

trace14.txt

- 这里面有几行用于错误测试的。
- 第1~2行测试执行不存在的脚本./bogus的错误处理；
- 第9~16行测试无效作业 ID 的错误处理。

验证

```
(base) xiaoye@localhost:~/CS/lab4$ 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] (16199) ./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] (16199) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (16199) ./myspin 4 &
tsh> jobs
[1] (16199) Running ./myspin 4 &
```

```
(base) xiaoye@localhost:~/CS/lab4$ 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] (16321) ./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] (16321) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (16321) ./myspin 4 &
tsh> jobs
[1] (16321) Running ./myspin 4 &
```

test15-综合测试

```
#
# trace15.txt - Putting it all together
#

/bin/echo tsh> ./bogus
./bogus             # 错误测试

/bin/echo tsh> ./myspin 10
./myspin 10         # 前台进程

SLEEP 2
```

```
INT                  # 发送信号SIGINT

/bin/echo -e tsh> ./myspin 3 \046
./myspin 3 &        # 后台进程

/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

/bin/echo tsh> jobs
jobs                # 查看进程列表

/bin/echo tsh> fg %1
fg %1               # 进程1切换到前台运行

SLEEP 2
TSTP                # 挂起前台进程

/bin/echo tsh> jobs
jobs               # 查看进程列表

/bin/echo tsh> bg %3
bg %3              # 进程3切换到前台运行

/bin/echo tsh> bg %1
bg %1              # 进程1切换到后台运行

/bin/echo tsh> jobs
jobs               # 查看进程列表

/bin/echo tsh> fg %1
fg %1             # 进程1切换到前台运行

/bin/echo tsh> quit
quit              # quit
```

验证

```
(base) xiaoye@localhost:~/CS/lab4$ 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] (16456) terminated by signal 2
tsh> ./myspin 3 &
[1] (16478) ./myspin 3 &
tsh> ./myspin 4 &
[2] (16481) ./myspin 4 &
tsh> jobs
[1] (16478) Running ./myspin 3 &
[2] (16481) Running ./myspin 4 &
tsh> fg %1
Job [1] (16478) stopped by signal 20
tsh> jobs
[1] (16478) Stopped ./myspin 3 &
[2] (16481) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (16478) ./myspin 3 &
tsh> jobs
[1] (16478) Running ./myspin 3 &
[2] (16481) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

```
(base) xiaoye@localhost:~/CS/lab4$ 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] (16545) terminated by signal 2
tsh> ./myspin 3 &
[1] (16564) ./myspin 3 &
tsh> ./myspin 4 &
[2] (16567) ./myspin 4 &
tsh> jobs
[1] (16564) Running ./myspin 3 &
[2] (16567) Running ./myspin 4 &
tsh> fg %1
Job [1] (16564) stopped by signal 20
```

```
tsh> jobs
[1] (16564) Stopped ./myspin 3 &
[2] (16567) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (16564) ./myspin 3 &
tsh> jobs
[1] (16564) Running ./myspin 3 &
[2] (16567) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

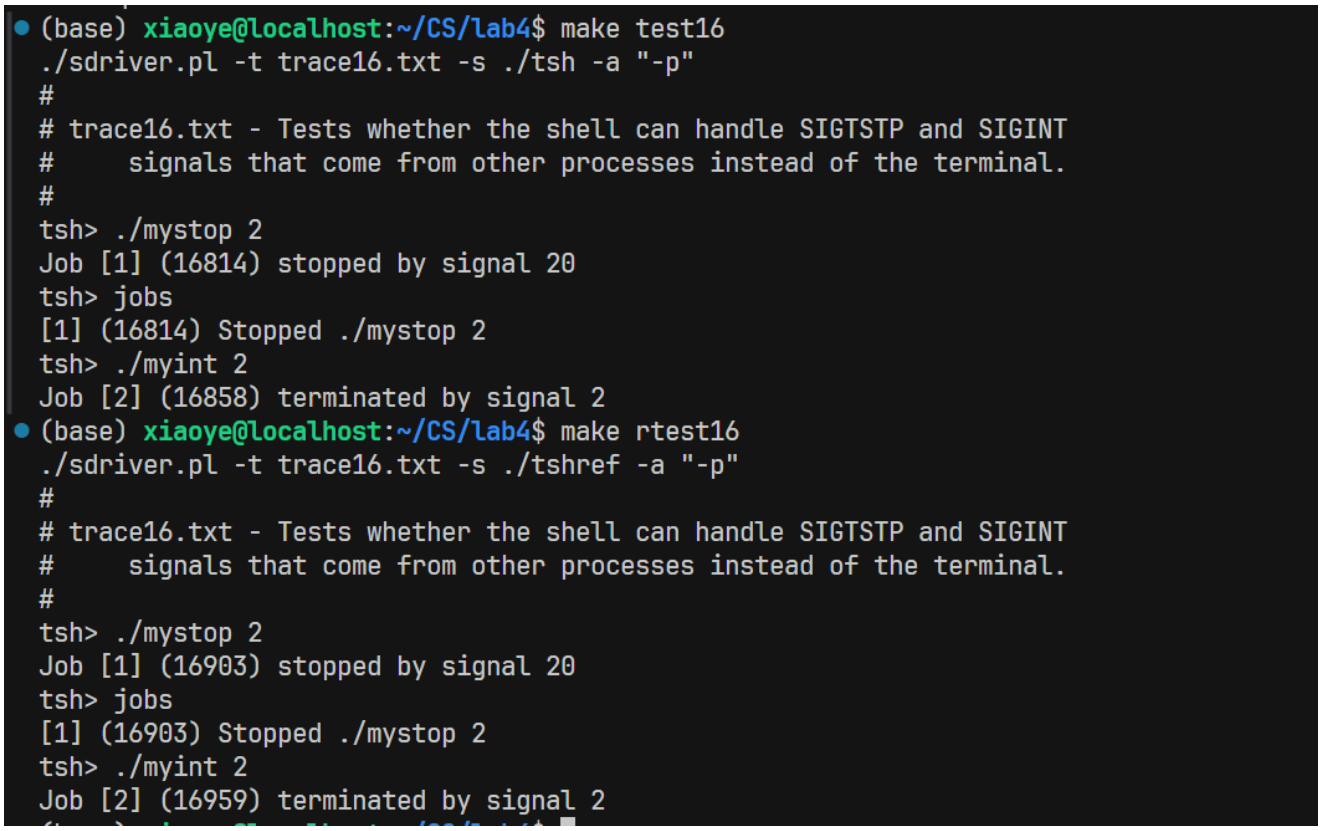
test16-来自其他进程的信号

```
#
# trace16.txt - Tests whether the shell can handle SIGTSTP and SIGINT
#     signals that come from other processes instead of the terminal.
#

/bin/echo tsh> ./mystop 2
./mystop 2

SLEEP 3

/bin/echo tsh> jobs
jobs

/bin/echo tsh> ./myint 2
./myint 2
```

trace16.txt

- mystop.c:程序休眠指定的秒数后，向自身所在的进程组发送 `SIGTSTP` 信号;
- myint.c:程序休眠指定的秒数后，向自身发送 `SIGINT` 信号;
- 测试shell是否可以处理来自其他进程而不是终端的SIGTSTP和SIGINT信号。

验证

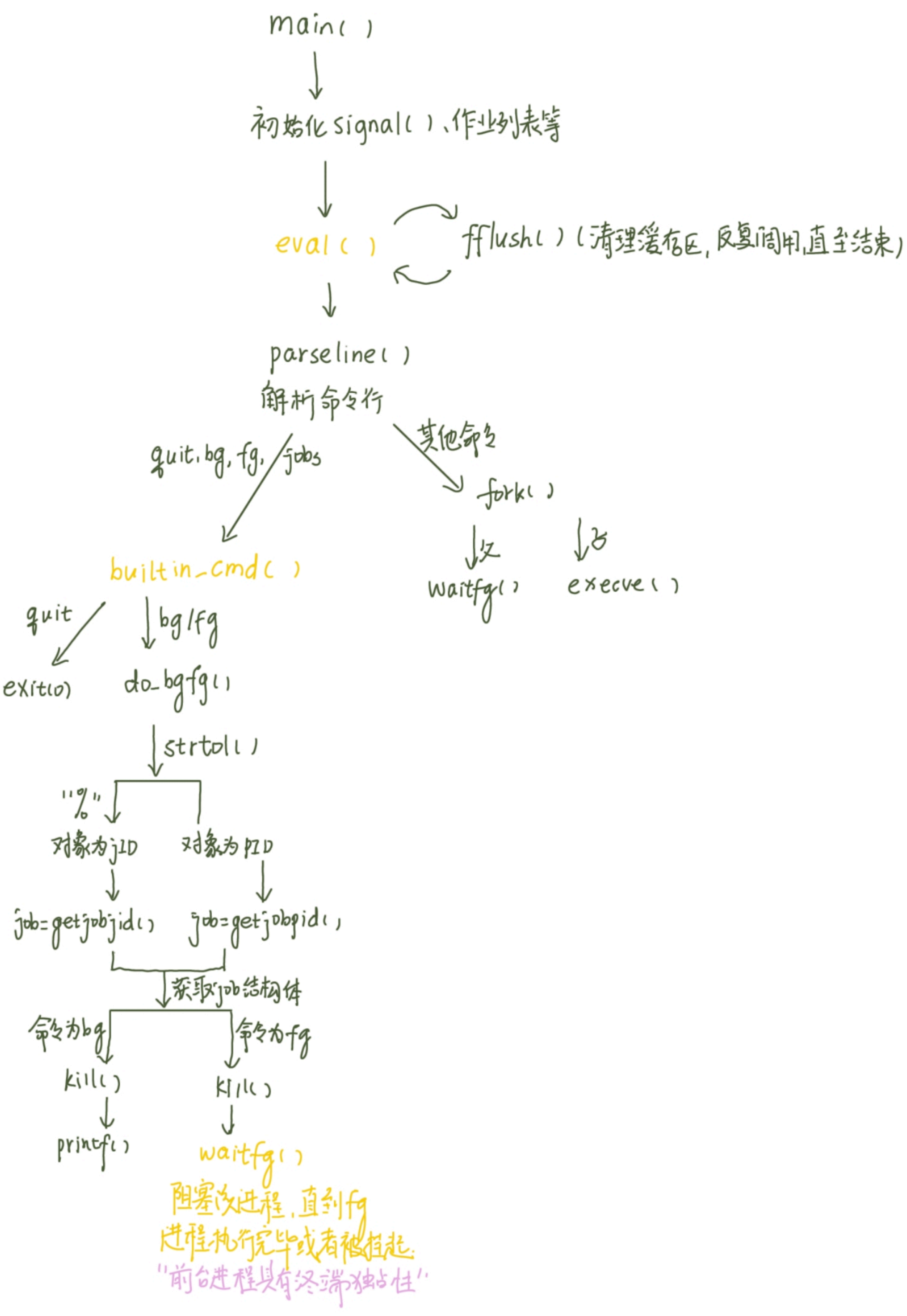


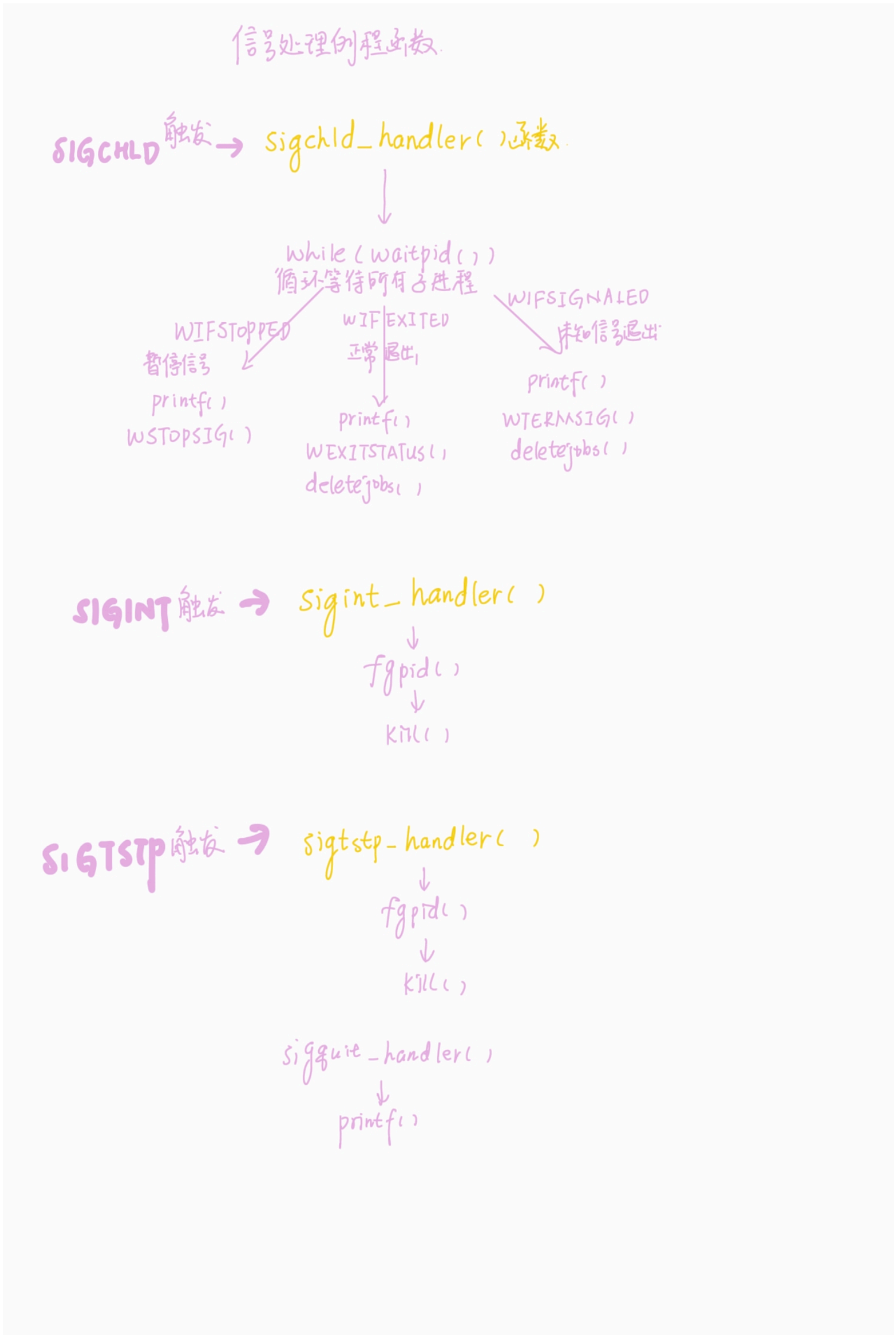
```
(base) xiaoye@localhost:~/CS/lab4$ make test16
./sdriver.pl -t trace16.txt -s ./tsh -a "-p"
#
# trace16.txt - Tests whether the shell can handle SIGTSTP and SIGINT
#     signals that come from other processes instead of the terminal.
#
tsh> ./mystop 2
Job [1] (16814) stopped by signal 20
tsh> jobs
[1] (16814) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (16858) terminated by signal 2
(base) xiaoye@localhost:~/CS/lab4$ 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] (16903) stopped by signal 20
tsh> jobs
[1] (16903) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (16959) terminated by signal 2
```

实验总结

进程的创建与销毁都是内核级的操作，这些操作由操作系统完成，一般来说用户是没有权限看到的。用户能做的只是请求创建进程、销毁进程、中断、结束等。Shell俗称壳（用来区别于核），是指“为使用者提供操作界面”的软件，它的作用是沟通用户与内核。我们的tsh（也就是tinyshell）所做的事情就是Shell的简化版，它简化到只有4个信号（SIGINT，SIGTSTP,SIGCHLD,SIGCONT），只有4个内置命令（bg,fg,jobs,quit）。它起到的作用也是沟通我们用户与内核，满足我们用户对于操作系统的要求，包括谁在前台运行，运行什么，谁在后台运行，运行什么等。

函数方面


main( )
初始化signal( )、作业列表等
eval( )
fflush( ) (清理缓存区,反复调用,直至结束)
parseline( )
解析命令行
quit, bg, fg, jobs
其他命令
fork( )
父
子
waitfg( )
execve( )
builtin_cmd( )
quit
bg/fg
exit(0)
do_bgfg( )
strtol( )
"%"
对象为JID
对象为PID
job=getjobjid( )
job=getjobpid( )
获取job结构体
命令为bg
命令为fg
kill( )
kill( )
printf( )
waitfg( )
阻塞该进程,直到fg
进程执行完毕或者被挂起.
"前台进程具有终端独占性"



这样我们可以清楚地看见整个过程程序在什么情况下调用什么函数做什么事情。

信号方面

其实前面在实现sigchld_handler（）、sigint_handler（）、sigtstp_handler（）的时候，我们已经介绍了sigsuspend（）、waitpid（）、kill（）和信号集等。
这里重点讲一下信号是如何知晓屏蔽与否的工作过程。

每个进程都维护着两个集合，称为阻塞信号集（也称为屏蔽词集）和未决信号集。

Linux内核的进程控制块PCB是一个结构体task_struct，除了包含进程id、状态、工作目录、用户id、组id、文件描述符表、还包含了信号相关的信息，主要指阻塞信号集和未决信号集。

- 阻塞信号集：也叫信号屏蔽字，将某些信号加入集合，对他们设置屏蔽，当屏蔽某个信号后，再收到该信号，该信号的处理将推后(解除屏蔽后)。
- 信号产生，未决信号集中描述该信号的位立刻翻转为1，表示信号处于未决状态；当信号被处理对应位翻转回为0，这一时刻往往非常短暂。
- 信号产生后由于某些原因主要是阻塞不能抵达，这类信号的集合称之为未决信号集。在屏蔽解除前，信号一直处于未决状态。
  未决信号集就是没有被处理的信号，未决信号集实际上是一个32位数，每一位代表一个信号，当信号产生的时候，就把对应的位反转为1，如果该信号未被处理就反转回0，处理了就保持为1。
  而阻塞信号集会影响到未决信号集，比如说我在阻塞信号集中将2号信号为置为1，也就是将2号信号屏蔽，那么未决信号集中2号信号对应的位就会变为1（未决状态），一直阻塞在这种状态。
  内核通过读取未决信号集来判断信号是否应被处理，信号屏蔽字mask可以影响未决信号集，而我们可以在应用程序中自定义set来改变mask来达到屏蔽指定信号的目的。

并发问题

（1）jobs数组是一个临界缓冲区，addjob函数和deletejob函数的访问必须持有锁。
（2）在 SIGCHLD 信号处理函数中处理多个子进程的并发终止时，必须使用循环结构（如 `while`）结合 `waitpid` 的非阻塞模式（`WNOHANG`），以确保所有已终止的子进程都被正确回收。