

Shelllab : Report

사범대학 국어교육과
2017-14342 도양훈

1. 구현 과정

1.1. 스켈레톤 코드 분석

1.1.1. Functions to implement

```
void eval(char *cmdline);  
int builtin_cmd(char **argv);  
void do_bgfg(char **argv);  
void waitfg(pid_t pid);  
  
void sigchld_handler(int sig);  
void sigtstp_handler(int sig);  
void sigint_handler(int sig);
```

구현해야 할 함수는 총 7개이다.

`eval`은 셸의 중추 역할을 하는 함수로서, input string에 따라 적절한 함수를 call한다.

`builtin_cmd`는 builtin 커맨드를 수행하는 함수인데, 모든 `eval`의 input에 대해 `builtin_cmd`를 수행하여 builtin이라면 그대로 수행, 아니라면 다시 `eval`로 return하는 형태로 구현할 수 있다.

`do_bgfg`는 `bg`와 `fg`라는 본 셸의 builtin 커맨드를 구현하는 함수이다. 각각 프로세스를 `bg` 혹은 `fg`로 옮겨 수행할 수 있도록 하는데, 각 상태의 특성을 고려하면 (`st > fg`), (`st > bg`), (`bg > fg`), (`bg > bg`)를 수행할 수 있다.

`waitfg`는 `fg`에서 수행중인 child 프로세스를 기다릴 수 있도록 하는 함수이다.

아래의 세 signal handler는 각각 `sigchld`, `sigint`, `sigtstp`를 handle한다. `sigint`와 `sigtstp`는 signal 전달을, `sigchld`는 실제 종료되거나 stop된 child에 대한 처리를 하는 방향으로 구현을 계획하였다.

1.1.2. Helper routines

```
/* Here are helper routines that we've provided for you */  
int parseline(const char *cmdline, char **argv);  
void sigquit_handler(int sig);  
  
void clearjob(struct job_t *job);  
void initjobs(struct job_t *jobs);  
int maxjid(struct job_t *jobs);  
int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline);  
int deletejob(struct job_t *jobs, pid_t pid);  
pid_t fgpid(struct job_t *jobs);  
struct job_t *getjobpid(struct job_t *jobs, pid_t pid);  
struct job_t *getjobjid(struct job_t *jobs, int jid);  
int pid2jid(pid_t pid);  
void listjobs(struct job_t *jobs);  
  
void usage(void);  
void unix_error(char *msg);  
void app_error(char *msg);  
typedef void handler_t(int);  
handler_t *Signal(int signum, handler_t *handler);
```

Implement 과정에서 사용할 수 있는 함수는 위와 같았다.

parseline은 shell 인풋 스트링을 공백을 기준으로 분할하여 argv라는 string 배열에 정리한다. sighquit_handler는 프로세스를 종료시키는 시그널핸들러였으나 quit을 구현한 이후로는 적극적으로 사용하지 않았다.

addjob과 deletejob은 각각 프로세스가 실행되거나 종료될때, 이를 jobs라는 job의 배열에서 삭제한다.

fgpid는 현재 fg에서 수행되고 있는 프로세스의 pid를 반환한다. 내부적으로는 모든 jobs의 원소를 방문하면서 작동하는 것을 확인할 수 있었다.

getjobpid와 getjobjid는 각각 pid 혹은 jid를 인수로 하여 해당 job struct를 반환한다. job struct에 대해서는 1.1.4에서 설명하도록 하겠다.

pid2jid는 pid를 jid로 변환한다. jid는 현재 셸에서 수행되고 있는 프로세스의 번호로, 1씩 증가하는 순서를 가진다. pid는 시스템 종속적인 프로세스 id이다.

listjobs는 현재 백그라운드에서 수행 중인 jobs를 모두 출력한다. 구현 초기에 fg와 bg를 어떻게 나누어 출력할 수 있을지 걱정했다. 하지만 jobs 명령을 통해 listjobs를 수행할 때에는 fg에 jobs 명령이 수행되고 있으며, 이는 builtin command이기 때문에 jobs 배열에 입력되지 않으므로 해당 문제는 자동으로 해결됨을 확인할 수 있었다.

unix_error와 app_error는 각각 unix 스타일의, 혹은 프로그램 정의 스타일의 에러 처리 함수이다. 아래에서 구현한 system call check 과정에서 사용하였다. 다만, unix_error와 app_error 중 어떤 함수를 호출해야 하는지 고민이 되었다. man pages에서 검색한 결과 sigemptyset은 정의된 에러코드가 없는 반면, 나머지 함수는 에러코드가 구현되어 있었기 때문에 sigemptyset은 app_error로, 나머지 함수는 unix_error로 구현하였다.

signal은 signal을 다루기 위해 구현되어있었으며, 직접 활용할 일은 없었다.

1.1.3. Safe system call

```
/*
 * check the return value of every system call
 * sigemptyset, sigaddset, sigprocmask, fork, setpgid, kill
 */
int ch_sigemptyset(sigset_t *set); // should return 0
int ch_sigaddset(sigset_t *set, int signum); // should return 0
int ch_sigprocmask(int how, const sigset_t *set, sigset_t *oldset); // should return 0
pid_t ch_fork(void); // should return 0 or positive PID, should not return -1
int ch_setpgid(pid_t pid, pid_t pgid); // should return 0
int ch_kill(pid_t pid, int sig); // should return 0
```

코드 전반에 걸쳐 위의 여섯 개의 함수와 waitpid, sleep 총 8개의 system calls를 사용하였다. 이 함수들은 return 값을 통해 자신이 제대로 수행되었는지 알린다. 따라서 함수를 실행 후 return 값을 확인해 추가 오류를 방지할 수 있도록 하였다.

다만 sleep의 경우, signal이 발생했을 때 0이 아닌 값을 return하는데, 프로그램 특성 상 시그널의 발생이 자연스러운 현상인 경우가 많기 때문에 모든 return 값이 허용된다.

대표적으로 ch_sigprocmask는 다음과 같이 구현하였다.

```
int ch_sigprocmask(int how, const sigset_t *set, sigset_t *oldset)
{
    if(sigprocmask(how, set, oldset) != 0)
    {
        unix_error("ERROR from sigprocmask");
    }
    return 0;
}
```

앞서 설명한 바와 같이 함수 자체에 구현된 에러코드가 있는 경우 unix_error로, 그렇지 않은 경우 app_error로 처리하도록 하였으며, 코멘트로 에러 발생 장소를 알리도록 구현하였다. man pages에 검색해 정상 동작 후 0을 return하는 경우(대부분)에는 0을 return할 때, 0이나 양수를 return하는 경우에는 음수를 return할 때 에러 처리가 수행될 수 있도록 하였다. 본 코드에서 대부분의 system call은 return void의 형태로 사용하고 있었지만, 추가 구현 등을

고려해 본체 함수의 `return`을 그대로 `return`하도록 하였다. 그 예로, `fork`는 다음과 같이 구현하여 `return` 값을 보존할 수 있도록 하였다.

```
pid_t ch_fork(void)
{
    pid_t pid;
    if((pid = fork()) < 0)
    {
        unix_error("ERROR from fork");
    }
    return pid;
}
```

나머지의 경우에도 위의 두 함수와 유사하기 때문에 코드는 생략하겠다. 다만, 앞서 언급한 7개(sleep 제외)의 `system calls` 중 `waitpid`의 경우, `waitpid`를 사용하는 `sigchld handler`에서 `waitpid`의 모든 `return` 값에 대한 구현이 필요하기 때문에 추가 함수로 구현하지 않았다.

1.1.4. 그 밖의 주의점

```
struct job_t {
    /* The job struct */
    pid_t pid;          /* job PID */
    int jid;            /* job ID [1, 2, ...] */
    int state;          /* UNDEF, BG, FG, or ST */
    char cmdline[MAXLINE]; /* command line */
};
```

`struct job_t`는 프로세스의 `pid`, 셸 내부에서 통용되는 `jid`, `undef`, `bg`, `fg`, `st(stop)`을 정의하는 `state`, 그리고 출력을 위한 `cmdline`을 자산으로 가진다.

```
#define UNDEF 0 /* undefined */
#define FG 1    /* running in foreground */
#define BG 2    /* running in background */
#define ST 3    /* stopped */
```

`undef`, `bg`, `fg`, `st`는 `#define`으로 정의되어 `int`처럼 활용할 수 있다.

1.2. 구현

1.2.1. void eval(char *cmdline)

```
/*
 * eval - Evaluate the command line that the user has just typed in
 *
 * If the user has requested a built-in command (quit, jobs, bg or fg)
 * then execute it immediately. Otherwise, fork a child process and
 * run the job in the context of the child. If the job is running in
 * the foreground, wait for it to terminate and then return. Note:
 * each child process must have a unique process group ID so that our
 * background children don't receive SIGINT (SIGTSTP) from the kernel
 * when we type ctrl-c (ctrl-z) at the keyboard.
 */
void eval(char *cmdline)
{
    char *argv[MAXARGS];
    char buf[MAXLINE];
    sigset_t m_sigchld;
    pid_t pid;
    int bg;
```

```

strcpy(buf, cmdline);
bg = parseline(buf, argv);
/* handle no input */
if(argv[0] == NULL)
{
    return;
}
if(!builtin_cmd(argv))
{
    /* mask parent from SIGCHLD, SIGINT, SIGTSTP during fork */
    ch_sigemptyset(&m_sigchld);
    ch_sigaddset(&m_sigchld, SIGCHLD);
    ch_sigaddset(&m_sigchld, SIGINT);
    ch_sigaddset(&m_sigchld, SIGTSTP);
    ch_sigprocmask(SIG_BLOCK, &m_sigchld, NULL);
    /* fork */
    pid = ch_fork();
    if(pid == 0)
    {
        /* this is a child process */
        /* unmask child from SIGCHLD after fork */
        ch_sigprocmask(SIG_UNBLOCK, &m_sigchld, NULL);
        /*
         * get new process group ID
         * so that each process leads group of itself
         * first 0 : set group of this process
         * second 0 : this process leads group of itself
         */
        ch_setpgid(0, 0);
        /* load and run new program by execve */
        execve(argv[0], argv, environ);
        /*
         * legal input never reaches here
         * not using ch_execve since the shell must go on
         */
        printf("%s: Command not found\n", argv[0]);
        exit(0);
    }
    else
    {
        /* this is a parent process */
        if(bg)
        {
            /* add job */
            addjob(jobs, pid, BG, buf);
            /* unmask parent from SIGCHLD after fork */
            ch_sigprocmask(SIG_UNBLOCK, &m_sigchld, NULL);
            /* bg : print log message */
            printf("[%d] (%4d) %s", pid2jid(pid), (int) pid, buf);
        }
        else
        {
            /* add job */
            addjob(jobs, pid, FG, buf);
            /* unmask parent from SIGCHLD after fork */
            ch_sigprocmask(SIG_UNBLOCK, &m_sigchld, NULL);
            /* fg : wait for child */
            waitfg(pid);
        }
    }
}
}

```

```

    return;
}

```

`eval`은 커맨드라인을 읽어서 적절한 동작을 수행하는 역할을 한다. 우선 `builtin_cmd`로 커맨드를 보내서 `builtin` 커맨드인지 확인한 뒤 그렇지 않은 경우 새로운 프로세스를 포크 후 `execve`하는 것이 주 기능이다. `buf` 변수를 사용해서 커맨드라인 `input`을 함수 내부 변수로 변환하여 사용하도록 하였고, 개행(엔터)만 입력되는 경우 해당 페이지에 대한 모든 처리가 끝난 것으로 간주한 뒤 `return`하도록 하였다.

`builtin`이 아닌 커맨드를 수행할 때에는 `sigemptyset`, `sigaddset`, `sigprocmask`를 활용하여 `sigint`, `sigtstp`, `sigchld` 세 `signal`을 `block`한 뒤 `child`를 `fork`하는 과정을 거친다. 이때 중요한 것은 이후 발생한 `child` 프로세스와 기존의 `parent` 프로세스에서 모두 `sigprocmask unblock`을 해줘야한다는 것이다. 또한 `child` 프로세스의 경우, 새로운 프로세스 그룹을 형성할 수 있도록 `setpgid(0,0)`을 호출한다. 모든 `system call`은 `ch_`를 이용한 `return value checker` 함수를 통해 호출하여 안정성을 확보하였다.

1.2.2. `int builtin_cmd(char **argv)`

```

/*
 * builtin_cmd - If the user has typed a built-in command then execute
 *               it immediately.
 */
int builtin_cmd(char **argv)
{
    char *name = argv[0];
    /*
     * return value - quit : 3, jobs : 4, bg : 2, fg : 1
     *               not a builtin command : 0
     */
    if(!strcmp(name, "quit"))
    {
        exit(0); /* not using SIGQUIT since it prints extra comments */
        return 3;
    }
    if(!strcmp(name, "jobs"))
    {
        listjobs(jobs);
        return 4;
    }
    if(!strcmp(name, "bg"))
    {
        do_bgfg(argv);
        return 2; /* return same int with BG */
    }
    if(!strcmp(name, "fg"))
    {
        do_bgfg(argv);
        return 1; /* return same int with FG */
    }
    return 0; /* not a builtin command */
}

```

`builtin_cmd`는 `builtin` 커맨드를 수행하는 함수이다. 다만, `builtin` 커맨드가 입력된 경우에만 호출하는 것이 아닌, 모든 커맨드에 대하여 호출한 뒤 `builtin`이 아닌 경우 `return`하도록 구현하였다. 각 `builtin` 커맨드는 3, 4, 2, 1을 `return`하는데, 이는 디버깅 용으로 실제 `implementation`에서 사용되지는 않는다. 다만 `builtin` 커맨드가 아니기 때문에 0을 `return`하는 경우에는 `eval`에서 적절한 동작을 수행하도록 한다.

`quit`의 경우 `sigquit`을 사용하고 싶었지만, `exit`를 이용하여 정상적으로 종료할 수 있는 상황에는 정상 종료를 하도록 하였다.

1.2.3. void do_bgfg(char **argv)

```
/*
 * do_bgfg - Execute the builtin bg and fg commands
 */
void do_bgfg(char **argv)
{
    struct job_t *job;

    /* handle input errors */
    if(argv[1] == NULL) /* no argument */
    {
        printf("%s command requires PID or %%jobid argument\n", argv[0]);
        return;
    }
    if(argv[1][0] == '%') /* jid is given */
    {
        if((job = getjobjid(jobs, atoi(&argv[1][1]))) == NULL) /* no job with
given jid */
        {
            printf("%s: No such job\n", argv[1]);
            return;
        }
    }
    else if(isdigit(argv[1][0])) /* pid is given */
    {
        if((job = getjobpid(jobs, atoi(&argv[1][0]))) == NULL) /* no job with
given pid */
        {
            printf("(%s): No such process\n", argv[1]);
            return;
        }
    }
    else /* argument is in wrong format */
    {
        printf("%s: argument must be a PID or %%jobid\n", argv[0]);
        return;
    }
    if(!strcmp(argv[0], "bg"))
    {
        /* continue in BG : ST -> BG */
        job -> state = BG;
        ch_kill(job -> pid, SIGCONT);
        printf("[%d] (%4d) %s", job -> jid, (int) job -> pid, job -> cmdline);
    }
    else
    {
        /* continue in FG : ST -> FG / BG -> FG */
        job -> state = FG;
        ch_kill(-(job -> pid), SIGCONT);
        waitfg(job -> pid);
    }
    return;
}
```

do_bgfg 역시 구현이 까다로운 함수 중 하나였다. 코드의 대부분은 input exception을 handle하는 기능을 한다. element의 첫 글자가 %이거나 element가 숫자(isdigit)이면 우선 valid한 input이라고 볼 수 있지만, 그것이 실제 존재하지 않는 프로세스의 pid 혹은 jid인 경우 어쩔 수 없이 return하는 수밖에 없다. 이를 통해 실제 존재하는 pid 혹은 jid를 입력받은

경우만 추린 후, **bg**와 **fg**로 나누어 각각 **eval**에서 **bg**와 **fg**를 구현하는 코드와 유사하게 구현하였다.

1.2.4. void waitfg(pid_t pid)

```
/*
 * waitfg - Block until process pid is no longer the foreground process
 */
void waitfg(pid_t pid)
{
    while(1) /* loop with sleep */
    {
        if(pid != fgpid(jobs)) /* no job with given pid is in foreground */
        {
            return;
        }
        else
        {
            sleep(1);
        }
    }
    return;
}
```

sleep가 포함된 무한 루프를 수행하면서 **child** 프로세스의 종료 여부, 즉 **child** 프로세스의 **pid**가 **fgetpid**와 같은지 여부를 확인한다.

1.2.5. void sigchld_handler(int sig)

```
/*
 * sigchld_handler - The kernel sends a SIGCHLD to the shell whenever
 * a child job terminates (becomes a zombie), or stops because it
 * received a SIGSTOP or SIGTSTP signal. The handler reaps all
 * available zombie children, but doesn't wait for any other
 * currently running children to terminate.
 */
void sigchld_handler(int sig)
{
    pid_t pid;
    int status;
    /* loop until there is no more unhandled child left */
    while((pid = waitpid(-1, &status, WUNTRACED | WNOHANG)) > 0)
    {
        if(WIFSIGNALED(status)) /* unknown child terminated - delete job*/
        {
            printf("Job [%d] (%4d) terminated by signal %d\n", pid2jid(pid),
(int) pid, WTERMSIG(status));
            deletejob(jobs, pid);
        }
        else if(WIFSTOPPED(status)) /* unknown child stopped - should not delete
job but make state ST */
        {
            printf("Job [%d] (%4d) stopped by signal %d\n", pid2jid(pid), (int)
pid, WSTOPSIG(status));
            getjobpid(jobs, pid) -> state = ST;
        }
        else
        {
            deletejob(jobs, pid);
        }
    }
}
```

```

    }
}
return;
}

```

sigchld handler는 child 프로세스에 대하여 signal에 의한 종료나 stop, 정상 종료를 모두 확인한다. 특히 한 번의 sigchld handler 호출이 여러 child의 상태를 확인해야 하기 때문에 while loop을 통해 waitpid를 통해 확인되는 종료 혹은 stop된 child 프로세스가 단 하나도 남지 않을 때까지 이를 반복한다.

이후 sigchld handler는 종료를 경우 deletejob을 호출하고, stop의 경우 job의 state를 바꾼다. signal에 의해 종료되거나 stop된 경우 이를 출력한다. 따라서 이어 소개하는 sigtstp handler와 sigint handler는 로그를 출력하거나 deletejob, state 변환을 하지 않고 해당 시그널을 전달만 하도록 한다.

1.2.6. void sigtstp_handler(int sig) & void sigint_handler(int sig)

```

/*
 * sigint_handler - The kernel sends a SIGINT to the shell whenever the
 * user types ctrl-c at the keyboard. Catch it and send it along
 * to the foreground job.
 */
void sigint_handler(int sig)
{
    pid_t pid = fgpid(jobs);
    if(pid != 0)
    {
        /* this is a parent process */
        ch_kill(-pid, sig);
        /* deletion will be handled by SIGCHLD handler */
    }
    return;
}

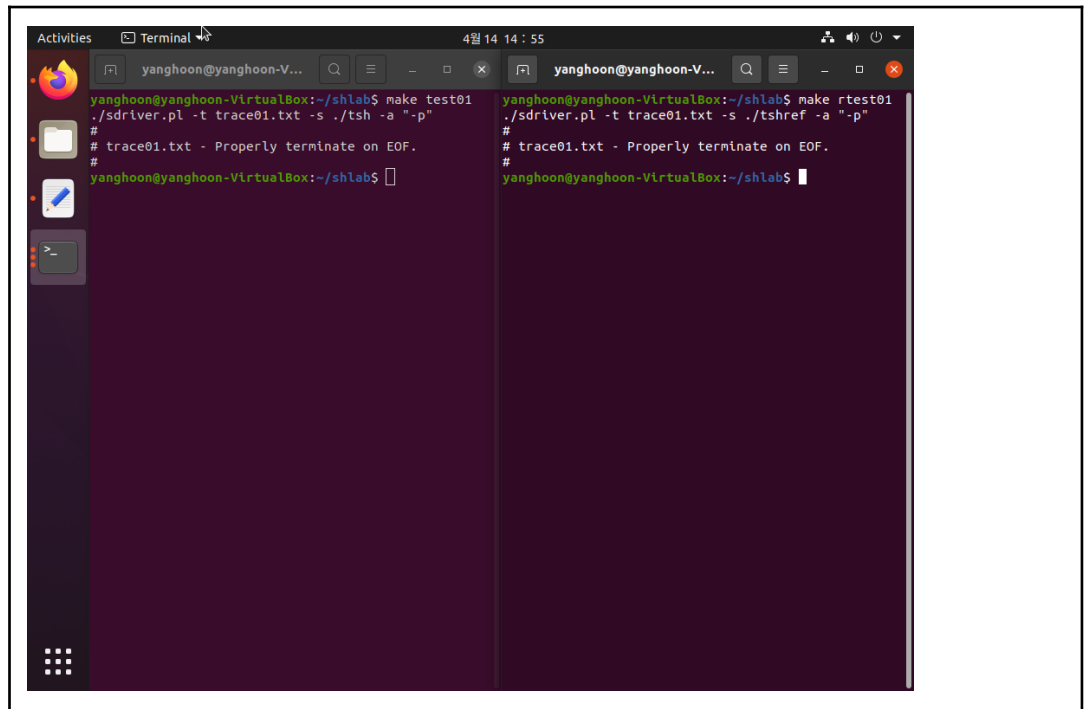
/*
 * sigtstp_handler - The kernel sends a SIGTSTP to the shell whenever
 * the user types ctrl-z at the keyboard. Catch it and suspend the
 * foreground job by sending it a SIGTSTP.
 */
void sigtstp_handler(int sig)
{
    pid_t pid = fgpid(jobs);
    if(pid != 0)
    {
        /* this is a parent process */
        ch_kill(-pid, sig);
        /* deletion will be handled by SIGCHLD handler */
    }
    return;
}

```

두 함수는 유사한 기능을 유사하게 구현하였다. 각각 자신이 handle하는 signal을 현재 fg의 프로세스 그룹으로 전달한다. 이를 통해 signal은 셸 전체에 전달된 것이지만 이를 셸 상의 프로세스로 옮기는 것이다.

1.3. tracefiles

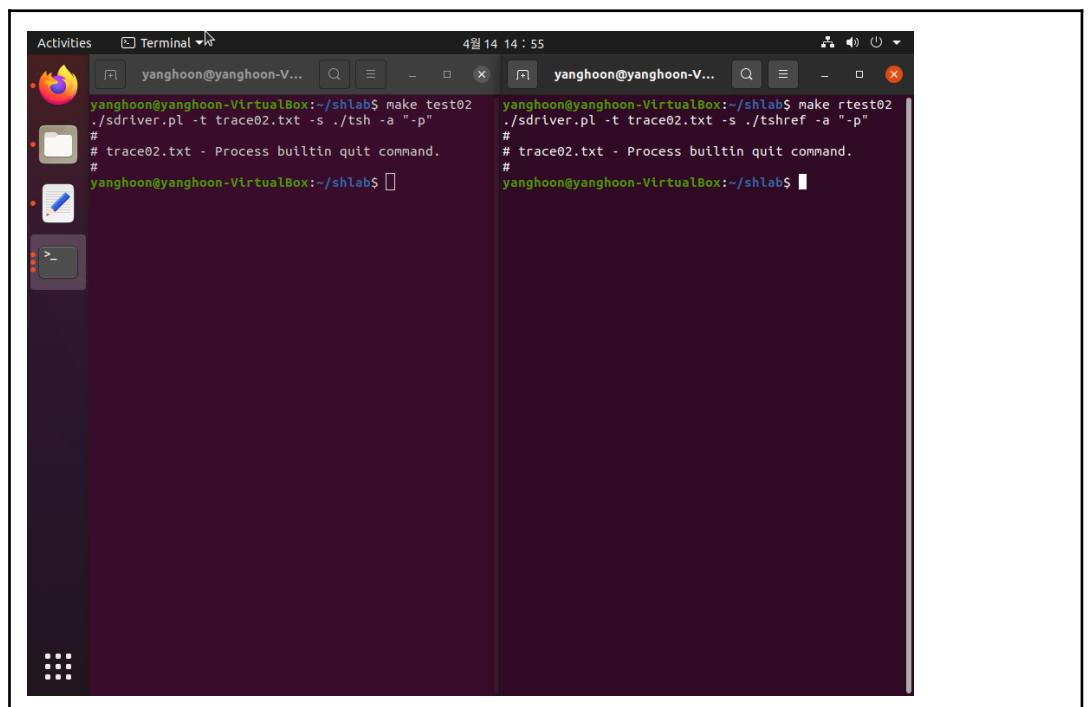
1.3.1. trace01



trace01은 EOF에 도달하였을 때 shell이 적절히 종료되는 것을 확인한다. 이는 main함수에 구현되어 있으므로 추가로 구현할 사항이 없다. 개행(엔터)만 입력될 때에도 프로그램이 적절히 작동하도록 다음을 구현하였다.

```
if(argv[0] == NULL)
{
    return;
}
```

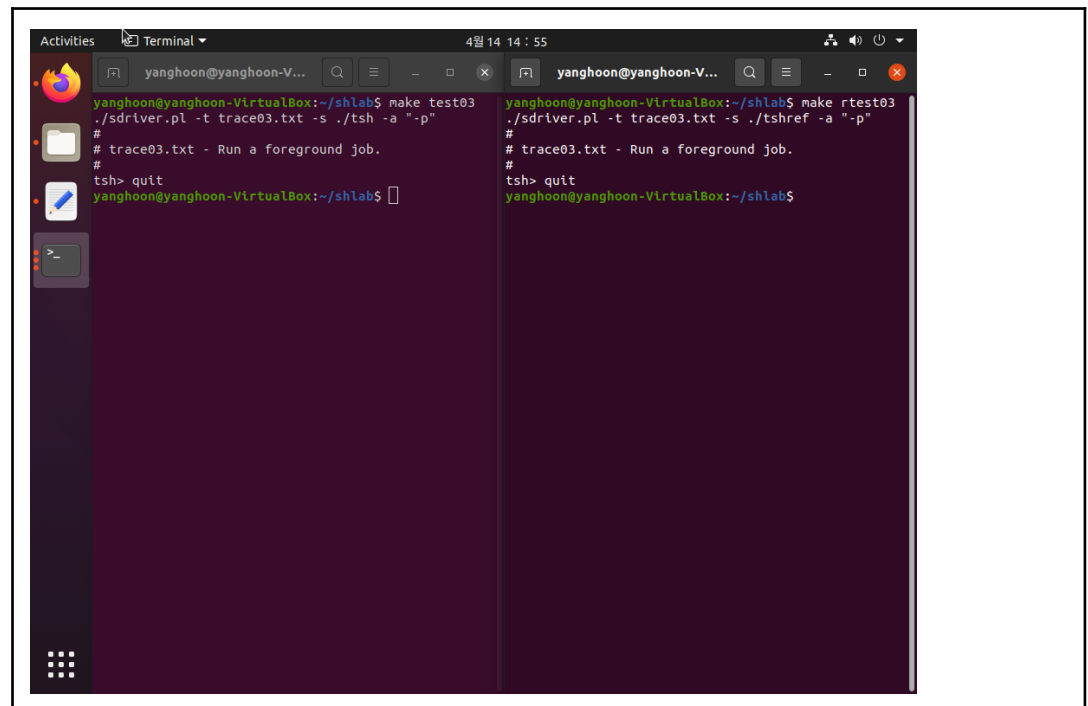
1.3.2. trace02



trace02는 builtin quit 커맨드 수행을 확인한다. 모든 입력은 builtin_cmd 함수로 인풋되고, 해당 함수에서 builtin 여부를 판단하도록 구현하였다. quit의 경우, 처음에는 sigquit을 사용하려고 했으나 helper routines에서 sigquit이 호출시 시그널과 관련된 코멘트를 출력하도록 정의되었기 때문에 해당 함수의 의도와 맞지 않다고 판단하였다. 따라서 quit 수행을 위해 exit(0)을 이용하였다. 아래의 return 구문은 정상적인 동작 하에서는 도달되지 않는다.

```
if(!strcmp(name, "quit"))
{
    exit(0); /* not using SIGQUIT since it prints extra comments */
    return 3;
}
```

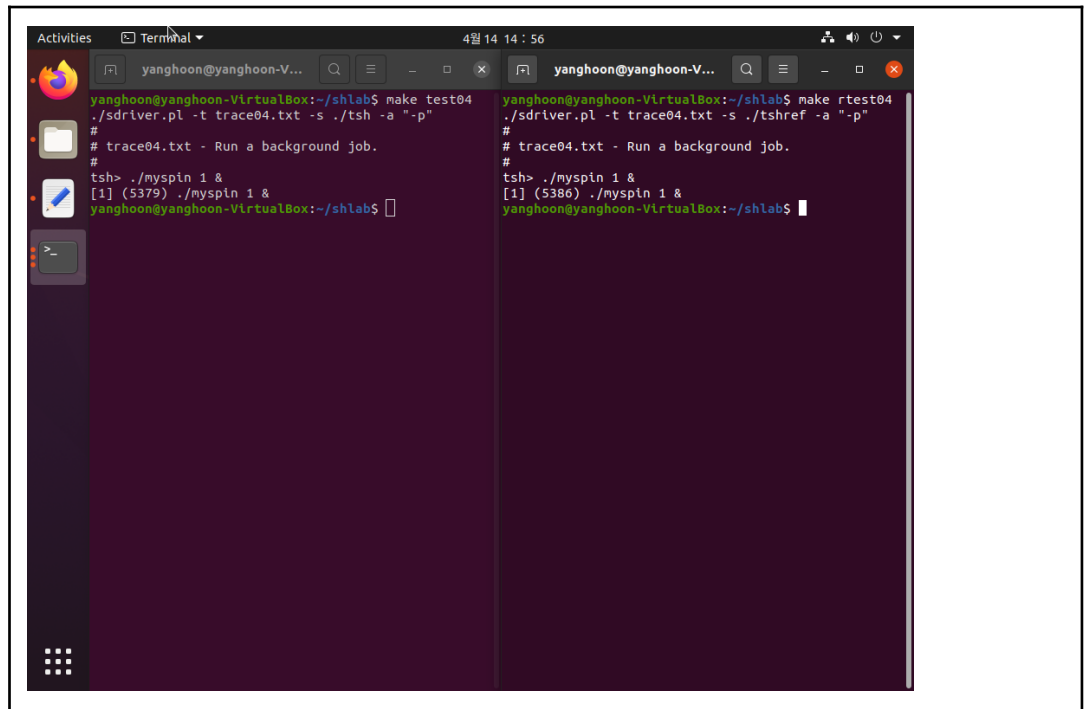
1.3.3. trace03



trace03은 /bin/echo라는 foreground job이 적절히 수행되는지 확인한다. &가 뒤에 붙으면 background로 인식하도록 parseline이 이미 구현되어 있다. 따라서 1.3.3의 코드에서 구현한 바와 같이 이 값을 반환받아 bg와 fg 여부를 확인할 수 있다. fg 옵션으로 프로세스가 수행되는 경우, parent 프로세스는 waitfg 함수를 통해 기다리며 종료를 확인한다.

```
else
{
    /* add job */
    addjob(jobs, pid, FG, buf);
    /* unmask parent from SIGCHLD after fork */
    ch_sigprocmask(SIG_UNBLOCK, &m_sigchld, NULL);
    /* fg : wait for child */
    waitfg(pid);
}
```

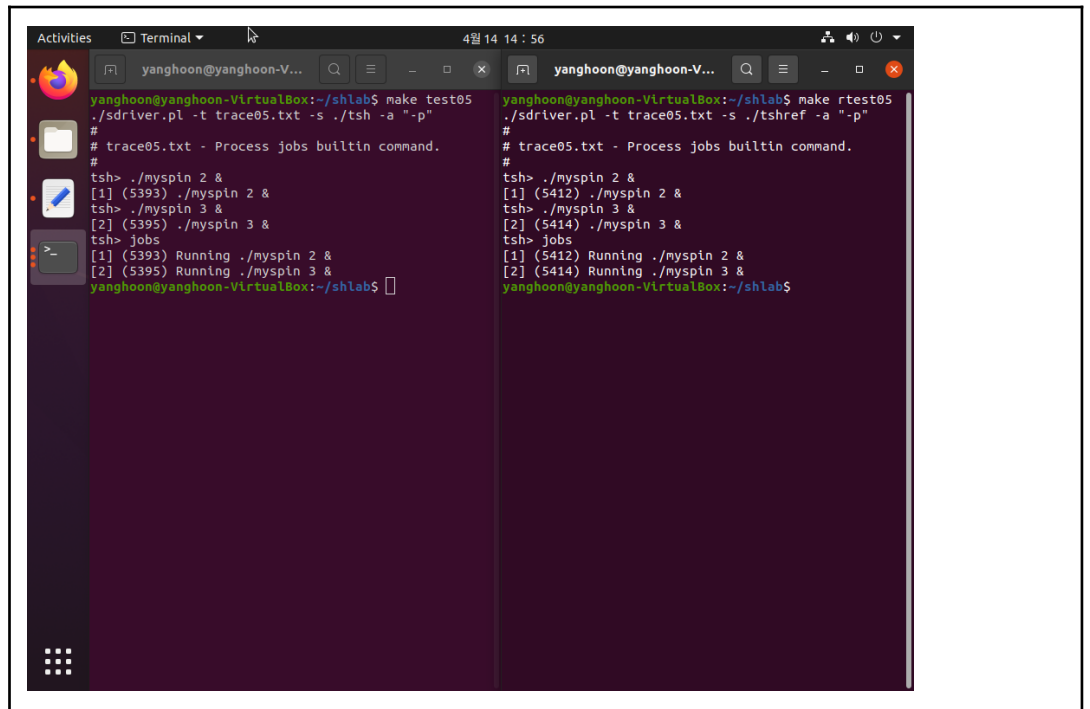
1.3.4. trace04



trace04는 trace03과 비슷하지만 background job에 대해 확인한다. bg인 경우, 정보를 출력한다는 특이사항이 있다.

```
if (bg)
{
    /* add job */
    addjob(jobs, pid, BG, buf);
    /* unmask parent from SIGCHLD after fork */
    ch_sigprocmask(SIG_UNBLOCK, &m_sigchld, NULL);
    /* bg : print log message */
    printf("[%d] (%4d) %s", pid2jid(pid), (int) pid, buf);
}
```

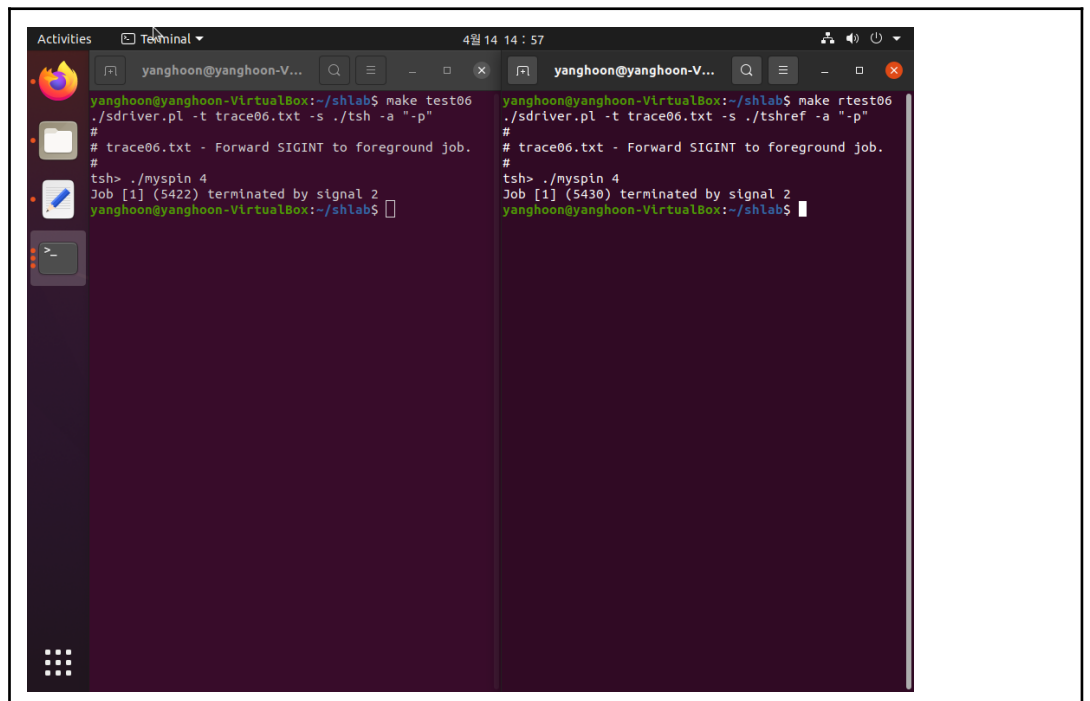
1.3.5. trace05



trace05는 builtin jobs 커맨드 수행을 확인한다. 현재 bg에서 수행중인 프로세스를 출력하는 것으로, 처음에는 이때까지의 모든 job이 출력되는 등 오류를 겪었지만, deletejob 등을 적절히 구현하고 나니 listjobs 함수를 통해 쉽게 구현할 수 있었다.

```
if(!strcmp(name, "jobs"))
{
    listjobs(jobs);
    return 4;
}
```

1.3.6. trace06



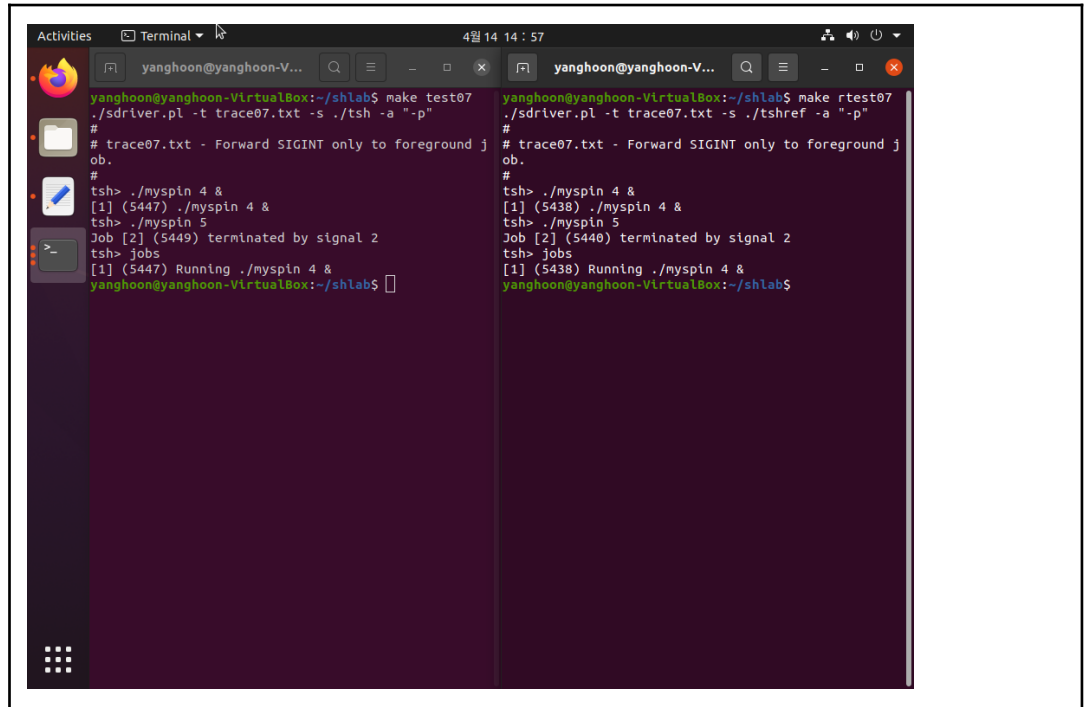
trace06은 fg에 대한 sigint의 동작을 확인한다. signal 2를 출력하며 프로세스가 terminate되면 성공이다. 본 코드에서는 sigint handler를 통해 fg 프로세스 그룹에 sigint가 전달되도록 구현하였지만, deletejob 등은 따로 구현하지 않았다. 대신 sigchld를 통해 parent 프로세스에서 일괄적으로 처리되도록 하였다. pid 대신 -pid를 사용하여 모든 프로세스 그룹에 signal이 전달되도록 하였다. 다만 이때 모든 프로세스가 하나의 그룹에 속하는 것을 방지하기 위해 child fork시 새로운 프로세스 그룹을 형성하도록 하는 구현을 추가하였다.

```
/*
 * sigint_handler - The kernel sends a SIGINT to the shell whenever the
 *                 user types ctrl-c at the keyboard.  Catch it and send it along
 *                 to the foreground job.
 */
void sigint_handler(int sig)
{
    pid_t pid = fgpid(jobs);
    if(pid != 0)
    {
        /* this is a parent process */
        ch_kill(-pid, sig);
        /* deletion will be handled by SIGCHLD handler */
    }
    return;
}
```

```
if(pid == 0)
{
    (생략)

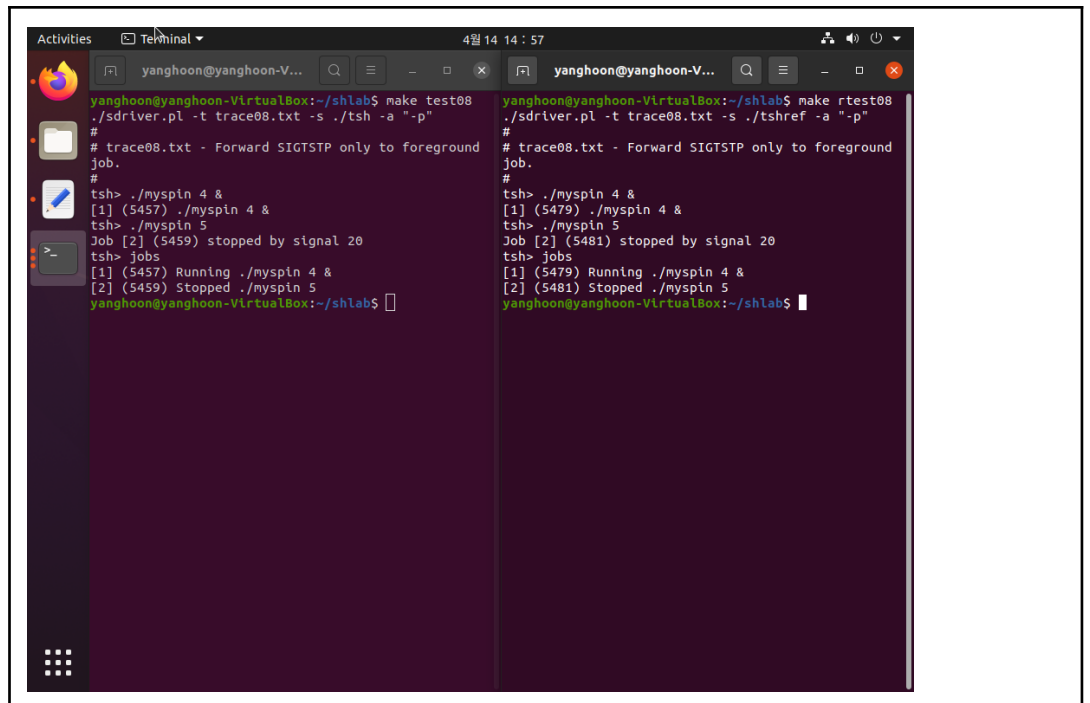
    /*
     * get new process group ID
     * so that each process leads group of itself
     * first 0 : set group of this process
     * second 0 : this process leads group of itself
     */
    ch_setpgid(0, 0);
    (생략)
}
```

1.3.7. trace07



trace07은 sigint 이후 bg 프로세스가 함께 종료되지는 않았는지 확인한다. 구현대로라면, sigint handler는 fgetpid를 통해 fg에서 수행 중인 프로세스의 pid만 알고 있으므로 bg 프로세스에는 시그널이 전달되지 않을 것이며, 앞서 언급한 바와 같이 fork 시 프로세스 그룹을 모두 분리해줌으로써 -pid 인수를 통해 bg 프로세스로 signal이 전달될 가능성을 차단하였으므로 bg 프로세스가 종료되지 않음을 보장할 수 있다.

1.3.8. trace08



trace08은 sigtstp의 동작을 확인한다. 앞의 sigint와 유사하게 stop을 직접 구현하기보다는 signal만 전달한 뒤 sigchld가 일괄적으로 이를 처리하도록 구현하였다.

```
/*
 * sigtstp_handler - The kernel sends a SIGTSTP to the shell whenever
 *                   the user types ctrl-z at the keyboard. Catch it and suspend the
```

```

*      foreground job by sending it a SIGTSTP.
*/
void sigtstp_handler(int sig)
{
    pid_t pid = fgpid(jobs);
    if(pid != 0)
    {
        /* this is a parent process */
        ch_kill(-pid, sig);
        /* stop will be handled by SIGCHLD handler */
    }
    return;
}

```

1.3.9. trace09

```

yanghoon@yanghoon-VirtualBox:~/shlab$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (5490) ./myspin 4 &
tsh> ./myspin 5
Job [2] (5492) stopped by signal 20
tsh> jobs
[1] (5490) Running ./myspin 4 &
[2] (5492) Stopped ./myspin 5
tsh> bg %2
[2] (5492) ./myspin 5
tsh> jobs
[1] (5490) Running ./myspin 4 &
[2] (5492) Running ./myspin 5
yanghoon@yanghoon-VirtualBox:~/shlab$

yanghoon@yanghoon-VirtualBox:~/shlab$ make rtest09
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (5513) ./myspin 4 &
tsh> ./myspin 5
Job [2] (5515) stopped by signal 20
tsh> jobs
[1] (5513) Running ./myspin 4 &
[2] (5515) Stopped ./myspin 5
tsh> bg %2
[2] (5515) ./myspin 5
tsh> jobs
[1] (5513) Running ./myspin 4 &
[2] (5515) Running ./myspin 5
yanghoon@yanghoon-VirtualBox:~/shlab$

```

trace09는 builtin bg 커맨드 수행을 확인한다. bg 이후 pid나 jid를 입력하면 해당 job이 bg에서 이어 수행된다. 단, fg 상태의 프로세스의 경우, 해당 프로세스가 종료되거나 stop되기 전에는 bg 커맨드를 입력할 수 없기 때문에 실제 작동하는 것은 st > bg의 경우밖에 없다.

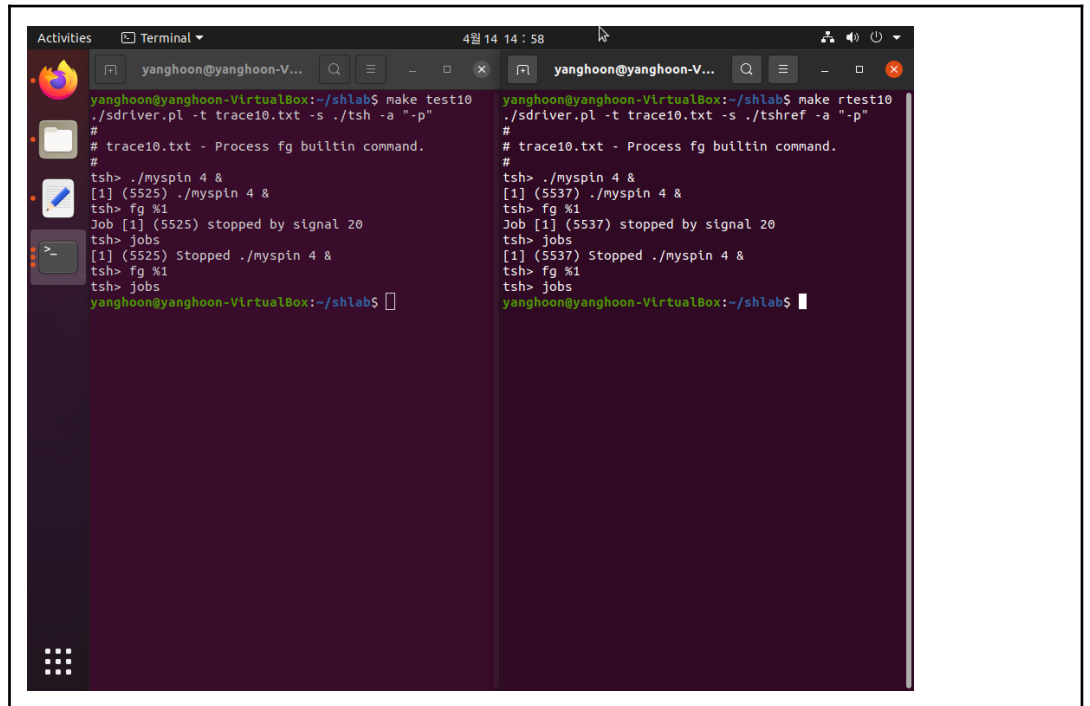
```

/*
* do_bgfg - Execute the builtin bg and fg commands
*/
void do_bgfg(char **argv)
{
    (생략)

    if(!strcmp(argv[0], "bg"))
    {
        /* continue in BG : ST -> BG */
        job -> state = BG;
        ch_kill(job -> pid, SIGCONT);
        printf("[%d] (%4d) %s", job -> jid, (int) job -> pid, job -> cmdline);
    }
    (생략)
}

```

1.3.10. trace10



trace09는 builtin fg 커맨드 수행을 확인한다. fg 이후 pid나 jid를 입력하면 해당 job이 fg에서 이어 수행된다. fg와 다르게, bg 상태의 프로세스의 경우, 해당 프로세스가 종료되거나 stop되기 전에도 bg 커맨드를 입력할 수 있기 때문에 st > fg 혹은 bg > fg의 작동을 모두 확인할 수 있다.

```
/*
 * do_bgfg - Execute the builtin bg and fg commands
 */
void do_bgfg(char **argv)
{
    (생략)

    else
    {
        /* continue in FG : ST -> FG / BG -> FG */
        job -> state = FG;
        ch_kill(-(job -> pid), SIGCONT);
        waitfg(job -> pid);
    }

    (생략)
}
```

1.3.11. trace11


```

yanghoon@yanghoon-VirtualBox:~/shlab$ 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] (5548) terminated by signal 2
tsh> /bin/ps a
PID TTY STAT TIME COMMAND
826 tty1 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
828 tty1 Sl+ 0:04 /usr/lib/xorg/Xorg vt1 -displayfd 3 -auth /run/user/125/gdm/Xauthority -background none -noreset -keeptty -verbose 3
834 tty1 S+ 0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
835 tty1 S+ 0:00 dbus-daemon --nofork --print-address 4 --session
836 tty1 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --autostart /usr/share/gdm/greeter/autostart
844 tty1 Sl+ 0:00 /usr/libexec/at-spi-bus-launcher
849 tty1 S+ 0:00 /usr/bin/dbus-daemon -config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
886 tty1 Sl+ 0:13 /usr/bin/gnome-shell
904 tty1 Sl 0:00 ibus-daemon --panel disable --xtn
907 tty1 Sl 0:00 /usr/libexec/ibus-memcached
910 tty1 Sl 0:00 /usr/libexec/ibus-x11 --kill-daemon
915 tty1 Sl+ 0:00 /usr/libexec/ibus-portals
yanghoon@yanghoon-VirtualBox:~/shlab$

yanghoon@yanghoon-VirtualBox:~/shlab$ 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] (5567) terminated by signal 2
tsh> /bin/ps a
PID TTY STAT TIME COMMAND
826 tty1 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
828 tty1 Sl+ 0:04 /usr/lib/xorg/Xorg vt1 -displayfd 3 -auth /run/user/125/gdm/Xauthority -background none -noreset -keeptty -verbose 3
834 tty1 S+ 0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
835 tty1 S+ 0:00 dbus-daemon --nofork --print-address 4 --session
836 tty1 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --autostart /usr/share/gdm/greeter/autostart
844 tty1 Sl+ 0:00 /usr/libexec/at-spi-bus-launcher
849 tty1 S+ 0:00 /usr/bin/dbus-daemon -config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
886 tty1 Sl+ 0:13 /usr/bin/gnome-shell
904 tty1 Sl 0:00 ibus-daemon --panel disable --xtn
907 tty1 Sl 0:00 /usr/libexec/ibus-memcached
910 tty1 Sl 0:00 /usr/libexec/ibus-x11 --kill-daemon
915 tty1 Sl+ 0:00 /usr/libexec/ibus-portals
yanghoon@yanghoon-VirtualBox:~/shlab$

```

trace11은 sigint를 통해 mysplit이 적절히 종료되는지 확인한다. sigint 후에 실행시킨 /bin/ps a만 남도록 해야 한다. 앞서 sigint가 pid가 아닌 -pid에 전달되도록 구현하였기 때문에 해당 프로세스 group에 모두 전달됨을 보장할 수 있다.

1.3.12. trace12

```

yanghoon@yanghoon-VirtualBox:~/shlab$ 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] (5581) stopped by signal 20
tsh> jobs
[1] (5581) Stopped ./mysplit 4
tsh> /bin/ps a
PID TTY STAT TIME COMMAND
826 tty1 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
828 tty1 Sl+ 0:04 /usr/lib/xorg/Xorg vt1 -displayfd 3 -auth /run/user/125/gdm/Xauthority -background none -noreset -keeptty -verbose 3
834 tty1 S+ 0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
835 tty1 S+ 0:00 dbus-daemon --nofork --print-address 4 --session
836 tty1 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --autostart /usr/share/gdm/greeter/autostart
844 tty1 Sl+ 0:00 /usr/libexec/at-spi-bus-launcher
849 tty1 S+ 0:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
886 tty1 Sl+ 0:13 /usr/bin/gnome-shell
904 tty1 Sl 0:00 ibus-daemon --panel disable --xtn
907 tty1 Sl 0:00 /usr/libexec/ibus-memcached
910 tty1 Sl 0:00 /usr/libexec/ibus-x11
--kill-daemon
915 tty1 Sl 0:00 /usr/libexec/ibus-x11

yanghoon@yanghoon-VirtualBox:~/shlab$ 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] (5592) stopped by signal 20
tsh> jobs
[1] (5592) Stopped ./mysplit 4
tsh> /bin/ps a
PID TTY STAT TIME COMMAND
826 tty1 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
828 tty1 Sl+ 0:04 /usr/lib/xorg/Xorg vt1 -displayfd 3 -auth /run/user/125/gdm/Xauthority -background none -noreset -keeptty -verbose 3
834 tty1 S+ 0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
835 tty1 S+ 0:00 dbus-daemon --nofork --print-address 4 --session
836 tty1 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --autostart /usr/share/gdm/greeter/autostart
844 tty1 Sl+ 0:00 /usr/libexec/at-spi-bus-launcher
849 tty1 S+ 0:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
886 tty1 Sl+ 0:13 /usr/bin/gnome-shell
904 tty1 Sl 0:00 ibus-daemon --panel disable --xtn
907 tty1 Sl 0:00 /usr/libexec/ibus-memcached
910 tty1 Sl 0:00 /usr/libexec/ibus-x11
--kill-daemon
915 tty1 Sl 0:00 /usr/libexec/ibus-x11

```

```

yanghoon@yanghoon-VirtualBox:~/shlab$ 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] (5581) stopped by signal 20
tsh> jobs
[1] (5581) Stopped ./mysplit 4
tsh> /bin/ps a
PID TTY STAT TIME COMMAND
984 tty1 Sl+ 0:00 /usr/libexec/gsd-datetime
985 tty1 Sl+ 0:00 /usr/libexec/gsd-media-keys
986 tty1 Sl+ 0:00 /usr/libexec/gsd-screensaver-proxy
987 tty1 Sl 0:00 /usr/libexec/ibus-engine-simple
995 tty1 Sl+ 0:00 /usr/libexec/gsd-sound
1002 tty1 Sl+ 0:00 /usr/libexec/gsd-a11y-settings
1007 tty1 Sl+ 0:00 /usr/libexec/gsd-housekeeping
1016 tty1 Sl+ 0:00 /usr/libexec/gsd-power
1075 tty1 Sl+ 0:00 /usr/libexec/gsd-printer
1340 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
1342 tty2 Sl+ 29:25 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
1348 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
1876 pts/0 Ss+ 0:00 bash
3411 pts/1 Ss+ 0:00 bash
3798 pts/2 Ss+ 0:00 bash
5576 pts/1 S+ 0:00 make test12
5577 pts/1 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
5578 pts/1 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
5579 pts/1 S+ 0:00 ./tsh -p
5581 pts/1 T 0:00 ./mysplit 4
5582 pts/1 T 0:00 ./mysplit 4
5585 pts/1 R 0:00 /bin/ps a

yanghoon@yanghoon-VirtualBox:~/shlab$

yanghoon@yanghoon-VirtualBox:~/shlab$ 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] (5592) stopped by signal 20
tsh> jobs
[1] (5592) Stopped ./mysplit 4
tsh> /bin/ps a
PID TTY STAT TIME COMMAND
984 tty1 Sl+ 0:00 /usr/libexec/gsd-datetime
985 tty1 Sl+ 0:00 /usr/libexec/gsd-media-keys
986 tty1 Sl+ 0:00 /usr/libexec/gsd-screensaver-proxy
987 tty1 Sl 0:00 /usr/libexec/ibus-engine-simple
995 tty1 Sl+ 0:00 /usr/libexec/gsd-sound
1002 tty1 Sl+ 0:00 /usr/libexec/gsd-a11y-settings
1007 tty1 Sl+ 0:00 /usr/libexec/gsd-housekeeping
1016 tty1 Sl+ 0:00 /usr/libexec/gsd-power
1075 tty1 Sl+ 0:00 /usr/libexec/gsd-printer
1340 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
1342 tty2 Rl+ 29:26 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
1348 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
1876 pts/0 Ss 0:00 bash
3411 pts/1 Ss+ 0:00 bash
3798 pts/2 Ss+ 0:00 bash
5587 pts/0 S+ 0:00 make rtest12
5588 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
5589 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
5590 pts/0 S+ 0:00 ./tshref -p
5592 pts/0 T 0:00 ./mysplit 4
5593 pts/0 T 0:00 ./mysplit 4
5596 pts/0 R 0:00 /bin/ps a

yanghoon@yanghoon-VirtualBox:~/shlab$

```

trace12는 trace11과 유사하게 sigtstp이 모든 프로세스 group에 적절히 적용되는지 확인한다. 마찬가지로 pid가 아닌 -pid에 sigtstp이 전달되도록 구현하였으므로 이를 보장할 수 있다.

1.3.13. trace13

```
Activities Terminal 4월 14 15:02
yanghoon@yanghoon-V... yanghoon@yanghoon-V...

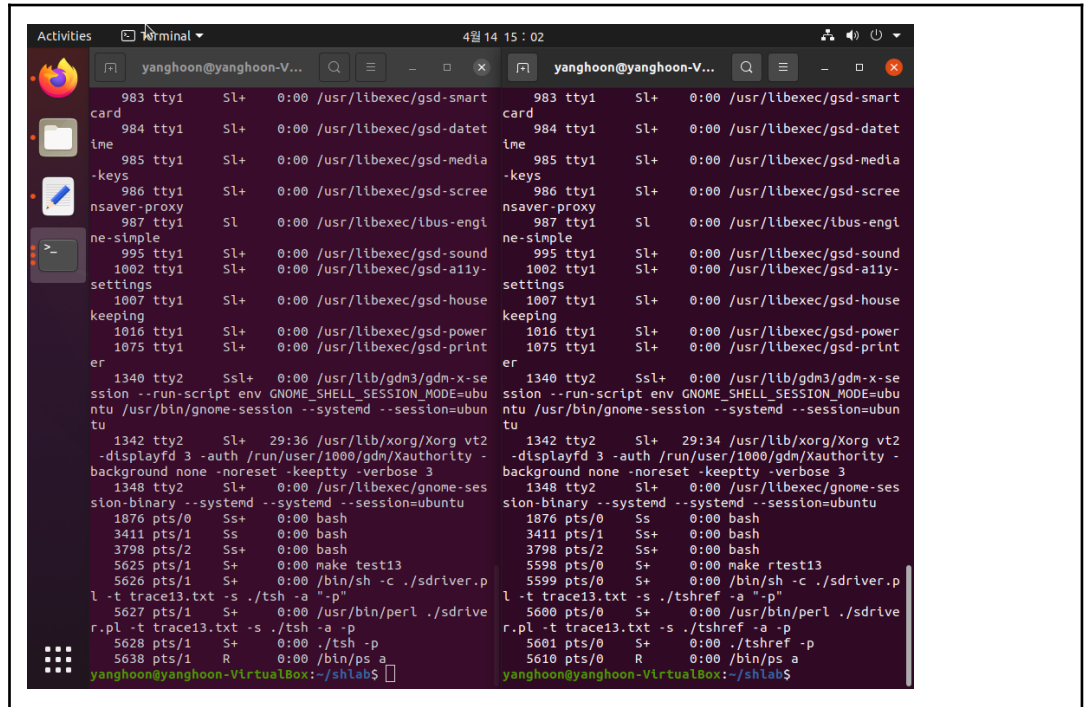
yanghoon@yanghoon-VirtualBox:~/shlab$ make test13
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (5630) stopped by signal 20
tsh> jobs
[1] (5630) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY          STAT TIME  COMMAND
  826 tty1    Ssl+   0:00 /usr/lib/gdm3/gdm-x-session dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
  828 tty1    Sl+    0:04 /usr/lib/xorg/Xorg vt1 -displayfd 3 -auth /run/user/125/gdm/Xauthority -background none -noreset -keeptty -verbose 3
  834 tty1    S+     0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
  835 tty1    S+     0:00 dbus-daemon --nofork --print-address 4 --session
  836 tty1    Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --autostart /usr/share/gdm/greeter/autostart
  844 tty1    Sl+    0:00 /usr/libexec/at-spi-bus-launcher
  849 tty1    S+     0:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
  886 tty1    Sl+    0:13 /usr/bin/gnome-shell
  904 tty1    Sl     0:00 ibus-daemon --panel disable --xln
  907 tty1    Sl     0:00 /usr/libexec/ibus-memconf
  910 tty1    Sl     0:00 /usr/libexec/ibus-x11 --kill-daemon
  915 tty1    Sl     0:00 /usr/libexec/ibus-x11 --kill-daemon

yanghoon@yanghoon-VirtualBox:~/shlab$ 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] (5603) stopped by signal 20
tsh> jobs
[1] (5603) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY          STAT TIME  COMMAND
  826 tty1    Ssl+   0:00 /usr/lib/gdm3/gdm-x-session dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
  828 tty1    Sl+    0:04 /usr/lib/xorg/Xorg vt1 -displayfd 3 -auth /run/user/125/gdm/Xauthority -background none -noreset -keeptty -verbose 3
  834 tty1    S+     0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
  835 tty1    S+     0:00 dbus-daemon --nofork --print-address 4 --session
  836 tty1    Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --autostart /usr/share/gdm/greeter/autostart
  844 tty1    Sl+    0:00 /usr/libexec/at-spi-bus-launcher
  849 tty1    S+     0:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
  886 tty1    Sl+    0:13 /usr/bin/gnome-shell
  904 tty1    Sl     0:00 ibus-daemon --panel disable --xln
  907 tty1    Sl     0:00 /usr/libexec/ibus-memconf
  910 tty1    Sl     0:00 /usr/libexec/ibus-x11 --kill-daemon
  915 tty1    Sl     0:00 /usr/libexec/ibus-x11 --kill-daemon
```

```
Activities Terminal 4월 14 15:02
yanghoon@yanghoon-V... yanghoon@yanghoon-V...

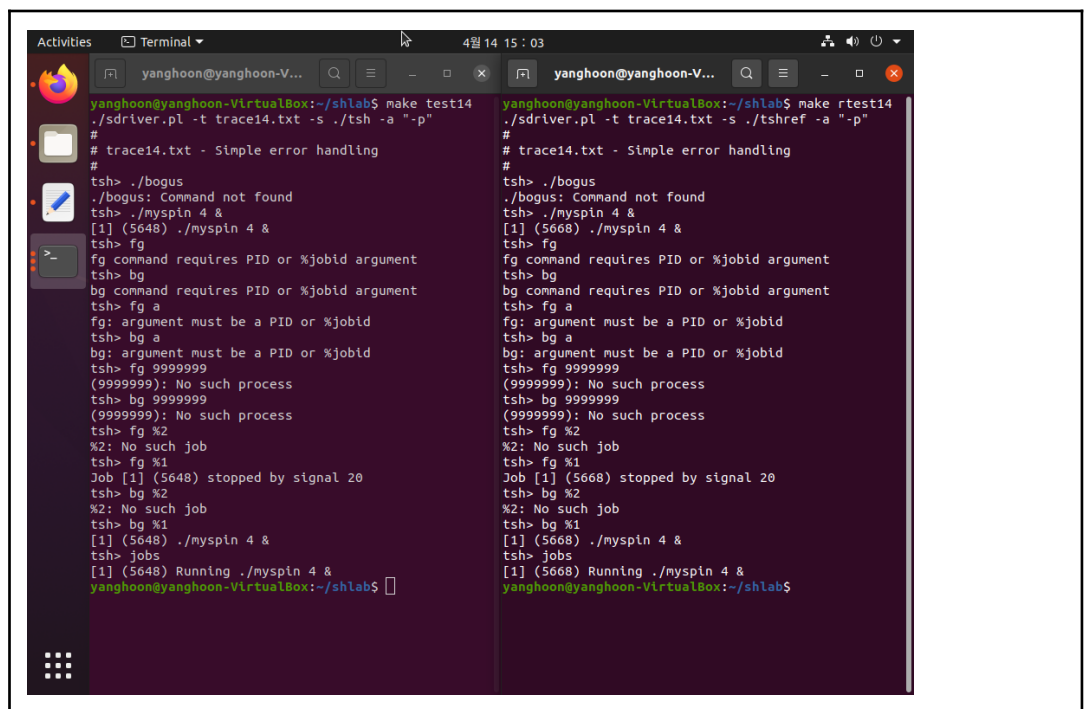
er 1075 tty1    Sl+    0:00 /usr/libexec/gsd-print
1340 tty2    Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
1342 tty2    Sl+    29:35 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
1348 tty2    Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
1876 pts/0    Ss+    0:00 bash
3411 pts/1    Ss+    0:00 bash
3798 pts/2    Ss+    0:00 bash
5625 pts/1    S+     0:00 make test13
5626 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
5627 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
5628 pts/1    S+     0:00 ./tsh -p
5630 pts/1    T      0:00 ./mysplit 4
5631 pts/1    T      0:00 ./mysplit 4
5635 pts/1    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY          STAT TIME  COMMAND
  826 tty1    Ssl+   0:00 /usr/lib/gdm3/gdm-x-session dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
  828 tty1    Sl+    0:04 /usr/lib/xorg/Xorg vt1 -displayfd 3 -auth /run/user/125/gdm/Xauthority -background none -noreset -keeptty -verbose 3
  834 tty1    S+     0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
  835 tty1    S+     0:00 dbus-daemon --nofork --print-address 4 --session
  836 tty1    Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --autostart /usr/share/gdm/greeter/autostart
  844 tty1    Sl+    0:00 /usr/libexec/at-spi-bus-launcher
  849 tty1    S+     0:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
  886 tty1    Sl+    0:13 /usr/bin/gnome-shell
  904 tty1    Sl     0:00 ibus-daemon --panel disable --xln
  907 tty1    Sl     0:00 /usr/libexec/ibus-memconf
  910 tty1    Sl     0:00 /usr/libexec/ibus-x11 --kill-daemon
  915 tty1    Sl     0:00 /usr/libexec/ibus-x11 --kill-daemon

er 1075 tty1    Sl+    0:00 /usr/libexec/gsd-print
1340 tty2    Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
1342 tty2    Sl+    29:34 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
1348 tty2    Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
1876 pts/0    Ss     0:00 bash
3411 pts/1    Ss+    0:00 bash
3798 pts/2    Ss+    0:00 bash
5598 pts/0    S+     0:00 make rtest13
5599 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
5600 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
5601 pts/0    S+     0:00 ./tshref -p
5603 pts/0    T      0:00 ./mysplit 4
5604 pts/0    T      0:00 ./mysplit 4
5607 pts/0    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY          STAT TIME  COMMAND
  826 tty1    Ssl+   0:00 /usr/lib/gdm3/gdm-x-session dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
  828 tty1    Sl+    0:04 /usr/lib/xorg/Xorg vt1 -displayfd 3 -auth /run/user/125/gdm/Xauthority -background none -noreset -keeptty -verbose 3
  834 tty1    S+     0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
  835 tty1    S+     0:00 dbus-daemon --nofork --print-address 4 --session
  836 tty1    Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --autostart /usr/share/gdm/greeter/autostart
  844 tty1    Sl+    0:00 /usr/libexec/at-spi-bus-launcher
  849 tty1    S+     0:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
  886 tty1    Sl+    0:13 /usr/bin/gnome-shell
  904 tty1    Sl     0:00 ibus-daemon --panel disable --xln
  907 tty1    Sl     0:00 /usr/libexec/ibus-memconf
  910 tty1    Sl     0:00 /usr/libexec/ibus-x11 --kill-daemon
  915 tty1    Sl     0:00 /usr/libexec/ibus-x11 --kill-daemon
```



trace13은 fg를 통해 sigcont의 작동을 확인한다. 앞서 fg를 구현하였으므로, 모든 프로세스에 대해 fg가 작동함을 보이면 된다. 사진과 같이 현재 시스템 정보를 출력하는 bin/ps a는 중간에 한 번 stop되었다가 fg에 의해 다시 작동되는 것을 확인할 수 있다.

1.3.14. trace14



trace14는 에러 인풋을 적절히 처리하는지 확인한다. invalid command, fgbg에서 invalid argument 혹은 none argument, valid argument이지만 해당 프로세스는 존재하지 않는 경우를 처리하도록 하면 된다. if 문을 통해 구현하였다. invalid command의 경우, 처음에는 if문을 통해 구현하였지만, execve는 제대로 작동하였을 때 return을 하지 않는 함수라는 점을 활용하여 execve 이후 구문에 reach하는 경우 무조건 invalid command로 간주하도록 구현하였다.

```
execve(argv[0], argv, environ);
```

```

/*
 * legal input never reaches here
 * not using ch_execve since the shell must go on
 */
printf("%s: Command not found\n", argv[0]);
exit(0);

```

```

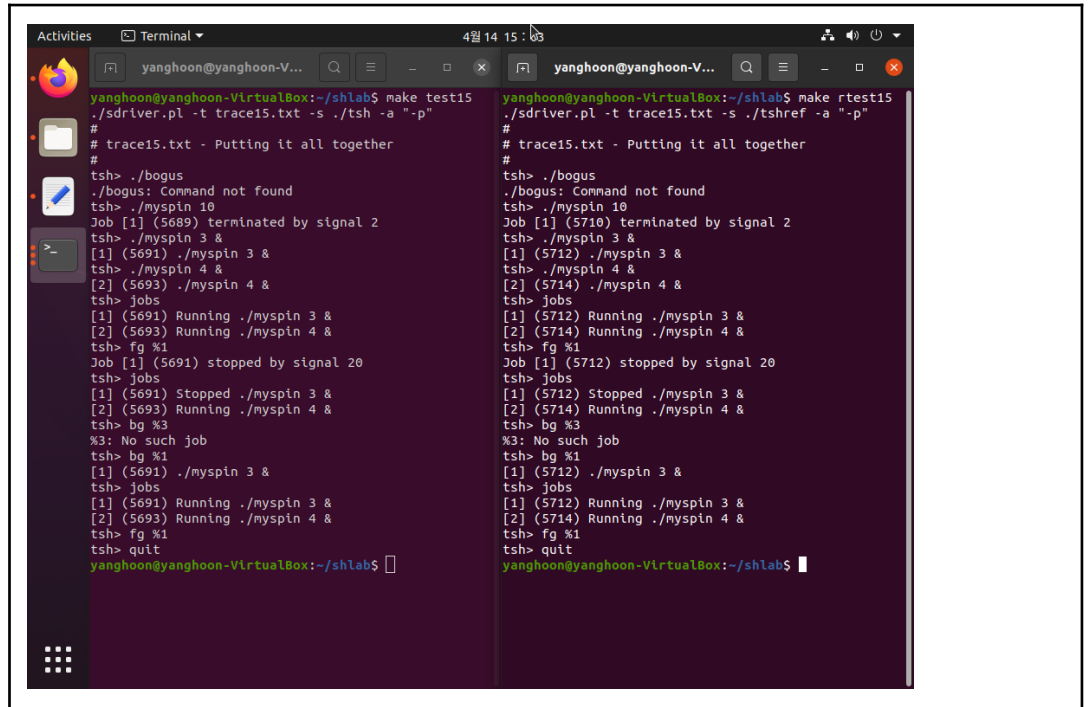
void do_bgfg(char **argv)
{
    struct job_t *job;

    /* handle input errors */
    if(argv[1] == NULL) /* no argument */
    {
        printf("%s command requires PID or %%jobid argument\n", argv[0]);
        return;
    }
    if(argv[1][0] == '%') /* jid is given */
    {
        if((job = getjobjid(jobs, atoi(&argv[1][1]))) == NULL) /* no job with
given jid */
        {
            printf("%s: No such job\n", argv[1]);
            return;
        }
    }
    else if(isdigit(argv[1][0])) /* pid is given */
    {
        if((job = getjobpid(jobs, atoi(&argv[1][0]))) == NULL) /* no job with
given pid */
        {
            printf("(%s): No such process\n", argv[1]);
            return;
        }
    }
    else /* argument is in wrong format */
    {
        printf("%s: argument must be a PID or %%jobid\n", argv[0]);
        return;
    }

    (생략)
}

```

1.3.15. trace15

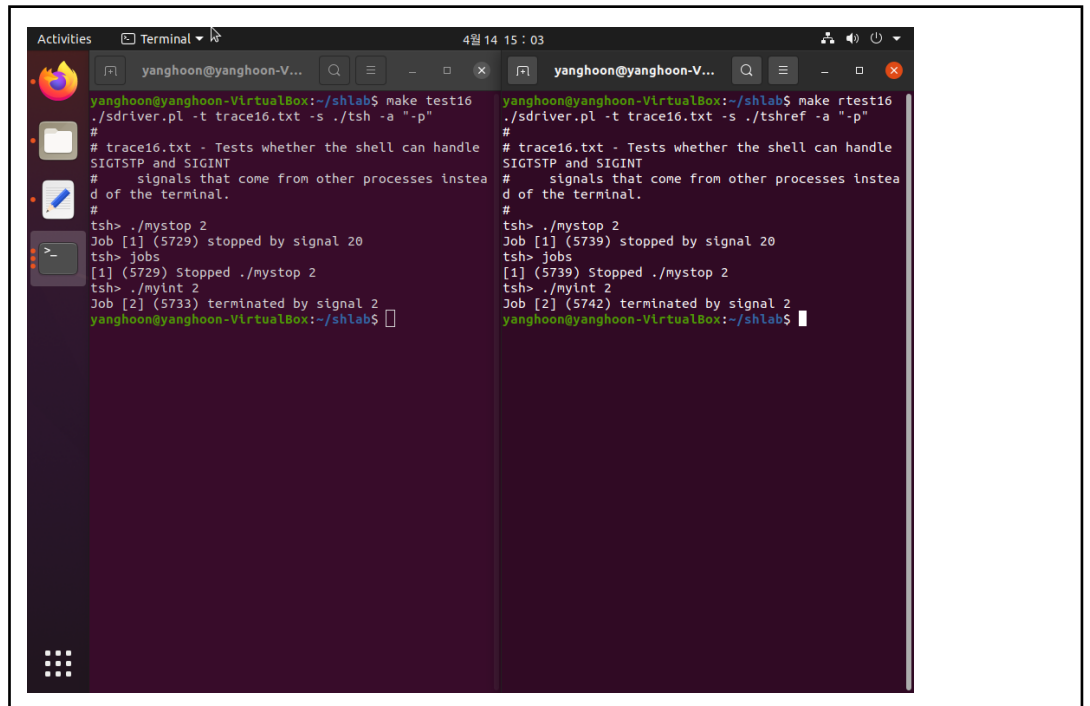


```
yanghoon@yanghoon-VirtualBox:~/shlab$ make test15
./sdriver.pl -t trace15.txt -s ./tsh -a "-p"
#
# trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 10
Job [1] (5689) terminated by signal 2
tsh> ./myspin 3 &
[1] (5691) ./myspin 3 &
tsh> ./myspin 4 &
[2] (5693) ./myspin 4 &
tsh> jobs
[1] (5691) Running ./myspin 3 &
[2] (5693) Running ./myspin 4 &
tsh> fg %1
Job [1] (5691) stopped by signal 20
tsh> jobs
[1] (5691) Stopped ./myspin 3 &
[2] (5693) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (5691) ./myspin 3 &
tsh> jobs
[1] (5691) Running ./myspin 3 &
[2] (5693) Running ./myspin 4 &
tsh> fg %1
tsh> quit
yanghoon@yanghoon-VirtualBox:~/shlab$

yanghoon@yanghoon-VirtualBox:~/shlab$ 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] (5710) terminated by signal 2
tsh> ./myspin 3 &
[1] (5712) ./myspin 3 &
tsh> ./myspin 4 &
[2] (5714) ./myspin 4 &
tsh> jobs
[1] (5712) Running ./myspin 3 &
[2] (5714) Running ./myspin 4 &
tsh> fg %1
Job [1] (5712) stopped by signal 20
tsh> jobs
[1] (5712) Stopped ./myspin 3 &
[2] (5714) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (5712) ./myspin 3 &
tsh> jobs
[1] (5712) Running ./myspin 3 &
[2] (5714) Running ./myspin 4 &
tsh> fg %1
tsh> quit
yanghoon@yanghoon-VirtualBox:~/shlab$
```

trace15는 여러 경우를 모두 확인한다. 구현한 코드로 수행해보니 command not found부터 fg, bg 등 다양한 기능이 문제 없이 수행됨을 확인할 수 있었다.

1.3.16. trace16



```
yanghoon@yanghoon-VirtualBox:~/shlab$ 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] (5729) stopped by signal 20
tsh> jobs
[1] (5729) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (5733) terminated by signal 2
yanghoon@yanghoon-VirtualBox:~/shlab$

yanghoon@yanghoon-VirtualBox:~/shlab$ 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] (5739) stopped by signal 20
tsh> jobs
[1] (5739) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (5742) terminated by signal 2
yanghoon@yanghoon-VirtualBox:~/shlab$
```

trace16은 다른 프로세스에서 전달되는 signal을 처리하는지에 대해 확인한다. mystop은 2를 인수로 가질 경우 자신의 프로세스 그룹에 sigtstp를 보낸다. myint 역시 2를 인수로 가질 경우 자신의 프로세스 그룹에 sigint를 보낸다. 하지만 signal을 catch하는 것은 같으므로 앞선 구현만으로도 이를 확인할 수 있다.

2. 어려웠던 점

그동안 수업으로만 접하던 system call들을 직접 사용하려니, 잘 안다고 생각했지만 어떤 값을 return하는지, error는 어떻게 처리하는지 등이 헛갈렸다. man page에 함수를 검색해가며 하나씩

코드를 작성하였는데, **unix system call**에 대해 확실히 활용하면서 배울 수 있었다. 구현 상으로는 **trace**를 작동시켰을 때 무한루프에 갇히는 일이 발생하여 이에 대한 디버깅을 오래 하였다. 코드를 분석해보니 **sigint**나 **sigstp** 에서와는 다르게 **sigcont**는 **-pid**가 아닌 **pid**에 전달하고 있었음을 확인할 수 있었다. 이 이슈를 해결하니 모든 **stop**되었던 프로세스가 정상적으로 **fg** 혹은 **bg**를 통해 **sigcont**되어 무한루프에 갇히지 않게 되었다.

3. 놀라웠던 점

shell lab의 경우 **signal**을 **handle**하는 것이 가장 큰 **implementation** 주제였다. **signal**에 대해 수업 시간에 배웠지만 시스템 단계에서만 사용되는 기능이라고 생각했는데, 직접 **handler**를 구현하면서 **implementation** 단계에서 **signal**을 하나의 **input element**처럼 취급할 수도 있다는 것을 알게 되었다. 컴퓨터구조를 수강할 때 더 작은 단계에서 조각을 맞춰나가면서 프로세서와 컴퓨터에 대해 알게된 것처럼 본 과목을 통해 시스템 단계에서 조각을 맞춰나가 소프트웨어적 역량을 기르는 것을 목표로 하고 있는데, 실제 시스템이 시그널을 이용해 서로 컨트롤하는 방법에 대해 배웠다. 또 사소하지만 지금까지 **windows**만 이용하면서 **unix**에 대해서는 잘 몰랐는데, **ctrl-c**를 통해 **sigint**를 보낼 수 있다는 것도 놀라웠던 점 중 하나이다.