CSED211 컴퓨터 SW 시스템개론

Lab Assignment #6: Shell Lab:

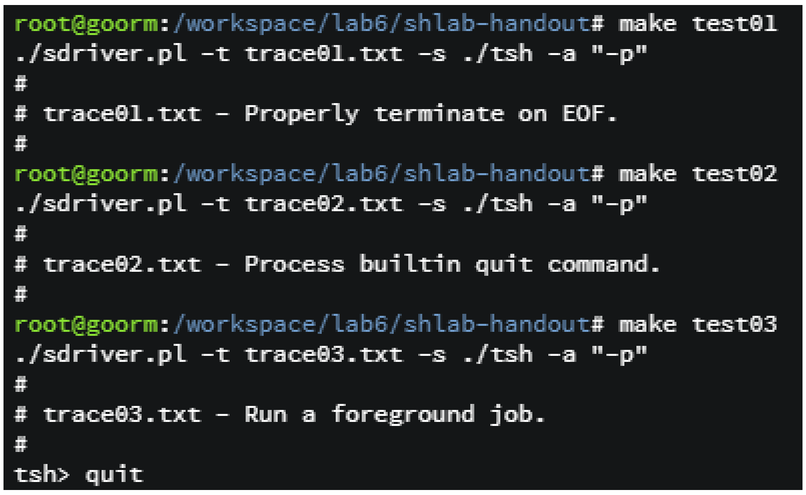
Writing Your Own Unix Shell

20180551

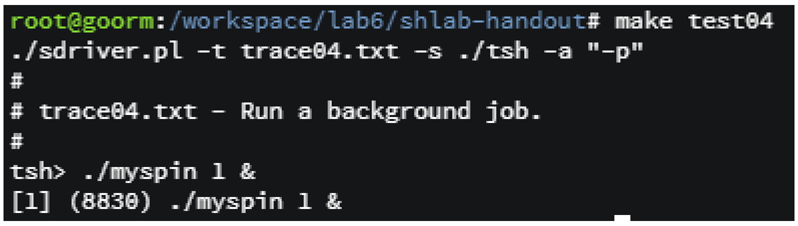
컴퓨터공학과

이준석

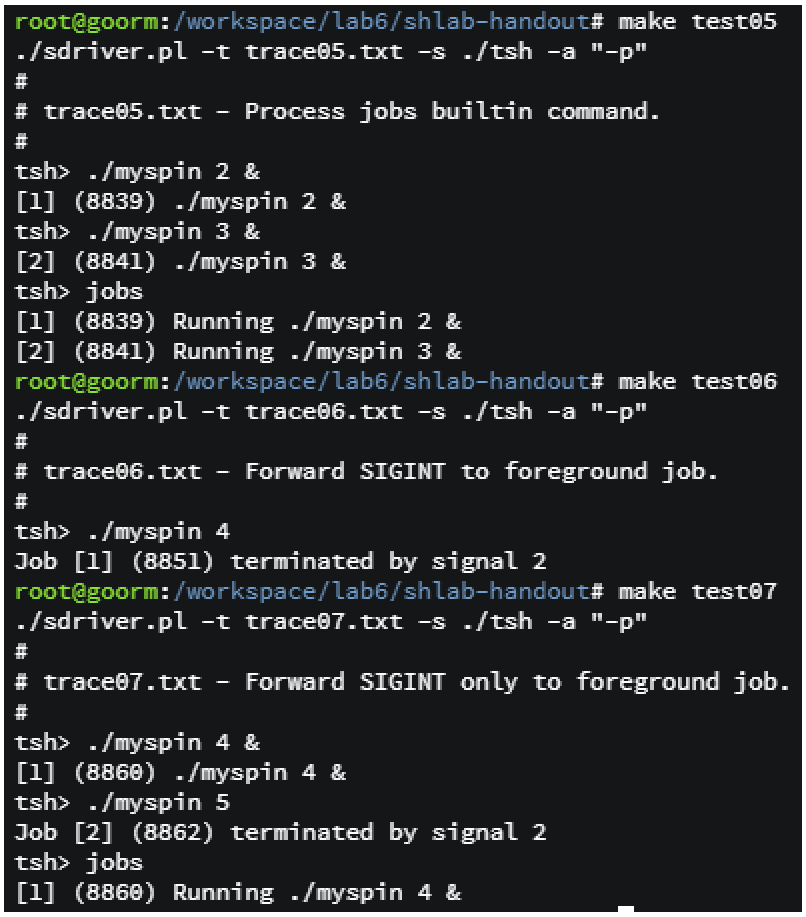
1. 실행 모습



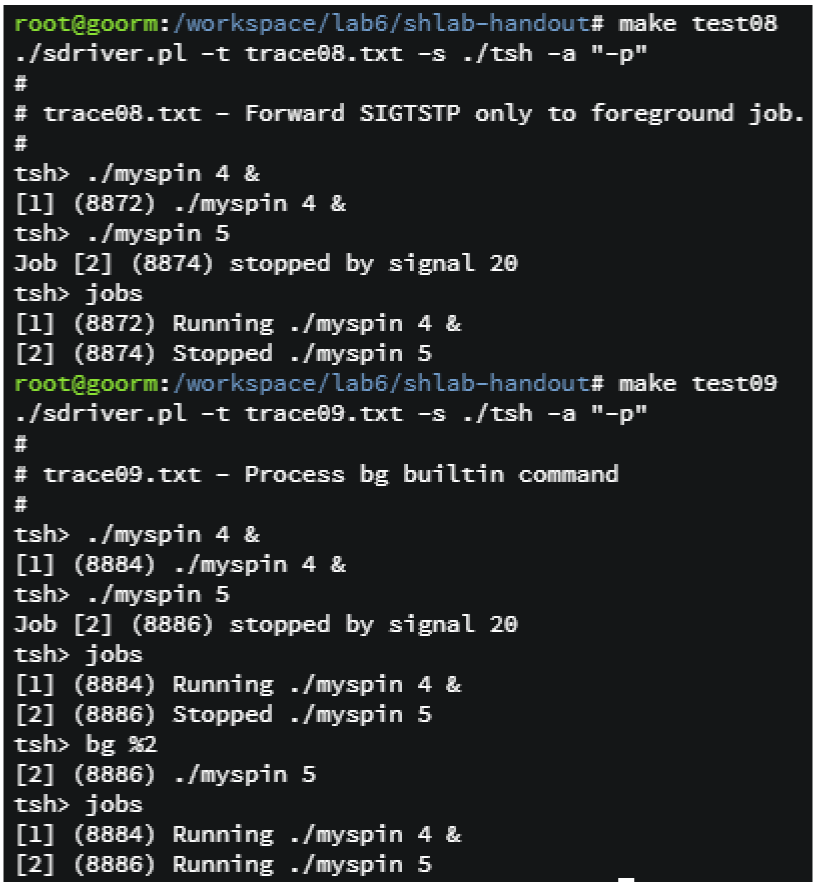
```
root@goorm:/workspace/lab6/shlab-handout# make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
root@goorm:/workspace/lab6/shlab-handout# make test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
root@goorm:/workspace/lab6/shlab-handout# make test03
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
```



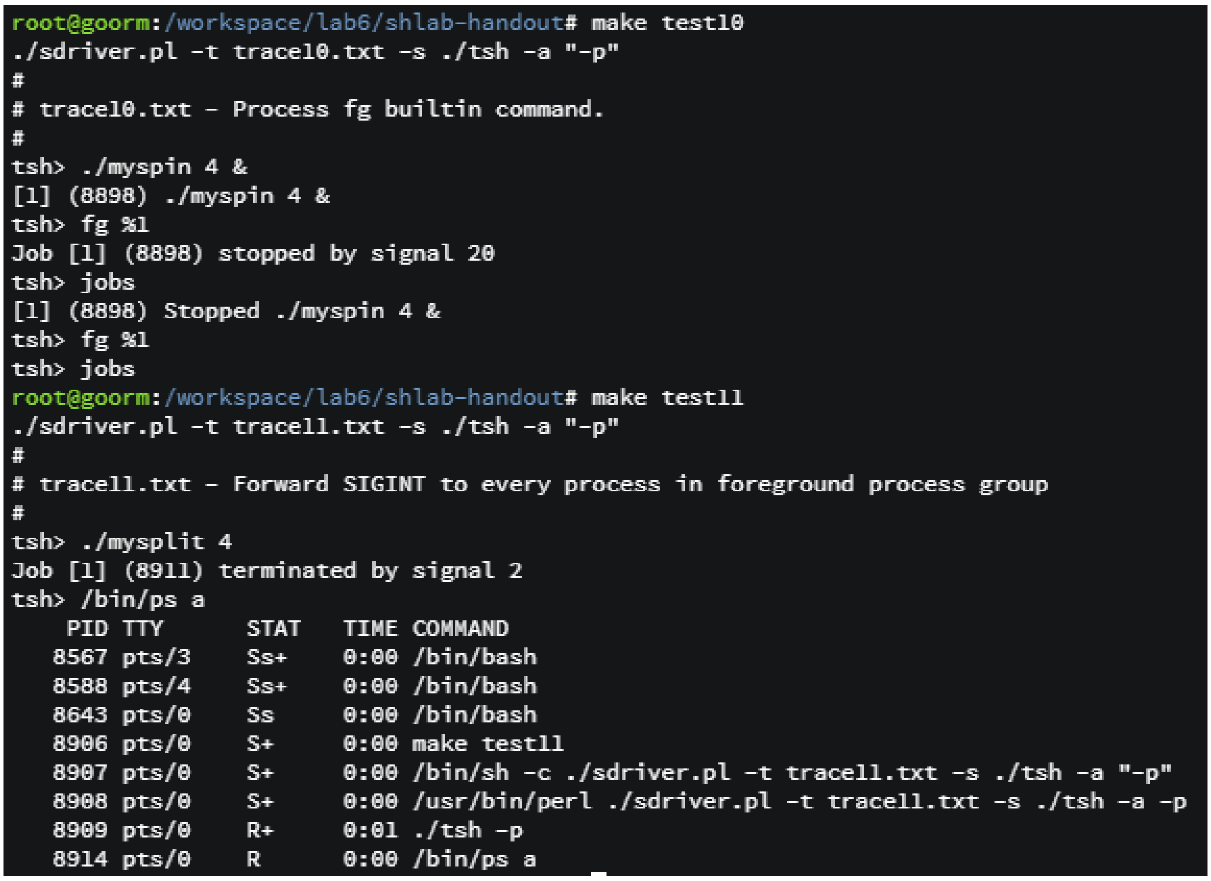
```
root@goorm:/workspace/lab6/shlab-handout# make test04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (8830) ./myspin 1 &
```



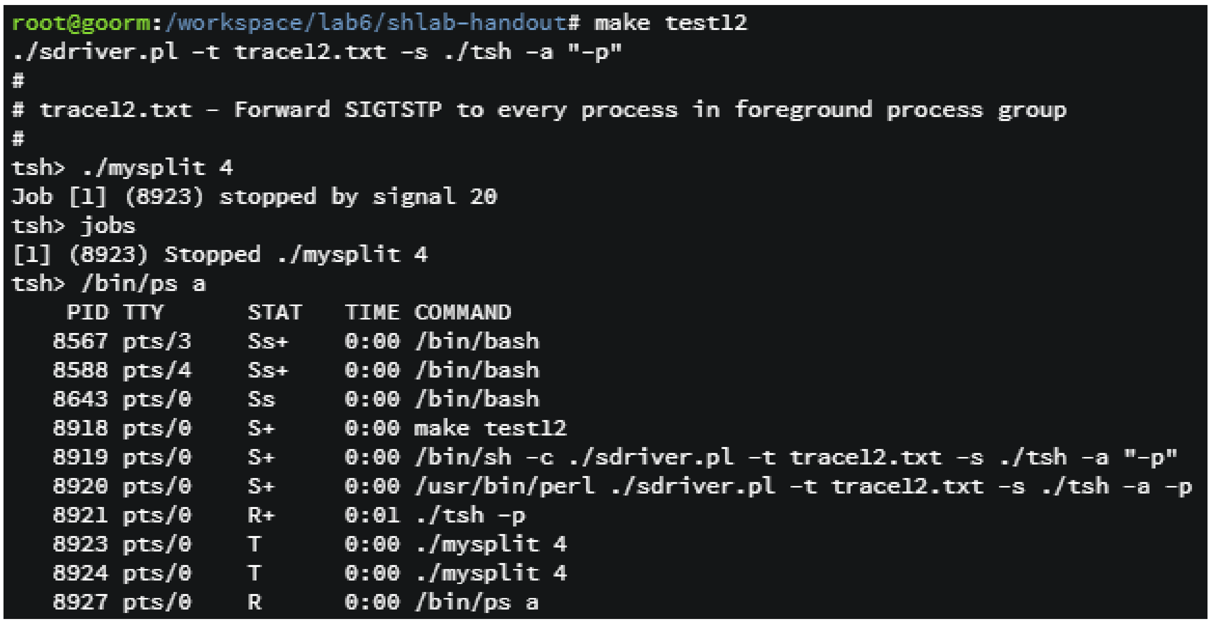
```
root@goorm:/workspace/lab6/shlab-handout# make test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (8839) ./myspin 2 &
tsh> ./myspin 3 &
[2] (8841) ./myspin 3 &
tsh> jobs
[1] (8839) Running ./myspin 2 &
[2] (8841) Running ./myspin 3 &
root@goorm:/workspace/lab6/shlab-handout# make test06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (8851) terminated by signal 2
root@goorm:/workspace/lab6/shlab-handout# make test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (8860) ./myspin 4 &
tsh> ./myspin 5
Job [2] (8862) terminated by signal 2
tsh> jobs
[1] (8860) Running ./myspin 4 &
```



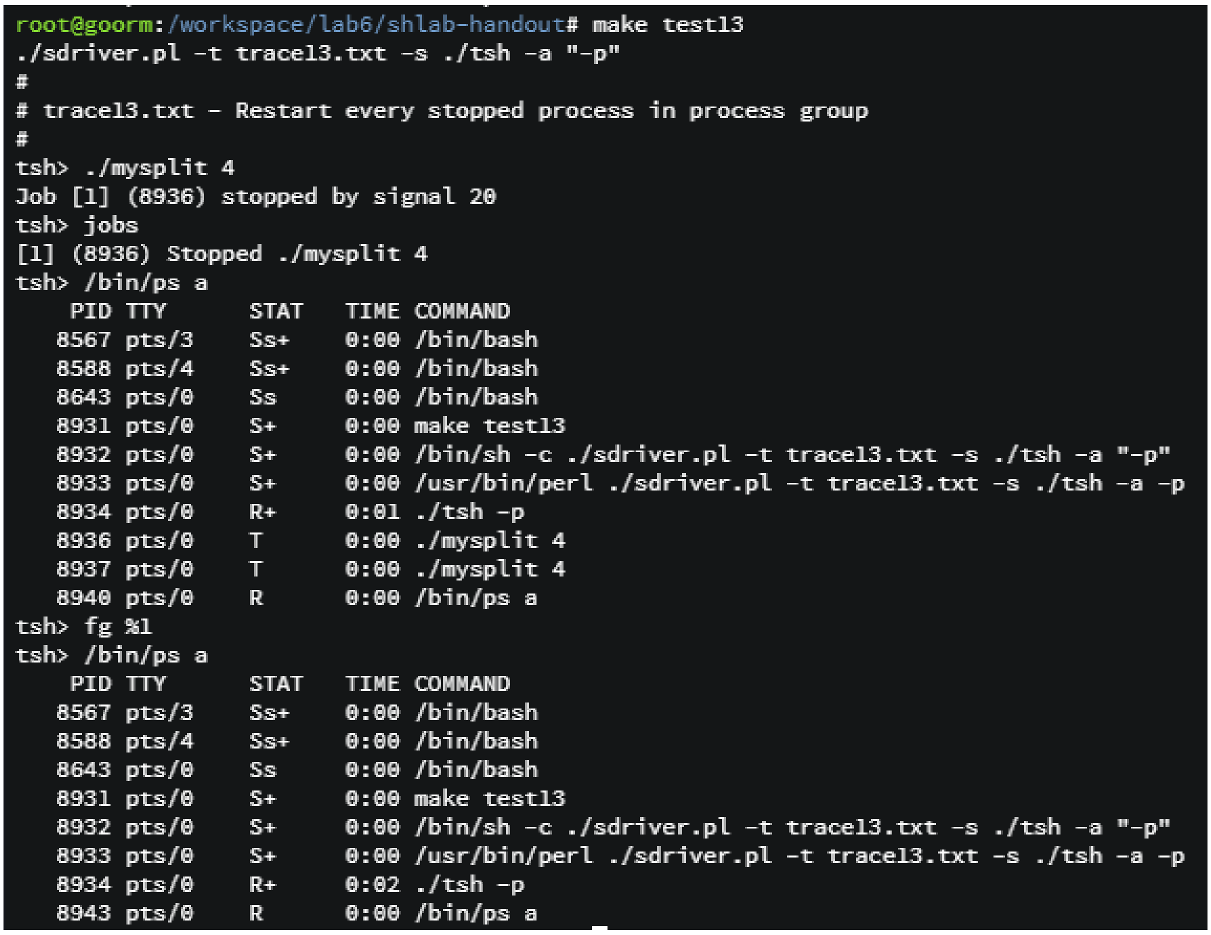
```
root@goorm:/workspace/lab6/shlab-handout# make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (8872) ./myspin 4 &
tsh> ./myspin 5
Job [2] (8874) stopped by signal 20
tsh> jobs
[1] (8872) Running ./myspin 4 &
[2] (8874) Stopped ./myspin 5
root@goorm:/workspace/lab6/shlab-handout# make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (8884) ./myspin 4 &
tsh> ./myspin 5
Job [2] (8886) stopped by signal 20
tsh> jobs
[1] (8884) Running ./myspin 4 &
[2] (8886) Stopped ./myspin 5
tsh> bg %2
[2] (8886) ./myspin 5
tsh> jobs
[1] (8884) Running ./myspin 4 &
[2] (8886) Running ./myspin 5
```



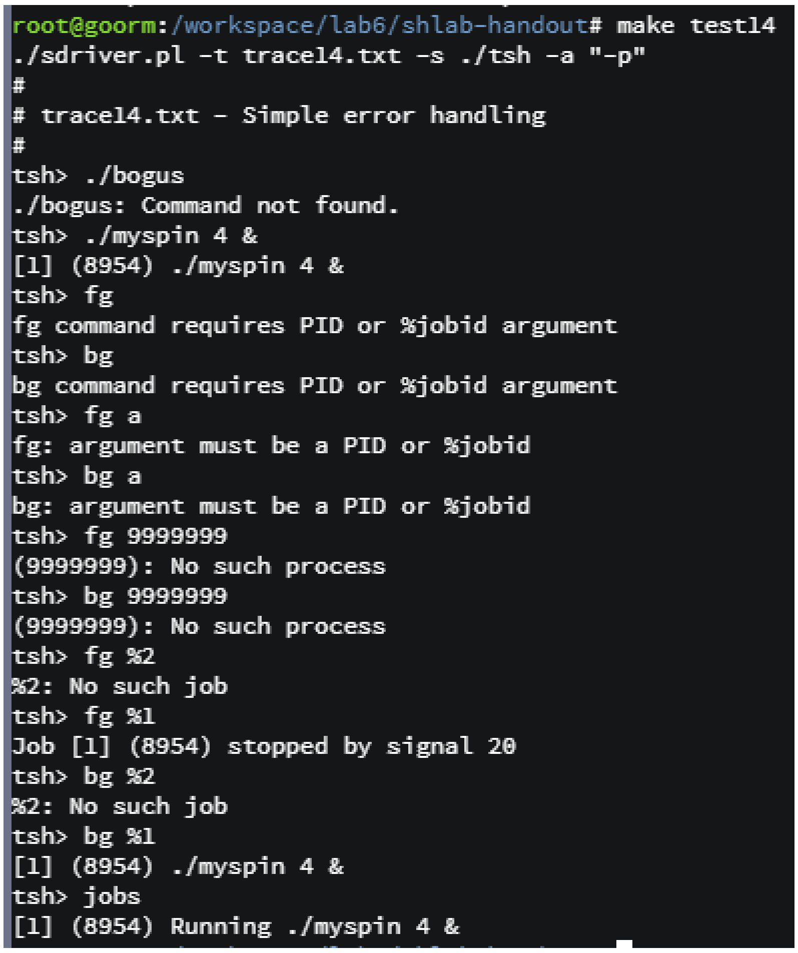
```
root@goorm:/workspace/lab6/shlab-handout# make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (8898) ./myspin 4 &
tsh> fg %1
Job [1] (8898) stopped by signal 20
tsh> jobs
[1] (8898) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
root@goorm:/workspace/lab6/shlab-handout# 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] (8911) terminated by signal 2
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   8567 pts/3    Ss+    0:00 /bin/bash
   8588 pts/4    Ss+    0:00 /bin/bash
   8643 pts/0    Ss     0:00 /bin/bash
   8906 pts/0    S+     0:00 make test11
   8907 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -a "-p"
   8908 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tsh -a -p
   8909 pts/0    R+     0:01 ./tsh -p
   8914 pts/0    R      0:00 /bin/ps a
```

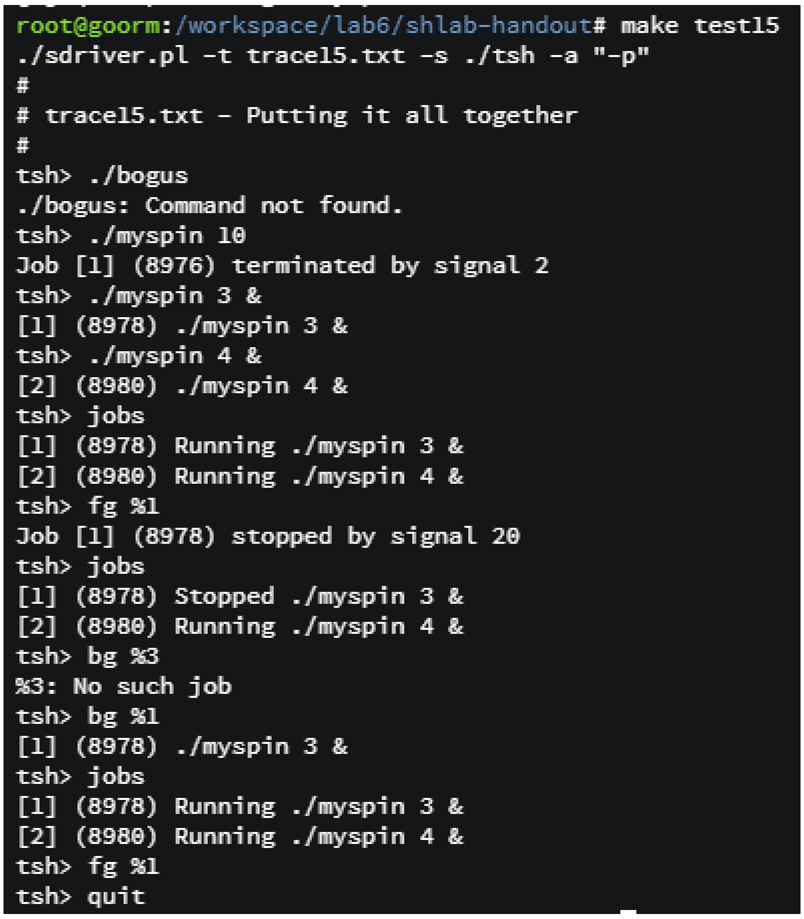


```
root@goorm:/workspace/lab6/shlab-handout# make test12
./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (8923) stopped by signal 20
tsh> jobs
[1] (8923) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   8567 pts/3    Ss+    0:00 /bin/bash
   8588 pts/4    Ss+    0:00 /bin/bash
   8643 pts/0    Ss     0:00 /bin/bash
   8918 pts/0    S+     0:00 make test12
   8919 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
   8920 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p
   8921 pts/0    R+     0:01 ./tsh -p
   8923 pts/0    T      0:00 ./mysplit 4
   8924 pts/0    T      0:00 ./mysplit 4
   8927 pts/0    R      0:00 /bin/ps a
```

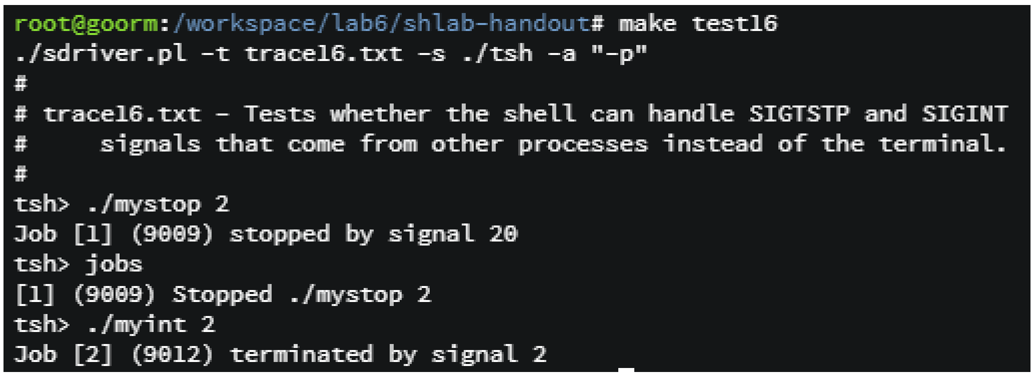


```
root@goorm:/workspace/lab6/shlab-handout# make test13
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (8936) stopped by signal 20
tsh> jobs
[1] (8936) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   8567 pts/3    Ss+    0:00 /bin/bash
   8588 pts/4    Ss+    0:00 /bin/bash
   8643 pts/0    Ss     0:00 /bin/bash
   8931 pts/0    S+     0:00 make test13
   8932 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
   8933 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
   8934 pts/0    R+     0:01 ./tsh -p
   8936 pts/0    T      0:00 ./mysplit 4
   8937 pts/0    T      0:00 ./mysplit 4
   8940 pts/0    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   8567 pts/3    Ss+    0:00 /bin/bash
   8588 pts/4    Ss+    0:00 /bin/bash
   8643 pts/0    Ss     0:00 /bin/bash
   8931 pts/0    S+     0:00 make test13
   8932 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
   8933 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
   8934 pts/0    R+     0:02 ./tsh -p
   8943 pts/0    R      0:00 /bin/ps a
```


```
root@goorm:/workspace/lab6/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] (8954) ./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] (8954) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (8954) ./myspin 4 &
tsh> jobs
[1] (8954) Running ./myspin 4 &
```



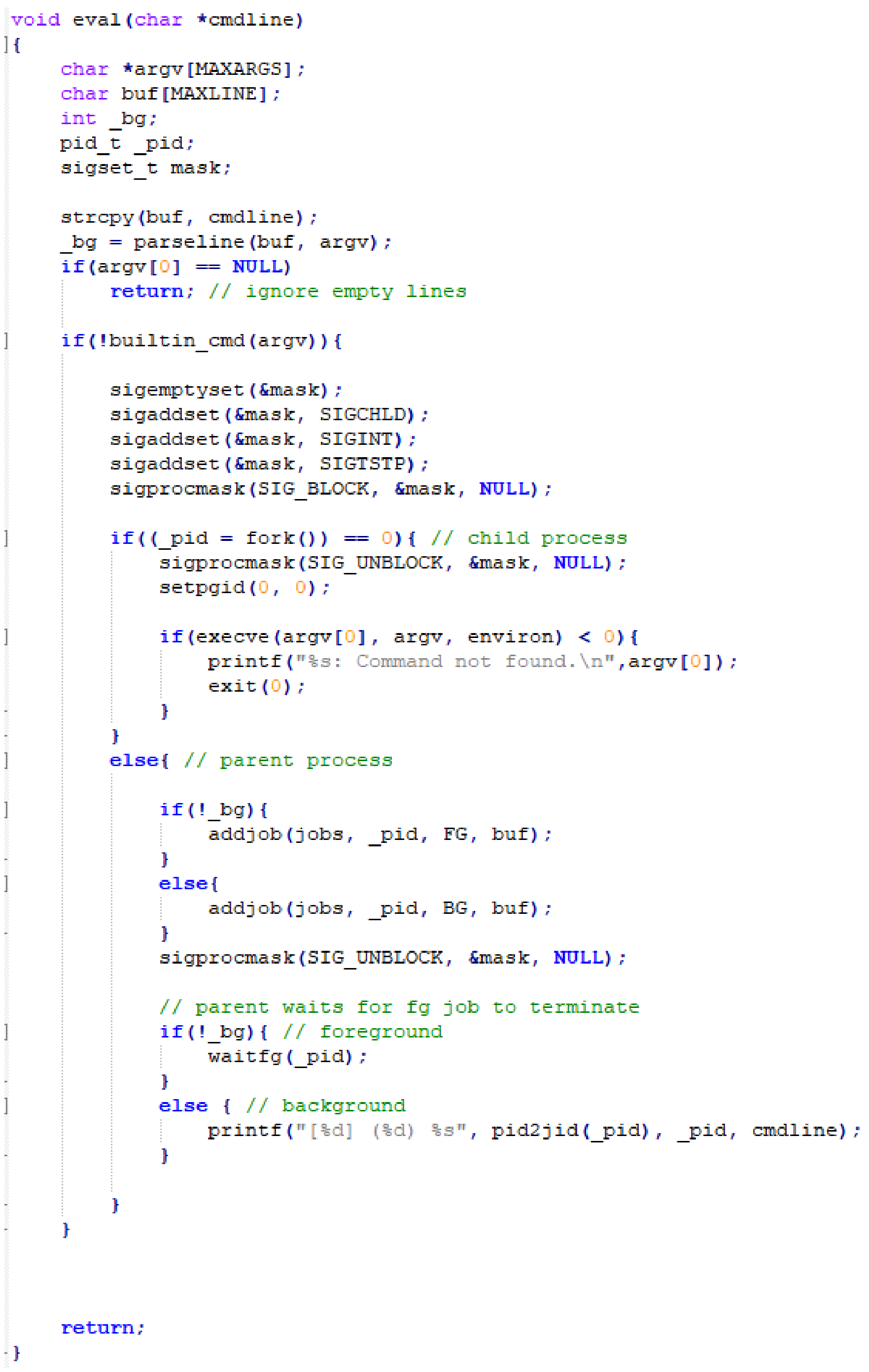
```
root@goorm:/workspace/lab6/shlab-handout# make test15
./sdriver.pl -t trace15.txt -s ./tsh -a "-p"
#
# trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: Command not found.
tsh> ./myspin 10
Job [1] (8976) terminated by signal 2
tsh> ./myspin 3 &
[1] (8978) ./myspin 3 &
tsh> ./myspin 4 &
[2] (8980) ./myspin 4 &
tsh> jobs
[1] (8978) Running ./myspin 3 &
[2] (8980) Running ./myspin 4 &
tsh> fg %1
Job [1] (8978) stopped by signal 20
tsh> jobs
[1] (8978) Stopped ./myspin 3 &
[2] (8980) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (8978) ./myspin 3 &
tsh> jobs
[1] (8978) Running ./myspin 3 &
[2] (8980) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```



```
root@goorm:/workspace/lab6/shlab-handout# 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] (9009) stopped by signal 20
tsh> jobs
[1] (9009) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (9012) terminated by signal 2
```

2. 코드 설명

2.1. eval 함수

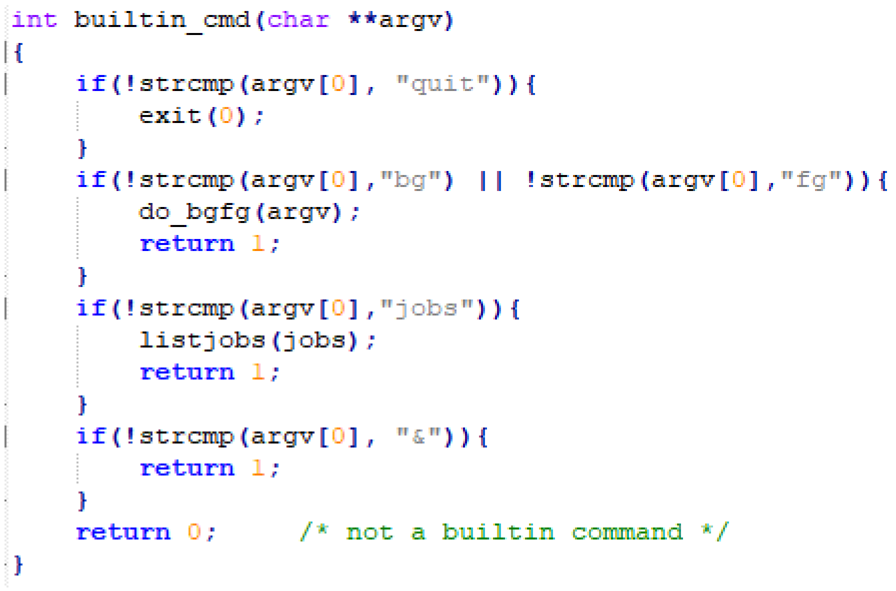


```c
void eval(char *cmdline)
{
    char *argv[MAXARGS];
    char buf[MAXLINE];
    int _bg;
    pid_t _pid;
    sigset_t mask;

    strcpy(buf, cmdline);
    _bg = parseline(buf, argv);
    if(argv[0] == NULL)
        return; // ignore empty lines

    if(!builtin_cmd(argv)){

        sigemptyset(&mask);
        sigaddset(&mask, SIGCHLD);
        sigaddset(&mask, SIGINT);
        sigaddset(&mask, SIGTSTP);
        sigprocmask(SIG_BLOCK, &mask, NULL);

        if((_pid = fork()) == 0){ // child process
            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);
            }
        }
        else{ // parent process

            if(!_bg){
                addjob(jobs, _pid, FG, buf);
            }
            else{
                addjob(jobs, _pid, BG, buf);
            }
            sigprocmask(SIG_UNBLOCK, &mask, NULL);

            // parent waits for fg job to terminate
            if(!_bg){ // foreground
                waitfg(_pid);
            }
            else { // background
                printf("[%d] (%d) %s", pid2jid(_pid), _pid, cmdline);
            }

        }
    }


    return;
}
```

Eval은 커맨드라인으로 받은 문자열을 파싱해서 적절한 프로그램 혹은 기능을 수행하도록 하는 함수이다. Readme에 나와있는 것처럼 race condition을 해결하기 위해 적절한 signal masking을 해준다. Setpgid를 통해 내가 만든 shell에서 잘 동작하도록 process group ID를 설정해준다. 그리고 프로그램이 child process에서 동작하도록 한다. parent process에서는 background 혹은

foreground에 맞게 job 배열에 작업을 추가해준다.

2.2. builtin_cmd 함수

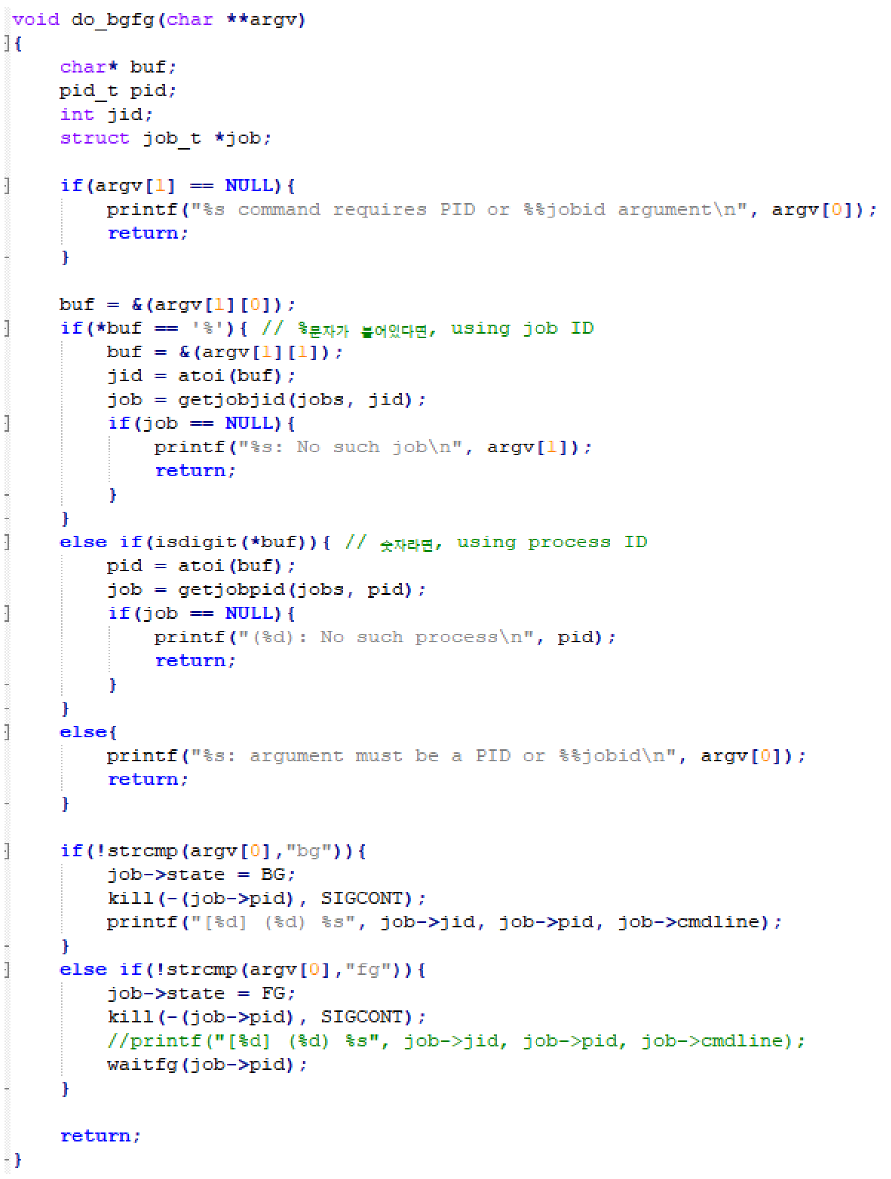


```c
int builtin_cmd(char **argv)
{
    if(!strcmp(argv[0], "quit")){
        exit(0);
    }
    if(!strcmp(argv[0],"bg") || !strcmp(argv[0],"fg")){
        do_bgfg(argv);
        return 1;
    }
    if(!strcmp(argv[0],"jobs")){
        listjobs(jobs);
        return 1;
    }
    if(!strcmp(argv[0],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

각각의 명령어에 맞는 기능을 수행하도록 한다.

Jobs는 기존에 주어진 함수를 이용하였고 bg와 fg는 함수를 추가로 만들어서 사용하였다.

2.3. do_bgfg 함수



```c
void do_bgfg(char **argv)
{
    char* buf;
    pid_t pid;
    int jid;
    struct job_t *job;

    if(argv[1] == NULL){
        printf("%s command requires PID or %%jobid argument\n", argv[0]);
        return;
    }

    buf = &(argv[1][0]);
    if(*buf == '%'){ // %문자가 붙어있다면, using job ID
        buf = &(argv[1][1]);
        jid = atoi(buf);
        job = getjobjid(jobs, jid);
        if(job == NULL){
            printf("%s: No such job\n", argv[1]);
            return;
        }
    }
    else if(isdigit(*buf)){ // 숫자라면, using process ID
        pid = atoi(buf);
        job = getjobpid(jobs, pid);
        if(job == NULL){
            printf("(%d): No such process\n", pid);
            return;
        }
    }
    else{
        printf("%s: argument must be a PID or %%jobid\n", argv[0]);
        return;
    }

    if(!strcmp(argv[0],"bg")){
        job->state = BG;
        kill(-(job->pid), SIGCONT);
        printf("[%d] (%d) %s", job->jid, job->pid, job->cmdline);
    }
    else if(!strcmp(argv[0],"fg")){
        job->state = FG;
        kill(-(job->pid), SIGCONT);
        //printf("[%d] (%d) %s", job->jid, job->pid, job->cmdline);
        waitfg(job->pid);
    }

    return;
}
```

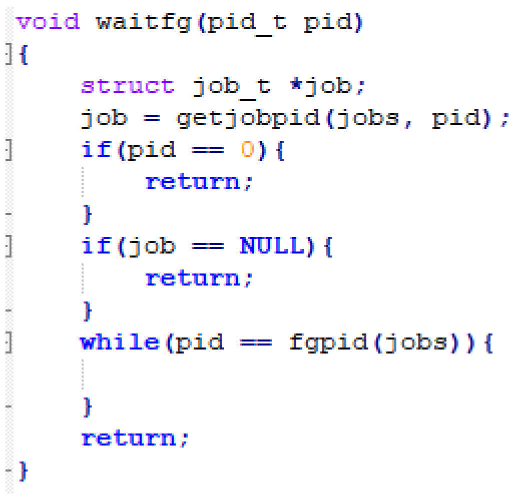
빌트인 명령어인 bg 혹은 fg를 받았을 때 처리하는 함수이다.

명령어 뒤에 대상 프로세스가 입력되지 않았다면 오류 메시지를 출력한다.

해당 프로세스(혹은 job ID)가 존재하지 않을 경우에도 오류 메시지를 출력한다.

Stopped process일 경우에는 다시 시작하도록 해야 한다. 따라서 SIGCONT 시그널을 보낸다.

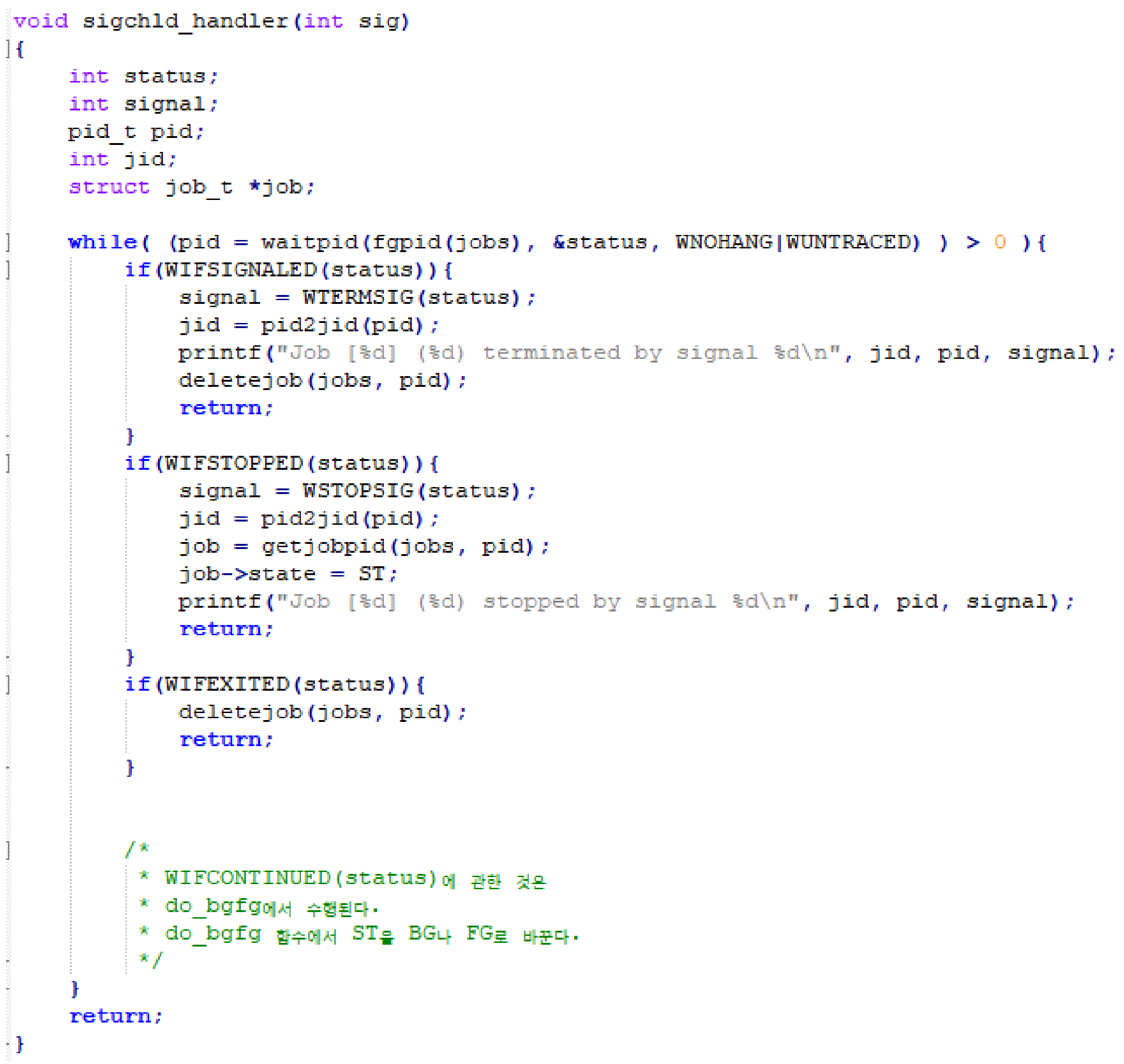
2.4. waitfg 함수



```
void waitfg(pid_t pid)
{
    struct job_t *job;
    job = getjobpid(jobs, pid);
    if(pid == 0){
        return;
    }
    if(job == NULL){
        return;
    }
    while(pid == fgpid(jobs)){

    }
    return;
}
```

Foreground 프로세스가 종료되기를 기다리기 위하여 busy loop를 둔다. Busy loop란 저렇게 반복문 안에서 아무런 기능도 수행되지 않으면서 반복문의 조건이 거짓이 되기를 기다리는 것을 말한다.

2.5. sigchld_handler 함수



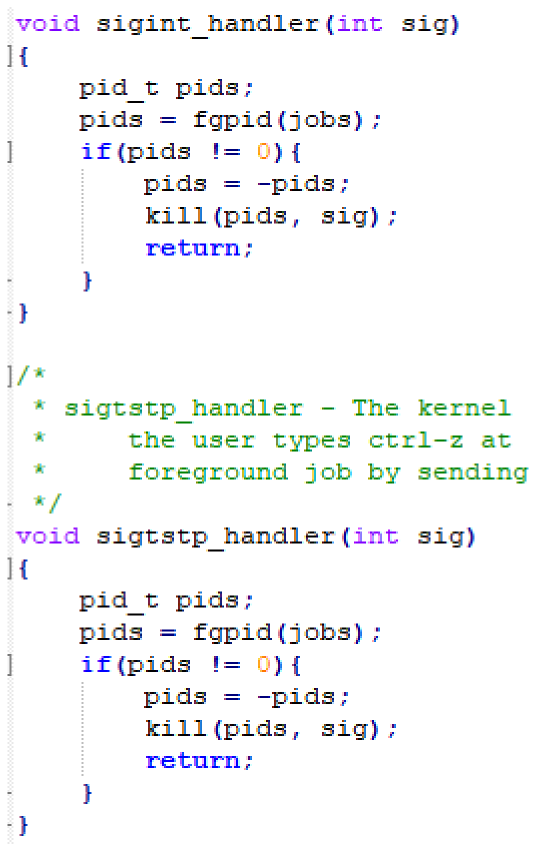
```c
void sigchld_handler(int sig)
{
    int status;
    int signal;
    pid_t pid;
    int jid;
    struct job_t *job;

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


        /*
         * WIFCONTINUED(status)에 관한 것은
         * do_bgfg에서 수행된다.
         * do_bgfg 함수에서 ST을 BG나 FG로 바꾼다.
         */
    }
    return;
}
```

SIGCHLD는 child process가 정지하거나 종료되거나 정지되어 있다가 다시 시작될 때 발생한다. 그래서 그러한 경우들에 대해 모두 처리를 해주도록 한다.

그것을 위해 WIFSIGNALED, WIFSTOPPED, WIFEXITED 등의 미리 지정되어 있는 것들을 사용하였다.

WIFCONTINUED의 경우에는 do_bgfg에서 처리되므로 따로 처리하지 않았다.

2.6. 나머지 handler



```
void sigint_handler(int sig)
{
    pid_t pids;
    pids = fgpid(jobs);
    if(pids != 0){
        pids = -pids;
        kill(pids, sig);
        return;
    }
}

/*
 * sigtstp_handler - The kernel
 *     the user types ctrl-z at
 *     foreground job by sending
 */
void sigtstp_handler(int sig)
{
    pid_t pids;
    pids = fgpid(jobs);
    if(pids != 0){
        pids = -pids;
        kill(pids, sig);
        return;
    }
}
```

사실 sigint handler와 sigtstp handler는 구조적인 차이가 없다.

각각의 handler들이 호출될 때 어떤 시그널에 의해서 호출되었는지 sig라는 매개변수에 의해 알 수 있으므로 그것을 다시 그대로 내가 원하는 process들에 던져주면 되기 때문이다.

따라서 오류 처리를 해주고 대상이 될 process group 전체에 시그널을 보내도록 하였다.