CSED211 LAB10 REPORT

20220778 표승현

1. eval

```
char *argv[MAXARGS];
int bg_flag;

sigset_t mask;
pid_t pid;
```

다음과 같이 사용할 변수를 선언한다.

```
bg_flag = parseline(cmdline, argv);
if(argv[0]==NULL) return;
```

parseline 은 bg 인지 fg 인지에 따라 다른 값을 리턴한다. 더불어 argv 에 커맨드 값을 저장하게 하는데 NULL 인 경우를 예외처리해준다.

```
// Not a built-in command
if(!builtin_cmd(argv)){
    sigemptyset(&mask);
    sigaddset(&mask, SIGCHLD);
    sigprocmask(SIG_BLOCK,&mask,NULL);

    if(!(pid=fork())){
        sigprocmask(SIG_UNBLOCK, &mask, NULL);
        setpgid(0,0);

        if(execve(argv[0],argv,environ)<0){
            printf("%s: Command not found\n", argv[0]);
            exit(0);
        }
    }
}
```

built in command 가 아닌 경우이다. signal set 을 초기화해주고, SIGCHLD signal 을 마스크에 저장하여 mask 를 블록한다. fork 를 호출한 결과가 0 이면 자식 프로세스인 경우로 setpgid 를 통해 새로운 group 으로 묶어준다.

```
if(pid<0){
    return;
}
else{
    sigprocmask(SIG_UNBLOCK, &mask, NULL);

    if(!bg_flag){
        addjob(jobs, pid, FG, cmdline);
        waitfg(pid);
    }
    else{
        addjob(jobs, pid, BG, cmdline);
        printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
    }
}
```

pid 가 음수인 경우는 프로세스 생성에 실패한 경우이므로 예외처리 해준다. Built in command 인 경우는 sigprocmask 를 해제해주고 bg, fg 에 대한 처리를 수행한다. bg_flag 에 1 이 저장되어 있으면 bg 를 처리하고 아닌 경우에는 fg 를 처리한다. bg_flag 의 값은 parseline 함수의 return 값에 의존하는데 명령어 끝에 &의 유무를 판단하여 이를 결정한다. addjob 을 통해 각 명령을 처리한다.

2. builtin_cmd

```
int builtin_cmd(char **argv)
{
    char *cmd=argv[0];
    int bin_flag = 1;

    if(!strcmp(cmd, "quit")){
        exit(0);
    }
    else if(!strcmp(cmd, "jobs")){
        listjobs(jobs);
    }
    else if(!strcmp(cmd, "bg")||!strcmp(cmd,"fg")){
        do_bgfg(argv);
    }
    else{
        bin_flag=0; //wrong command
    }

    if(bin_flag) return 1;

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

cmd 에 argv 의 문자열을 저장하고 이를 특정 명령어들과 비교하여 함수를 동작시킨다. quit 을 입력받으면 exit 함수를 통해 프로그램을 종료한다. jobs 를 입력받으면 job list 를 출력하게 한다. 마지막으로 bg 나 fg 를 입력받으면 do_bgfg 함수를 통해 각각에 알맞은 기능을 수행한다. 어느 명령어에도 맞지 않으면 built in 명령이 아닌 것으로 판단한다.

3. do_bgfg

```
struct job_t *job;
pid_t pid;
int jid;

char *cmd1 = argv[0];
char *cmd2 = argv[1];
```

다음과 같이 선언해준다.

```
if(cmd2==NULL){ //wrong command
    printf("%s command requires PTD or %%jobid argument\n", cmd1);
    return;
}

if(cmd2[0]!='%'){
if(!isdigit(cmd2[0])){
    printf("%s: argument must be a PID or %%jobid\n", cmd1);
    return;
}
}
```

cmd2 에 아무 입력이 없는 경우와 cmd2 가 %으로 시작되지 않을 때 정수 입력값이 아닐 경우에 예외처리를 해준다.

```
// setting jid or pid
if(cmd2[0]=='%'){ // jid
    jid = atoi(&cmd2[1]);

    if(jid ==0){
        printf("%s: argument must be a PID or %%jobid\n", cmd1);
        return;
    }

    job=getjobjid(jobs,jid);
    if(job==NULL){
        printf("%s: No such job\n", cmd2);
        return;
    }

}
else if(isdigit(cmd2[0])){ // pid
    pid = atoi(cmd2);

    if(pid ==0){
        printf("%s: argument must be a PID or %%jobid\n", cmd1);
        return;
    }

    job=getjobpid(jobs,pid);
    if(job==NULL){
        printf("%s: No such process\n", cmd2);
        return;
    }
}
```

%으로 시작하는 경우는 jid 이다. jid 를 받아서 getjobjid 를 호출하여 해당 jid 를 가진 job 을 찾아 저장한다. %로 시작하지 않는 경우는 pid 를 입력받았을 때이다. pid 를 입력받아 getjobpid 를 호출하여 해당 pid 를 갖는 job 을 저장한다.

```
// output
if(!strcmp(cmd1,"fg")){
    job -> state = FG;
    kill(-job->pid, SIGCONT);
    waitfg(job->pid);
}
else if(!strcmp(cmd1,"bg")){
    job->state = BG;
    printf("[%d] (%d) %s", job->jid, job->pid, job->cmdline);
    kill(-job->pid, SIGCONT);
}
```

fg 인지 bg 인지 판단하고 각각에 맞는 기능을 수행한다. fg 인 경우 state 를 FG 로 바꾸고 kill 함수를 이용하여 해당 pid 프로세스에 SIGCONT 를 전송한다. 이후 해당 프로세스가 종료될 때까지 wait 한다. bg 의 경우 state 를 BG 로 바꾸고 해당 프로세스의 정보를 출력한다. 이후 fg 에서처럼 kill 함수를 이용해 SIGCONT 시그널을 전달한다.

4. waitfg

```
void waitfg(pid_t pid)
{
    if(pid==0) return;

    while((fgpid(jobs)==pid)){
        sleep(1);
    }

    return;
}
```

pid 가 잘못된 경우에 대해 예외처리를 한다. 이후 foreground 의 pid 가 파라미터의 pid 와 같은지 계속 체크한다. background 이면 반복문을 탈출하게 되므로 작업이 완료되었음을 의미한다.

5. sigchld_handler

```
pid_t pid;
int status;

while((pid=waitpid(-1,&status, WNOHANG|WUNTRACED))>0){
    if(WIFSTOPPED(status)){
        getjobpid(jobs, pid) -> state = ST;
        printf("Job [%d] (%d) stopped by signal %d\n", pid2jid(pid), pid, WSTOPSIG(status));
    }
    else if(WIFSIGNALED(status)|WIFEXITED(status)){
        if(WIFSIGNALED(status)){
            printf("Job [%d] (%d) terminated by signal %d\n",pid2jid(pid), pid, WTERMSIG(status));
        }
        deletejob(jobs, pid);
    }
}
return;
```

waitpid 함수를 이용해 임의의 자식 프로세스가 종료될 때까지 반복문을 돌린다. pid 가 0 보다 크면 waitpid 가 성공적으로 값을 반환한 것이므로 이를 조건으로 한다. WIFSTOPPED 는 프로세스가 정지하면 true 를 반환한다. 이 경우 getjobpid 를 이용해 job 을 불러오고 state 를 ST 로 바꿔준다. WIFSIGNALED 와 WIFEXITED 는 각각 프로세스가 시그널에 의해 종료되었거나 정상적으로 종료되었는지를 판단하여 true 를 반환한다. 이 경우 프로세스가 종료되었다는 뜻이므로 job list 에서 해당 pid 의 job 을 삭제한다.

6. signt_handler

```
{
    pid_t pid = fgpid(jobs);

    if(pid>0) kill(-pid, sig);

    return;
}
```

foreground 의 pid 를 불러오고, pid 가 0 이상일 때 해당 pid 에 시그널을 보낸다.

7. sigtstp_handler

```
pid_t pid = fgpid(jobs);

if(pid<=0) return;

if(kill(-pid, SIGTSTP)<0) unix_error("kill (tstp) error");
```

foreground 의 pid 를 불러온다. pid 가 0 보다 크지 않으면 바로 반환한다. 0 보다 크면 해당 pid 프로세스에 SIGTSTP 시그널을 보내고 반환값이 0 보다 작으면 error 문구를 출력하게 한다.

8. test

test1.

```
1   #
2   # trace01.txt - Properly terminate on EOF.
3   #
4   CLOSE
5   WAIT
6
```

```
[hyeony312@programming2 shellab2]$ make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
[hyeony312@programming2 shellab2]$ make rtest01
./sdriver.pl -t trace01.txt -s ./tshref -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
```

rtest 의 결과와 같은 모습이다.

test2.

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

```
[hyeony312@programming2 shellab2]$ make test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
[hyeony312@programming2 shellab2]$ make rtest02
./sdriver.pl -t trace02.txt -s ./tshref -a "-p"
#
# trace02.txt - Process builtin quit command.
#
```

quit 를 정상적으로 수행했다.

test3.

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

```
[hyeony312@programming2 shellab2]$ make test03
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
[hyeony312@programming2 shellab2]$ make rtest03
./sdriver.pl -t trace03.txt -s ./tshref -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
```

foreground 프로세스를 정상적으로 처리한 모습.

test4

```
1   #
2   # trace04.txt - Run a background job.
3   #
4   /bin/echo -e tsh> ./myspin 1 \046
5   ./myspin 1 &
```

```
[hyeony312@programming2 shellab2]$ make test04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (19060) ./myspin 1 &
[hyeony312@programming2 shellab2]$ make rtest04
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (19079) ./myspin 1 &
```

background 를 잘 처리한 모습이다.

test5

```
1   #
2   # trace05.txt - Process jobs builtin command.
3   #
4   /bin/echo -e tsh> ./myspin 2 \046
5   ./myspin 2 &
6
7   /bin/echo -e tsh> ./myspin 3 \046
8   ./myspin 3 &
9
10  /bin/echo tsh> jobs
11  jobs
```

```
[hyeony312@programming2 shellab2]$ make test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (19275) ./myspin 2 &
tsh> ./myspin 3 &
[2] (19277) ./myspin 3 &
tsh> jobs
[1] (19275) Running ./myspin 2 &
[2] (19277) Running ./myspin 3 &
[hyeony312@programming2 shellab2]$ make rtest05
./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (19304) ./myspin 2 &
tsh> ./myspin 3 &
[2] (19307) ./myspin 3 &
tsh> jobs
[1] (19304) Running ./myspin 2 &
[2] (19307) Running ./myspin 3 &
```

여러 개의 프로세스가 background 에서 돌아가고 있고 이를 보여주고 있다.

test6

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

SLEEP 2
INT
```

```
[hyeony312@programming2 shellab2]$ make test06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (21226) terminated by signal 2
[hyeony312@programming2 shellab2]$ make rtest06
./sdriver.pl -t trace06.txt -s ./tshref -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (21239) terminated by signal 2
```

프로세스에 INT 로 시그널을 보내 종료시켰다.

test7

```
#
# 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
```

```
[hyeony312@programming2 shellab2]$ make test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (23946) ./myspin 4 &
tsh> ./myspin 5
Job [2] (23948) terminated by signal 2
tsh> jobs
[1] (23946) Running ./myspin 4 &
[hyeony312@programming2 shellab2]$ make rtest07
./sdriver.pl -t trace07.txt -s ./tshref -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (23962) ./myspin 4 &
tsh> ./myspin 5
Job [2] (23964) terminated by signal 2
tsh> jobs
[1] (23962) Running ./myspin 4 &
```

foreground 인 myspin 5 만이 시그널을 받아 종료되었다.

test8

```
#
# 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
```

```
[hyeony312@programming2 shellab2]$ make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (24342) ./myspin 4 &
tsh> ./myspin 5
Job [2] (24346) stopped by signal 20
tsh> jobs
[1] (24342) Running ./myspin 4 &
[2] (24346) Stopped ./myspin 5
[hyeony312@programming2 shellab2]$ make rtest08
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (24404) ./myspin 4 &
tsh> ./myspin 5
Job [2] (24407) stopped by signal 20
tsh> jobs
[1] (24404) Running ./myspin 4 &
[2] (24407) Stopped ./myspin 5
```

test7 의 결과와 같이 foreground 의 프로세스만이 시그널을 받아 stop 됐다.

test9

```
#
# 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
```

```
[hyeony312@programming2 shellab2]$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (24589) ./myspin 4 &
tsh> ./myspin 5
Job [2] (24591) stopped by signal 20
tsh> jobs
[1] (24589) Running ./myspin 4 &
[2] (24591) Stopped ./myspin 5
tsh> bg %2
[2] (24591) ./myspin 5
tsh> jobs
[1] (24589) Running ./myspin 4 &
[2] (24591) Running ./myspin 5
[hyeony312@programming2 shellab2]$ make rtest09
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (24620) ./myspin 4 &
tsh> ./myspin 5
Job [2] (24622) stopped by signal 20
tsh> jobs
[1] (24620) Running ./myspin 4 &
[2] (24622) Stopped ./myspin 5
tsh> bg %2
[2] (24622) ./myspin 5
tsh> jobs
[1] (24620) Running ./myspin 4 &
[2] (24622) Running ./myspin 5
```

foreground 프로세스에 대해 bg 명령어를 처리한 모습이다.

test10

```
#
# 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
```

```
[hyeony312@programming2 shellab2]$ make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (25250) ./myspin 4 &
tsh> fg %1
Job [1] (25250) stopped by signal 20
tsh> jobs
[1] (25250) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
[hyeony312@programming2 shellab2]$ make rtest10
./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (25267) ./myspin 4 &
tsh> fg %1
Job [1] (25267) stopped by signal 20
tsh> jobs
[1] (25267) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

background job 을 fg 를 통해 foreground 로 바꿔주었다.

test11

```
#
# 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
```

foreground job 이 실행되는 도중에 시그널을 받아 종료됐고 이를 출력하게 된다. 출력값은 생략한다.

test12

```
#
# 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
```

test12 의 과정에서 시그널이 tstp 로 바뀌었다. 마찬가지로 test 와 rtest 값이 같아 사진은 생략한다.

test13

```
#
# 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
```

test12 의 과정 이후 fg %1 을 입력받아 정지되었던 프로세스가 다시 실행되는 모습을 보여준다.

test14

```
#
# 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
```

```
[hyeony312@programming2 shellab2]$ 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] (26385) ./myspin 4 &
tsh> fg
fg command requires PTD or %jobid argument
tsh> bg
bg command requires PTD 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] (26385) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (26385) ./myspin 4 &
tsh> jobs
[1] (26385) Running ./myspin 4 &
```

```
[hyeony312@programming2 shellab2]$ 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] (26424) ./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] (26424) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (26424) ./myspin 4 &
tsh> jobs
[1] (26424) Running ./myspin 4 &
[hyeony312@programming2 shellab2]$
```

단순한 몇 가지 예외처리를 보여준다.

test15

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

/bin/echo tsh> ./bogus
./bogus

/bin/echo tsh> ./myspin 10
./myspin 10

SLEEP 2
INT

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

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

/bin/echo tsh> jobs
jobs

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

SLEEP 2
TSTP

/bin/echo tsh> jobs
jobs

/bin/echo tsh> bg %3
bg %3

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

/bin/echo tsh> jobs
jobs

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

/bin/echo tsh> quit
quit
```

```
[hyeony312@programming2 shellab2]$ 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] (27279) terminated by signal 2
tsh> ./myspin 3 &
[1] (27283) ./myspin 3 &
tsh> ./myspin 4 &
[2] (27286) ./myspin 4 &
tsh> jobs
[1] (27283) Running ./myspin 3 &
[2] (27286) Running ./myspin 4 &
tsh> fg %1
Job [1] (27283) stopped by signal 20
tsh> jobs
[1] (27283) Stopped ./myspin 3 &
[2] (27286) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (27283) ./myspin 3 &
tsh> jobs
[1] (27283) Running ./myspin 3 &
[2] (27286) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

```
[hyeony312@programming2 shellab2]$ 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] (27312) terminated by signal 2
tsh> ./myspin 3 &
[1] (27318) ./myspin 3 &
tsh> ./myspin 4 &
[2] (27321) ./myspin 4 &
tsh> jobs
[1] (27318) Running ./myspin 3 &
[2] (27321) Running ./myspin 4 &
tsh> fg %1
Job [1] (27318) stopped by signal 20
tsh> jobs
[1] (27318) Stopped ./myspin 3 &
[2] (27321) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (27318) ./myspin 3 &
tsh> jobs
[1] (27318) Running ./myspin 3 &
[2] (27321) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

tsh 의 모든 기능을 한 번씩 실행해보는 모습이다.

test16

```
#
# 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
```

```
[hyeony312@programming2 shellab2]$ 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] (27455) stopped by signal 20
tsh> jobs
[1] (27455) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (27461) terminated by signal 2
[hyeony312@programming2 shellab2]$ 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] (27472) stopped by signal 20
tsh> jobs
[1] (27472) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (27477) terminated by signal 2
```

SIGTSTP 와 SIGINT 를 처리하는 모습이다.