实验报告

一、实验目标

本次实验的目标是通过编写一个简单的支持作业控制（job control）的 Unix shell 程序来熟悉进程控制和信号的概念。我们需要通过填充 eval、builtin_cmd、do_bgfg 等函数，自己实现一个简易的 shell，使其具备实验要求中的功能，并得到与参考程序相同的输出。

二、实验过程

1. eval 函数

根据说明，eval 函数负责**解析并解释命令行的主要部分**。是内建命令则立即执行；否则 fork 一个子进程并在其上下文中运行该任务。且对前台任务，要等待它运行结束才能返回。

需要注意的是，父进程 **fork 子进程之前需要先屏蔽 SIGCHLD 信号**，以防止子进程在父进程 addjob 之前就已经运行结束被回收。并且我们还需要**为子进程重新分配进程组 id**，以防止 ctrl+c、ctrl+z 命令的信号被发送给我们的 shell 和其创建的每一个进程。此外，**访问全局变量(如 jobs)时需要阻塞所有信号。**（这一点在之后也会用到，届时不再赘述）

通过以上分析，首先初始化各变量并设置信号阻塞合集。其次，通过 parseline 函数解析命令行，以判断是否为后台命令，并将结果存放在 bg 中。然后 builtin_cmd 函数判断是否为内建命令，是则由 builtin_cmd 函数立即执行，否则由 eval 函数阻塞信号后 fork 子进程：子进程调用执行命令文件后退出；父进程执行 addjob 添加子进程后解阻塞，如果是前台任务则调 waitfg 函数等待，后台任务直接打印消息即可。

因此，eval 函数实现如下：

```
void eval(char *cmdline)
{
    char *argv[MAXARGS];  //命令行参数
    char buf[MAXLINE];
    int bg;  // should the job run in bg or fg?
    pid_t pid;
    sigset_t mask_one, mask_all, mask_prev;

    sigemptyset(&mask_one);
    sigaddset(&mask_one, SIGCHLD);
    sigfillset(&mask_all);

    //解析命令行
    strcpy(buf, cmdline);
    bg = parseline(buf, argv);
    if(argv[0] == NULL)
        return;

    if(!builtin_cmd(argv)){
        //不是内置命令，为了防止子进程先结束导致addjob
        //和deletejob的顺序发生错误，需要先阻塞SIGCHID
        sigprocmask(SIG_BLOCK, &mask_one, &mask_prev);
        //需要fork一个子进程
        if((pid = fork()) == 0){
            //子进程
            fflush(stdout);
            sigprocmask(SIG_UNBLOCK, &mask_one, NULL);//解除阻塞
            setpgid(0, 0); //子进程放到新的进程组，和shell分开
            if(execve(argv[0], argv, environ) < 0){
                //没找到可执行文件，退出
                printf("%s: Command not found.\n", argv[0]);
                exit(0);
            }

        }
        //阻塞所有信号，保证父进程addjob执行期间不中断
        sigprocmask(SIG_BLOCK, &mask_all, NULL);
        if(bg){
            addjob(jobs, pid, BG, cmdline);
        }else{
            addjob(jobs, pid, FG, cmdline);
        }
        //解阻塞
        sigprocmask(SIG_SETMASK, &mask_prev, NULL);

        if(bg){//后台
            printf("[%d] (%d) %s", pid2jid(pid), pid, buf);
        }else{  //前台
            waitfg(pid);
        }
    }
    return;
}
```

2.builtin_cmd 函数

该函数的任务是识别并解释内建命令(quit、fg、bg 以及 jobs)。所以只需要**判断是哪个内建命令**，分别执行操作然后 return 1 即可。若与每个内建命令都不匹配，则不需要执行操作，return 0 即可。

其中，对 quit 命令直接 exit(0)退出，对单独的&指令直接无视，对 jobs 命令调用 listjobs 函数，对 bg 或 fg 命令则调用 do_bgfg 函数即可。

```c
int builtin_cmd(char **argv)
{
    if(!strcmp(argv[0], "quit")){
        exit(0);
    }
    if(!strcmp(argv[0], "&")){
        return 1;
    }
    if(!strcmp(argv[0], "jobs")){
        listjobs(jobs);
        return 1;
    }
    if(!strcmp(argv[0], "fg")||!strcmp(argv[0], "bg")){
        do_bgfg(argv);
        return 1;
    }
    return 0;     /* not a builtin command */
}
```

3.do_bgfg 函数

该函数的任务是实现内建命令 bg <job>和 fg <job>，既要**对 bg 和 fg 命令进行区分**，又要**对传入的参数是 pid 还是 jid 进行区分**。前者直接根据 argv[0]判断即可，后者则利用它们格式上的不同，尝试用 sscanf 以不同格式读入（sscanf 在正确读取时返回读取的参数个数，返回值大于 0；读取失败时返回值为-1）。此外，还要注意对参数数量不够、重启不存在的进程等等**错误进行处理**。最后使用 **kill 发送 SIGCONT 信号**，并根据前后台状态采取不同操作。

```c
void do_bgfg(char **argv)
{
    int jid;
    struct job_t *job;
    pid_t pid;
    sigset_t mask_all, mask_prev;

    if(argv[1] == NULL){
        printf("%s command requires PID or %%jobid argument\n", argv[0]);
        return;
    }
    if(sscanf(argv[1], "%%%d", &jid) > 0){//是jid
        job = getjobjid(jobs, jid);
        if(job == NULL || job->state == UNDEF){
            printf("%s: No such job\n", argv[1]);
            return;
        }
    }else if(sscanf(argv[1], "%d", &pid) > 0){  //是pid
        job = getjobpid(jobs, pid);
        if(job == NULL || job->state == UNDEF){
            printf("(%s): No such process\n", argv[1]);
            return;
        }
    }else{
        printf("%s: argument must be a PID or %%jobid\n", argv[0]);
        return;
    }

    sigfillset(&mask_all);
    sigprocmask(SIG_BLOCK, &mask_all, &mask_prev);  //阻塞所有信号
    if(!strcmp(argv[0], "fg")){  //执行fg命令，改为前台运行
        job->state = FG;
    }else{  //执行bg命令，改为后台运行
        job->state = BG;
    }
    sigprocmask(SIG_SETMASK, &mask_prev, NULL); //解阻塞

    pid = job->pid;
    kill(-pid, SIGCONT);//发送SIGCONT信号重启
    if(!strcmp(argv[0], "fg")){
        waitfg(pid);
    }else{
        printf("[%d] (%d) %s", job->jid, pid, job->cmdline);
    }

    return;
}
```

4.waitfg 函数

该函数等待一个前台作业结束，只需要重复**调用 sleep 函数**直至当前子进程的 pid 与前台进程的 pid 不再相等即可。

```c
void waitfg(pid_t pid)
{
    while(pid == fgpid(jobs)){
        sleep(1);
    }
    return;
}
```

5. sigchld_handler 函数

该函数捕捉 SIGCHLD 信号，并作出响应。首先需要注意**使用 while 循环**而不是 if 来接收信号，因为不可靠信号不支持排队，一个未处理信号可能对应多个信号。然后 waitpid 函数回收子进程时，要将**参数设置为 WNOHANG | WUNTRACED**，在当前进程中不存在已经停止或者终止的进程时立即返回，以防止卡在 while 循环中。最后则需要通过 state 参数和

WIFEXITED 等函数来**判断返回的进程终止或暂停的原因**，并进行相应的不同处理。

```c
void sigchld_handler(int sig)
{
    int old_errno = errno;
    pid_t pid;
    int state;
    sigset_t mask_all, mask_prev;
    struct job_t *job;

    sigfillset(&mask_all);
    while((pid = waitpid(-1, &state, WNOHANG | WUNTRACED)) > 0){
        sigprocmask(SIG_BLOCK, &mask_all, &mask_prev); //阻塞所有信号
        if(WIFEXITED(state)){   //子进程正常退出，deletejob
            deletejob(jobs, pid);
        }else if(WIFSIGNALED(state)){ //子进程因为信号退出，打印并deletejob
            printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(pid), pid, WTERMSIG(state));
            deletejob(jobs, pid);
        }else if(WIFSTOPPED(state)){  //子进程停止，需要修改状态
            job = getjobpid(jobs, pid);
            job->state = ST;
            printf("Job [%d] (%d) stopped by signal %d\n", job->jid, pid, WSTOPSIG(state));
        }
        sigprocmask(SIG_SETMASK, &mask_prev, NULL);  //解阻塞
    }
    errno = old_errno;
    return;
}
```

6. sigint_handler 函数和 sigtstp_handler 函数

这两个函数的功能类似，分别捕捉 SIGINT(ctrl-c)和 SIGTSTP(ctrl-z)信号，并将其发送给所有前台程序。只需要获得前台进程的 pid，然后使用 kill 发送信号即可。过程中注意保存和恢复 errno。注意：根据提示，使用-pid 作为 kill 的参数。

```c
void sigint_handler(int sig)
{
    int old_errno = errno;
    pid_t pid = fgpid(jobs);
    if(pid != 0){
        kill(-pid, SIGINT);
    }
    errno = old_errno;
    return;
}
```

```c
void sigtstp_handler(int sig)
{
    int old_errno = errno;
    pid_t pid = fgpid(jobs);
    if(pid != 0){
        kill(-pid, SIGTSTP);
    }
    errno = old_errno;
    return;
}
```

三、思维扩展

实现管道和输入输出重定向功能的思路：在解析命令行时，检测其中是否包含“< > |”标识符，如果不存在则正常执行命令即可。如果存在，对于输入输出重定向的情况，要保存标识符之后的文件名，通过 dup2 函数将标准输入或输出替换为相应文件；对于管道功能，要在当前子进程中再 fork 一个子进程，用新建的子进程执行管道符之前的命令，并重定向标准输出，待其完成后再由之前的子进程重定向标准输入。

四、正确性检验

对比 test 与 rtest 的 01-16 共 16 个案例的输出，除 PID 这类每次运行都会改变的值外，输出均与参考程序相同，证明了程序的正确性。

（以下随机展示几个样例）

```
[2020202279@work122 shlab-handout]$ make test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (234399) ./myspin 2 &
tsh> ./myspin 3 &
[2] (234401) ./myspin 3 &
tsh> jobs
[1] (234399) Running ./myspin 2 &
[2] (234401) Running ./myspin 3 &
```

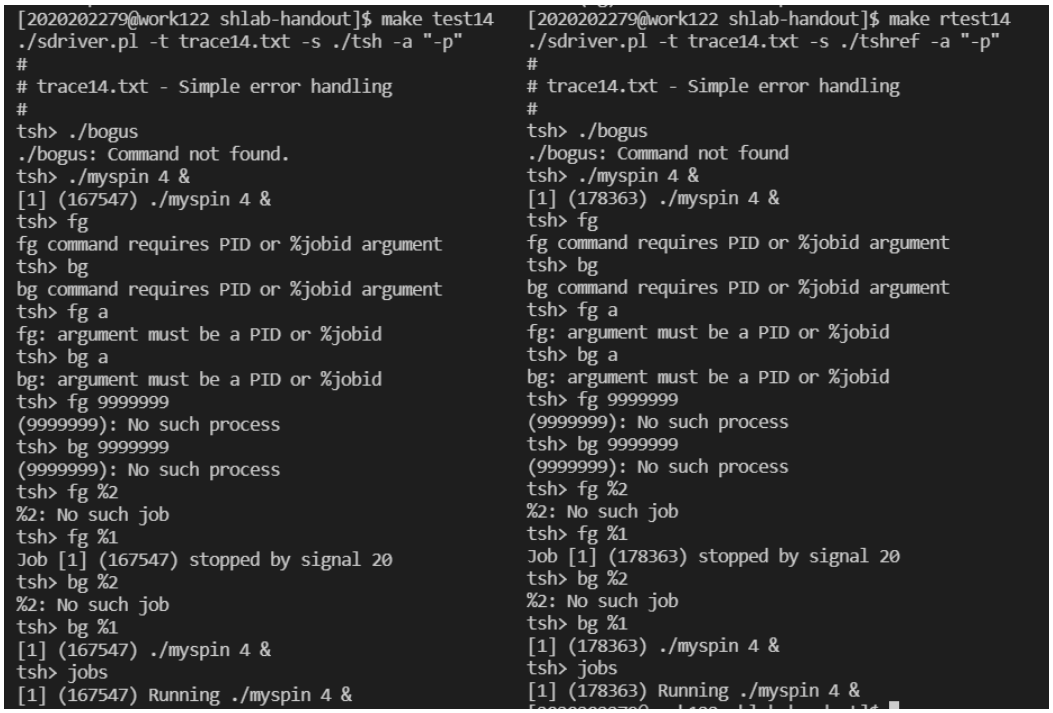
```
[2020202279@work122 shlab-handout]$ make rtest05
./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (235119) ./myspin 2 &
tsh> ./myspin 3 &
[2] (235121) ./myspin 3 &
tsh> jobs
[1] (235119) Running ./myspin 2 &
[2] (235121) Running ./myspin 3 &
```

```
[2020202279@work122 shlab-handout]$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (69433) ./myspin 4 &
tsh> ./myspin 5
Job [2] (69437) stopped by signal 20
tsh> jobs
[1] (69433) Running ./myspin 4 &
[2] (69437) Stopped ./myspin 5
tsh> bg %2
[2] (69437) ./myspin 5
tsh> jobs
[1] (69433) Running ./myspin 4 &
[2] (69437) Running ./myspin 5
```

```
[2020202279@work122 shlab-handout]$ make rtest09
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (71116) ./myspin 4 &
tsh> ./myspin 5
Job [2] (71119) stopped by signal 20
tsh> jobs
[1] (71116) Running ./myspin 4 &
[2] (71119) Stopped ./myspin 5
tsh> bg %2
[2] (71119) ./myspin 5
tsh> jobs
[1] (71116) Running ./myspin 4 &
[2] (71119) Running ./myspin 5
```

```
[2020202279@work122 shlab-handout]$ 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] (153059) stopped by signal 20
tsh> jobs
[1] (153059) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1205 tty1     Ss+    0:00 /sbin/agetty --noclear tty1 linux
  65354 pts/16   Ss     0:00 /bin/bash
 153055 pts/16   S+     0:00 make test13
 153056 pts/16   S+     0:00 /usr/bin/perl ./sdriver.pl -t trace1
 153057 pts/16   S+     0:00 ./tsh -p
 153059 pts/16   T      0:00 ./mysplit 4
 153060 pts/16   T      0:00 ./mysplit 4
 153177 pts/16   R      0:00 /bin/ps a
 204926 pts/20   Ss+    0:00 /usr/bin/bash
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1205 tty1     Ss+    0:00 /sbin/agetty --noclear tty1 linux
  65354 pts/16   Ss     0:00 /bin/bash
 153055 pts/16   S+     0:00 make test13
 153056 pts/16   S+     0:00 /usr/bin/perl ./sdriver.pl -t trace1
 153057 pts/16   S+     0:00 ./tsh -p
 153295 pts/16   R      0:00 /bin/ps a
 204926 pts/20   Ss+    0:00 /usr/bin/bash
```

```
[2020202279@work122 shlab-handout]$ 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] (155058) stopped by signal 20
tsh> jobs
[1] (155058) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1205 tty1     Ss+    0:00 /sbin/agetty --noclear tty1 linux
  65354 pts/16   Ss     0:00 /bin/bash
 155054 pts/16   S+     0:00 make rtest13
 155055 pts/16   S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.
 155056 pts/16   S+     0:00 ./tshref -p
 155058 pts/16   T      0:00 ./mysplit 4
 155059 pts/16   T      0:00 ./mysplit 4
 155153 pts/16   R      0:00 /bin/ps a
 204926 pts/20   Ss+    0:00 /usr/bin/bash
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1205 tty1     Ss+    0:00 /sbin/agetty --noclear tty1 linux
  65354 pts/16   Ss     0:00 /bin/bash
 155054 pts/16   S+     0:00 make rtest13
 155055 pts/16   S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.
 155056 pts/16   S+     0:00 ./tshref -p
 155272 pts/16   R      0:00 /bin/ps a
 204926 pts/20   Ss+    0:00 /usr/bin/bash
```



```
[2020202279@work122 shlab-handout]$ 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] (167547) ./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] (167547) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (167547) ./myspin 4 &
tsh> jobs
[1] (167547) Running ./myspin 4 &
```

```
[2020202279@work122 shlab-handout]$ 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] (178363) ./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] (178363) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (178363) ./myspin 4 &
tsh> jobs
[1] (178363) Running ./myspin 4 &
```