Intro. to Computer SW Systems Lab Report

[Shell Lab]

20220100 박기현

kihyun@postech.ac.kr

명예 서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

void eval(char *cmdline)

: Evaluate the command line that the user has just typed in

code/ecf/shellex.c

```
/* eval - Evaluate a command line */
void eval(char *cmdline)
{
    char *argv[MAXARGS]; /* Argument list execve() */
    char buf[MAXLINE];   /* Holds modified command line */
    int bg;              /* Should the job run in bg or fg? */
    pid_t pid;           /* Process id */

    strcpy(buf, cmdline);
    bg = parseline(buf, argv);
    if (argv[0] == NULL)
        return;   /* Ignore empty lines */

    if (!builtin_command(argv)) {
        if ((pid = Fork()) == 0) {   /* Child runs user job */
            if (execve(argv[0], argv, environ) < 0) {
                printf("%s: Command not found.\n", argv[0]);
                exit(0);
            }
        }

        /* Parent waits for foreground job to terminate */
        if (!bg) {
            int status;
            if (waitpid(pid, &status, 0) < 0)
                unix_error("waitfg: waitpid error");
        }
        else
            printf("%d %s", pid, cmdline);
    }
    return;
}
```

```
    int pid;
    sigset_t mask_all, mask_one, prev_one;

    Sigfillset(&mask_all);
    Sigemptyset(&mask_one);
    Sigaddset(&mask_one, SIGCHLD);
    Signal(SIGCHLD, handler);
    initjobs(); /* Initialize the job list */

    while (1) {
        Sigprocmask(SIG_BLOCK, &mask_one, &prev_one); /* Block SIGCHLD */
        if ((pid = Fork()) == 0) { /* Child process */
            Sigprocmask(SIG_SETMASK, &prev_one, NULL); /* Unblock SIGCHLD */
            Execve("/bin/date", argv, NULL);
        }
        Sigprocmask(SIG_BLOCK, &mask_all, NULL); /* Parent process */
        addjob(pid);  /* Add the child to the job list */
        Sigprocmask(SIG_SETMASK, &prev_one, NULL);  /* Unblock SIGCHLD */
    }
```

<CSAPP 교재 참고>

```
char *argv[MAXARGS];                    //argument list execve()
char buf[MAXLINE];                      //holds modified command line
int bg;                                 //should the job run in bg or fg?
pid_t pid;                              //process id
int state;                              //job state
sigset_t mask_all, mask_one, prev_one; //signal set
```

변수는 위와 같이 선언한다.

```
strcpy(buf, cmdline);
bg = parseline(buf, argv);
state = (bg == 1 ? BG : FG);
if(argv[0] == NULL)
    return; //ignore empty lines
```

strcpy 함수를 호출하여 사용자가 입력한 cmdline 문자열을 buf 배열에 복사한다. 이후 parseline 함수를 호출하여 cmdline, 즉 buf에 저장된 문자열의 단어 각각을 argv 배열에 순서대로 parsing하여 저장한다. 이때, parseline 함수는 foreground 작업인지, background 작업인지에 대해 '&'의 유무에 따라 다르게 반환한다. 즉, foreground 작업이면 bg의 값은 0으로, background 작업이면 bg의 값은 1로 설정된다. 설정된 bg의 값에 따라 작업의 state를 BG 혹은 FG로 설정한다. argv 배열의 첫 번째 원소는 built-in command의 이름, 또는 executable file의 경로 이름이다. 따라서 argv[0] 값이 NULL이면 command line이 비어 있다는 뜻이므로 함수를 종료한다.

```
if(!builtin_cmd(argv))
{
    if(sigfillset(&mask_all) < 0)                   //add every signal to set
        app_error("Sigfillset error");              //sigfillset error handling
    if(sigemptyset(&mask_one) < 0)                  //initialize set to the empty set
        app_error("Sigemptyset error");             //sigemptyset error handling
    if(sigaddset(&mask_one, SIGCHLD) < 0)           //add SIGCHLD to signal set
        app_error("Sigaddset error");               //sigaddset error handling
    if(sigprocmask(SIG_BLOCK, &mask_one, &prev_one) < 0)    //block SIGCHLD signals
        unix_error("Sigprocmask error");            //sigprocmask error handling

    if((pid = fork()) == 0) //child runs user job
    {
        if(setpgid(0, 0) < 0)                       //put the child in a new process group whose group ID is identical to the child's PID
            unix_error("Setpgid error");            //setpgid error handling
        if(sigprocmask(SIG_SETMASK, &prev_one, NULL) < 0)   //unblock SIGCHLD signals
            unix_error("Sigprocmask error");        //sigprocmask error handling

        if(execve(argv[0], argv, environ) < 0)      //load and run the executable object file argv[0] with the argument list argv and the environment variable list environ
        {
            printf("%s: Command not found\n", argv[0]);     //execve error handling
            exit(0);
        }
    }
    else if (pid < 0)
    {
        unix_error("Fork error"); //fork error handling
    }
    else
    {
        if(sigprocmask(SIG_BLOCK, &mask_all, NULL) < 0)     //block every signal
            unix_error("Sigprocmask error");        //sigprocmask error handling
        addjob(jobs, pid, state, cmdline);          //add a job to the job list
        if(sigprocmask(SIG_SETMASK, &prev_one, NULL) < 0)   //unblock every signal
            unix_error("Sigprocmask error");        //sigprocmask error handling

        if(!bg) //foreground
            waitfg(pid);                                    //block until process pid is no longer the foreground process
        else    //background
            printf("[%d] (%d) %s", pid2jid(pid), (int)pid, cmdline); //print the information of the background job
    }
}
```

shell은 argv의 첫 번째 원소가 built-in command의 이름이면 현재 프로세스에서 즉시 command를 실행하고, executable file의 경로 이름이면 자식 프로세스를 생성하여 자식 프로세스의 문맥 내에서 프로그램을 실행한다. 따라서 builtin_cmd 함수를 호출하여 실행 방법을 결정한다. builtin_cmd 함수의 결과로 0이 반환되면, executable file을 실행하는 상황이므로 자식 프로세스를 생성하는 과정을 수행한다.

```
if(sigfillset(&mask_all) < 0)                           //add every signal to set
    app_error("Sigfillset error");                      //sigfillset error handling
if(sigemptyset(&mask_one) < 0)                          //initialize set to the empty set
    app_error("Sigemptyset error");                     //sigemptyset error handling
if(sigaddset(&mask_one, SIGCHLD) < 0)                   //add SIGCHLD to signal set
    app_error("Sigaddset error");                       //sigaddset error handling
if(sigprocmask(SIG_BLOCK, &mask_one, &prev_one) < 0)    //block SIGCHLD signals
    unix_error("Sigprocmask error");                    //sigprocmask error handling
```

이때, 부모 프로세스는 자식 프로세스를 생성하기 전에 sigprocmask 함수를 호출하여 반드시 SIGCHLD 시그널을 block해야 하고, 생성 이후 addjob 함수를 호출하여 자식 프로세스를 작업 목록에 추가한 후에 다시 이 시그널을 unblock해야 한다. 부모 프로세스가 이러한 방식으로 SIGCHLD 시그널을 block하는 이유는 부모 프로세스가 addjob 함수를 호출하기 전에 sigchld_handler에 의해 자식 프로세스가 먼저 reaping되는 것, 즉 race condition을 피하기 위함이다. 이때, 자식 프로세스는 부모 프로세스의 blocked vectors를 상속받기 때문에 새 프로그램을 실행하기 전에 반드시 SIGCHLD 시그널을 unblock해주어야 한다.

sigfillset, sigemptyset, sigaddset, sigprocmask, 각 함수에 대한 반환 값을 확인하여 각 함수에 대한 error handling을 진행한다. 이는 error-reporting function인 unix_error 함수를 호출하여 수행한다.

```
if((pid = fork()) == 0) //child runs user job
{
    if(setpgid(0, 0) < 0)                               //put the child in a new process group whose group ID is identical to the child's PID
        unix_error("Setpgid error");                    //setpgid error handling
    if(sigprocmask(SIG_SETMASK, &prev_one, NULL) < 0)   //unblock SIGCHLD signals
        unix_error("Sigprocmask error");                //sigprocmask error handling

    if(execve(argv[0], argv, environ) < 0)              //load and run the executable object file argv[0] with the argument list argv and the environment variable list environ
    {
        printf("%s: Command not found\n", argv[0]);     //execve error handling
        exit(0);
    }
}
else if (pid < 0)
{
    unix_error("Fork error"); //fork error handling
}
else
{
    if(sigprocmask(SIG_BLOCK, &mask_all, NULL) < 0)     //block every signal
        unix_error("Sigprocmask error");                //sigprocmask error handling
    addjob(jobs, pid, state, cmdline);                  //add a job to the job list
    if(sigprocmask(SIG_SETMASK, &prev_one, NULL) < 0)   //unblock every signal
        unix_error("Sigprocmask error");                //sigprocmask error handling

    if(!bg) //foreground
        waitfg(pid);                                        //block until process pid is no longer the foreground process
    else    //background
        printf("[%d] (%d) %s", pid2jid(pid), (int)pid, cmdline); //print the information of the background job
}
```

fork 함수는 자식 프로세스인 경우 0을, 부모 프로세스인 경우 자식의 PID를 반환한다. 이를 통해 fork 함수의 반환 값을 pid 변수에 저장하여 각 프로세스에 맞는 동작을 수행한다.

```
if(setpgid(0, 0) < 0)                               //put the child in a new process group whose group ID is identical to the child's PID
    unix_error("Setpgid error");                    //setpgid error handling
if(sigprocmask(SIG_SETMASK, &prev_one, NULL) < 0)   //unblock SIGCHLD signals
    unix_error("Sigprocmask error");                //sigprocmask error handling

if(execve(argv[0], argv, environ) < 0)              //load and run the executable object file argv[0] with the argument list argv and the environment variable list environ
{
    printf("%s: Command not found\n", argv[0]);     //execve error handling
    exit(0);
}
```

pid가 0인 경우는 자식 프로세스를 수행하는 경우이므로 이 프로세스 내에서 executable file을 수행한다. 이때, setpgid 함수를 호출하여 자식 프로세스의 PID와 동일한 그룹 프로세스 ID를 가진 새 프로세스 그룹에 자식 프로세스를 두어야 한다. 그 이유는 다음과 같다. shell은 foreground

프로세스 그룹에서 실행되고 있다. 만약 shell이 자식 프로세스를 생성하면, 기본적으로 자식 프로세스는 foreground 프로세스 그룹 내에 위치하게 된다. 따라서 Ctrl+C를 입력하면, SIGINT 시그널이 foreground 프로세스 그룹 내의 전체 프로세스에 전달된다. 즉, shell과 shell이 생성한 모든 프로세스에 SIGINT 시그널이 전달되므로 우리가 구현하고자 하는 shell의 동작과 맞지 않다. 이를 해결하기 위해 foreground 프로세스 그룹에 오직 한 프로세스, 즉 shell만을 남겨두고, 이후 shell이 SIGINT 시그널을 확인하면, 이를 foreground 작업을 포함하는 프로세스 그룹으로 전달하도록 구현한다.

setpgid, sigprocmask, 각 함수에 대한 반환 값을 확인하여 각 함수에 대한 error handling을 진행한다.

pid가 음수인 경우 fork에 대한 error handling을 진행한다.

```
if(sigprocmask(SIG_BLOCK, &mask_all, NULL) < 0)     //block every signal
    unix_error("Sigprocmask error");                //sigprocmask error handling
addjob(jobs, pid, state, cmdline);                  //add a job to the job list
if(sigprocmask(SIG_SETMASK, &prev_one, NULL) < 0)   //unblock every signal
    unix_error("Sigprocmask error");                //sigprocmask error handling

if(!bg) //foreground
    waitfg(pid);                                            //block until process pid is no longer the foreground process
else    //background
    printf("[%d] (%d) %s", pid2jid(pid), (int)pid, cmdline); //print the information of the background job
```

pid가 양수인 경우는 부모 프로세스를 수행하는 경우이므로 addjob 함수를 호출하여 작업 목록에 자식 프로세스를 추가한다. foreground 프로세스는 한 번에 한 프로세스만을 실행할 수 있으므로, 자식 프로세스가 foreground 작업인 경우 자식 프로세스가 실행되는 동안 부모 프로세스를 대기시킨다. 이는 waitfg 함수를 호출하여 수행한다. background 작업인 경우 작업에 대한 정보를 출력한다.

int builtin_cmd(char **argv)

: If the user has typed a built-in command, then execute it immediately

```
34   /* If first arg is a builtin command, run it and return true */
35   int builtin_command(char **argv)
36   {
37       if (!strcmp(argv[0], "quit")) /* quit command */
38           exit(0);
39       if (!strcmp(argv[0], "&"))    /* Ignore singleton & */
40           return 1;
41       return 0;                     /* Not a builtin command */
42   }
```

<CSAPP 교재 참고>

```
int builtin_cmd(char **argv)
{
    if(!strcmp(argv[0], "quit"))      //quit command
    {
        exit(0);
    }
    else if(!strcmp(argv[0], "fg"))   //fg command
    {
        do_bgfg(argv);                //execute the builtin fg commands
        return 1;
    }
    else if(!strcmp(argv[0], "bg"))   //bg command
    {
        do_bgfg(argv);                //execute the builtin bg commands
        return 1;
    }
    else if(!strcmp(argv[0], "jobs")) //jobs command
    {
        listjobs(jobs);               //print the job list
        return 1;
    }
    return 0;     /* not a builtin command */
}
```

built-in command에는 'quit', 'fg', 'bg', 'jobs'가 있다. strcmp 함수를 호출하여 argv의 첫 번째 원소와 각 command를 비교하고, 해당하는 command에 맞는 동작을 즉시 실행한다.

먼저, 'quit'는 exit 함수를 호출하여 shell을 종료한다.

다음으로 'fg', 'bg'는 argv의 두 번째 원소로 주어진 PID 혹은 JID에 해당하는 프로세스를 SIGCONT 시그널을 보내어 foreground, background로 다시 실행한다. 이는 do_bgfg 함수를 호출하여 실행한다.

마지막으로 'jobs'는 background 작업의 목록을 모두 출력한다. 이는 listjobs 함수를 호출하여 실행한다.

위와 같은 built-in command인 경우에는 shell이 현재 프로세스에서 즉시 실행하므로 1을 반환하고, 그 이외의 경우에는 0을 반환하여 자식 프로세스를 생성하도록 한다.

void do_bgfg(char **argv)

: Execute the built-in bg and fg commands

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

if(argv[1] == NULL)
{
    printf("%s command requires PID or %%jobid argument\n", argv[0]); //error handling
    return;
}
else
{
    if(argv[1][0] == '%') //jid
    {
        jid = atoi(argv[1] + 1);

        if((job = getjobjid(jobs, jid)) == NULL)  //find a job (by JID) on the job list
        {
            printf("%s: No such job\n", argv[1]); //error handling
            return;
        }
    }
    else                    //pid
    {
        if((pid = atoi(argv[1])) == 0)
        {
            printf("%s: argument must be a PID or %%jobid\n", argv[0]); //error handling
            return;
        }
        else
        {
            if((job = getjobpid(jobs, pid)) == NULL)        //find a job (by PID) on the job list
            {
                printf("(%s): No such process\n", argv[1]); //error handling
                return;
            }
        }
    }
}
```

가장 먼저 argv의 두 번째 원소를 확인하여 작업의 PID 혹은 JID가 입력되었는지 확인한다. 만약 NULL이면, 에러 메시지를 출력하고 반환한다.

'%'의 유무로 PID와 JID를 구분하므로, argv의 두 번째 원소의 첫 번째 문자를 확인한다.

먼저, JID인 경우, '%'를 제외한 문자열을 정수로 변환하여 jid 변수에 저장한다. 그리고 나서 getjobjid 함수를 호출하여 해당하는 작업을 불러온다. 이때, 작업을 찾지 못하면 에러 메시지를 출력한다.

다음으로, PID인 경우, 문자열을 정수로 변환하여 pid 변수에 저장한다. atoi 함수는 문자열을 정수로 변환하는 함수로, 변환할 수 없는 문자열(숫자가 아닌 문자열)인 경우 0을 반환한다. 이를 이용하여 숫자가 아닌 경우 에러 메시지를 출력한다. 그리고 나서 getjobpid 함수를 호출하여 해당하는 작업을 불러온다. 이때, 작업을 찾지 못하면 에러 메시지를 출력한다.

```
if(kill(-job->pid, SIGCONT) < 0) //send SIGCONT signal to every process in process group |PID|
    unix_error("Kill error");    //kill error handling

if(!strcmp(argv[0], "fg"))       //fg command
{
    job->state = FG;             //set job state to FG
    waitfg(job->pid);            //block until process PID is no longer the foreground process
}
else if(!strcmp(argv[0], "bg"))  //bg command
{
    job->state = BG;                                    //set job state to BG
    printf("[%d] (%d) %s", job->jid, job->pid, job->cmdline); //print the information of the background job
}
```

입력 받은 PID 혹은 JID에 맞는 작업을 찾았다면, kill 함수를 호출하여 SIGCONT 시그널을 전달한다.

'fg' command를 수행하는 경우 작업의 state를 FG로 변경하고, waitfg 함수를 호출하여 부모 프로세스를 대기시킨다.

'bg' command를 수행하는 경우 작업의 state를 BG로 변경하고, 작업에 대한 정보를 출력한다.

void waitfg(pid_t pid)

: Block until process PID is no longer the foreground process

```
if(getjobpid(jobs, pid) == NULL) //find a job (by PID) on the job list
    return;

while(pid == fgpid(jobs)) //return PID of current foreground job, 0 if no such job
    sleep(1);             //suspend a process for a specified period of time
```

만약 전달 받은 PID에 해당하는 작업이 없으면 반환한다.

만약 있으면, 작업 목록에 그 작업이 있는 동안에는 sleep 함수를 호출하여 다른 foreground 프로세스를 대기시킨다.

void sigchld_handler(int sig)

: Reap all available zombie children, but doesn't wait for any other currently running children to terminate

```
    /* Parent reaps N children in no particular order */
    while ((pid = waitpid(-1, &status, 0)) > 0) {
        if (WIFEXITED(status))
            printf("child %d terminated normally with exit status=%d\n",
                   pid, WEXITSTATUS(status));
        else
            printf("child %d terminated abnormally\n", pid);
    }

    /* The only normal termination is if there are no more children */
    if (errno != ECHILD)
        unix_error("waitpid error");
```

```
void handler(int sig)
{
    int olderrno = errno;
    sigset_t mask_all, prev_all;
    pid_t pid;

    Sigfillset(&mask_all);
    while ((pid = waitpid(-1, NULL, 0)) > 0) { /* Reap a zombie child */
        Sigprocmask(SIG_BLOCK, &mask_all, &prev_all);
        deletejob(pid); /* Delete the child from the job list */
        Sigprocmask(SIG_SETMASK, &prev_all, NULL);
    }
    if (errno != ECHILD)
        Sio_error("waitpid error");
    errno = olderrno;
}
```

<CSAPP 교재 참고>

```
struct job_t* job;
pid_t wpid;
int child_status;
sigset_t mask_all, prev_all;

if(sigfillset(&mask_all) < 0)      //add every signal to set
    app_error("Sigfillset error"); //sigfillset error handling

while((wpid = waitpid(-1, &child_status, WNOHANG | WUNTRACED)) > 0) //suspend current process until specific process terminates
{
    job = getjobpid(jobs, wpid);       //find a job (by PID) on the job list

    if(WIFEXITED(child_status))        //return true if the child terminated normally, via a call to exit or a return
    {
        if(sigprocmask(SIG_BLOCK, &mask_all, &prev_all) < 0) //block every signal
            unix_error("Sigprocmask error");                 //sigprocmask error handling
        deletejob(jobs, wpid);                               //delete a job whose PID = pid from the job list
        if(sigprocmask(SIG_SETMASK, &prev_all, NULL) < 0)    //unblock every signal
            unix_error("Sigprocmask error");                 //sigprocmask error handling
    }
    else if(WIFSTOPPED(child_status))  //return true if the child that caused the return is currently stopped
    {
        job->state = ST;               //set job state to ST
        printf("Job [%d] (%d) stopped by signal %d\n", job->jid, job->pid, WSTOPSIG(child_status));
    }
    else if(WIFSIGNALED(child_status)) //return true if the child process terminated because of a signal that was not caught
    {
        printf("Job [%d] (%d) terminated by signal %d\n", job->jid, job->pid, WTERMSIG(child_status));
        if(sigprocmask(SIG_BLOCK, &mask_all, &prev_all) < 0) //block every signal
            unix_error("Sigprocmask error");                 //sigprocmask error handling
        deletejob(jobs, wpid);                               //delete a job whose PID = pid from the job list
        if(sigprocmask(SIG_SETMASK, &prev_all, NULL) < 0)    //unblock every signal
            unix_error("Sigprocmask error");                 //sigprocmask error handling
    }
}

if(wpid < 0 && errno != ECHILD)
    unix_error("Waitpid error"); //waitpid error handling
```

```
while((wpid = waitpid(-1, &child_status, WNOHANG | WUNTRACED)) > 0) //suspend current process until specific process terminates
{
    job = getjobpid(jobs, wpid);       //find a job (by PID) on the job list
```

자식 프로세스가 중단되었거나, 종료된 경우 SIGCHLD 시그널을 전달하여 위와 같은 handler를 호출한다. waitpid 함수의 WNOHANG | WUNTRACED 옵션을 인자로 전달하여 정지된, 혹은 종료된 자식 프로세스의 PID를 wpid 변수에 저장한다. getjobpid 함수를 호출하여 wpid에 해당하는 작업을 불러오고, 종료된 경우와 중단된 경우 각각에 대해서 각 동작을 수행한다.

WNOHANG | WUNTRACED: Return immediately, with a return value of 0, if none of the children in the wait set has stopped or terminated, or with a return value equal to the PID of one of the stopped or terminated children.

```
if(WIFEXITED(child_status))        //return true if the child terminated normally, via a call to exit or a return
{
    if(sigprocmask(SIG_BLOCK, &mask_all, &prev_all) < 0) //block every signal
        unix_error("Sigprocmask error");                 //sigprocmask error handling
    deletejob(jobs, wpid);                               //delete a job whose PID = pid from the job list
    if(sigprocmask(SIG_SETMASK, &prev_all, NULL) < 0)    //unblock every signal
        unix_error("Sigprocmask error");                 //sigprocmask error handling
}
```

WIFEXITED는 exit 호출 또는 return을 통해서 자식이 정상적으로 종료된 경우에 true를 반환하므로, deletejob 함수를 호출하여 작업 목록에서 작업을 삭제한다.

```
else if(WIFSTOPPED(child_status))  //return true if the child that caused the return is currently stopped
{
    job->state = ST;               //set job state to ST
    printf("Job [%d] (%d) stopped by signal %d\n", job->jid, job->pid, WSTOPSIG(child_status));
}
```

WIFSTOPPED는 리턴을 하게 한 자식 프로세스가 현재 정지된 상태라면 true를 반환하므로, 작업의 state를 ST로 변경하고, 작업이 중단되었다는 메시지를 출력한다.

```
else if(WIFSIGNALED(child_status)) //return true if the child process terminated because of a signal that was not caught
{
    printf("Job [%d] (%d) terminated by signal %d\n", job->jid, job->pid, WTERMSIG(child_status));
    if(sigprocmask(SIG_BLOCK, &mask_all, &prev_all) < 0) //block every signal
        unix_error("Sigprocmask error");                 //sigprocmask error handling
    deletejob(jobs, wpid);                               //delete a job whose PID = pid from the job list
    if(sigprocmask(SIG_SETMASK, &prev_all, NULL) < 0)    //unblock every signal
        unix_error("Sigprocmask error");                 //sigprocmask error handling
}
```

WIFSIGNALED는 시그널에 의해 자식 프로세스가 종료된 경우에 true를 반환하므로, 작업이 종료되었다는 메시지를 출력하고, deletejob 함수를 호출하여 작업 목록에서 작업을 삭제한다.

```
if(wpid < 0 && errno != ECHILD)
    unix_error("Waitpid error"); //waitpid error handling
```

waitpid 함수 반환 값과 errno을 확인하여 waitpid에 대한 error handling을 진행한다.

void sigint_handler(int sig)

: Catch SIGINT and send it along to the foreground job

```
pid_t pid;

pid = fgpid(jobs); //return PID of current foreground job, 0 if no such job

if(pid)
    if(kill(-pid, sig) < 0)             //send SIGINT signal to every process in process group |PID|
        unix_error("Kill (int) error"); //kill error handling
```

키보드로부터 Ctrl+C가 입력된 경우 SIGINT 시그널을 전달하여 위와 같은 handler를 호출한다. fgpid 함수를 호출하여 현재 foreground 작업의 PID를 pid 변수에 저장한다. 이때, 작업이 존재하지 않으면 pid는 0이므로, 존재하는 경우에 대해서 pid에 해당하는 프로세스에 SIGINT 시그널을 전달한다.

void sigtstp_handler(int sig)

: Catch SIGTSTP and suspend the foreground job by sending it a SIGTSTP

```
pid_t pid;

pid = fgpid(jobs); //return PID of current foreground job, 0 if no such job

if(pid)
    if(kill(-pid, sig) < 0)             //send SIGTSTP signal to every process in process group |PID|
        unix_error("Kill (tstp) error"); //kill error handling
```

키보드로부터 Ctrl+Z가 입력된 경우 SIGTSTP 시그널을 전달하여 위와 같은 handler를 호출한다. fgpid 함수를 호출하여 현재 foreground 작업의 PID를 pid 변수에 저장한다. 이때, 작업이 존재하지 않으면 pid는 0이므로, 존재하는 경우에 대해서 pid에 해당하는 프로세스에 SIGTSTP 시그널을 전달한다.

[test01]

./tsh

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

./tshref

```
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
```

[test02]

./tsh

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

./tshref

```
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
```

[test03]

./tsh

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

./tshref

```
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
```

[test04]

./tsh

```
[kihyun@programming2 shelllab]$ make test04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (23161) ./myspin 1 &
```

./tshref

```
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (26252) ./myspin 1 &
```

[test05]

./tsh

```
[kihyun@programming2 shelllab]$ make test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (23549) ./myspin 2 &
tsh> ./myspin 3 &
[2] (23552) ./myspin 3 &
tsh> jobs
[1] (23549) Running ./myspin 2 &
[2] (23552) Running ./myspin 3 &
```

./tshref

```
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (26256) ./myspin 2 &
tsh> ./myspin 3 &
[2] (26258) ./myspin 3 &
tsh> jobs
[1] (26256) Running ./myspin 2 &
[2] (26258) Running ./myspin 3 &
```

[test06]

./tsh

```
[kihyun@programming2 shelllab]$ make test06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (23674) terminated by signal 2
```

./tshref

```
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (26263) terminated by signal 2
```

[test07]

./tsh

```
[kihyun@programming2 shelllab]$ make test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (23814) ./myspin 4 &
tsh> ./myspin 5
Job [2] (23819) terminated by signal 2
tsh> jobs
[1] (23814) Running ./myspin 4 &
```

./tshref

```
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (26267) ./myspin 4 &
tsh> ./myspin 5
Job [2] (26269) terminated by signal 2
tsh> jobs
[1] (26267) Running ./myspin 4 &
```

[test08]

./tsh

```
[kihyun@programming2 shelllab]$ make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (24119) ./myspin 4 &
tsh> ./myspin 5
Job [2] (24125) stopped by signal 20
tsh> jobs
[1] (24119) Running ./myspin 4 &
[2] (24125) Stopped ./myspin 5
```

./tshref

```
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (26274) ./myspin 4 &
tsh> ./myspin 5
Job [2] (26276) stopped by signal 20
tsh> jobs
[1] (26274) Running ./myspin 4 &
[2] (26276) Stopped ./myspin 5
```

[test09]

./tsh

```
[kihyun@programming2 shelllab]$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (24312) ./myspin 4 &
tsh> ./myspin 5
Job [2] (24314) stopped by signal 20
tsh> jobs
[1] (24312) Running ./myspin 4 &
[2] (24314) Stopped ./myspin 5
tsh> bg %2
[2] (24314) ./myspin 5
tsh> jobs
[1] (24312) Running ./myspin 4 &
[2] (24314) Running ./myspin 5
```

./tshref

```
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (26281) ./myspin 4 &
tsh> ./myspin 5
Job [2] (26283) stopped by signal 20
tsh> jobs
[1] (26281) Running ./myspin 4 &
[2] (26283) Stopped ./myspin 5
tsh> bg %2
[2] (26283) ./myspin 5
tsh> jobs
[1] (26281) Running ./myspin 4 &
[2] (26283) Running ./myspin 5
```

[test10]

./tsh

```
[kihyun@programming2 shelllab]$ make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (24421) ./myspin 4 &
tsh> fg %1
Job [1] (24421) stopped by signal 20
tsh> jobs
[1] (24421) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

./tshref

```
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (26290) ./myspin 4 &
tsh> fg %1
Job [1] (26290) stopped by signal 20
tsh> jobs
[1] (26290) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

[test11]

./tsh

```
[kihyun@programming2 shelllab]$ 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] (24609) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
24602 pts/156  S+     0:00 ./tsh -p
24611 pts/60   S+     0:00 make test06
```

./tshref

```
./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] (26298) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
25181 pts/3    S      0:00 -usr/local/bin/tcsh -i
26239 pts/3    S      0:00 make tshrefout
26240 pts/3    S      0:00 /bin/sh -c make tests > tshref.out 2>&1
26241 pts/3    S      0:00 make tests
26295 pts/3    S      0:00 perl ./sdriver.pl -t trace11.txt -s ./tsh -a -p
26296 pts/3    S      0:00 ./tsh -p
26301 pts/3    R      0:00 /bin/ps a
```

[test12]

./tsh

```
[kihyun@programming2 shelllab]$ 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] (25340) stopped by signal 20
tsh> jobs
[1] (25340) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
25340 pts/156  T      0:00 ./mysplit 4
25342 pts/156  T      0:00 ./mysplit 4
```

./tshref

```
./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] (26305) stopped by signal 20
tsh> jobs
[1] (26305) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
25181 pts/3    S      0:00 -usr/local/bin/tcsh -i
26239 pts/3    S      0:00 make tshrefout
26240 pts/3    S      0:00 /bin/sh -c make tests > tshref.out 2>&1
26241 pts/3    S      0:00 make tests
26302 pts/3    S      0:00 perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p
26303 pts/3    S      0:00 ./tsh -p
26305 pts/3    T      0:00 ./mysplit 4
26306 pts/3    T      0:00 ./mysplit 4
26309 pts/3    R      0:00 /bin/ps a
```

[test13]

./tsh

```
[kihyun@programming2 shelllab]$ 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] (26151) stopped by signal 20
tsh> jobs
[1] (26151) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
26151 pts/156  T      0:00 ./mysplit 4
26153 pts/156  T      0:00 ./mysplit 4
tsh> fg %1
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
26148 pts/156  S+     0:00 ./tsh -p
26197 pts/156  R      0:00 /bin/ps a
```

./tshref

```
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (26313) stopped by signal 20
tsh> jobs
[1] (26313) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
25181 pts/3    S      0:00 -usr/local/bin/tcsh -i
26239 pts/3    S      0:00 make tshrefout
26240 pts/3    S      0:00 /bin/sh -c make tests > tshref.out 2>&1
26241 pts/3    S      0:00 make tests
26310 pts/3    S      0:00 perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
26311 pts/3    S      0:00 ./tsh -p
26313 pts/3    T      0:00 ./mysplit 4
26314 pts/3    T      0:00 ./mysplit 4
26317 pts/3    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
25181 pts/3    S      0:00 -usr/local/bin/tcsh -i
26239 pts/3    S      0:00 make tshrefout
26240 pts/3    S      0:00 /bin/sh -c make tests > tshref.out 2>&1
26241 pts/3    S      0:00 make tests
26310 pts/3    S      0:00 perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
26311 pts/3    S      0:00 ./tsh -p
26320 pts/3    R      0:00 /bin/ps a
```

[test14]

./tsh

```
[kihyun@programming2 shelllab]$ 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] (24914) ./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] (24914) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (24914) ./myspin 4 &
tsh> jobs
[1] (24914) Running ./myspin 4 &
```

./tshref

```
./sdriver.pl -t trace14.txt -s ./tsh -a "-p"
#
# 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 &
```

[test15]

./tsh

```
[kihyun@programming2 shelllab]$ 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] (27949) terminated by signal 2
tsh> ./myspin 3 &
[1] (27969) ./myspin 3 &
tsh> ./myspin 4 &
[2] (27975) ./myspin 4 &
tsh> jobs
[1] (27969) Running ./myspin 3 &
[2] (27975) Running ./myspin 4 &
tsh> fg %1
Job [1] (27969) stopped by signal 20
tsh> jobs
[1] (27969) Stopped ./myspin 3 &
[2] (27975) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (27969) ./myspin 3 &
tsh> jobs
[1] (27969) Running ./myspin 3 &
[2] (27975) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

./tshref

```
./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] (26343) terminated by signal 2
tsh> ./myspin 3 &
[1] (26345) ./myspin 3 &
tsh> ./myspin 4 &
[2] (26347) ./myspin 4 &
tsh> jobs
[1] (26345) Running ./myspin 3 &
[2] (26347) Running ./myspin 4 &
tsh> fg %1
Job [1] (26345) stopped by signal 20
tsh> jobs
[1] (26345) Stopped ./myspin 3 &
[2] (26347) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (26345) ./myspin 3 &
tsh> jobs
[1] (26345) Running ./myspin 3 &
[2] (26347) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

[test16]

./tsh

```
[kihyun@programming2 shelllab]$ 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] (28160) stopped by signal 20
tsh> jobs
[1] (28160) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (28205) terminated by signal 2
```

./tshref

```
./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] (26359) stopped by signal 20
tsh> jobs
[1] (26359) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (26362) terminated by signal 2
```