컴퓨터 SW시스템개론 Lab6 보고서

20210207 이지현

Ⅰ. 소개

process control과 signalling을 다뤄보는 것을 목표로 하고, eval builtin_cmd, do_bgfg, waitfg, sigchld_handler, sigint_handler, sigtstp_handler를 구현하고 16가지 trace에 대해 올바른 구현이 이루어지는 것을 목표로 한다.

Ⅱ. Lab 내용 요약 (10)

-Shell과 Shell Lab에 대해서 배웠다. command line을 통해 상호작용 하며 프로그램을 실행시킨다. shell의 BASIC FUNCTION에는 jobs, bg, fg, cirl+c, cirl+z 등이 있고, job은 background job을 list하며, bg는 stopped background job을 running background로 바꾸며, fg는 background job을 running foreground로 바꾼다. cirl+c는 sigint signal을 foreground process에 보내고 cirl+z는 sigtstp signal을 foreground process에 보낸다.

- Shell lab은 7개의 specified function을 작성해야하고, 4개의 프로그램 myint myspin mysplit mystop이 tsh에 의해 사용된다.

Ⅲ. 코드 설명

(1) Trace01

```
[io0818@programming2 ~]$ make rtest01
./sdriver.pl -t trace01.txt -s ./tshref -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
```

EOF (컨트롤 + D)가 입력되면 terminate되도록 구현해야하는데, main 함수에 보면 이미 구현되어 있기 때문에 따로 아무 코드도 추가하지 않아도 레퍼런스와 똑같이 실행됨을 알 수 있다.

```
if (feof(stdin)) { /* End of file (ctrl-d) */
    fflush(stdout);
    exit(0);
}
```

(2) Trace02

```
[io0818@programming2 ~]$ make rtest02
./sdriver.pl -t trace02.txt -s ./tshref -a "-p"
#
# trace02.txt - Process builtin quit command.
#
```

입력받은 command가 built-in 명령어 중 quit이라면 shelll을 종료하도록 구현해야한다. 입력 받은 명령어를 eval() 함수에서 pasrseline() 함수를 통해 분할하고 builtin_cmd() 함수를 통해 quit에 따른 쉘 종료 조건을 설정해줄 수 있다.

```c
void eval(char *cmdline)
{
    char* argv[MAXARGS];
    parseline(cmdline, argv);
    builtin_cmd(argv);
    return;
}
```

```c
int builtin_cmd(char **argv)
{
    char *command = argv[0];
    if(strcmp(command, "quit") == 0){
        exit(0);
    }

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

argv[0]는 입력받은 명령어에서 공백 이전까지 읽은 값을 나타낸다 (여기선 command가 argv[0]의 string("quit")이 될 것이고, 특히 2차원 배열로 분할해서 각각의 char 1개씩을 분할해서 사용하기도 한다 -> argv[0][0] = q, argv[0][1] = u, argv[0][2] = I, argv[0][3] = t) 이렇게 eval에서 호출한 builtin_cmd 함수에서 argv[0]에 있는 comamnd를 변수를 만들어 담고, command와 quit 문자열을 비교하여 같으면 exit(0)을 통해 shell을 종료하도록 설정할 수 있다.

(3) Trace03

```
[io0818@programming2 ~]$ make rtest03
./sdriver.pl -t trace03.txt -s ./tshref -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
```

command가 built-in 명령어가 아닐 경우 foreground 형태로 프로그램이 실행되도록 구현해야한다. builtin_cmd(argv) 함수에서 return 값은 built-in 명령어가 아닐 경우 0이 리턴되므로 !builtin_cmd를 통해 조건을 구현할 수 있고, fork()를 통해 child process를 생성하고, 프로그램이 정상적으로 실행되지 않을 경우 execve()가 음수를 반환하므로 이 때

Command not found 문구를 출력하고 shell을 종료시킨다. (execve함수는 현재 프로세스를 다른 프로그램으로 대체하는 함수로, 에러가 발생하지 않으면 반환값은 없다)

```
void eval(char *cmdline)
{
    char* argv[MAXARGS];
    parseline(cmdline, argv);

    //if cmd is not bulitin command
    if(!builtin_cmd(argv)){
        if(fork()==0){
            if(execve(argv[0], argv, environ)<0){
                printf("%s : Command not found\n\n", argv);
                exit(0);
            }
        }
    }
    return;
}
```

(4) Trace04

```
[io0818@programming2 ~]$ make rtest04
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (5018) ./myspin 1 &
```

명령어 끝에 & 가 붙으면 Background 형태로 프로그램이 실행되어 내용을 출력되도록 하도록 구현해야한다. 단순히 bg를 parseline으로 받아서, bg냐 fg냐에 따라 자식 프로세스를 job list에 등록하는 addjob함수를 호출하였다.

```
/* should the job run in the background? */
if ((bg = (*argv[argc-1] == '&')) != 0) {
argv[--argc] = NULL;
}
return bg;
```

위는 parseline()의 주어진 코드인데, 마지막 문자가 &일 때, background이면 BG(=1), 를 bg로 리턴해주는 것을 볼 수 있다. 따라서 !bg를 조건으로 설정해서 child process가 foreground인 경우 waitpid를 통해 기다리다가 deletejob을 통해 제거하고, background인 경우 레퍼런스처럼 child process의 PID와 command를 출력하는 코드를 printf로 작성할 수 있다.

이 때 trace03과 살짝 다른 부분이 fork() == 0 (부모는 양수, 자식은 0, 실패는 음수)이라는 child process를 특정짓기 위한 조건문 대신에 pid = fork() 이후 pid == 0 이라는 조건

문을 사용함으로써 pid도 동시에 정의되게 할 수 있다.

```c
void eval(char *cmdline)
{
    char* argv[MAXARGS];
    pid_t pid;
    int bg = parseline(cmdline, argv);

    //if cmd is not bulitin command
    if(!builtin_cmd(argv)){
        pid = fork();
        if(pid==0){
            //if execve = -1, error
            if(execve(argv[0], argv, environ)<0){
                printf("%s : Command not found\n\n", argv);
                exit(0);
            }
        }
    }

    //whether it is bg or fg
    int x;
    if(bg == 1) x = BG;
    else x = FG;

    addjob(jobs, pid, x, cmdline);

    if(!bg){
        waitpid(pid,NULL,0);
        deletejob(jobs,pid);
    }
    else{
        printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
    }
    return;
}
```

(5) Trace05

```
[io0818@programming2 ~]$ make rtest05
./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (7131) ./myspin 2 &
tsh> ./myspin 3 &
[2] (7133) ./myspin 3 &
tsh> jobs
[1] (7131) Running ./myspin 2 &
[2] (7133) Running ./myspin 3 &
```

background process 2개를 실행하고 jobs 명령어를 받아 job list를 출력하는 것을 구현해야한다. jobs 또한 built-in 명령어이므로 builtin_cmd 함수를 수정해야하고, quit과 마찬가지로 command == "jobs"이면 listjobs() (이미 구현되어 잇는 jobs 출력 함수)를 호출하도록 작성하면 된다.

```
int builtin_cmd(char **argv)
{
    char *command = argv[0];
    if(strcmp(command, "quit") == 0){
        exit(0);
    }
    else if(strcmp(command, "jobs") == 0){
        listjobs(jobs);
        return 1; //built-in command -> return 1
    }

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

앞서 작성한 builtin_cmd 함수에 command가 jobs인 조건도 추가해서 listjob을 호출하고 return 1 (builtin command 이므로) 하는 statement를 추가할 수 있다.

(6) Trace06

```
[io0818@programming2 ~]$ make rtest06
./sdriver.pl -t trace06.txt -s ./tshref -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (14660) terminated by signal 2
[io0818@programming2 ~]$
```

./myspin 4를 입력시 child process가 Foreground으로 실행하는 프로세스를 부모 프로세스에게 SIGINT를 전달해 자식의 해당 프로세스 작업을 terminate 시키는 것을 구현해야한다.

우선 eval() 함수에서 signal set을 초기화하고, SIGCHLD signal을 추가하였다. 이후, fork로 자식이 생성되는 동안 kill signal을 전달받지 않도록 signal set을 blocking해 놓고, addjob 실행 이전에 부모 process가 child process를 종료시키면 안되기 때문에 addjob이 끝나고 난 이후에 signal unblocking을 해준다. 주의해야하는 점은, fork는 복제 프로세스이기 때문에 blocking까지 복사해버려 다음 signal을 받을 수 없기에 fork 이후에도 unblocking을 해주어야 한다. 즉, unblocking을 부모 자식에 각각 2번 필요하다. 또, 중간에 setpgid(0,0)를 통해 현재 process (pid=0)의 group id를 0으로 설정했다.

```c
void eval(char *cmdline)
{
    char *argv[MAXARGS];
    int bg = parseline(cmdline, argv);
    pid_t pid;

    // SIGNAL setting
    sigset_t mask, prev;
    sigemptyset(&mask);
    sigaddset(&mask, SIGCHLD);
    // if cmd is not builtin command
    if (!builtin_cmd(argv))
    {
        sigprocmask(SIG_BLOCK, &mask, &prev); // BLOCKING
        if ((pid = fork()) == 0)
        {
            sigprocmask(SIG_SETMASK, &prev, NULL); // UNBLOCKING
            setpgid(0,0);

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

```c
    //wheter it is bg or fg
    int x;
    if(bg == 1) x = BG;
    else x = FG;

  addjob( jobs, pid, x, cmdline);
  sigprocmask(SIG_SETMASK, &prev, NULL); //UNBLOKCING

  if (!bg)
   {
       //until there is no foreground in job lists
     while(pid == fgpid(jobs)){
           sleep(1);
       }
  }
  else{
      printf("[%d] (%d) %s",pid2jid(pid), pid, cmdline);
    }

}
 return;
```

그 다음 sigint handler를 작성해야했는데, foreground job이 있으면 그 job의 PID를 반환시키고, 아니면 0을 반환시켜 pid 변수에 넣었고 (fgpid), pid가 0이면 (fg 없으면) sigint를 받고 kill 되므로 SIGCHLD를 발생시키게 되고, parent process는 SIGCHLD handler로 child를 처리해줘야 한다.

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

SIGCHLD handler도 구현을 해보면, waitpid로 WNOHANG (기다리는 PID가 종료되지 않아서 즉시 종료할 수 없는 상황)이나 WUNTRACED(중단된 상태)인 임의의 자식 프로세스(-1)를 기다리고, 해당 PID를 받는다. WIFSIGNALED 일 때, child가 signal에 의해 종료되었으면 Jobs 내용을 반환하고 deletejob한다.

```c
void sigchld_handler(int sig)
{
    int status = 0;
    pid_t pid;
    while ((pid = waitpid(-1, &status, WNOHANG|WUNTRACED))>0){

        //There is Terminate signal -> delete
        if(WIFSIGNALED(status)){
            printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(pid), pid, WTERMSIG(status));
            deletejob(jobs,pid);
        }
        else{
            deletejob(jobs, pid);
        }
    }
    return;
}
```

(7) Trace07

```
[io0818@programming2 ~]$ make rtest07
./sdriver.pl -t trace07.txt -s ./tshref -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (11297) ./myspin 4 &
tsh> ./myspin 5
Job [2] (11299) terminated by signal 2
tsh> jobs
[1] (11297) Running ./myspin 4 &
```

foreground는 SIGINT 보내고 background만 jobs 하는 process를 구현해야한다. 위와 같은 기능을 담당하므로 같은 코드를 사용해도 무방하다.

(8) Trace08

```
[io0818@programming2 ~]$ make rtest08
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (17937) ./myspin 4 &
tsh> ./myspin 5
Job [2] (17941) stopped by signal 20
tsh> jobs
[1] (17937) Running ./myspin 4 &
[2] (17941) Stopped ./myspin 5
```

이번엔 foreground job에 대해 SIGTSTP SIGNAL을 보내는 것을 구현해야한다. 8번과 유사한 방법으로 SIGTSTP HANLDER를 구현해주면 된다.

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

이후, child handler에서도 WIFSTOPPED 일 때, Signal이 멈춘 상태이면 ST state로 변환시키고 마찬가지로 job 프린트한다. 이는 SIGTSTOP을 부모 프로세스에게 전달 했을 때, 자식의 foreground state를 ST으로 바꾸어주는 역할을 맡는다.

```c
void sigchld_handler(int sig)
{
    int status = 0;
    pid_t pid;
    while ((pid = waitpid(-1, &status, WNOHANG|WUNTRACED))>0){

        //There is Terminate signal -> delete
        if(WIFSIGNALED(status)){
            printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(pid), pid, WTERMSIG(status));
            deletejob(jobs,pid);
        }
        //child to ST state
        else if (WIFSTOPPED(status)){
            getjobpid(jobs, pid)->state = ST;
            printf("Job [%d] (%d) stopped by signal %d\n", pid2jid(pid), pid, WSTOPSIG(status));
        }
        else{
            deletejob(jobs, pid);
        }
    }
    return;
}
```

또, bg command에 대해 builtin_cmd를 설정해줄 수 있다. command가 bf이면 do_bgfg() 함수로 가도록 설정하고, 함수 내에서는

```c
int builtin_cmd(char **argv)
{
    char *command = argv[0];
    if(strcmp(command, "quit") == 0){
        exit(0);
    }
    else if(strcmp(command, "jobs") == 0){
        listjobs(jobs);
        return 1; //built-in command -> return 1
    }

    if(strcmp(command, "bg")){
        do_bgfg(); //if command is bg of fg -> call
        return 1; //built-in command -> return 1
    }

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

(9) Trace09

```
[io0818@programming2 ~]$ make rtest09
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (2566) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2569) stopped by signal 20
tsh> jobs
[1] (2566) Running ./myspin 4 &
[2] (2569) Stopped ./myspin 5
tsh> bg %2
[2] (2569) ./myspin 5
tsh> jobs
[1] (2566) Running ./myspin 4 &
[2] (2569) Running ./myspin 5
[io0818@programming2 ~]$
```

built-in command 중 bf process를 구현해야한다. 우선 첫째로 builtin_cmd 함수에서 bg인 경우를 처리하는 함수 do_bgfg()를 call하고 built-in command이므로 return 1을 하는 코드를 작성한다. 이후, do_bfgf() 함수에서 argv를 넘겨주고 두번째 값이 %인 경우 jid을 넘겨 받아 job을 생성할 수도 있고, 두번째 값이 숫자(0~9)인 경우 pid를 넘겨 받아 job을 생성할 수도 있다. 첫번째 값이 bg이면 jb의 state가 BG(background)가 되 도록 변경해주고 해당 상태를 printf하도록 한다. 단, 여기서 block된 signal은 무시하는 예외처리를 해줘야 정상적으로 process가 끝날 수 있다. (kill = signal 죽이기, sigcont = blocked된 signal)

```c
int builtin_cmd(char **argv)
{
    char *command = argv[0];
    if (strcmp(command, "quit") == 0)
    {
        exit(0);
    }
    else if (strcmp(command, "jobs") == 0)
    {
        listjobs(jobs);
        return 1; // built-in command -> return 1
    }

    if (strcmp(command, "bg") == 0)
    {
        do_bgfg(argv); // if command is bg of fg -> call
        return 1;      // built-in command -> return 1
    }

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

```c
//number 0~9 -> pid
if(argv[1][0] == '0' || argv[1][0] == '1' || argv[1][0] == '2' || argv[1][0] == '3' || argv[1][0] == '4'
|| argv[1][0] == '5' || argv[1][0] == '6' || argv[1][0] == '7' || argv[1][0] == '8' || argv[1][0] == '9')
{
    pid = atoi(argv[1]);
    jb = getjobpid(jobs, pid);
    if(jb == NULL){
        printf("(%s): No such process\n", argv[1]);
        return;
    }
}

//% job
else if (argv[1][0] == '%')
{
    jid = atoi(&argv[1][1]);
    jb = getjobjid(jobs, jid);
    if(jb == NULL){
        printf("%s: No such job\n", argv[1]);
        return;
    }
}
```

```c
//background job
 if (strcmp(argv[0], "bg") == 0)
 {
   if(kill(-pid, SIGCONT)<0){ //blocked signal kill
        printf("do_bgfg(): kill error");
     }
     jb->state = BG;
     printf("[%d] (%d) %s", jb->jid, jb->pid, jb->cmdline);

 }
```

(10) Trace10

```
[io0818@programming2 ~]$ make rtest10
./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (19632) ./myspin 4 &
tsh> fg %1
Job [1] (19632) stopped by signal 20
tsh> jobs
[1] (19632) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
[io0818@programming2 ~]$
```

built-in command 중 fg command를 처리하는 process 구현을 해야한다. 위 레퍼런스를 보면 알 수 있듯이 fg를 통해 foreground로 변경시키고 job list에서 사라짐을 알 수 있다. trace09와 마찬가지로 builtin_cmd 함수에 fg인 경우를 추가해서 do_bgfg로 가게 한 다음, 두 번째 인자가 digit인 경우와 첫번째 인자가 fg인 경우를 추가해주면 된다.

```c
int builtin_cmd(char **argv)
{
    char *command = argv[0];
    if (strcmp(command, "quit") == 0)
    {
        exit(0);
    }

    if (strcmp(command, "jobs") == 0)
    {
        listjobs(jobs);
        return 1; // built-in command -> return 1
    }

    if (strcmp(command, "bg") == 0)
    {
        do_bgfg(argv); // if command is bg of fg -> call
        return 1;      // built-in command -> return 1
    }

    if (strcmp(command, "fg") == 0)
    {
        do_bgfg(argv); // if command is bg of fg -> call
        return 1;// built-in command -> return 1
    }

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

특히 do_bgfg의 fg에서 foreground process가 들어올때까지 기다리는 함수가 필요했는데, 이는 waitfg함수를 따로 구현해서 해결할 수 있었다. (앞서도 foreground process를 기다려야하는 코드가 있었지만 그냥 이 파트에서만 사용하기로 했다)

```c
//number 0~9 -> pid
if(argv[1][0] == '0' || argv[1][0] == '1' || argv[1][0] == '2' || argv[1][0] == '3' || argv[1][0] == '4'
|| argv[1][0] == '5' || argv[1][0] == '6' || argv[1][0] == '7' || argv[1][0] == '8' || argv[1][0] == '9')
{
    pid = atoi(argv[1]);
    jb = getjobpid(jobs, pid);
    if(jb == NULL){
        printf("(%s): No such process\n", argv[1]);
        return;
    }
}

//% job
else if (argv[1][0] == '%')
{
    jid = atoi(&argv[1][1]);
    jb = getjobjid(jobs, jid);
    if(jb == NULL){
        printf("%s: No such job\n", argv[1]);
        return;
    }
}
```

```
//background job
 if (strcmp(argv[0], "bg") == 0)
 {
   if(kill(-pid, SIGCONT)<0){ //blocked signal kill
        printf("do_bgfg(): kill error");
     }
     jb->state = BG;
     printf("[%d] (%d) %s", jb->jid, jb->pid, jb->cmdline);

 }

//foreground job
 if (strcmp(argv[0], "fg") == 0)
 {
   if(kill(-pid, SIGCONT)<0){ //blocked signal kill
        printf("do_bgfg(): kill error");
     }
     jb->state = FG;
     //until foreground is done
     waitfg(jb->pid);

 }
 return;
```

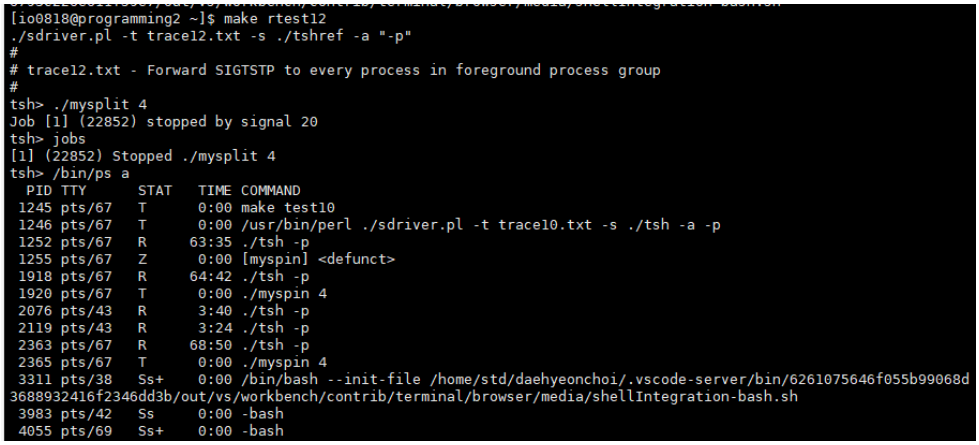
```
/*
 * waitfg - Block until process pid is no longer the foreground process
 */
void waitfg(pid_t pid)
{
    struct job_t *job = getjobpid(jobs, pid);

    //pid cannot be 0
    if(pid==0) return;
    else{
       //job cannot be 0
       if(job == NULL) return;
         else{
            while(pid == fgpid(jobs)){
                sleep(1);
            }
       }
      }
     return;
}
```

(11) Trace11

```
[io0818@programming2 ~]$ 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] (21941) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  966 pts/63   Ss+    0:00 /usr/bin/bash --init-file /home/std/jueunk/.vscode-server/bin/74b1f979648cc44d385a228
6793c226e611f59e7/out/vs/workbench/contrib/terminal/browser/media/shellIntegration-bash.sh
 1245 pts/67   T      0:00 make test10
 1246 pts/67   T      0:00 /usr/bin/perl ./sdriver.pl -t trace10.txt -s ./tsh -a -p
 1252 pts/67   R     63:22 ./tsh -p
 1255 pts/67   Z      0:00 [myspin] <defunct>
 1918 pts/67   R     64:29 ./tsh -p
 1920 pts/67   T      0:00 ./myspin 4
 2076 pts/43   R      3:26 ./tsh -p
 2119 pts/43   R      3:10 ./tsh -p
 2363 pts/67   R     68:36 ./tsh -p
 2365 pts/67   T      0:00 ./myspin 4
 3311 pts/38   Ss+    0:00 /bin/bash --init-file /home/std/daehyeonchoi/.vscode-server/bin/6261075646f055b99068d
3688932416f2346dd3b/out/vs/workbench/contrib/terminal/browser/media/shellIntegration-bash.sh
 3983 pts/42   Ss     0:00 -bash
 4055 pts/69   Ss+    0:00 -bash
 4299 pts/30   Ss+    0:00 -bash
 4438 pts/45   Ss     0:00 -bash
 4467 pts/50   Ss+    0:00 -bash
 4687 pts/29   Ss+    0:00 -bash
 5162 pts/51   Ss     0:00 -bash
 5234 tty1     Ss+    0:00 /sbin/agetty --noclear tty1 linux
```

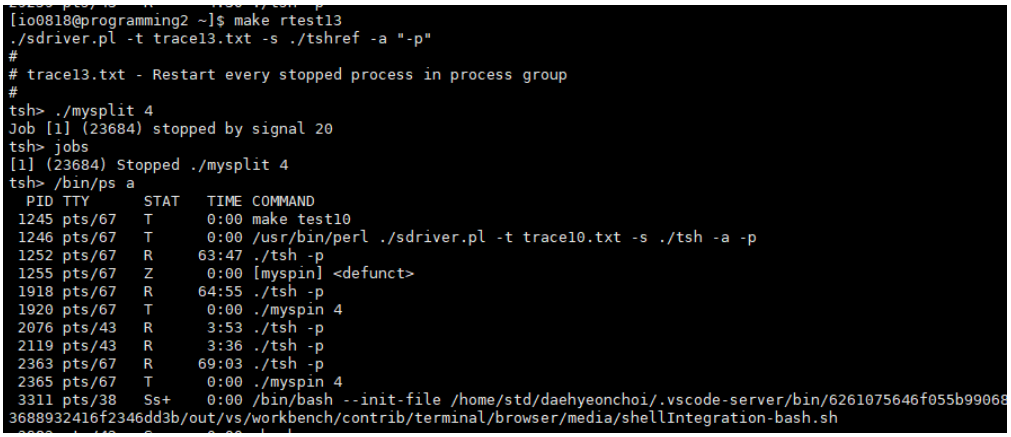
모든 process에 대해 foreground process group에 SIGINT SIGNAL을 보내지는 지 확인하면 된다. 이전 Trace부터 줄곧 모든 fg에 대해 wait해서 없어질때까지 job list에서 없애고 deletejob을 통해 terminate 시켜 왔으므로 같은 코드에 대해서도 Trace11에 적용된다고 볼 수 있다.

(12) Trace12



모든 process에 대해 foreground process group에 SIGTSTP SIGNAL가 보내지는 지 확인하면 된다.. 여기도 마찬가지로 이전 Trace부터 줄곧 모든 fd에 대해 wait해서 없어질때까지 process의 state를 ST (STOP)으로 바꿔왔기에, 같은 코드에 대해서도 Trace11에 적용된다고 볼 수 있다.

(13) Trace13



모든 stopped된 process를 재시작하는지를 확인하면 된다. fg %1 이후에 ps를 확인해보니 terminated된 mysplit 4가 없어짐을 알 수 있다. 직접 실행해보니 똑같이 다시 재시작 됨을 알 수 있었다.

(14) Trace14

```
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (26326) ./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] (26326) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (26326) ./myspin 4 &
tsh> jobs
[1] (26326) Running ./myspin 4 &
```

간단한 에러 핸들링을 구현해야한다. bogus, fg, bg, fg a, bg a 등에 대한 에러 핸들링을 레퍼런스와 같이 구현하면 된다. 우선 지정된 command 이외의 커맨드를 입력하면 (ex bogus) Command not found가 입력되도록 구현해야한다. 이는 execve <0일 때의 조건문으로 나타낼 수 있다.

```c
void eval(char *cmdline)
{
    char *argv[MAXARGS];
    int bg = parseline(cmdline, argv);
    pid_t pid;

    // SIGNAL setting
    sigset_t mask, prev;
    sigemptyset(&mask);
    sigaddset(&mask, SIGCHLD);
    // if cmd is not builtin command
    if (!builtin_cmd(argv))
    {
        sigprocmask(SIG_BLOCK, &mask, &prev); // BLOCKING
        if ((pid = fork()) == 0)
        {
            sigprocmask(SIG_SETMASK, &prev, NULL); // UNBLOCKING
            setpgid(0,0);

            if (execve(argv[0], argv, environ) < 0) // if execve = -1, error
            {
                printf("%s: Command not found\n", argv[0]);
                exit(0);
            }
        }
```

이후 입력값의 매개변수의 2번째 입력값이 없거나, Pid나 jobid를 넘겨주지 않았거나, 올바른 pid나 jobid가 아닐경우 에러를 출력하도록 해야한다.

이후 두번째 인자값에 아무것도 들어오지 않았을 경우 에러를 출력하고, pid를 이용해서 받은 job이 NULL인 경우, jid를 이용해 받은 job이 NULL인 경우, PID나 %jobid가 아닌 경우, BLOCKED된 경우 kill error를 뜨게 하면서 에러 처리를 모두 할 수 있고, 각각의 자세한 코드는 아래를 참고할 수 있다.

```c
void do_bgfg(char **argv)
{
    struct job_t *jb = NULL;
    int check1 = 0;
    int check2 = 0;

    //2nd argument is null
    if(argv[1] == NULL){
        if(strcmp(argv[0], "fg") == 0){
            printf("fg command requires PID or %%jobpid argument\n");
        }
        else if (strcmp(argv[0], "bg") == 0){
            printf("bg command requires PID or %%jobpid argument\n");
        }
        return;
    }

    //number 0~9 -> pid
    if(argv[1][0] == '0' || argv[1][0] == '1' || argv[1][0] == '2' || argv[1][0] == '3' || argv[1][0] == '4'
    || argv[1][0] == '5' || argv[1][0] == '6' || argv[1][0] == '7' || argv[1][0] == '8' || argv[1][0] == '9')
    {
        pid_t pid = atoi(argv[1]);
        jb
            if(jb == NULL){
            printf("(%s): No such process\n", argv[1]);
            return;
        }
        check1++;
    }
```

```
//% job
else if (argv[1][0] == '%')
{
    jid = atoi(&argv[1][1]);
    jb = getjobjid(jobs, jid);
    if(jb == NULL){
        printf("%s: No such job\n", argv[1]);
        return;
    }
}


//behind fg, bg -> there is no Pid or jobpid
else{
    if(strcmp(argv[0], "fg") == 0){
        printf("fg: argument must be a PID or %%jobid\n");
    }
    else if (strcmp(argv[0], "bg") == 0){
        printf("bg: argument must be a PID or %%jobid\n");
    }
    return;
}

pid = jb->pid;

//background job
if (strcmp(argv[0], "bg") == 0)
{
    if(kill(-pid, SIGCONT)<0){ //blocked signal kill
        printf("do_bgfg(): kill error");
    }
    jb->state = BG;
    printf("[%d] (%d) %s", jb->jid, jb->pid, jb->cmdline);

}

//foreground job
if (strcmp(argv[0], "fg") == 0)
{
    if(kill(-pid, SIGCONT)<0){ //blocked signal kill
        printf("do_bgfg(): kill error");
    }
    jb->state = FG;
    //until foreground is done
    waitfg(jb->pid);

}
return;
```

(15) Trace15

```
[ioo818@programming2 ~]$ 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] (26054) terminated by signal 2
tsh> ./myspin 3 &
[1] (26060) ./myspin 3 &
tsh> ./myspin 4 &
[2] (26063) ./myspin 4 &
tsh> jobs
[1] (26060) Running ./myspin 3 &
[2] (26063) Running ./myspin 4 &
tsh> fg %1
Job [1] (26060) stopped by signal 20
tsh> jobs
[1] (26060) Stopped ./myspin 3 &
[2] (26063) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (26060) ./myspin 3 &
tsh> jobs
[1] (26060) Running ./myspin 3 &
[2] (26063) Running ./myspin 4 &
tsh> fg %1
tsh> quit
[ioo818@programming2 ~]$
```

여러가지 조합의 명령어들에 대해 레퍼런스와 정확하게 동작하는지 확인하면 된다. 위의 trace 1~14에서 사용한 명령어들의 조합이므로 쉽게 통과됨을 확인할 수 있었다.

(16) Trace16

```
[io0818@programming2 ~]$ 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] (26229) stopped by signal 20
tsh> jobs
[1] (26229) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (26235) terminated by signal 2
[io0818@programming2 ~]$
```

STGTSIP와 SIGINT signal을 통해 process의 handle을 하는 trace이다. 마찬가지로 위의 구현들을 사용하는 것이기에 쉽게 통과가 가능했다.

VI. 실행 결과 (test)

```
[io0818@programming2 ~]$ make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
[io0818@programming2 ~]$ make test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
[io0818@programming2 ~]$ make test03
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
tsh> ./myspin 1 &
[1] (8105) ./myspin 1 &
[io0818@programming2 ~]$ make rtest04
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
[io0818@programming2 ~]$ make test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (9765) ./myspin 2 &
tsh> ./myspin 3 &
[2] (9767) ./myspin 3 &
tsh> jobs
[1] (9765) Running ./myspin 2 &
[2] (9767) Running ./myspin 3 &
```

```
[io0818@programming2 ~]$ make test06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (16901) terminated by signal 2
```

```
[io0818@programming2 ~]$ make test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (17356) ./myspin 4 &
tsh> ./myspin 5
Job [2] (17358) terminated by signal 2
tsh> jobs
[1] (17356) Running ./myspin 4 &
```

```
[io0818@programming2 ~]$ make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (18203) ./myspin 4 &
tsh> ./myspin 5
Job [2] (18205) stopped by signal 20
tsh> jobs
[1] (18203) Running ./myspin 4 &
```

```
[io0818@programming2 ~]$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (16432) ./myspin 4 &
tsh> ./myspin 5
Job [2] (16434) stopped by signal 20
tsh> jobs
[1] (16432) Running ./myspin 4 &
[2] (16434) Stopped ./myspin 5
tsh> bg %2
[2] (16434) ./myspin 5
tsh> jobs
[1] (16432) Running ./myspin 4 &
[2] (16434) Running ./myspin 5
```

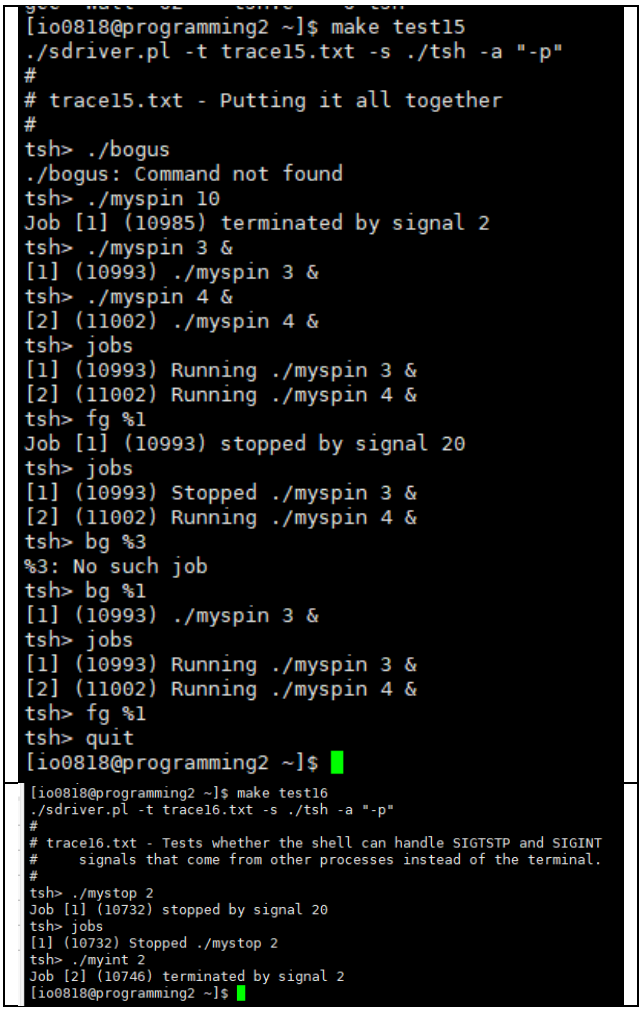
```
[io0818@programming2 ~]$ make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (21638) ./myspin 4 &
tsh> fg %1
Job [1] (21638) stopped by signal 20
tsh> jobs
[1] (21638) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

```
[io0818@programming2 ~]$ 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] (21941) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  966 pts/63   Ss+    0:00 /usr/bin/bash --init-file /home/std/jueunk/.vscode-server/bin/74b1f979648cc44d385a2286793c226e611f59e7/out/vs/workbench/contrib/terminal/browser/media/shellIntegration-bash.sh
 1245 pts/67   T      0:00 make test10
 1246 pts/67   T      0:00 /usr/bin/perl ./sdriver.pl -t trace10.txt -s ./tsh -a -p
 1252 pts/67   R     63:22 ./tsh -p
 1255 pts/67   Z      0:00 [myspin] <defunct>
 1918 pts/67   R     64:29 ./tsh -p
 1920 pts/67   T      0:00 ./myspin 4
 2075 pts/43   R      3:26 ./tsh -p
 2119 pts/43   R      3:10 ./tsh -p
 2363 pts/67   R     68:36 ./tsh -p
 2365 pts/67   T      0:00 ./myspin 4
 3311 pts/38   Ss+    0:00 /bin/bash --init-file /home/std/daehyeonchoi/.vscode-server/bin/6261075646f055b99068d3688932416f2346dd3b/out/vs/workbench/contrib/terminal/browser/media/shellIntegration-bash.sh
 3983 pts/42   Ss     0:00 -bash
 4055 pts/69   Ss+    0:00 -bash
 4299 pts/30   Ss+    0:00 -bash
 4438 pts/45   Ss     0:00 -bash
 4467 pts/50   Ss+    0:00 -bash
 4687 pts/29   Ss+    0:00 -bash
 5162 pts/51   Ss     0:00 -bash
 5234 tty1     Ss+    0:00 /sbin/agetty --noclear tty1 linux
```

```
[io0818@programming2 ~]$ 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] (23246) stopped by signal 20
tsh> jobs
[1] (23246) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
 1245 pts/67   T      0:00 make test10
 1246 pts/67   T      0:00 /usr/bin/perl ./sdriver.pl -t trace10.txt -s ./tsh -a -p
 1252 pts/67   R     63:40 ./tsh -p
 1255 pts/67   Z      0:00 [myspin] <defunct>
 1918 pts/67   R     64:48 ./tsh -p
 1920 pts/67   T      0:00 ./myspin 4
 2076 pts/43   R      3:46 ./tsh -p
 2119 pts/43   R      3:29 ./tsh -p
 2363 pts/67   R     68:55 ./tsh -p
 2365 pts/67   T      0:00 ./myspin 4
 3311 pts/38   Ss+    0:00 /bin/bash --init-file /home/std/daehyeonchoi/.vscode-server/bin/6261075646f055b99068d3688932416f2346dd3b/out/vs/workbench/contrib/terminal/browser/media/shellIntegration-bash.sh
 3983 pts/42   Ss     0:00 -bash
 4055 pts/69   Ss+    0:00 -bash
 4299 pts/38   Ss+    0:00 -bash
 4438 pts/45   Ss+    0:00 -bash
 4467 pts/50   Ss+    0:00 -bash
 4687 pts/29   Ss+    0:00 -bash
 5162 pts/51   Ss     0:00 -bash
 5234 tty1     Ss+    0:00 /sbin/agetty --noclear tty1 linux
 5270 pts/15   Ss+    0:00 /usr/bin/bash --init-file /home/std/pjy0422/.vscode-server/bin/6261075646f055b99068d3688932416f2346dd3b/out/vs/workbench/contrib/terminal/browser/media/shellIntegration-bash.sh
```

```
[io0818@programming2 ~]$ 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] (24071) stopped by signal 20
tsh> jobs
[1] (24071) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
 1245 pts/67   T      0:00 make test10
 1246 pts/67   T      0:00 /usr/bin/perl ./sdriver.pl -t trace10.txt -s ./tsh -a -p
 1252 pts/67   R     63:51 ./tsh -p
 1255 pts/67   Z      0:00 [myspin] <defunct>
 1918 pts/67   R     64:58 ./tsh -p
 1920 pts/67   T      0:00 ./myspin 4
 2076 pts/43   R      3:57 ./tsh -p
 2119 pts/43   R      3:40 ./tsh -p
 2363 pts/67   R     69:07 ./tsh -p
 2365 pts/67   T      0:00 ./myspin 4
```

```
[io0818@programming2 ~]$ 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] (11139) ./myspin 4 &
tsh> fg
fg command requires PID or %jobpid argument
tsh> bg
bg command requires PID or %jobpid 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
tsh> bg %2
%2: No such job
tsh> bg %1
%1: No such job
tsh> jobs
[io0818@programming2 ~]$
```



V.결론 및 고찰

- Shell의 원리와 사용자와 어떻게 interaction하는지 알 수 있었다.
- handler를 구현하며 signal에 따른 반응이 어떻게 조작되는지 이해할 수 있었다.
- foreground, background process들과, tsh의 basic function들의 원리를 이해할 수 있었다.