2016년 시스템 프로그래밍

-Shell Lab 2-

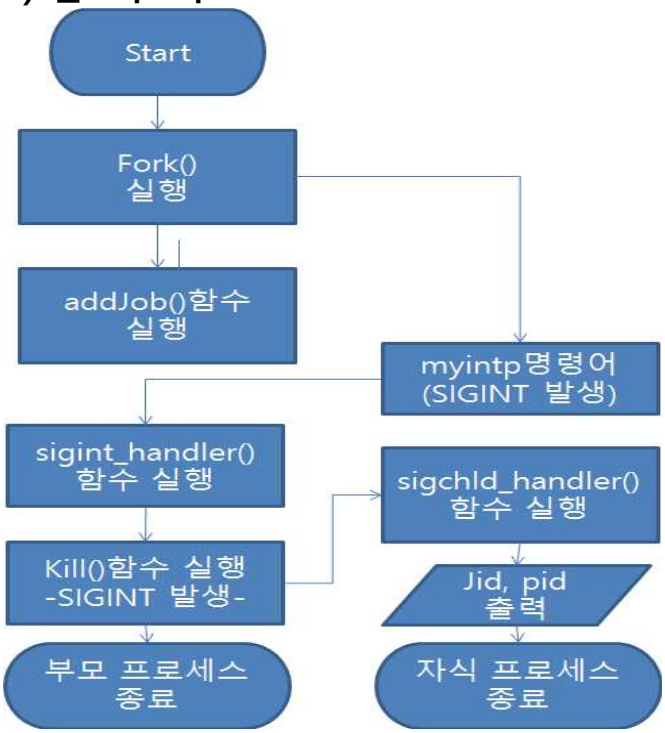
제출일자	2016.11.28.
이 름	정윤수
학 번	201302482
분 반	00

Trace 08

1) 정상 작동 모습

```
a201302482@host-192-168-0-5:~/shlab-handout$ ./sdriver -t 08 -s ./tsh
Running trace08.txt...
Success: The test and reference outputs for trace08.txt matched!
```

2) 플로우 차트



3) 해결 방법

trace08은 SIGINT를 발생시키는 명령어 ./myintp를 자식 프로세스의 execve()함수를 이용을 호출을 한다. 명령어가 호출이 되면 SIGINT가 발생하게 됨으로 자식 프로세스에서 발생하여 부모프로세스에 있는 handler에서 처리가 된다. 핸들러에서는 pid의 값을 함수를 이용을 하여 얻은 후 kill()함수를 이용을하여 SIGINT 시그널을 송신을 한다. 프로세스가 SIGINT 시그널로 인해 종료가 되면 자식 프로세스의 종료로 인해 부모 프로세스로 SIGCHLD시그널을 송신을 한다. 부모프로세스에서 sigchld_handler를 사용을 하여 자식 프로세스를 완전히 종료를 시킨다.

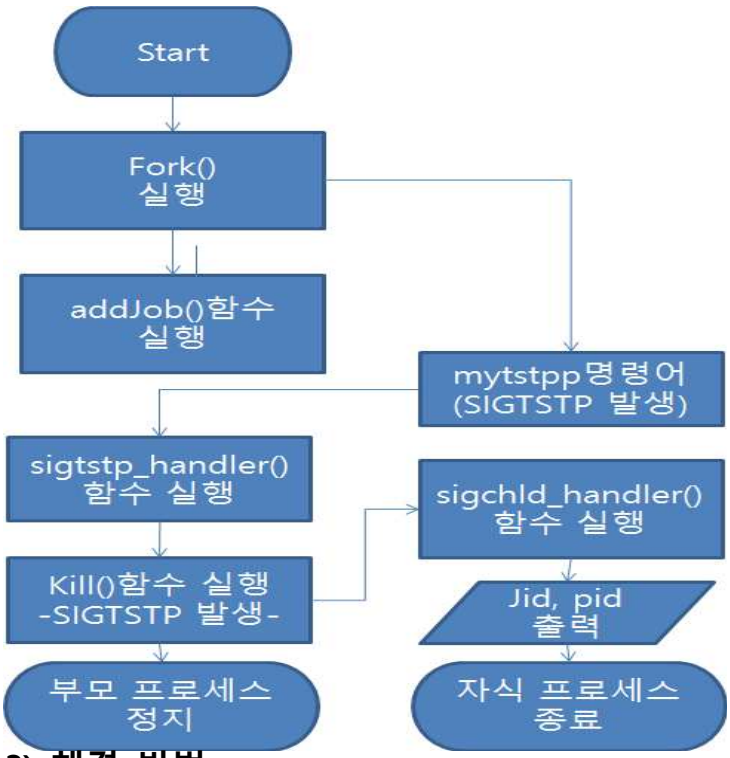
```
void sigint_handler(int sig)
{
    pid_t pid = fgpid(jobs);
    if(pid != 0){
        kill(-pid,sig);
    }
    return;
}
```

Trace 09

1) 정상 작동 모습

```
a201302482@host-192-168-0-5:~/shlab-handout$ ./sdriver -t 09 -s ./tsh
Running trace09.txt...
Success: The test and reference outputs for trace09.txt matched!
```

2) 플로우 차트



3) 해결 방법

Trace09은 mytstpp 명령어가 입력이 되어지면, SIGTSTP 시그널이 발생이 되어 sigtstp_hanlder 함수를 이용을 하여서 처리를 하는 문제이다. 명령어를 입력을 하면 시그널이 발생이되어서 부모 프로세스에 있는 sigtstp_handler()에서 처리를 하게 된다. hanlder에서는 phase08 때와 똑같이 pid값을 얻어 kill함수를 사용을 하여 SIGTSTP 시그널을 송신을 한다. 시그널을 송신을 한 후에는 자식 프로세스가 종료가 되어 SIGCHLD시그널이 발생이 된다. 부모 프로세스에서 sigchld_handler를 사용을하여 정지된 프로세스를 탐색을 한다. 프로세스를 찾으면 그 프로세스의 pid,jid, 등을 출력을 해주고 job에서의 state 또한 Stopped 상태로 변경을 해준다.

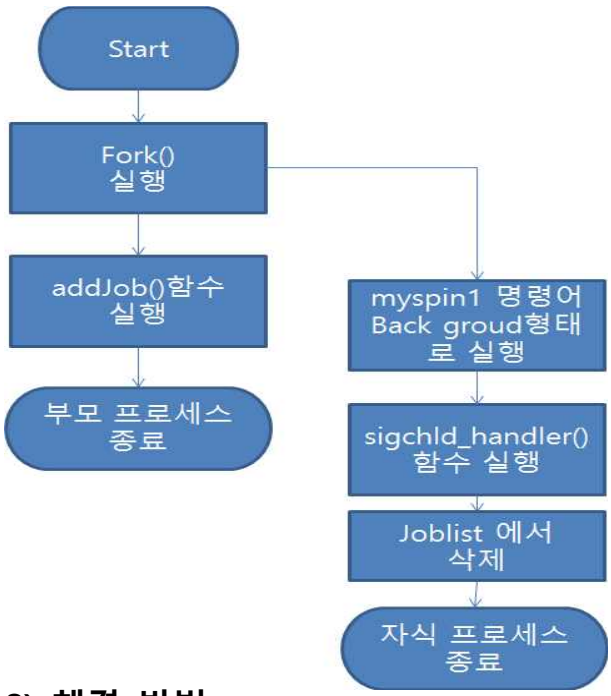
```
void sigtstp_handler(int sig)
{
    pid_t pid = fgpid(jobs);
    if(pid != 0)
    {
        kill(-pid,sig);
    }
    return ;
}
```

Trace 10

1) 정상 작동 모습

```
a201302482@host-192-168-0-5:~/shlab-handout$ ./sdriver -t 10 -s ./tsh
Running trace10.txt...
Success: The test and reference outputs for trace10.txt matched!
```

2) 플로우 차트



3) 해결 방법

phase10 에서는 Background 작업이 정상적으로 종료가 되면 sigchld_handler에서 사용을 했던 자원을 반환을 하는 것을 구현을 해야한다. 정상적으로 프로그램이 끝났는지 확인을 하기 위해서 status라는 변수를 하나 만들고 waitpid()의 2번째 인수로 포인터 값을 넘겨준다. waitpid()가 정상적으로 끝나게 되면 status 변수 안에는 자식프로세스가 어떤식으로 종료가 되었는지 알 수 있는 정보가 있다. WIFEXITED()는 정상적으로 종료가 되었을 때 참값을 반환을 해준다. 이것을 이용을하여 정상적으로 자식프로세스가 종료가 되었을 때 자원을 성공적으로 반환을 할 수 있다.

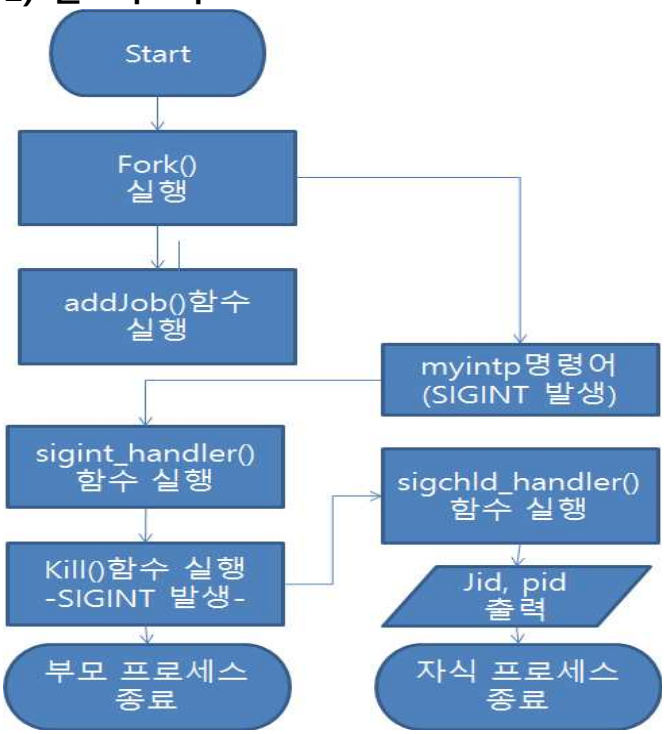
```c
void sigchld_handler(int sig)
{
    struct job_t *t;
    int status = -1;
    pid_t pid;
    while((pid=waitpid(-1,&status,WNOHANG|WUNTRACED))>0){
        t = getjobpid(jobs,pid);
        if(WIFEXITED(status))
        {
            deletejob(jobs,pid);

        }
        if(WIFSIGNALED(status))
        {
            printf("Job [%d] (%d) terminated by signal 2\n",t->jid,t->pid);
            deletejob(jobs,pid);
        }
        if(WIFSTOPPED(status))
        {
            printf("Job [%d] (%d) stopped by signal 20\n",t->jid,t->pid);
            t->state = 3;
        }
    }
    return;
}
```

Trace 11

1) 정상 작동 모습

```
a201302482@host-192-168-0-5:~/shlab-handout$ ./sdriver -t 11 -s ./tsh
Running trace11.txt...
Success: The test and reference outputs for trace11.txt matched!
```

2) 플로우 차트



3) 해결 방법

trace11은 자식의 pid로 SIGINT를 전달하였을 때 정상적으로 SIGINT에 대한 처리가 되도록 구현을 하는 것이다. 자식의 pid로 SIGINT를 전달을 하였다면 자식은 시그널로 인해 종료가 된 것이다. 그러므로 sigchld_hander 함수에 WIFSIGNALED()를 이용을하여 시그널로 종료가 된 것을 확인을 하여 구현을 한다.

Trace 12

1) 정상 작동 모습

```
a201302482@host-192-168-0-5:~/shlab-handout$ ./sdriver -t 12 -s ./tsh
Running trace12.txt...
Success: The test and reference outputs for trace12.txt matched!
```

2) 플로우 차트

3) 해결 방법

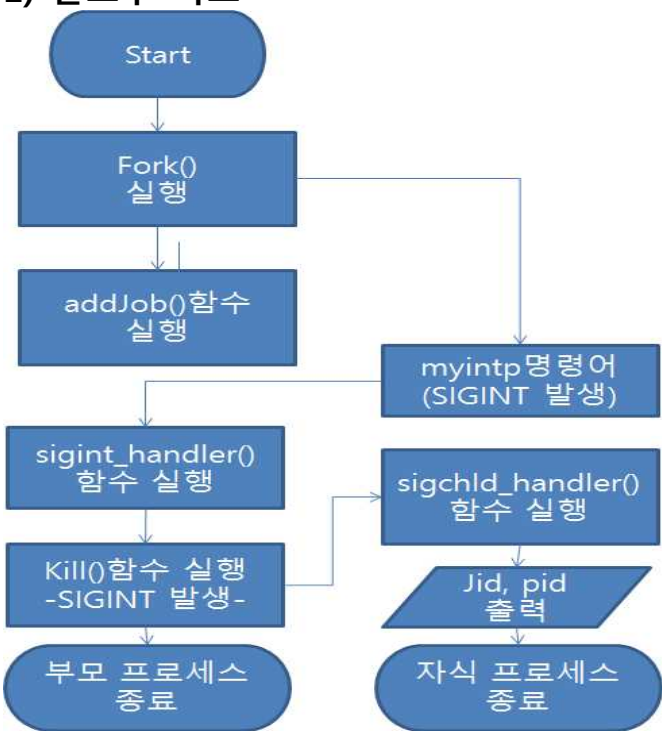
trace12는 trace11과 비슷한 방법으로 구현을 하면 문제를 해결을 할 수 있다. 자식 프로세스로 SIGTSTP가 전달이 되면 자식 프로세스는 정지가 된 상태로 대기를 한다. 이때 sigchld_handlder에 waitpid()함수의 option으로 WUNTRACED와 WNOHANG을 설정을 해주고 정지가 되었을 때 참값을 반환을 해주는 WIFSTOPED를 이용을 하여서 SIGTSTP에 대한 처리가 정상적으로 처리가 되게 구현을 할 수 있다.

Trace 13

1) 정상 작동 모습

```
a201302482@host-192-168-0-5:~/shlab-handout$ ./sdriver -t 13 -s ./tsh
Running trace13.txt...
Success: The test and reference outputs for trace13.txt matched!
```

2) 플로우 차트



3) 해결 방법

오직 foreground 작업에 대해서만 SIGINT가 처리가 되도록 할려면 fgpid()함수를 이용을히야 한다. fgpid()함수는 joblist중에 foreground로 실행이 되는 작업의 pid를 반환을 해주는 함수이다. 이것을 이용을하여 오직 foreground작업에 대해서만 SIGINT가 처리되도록 한다.

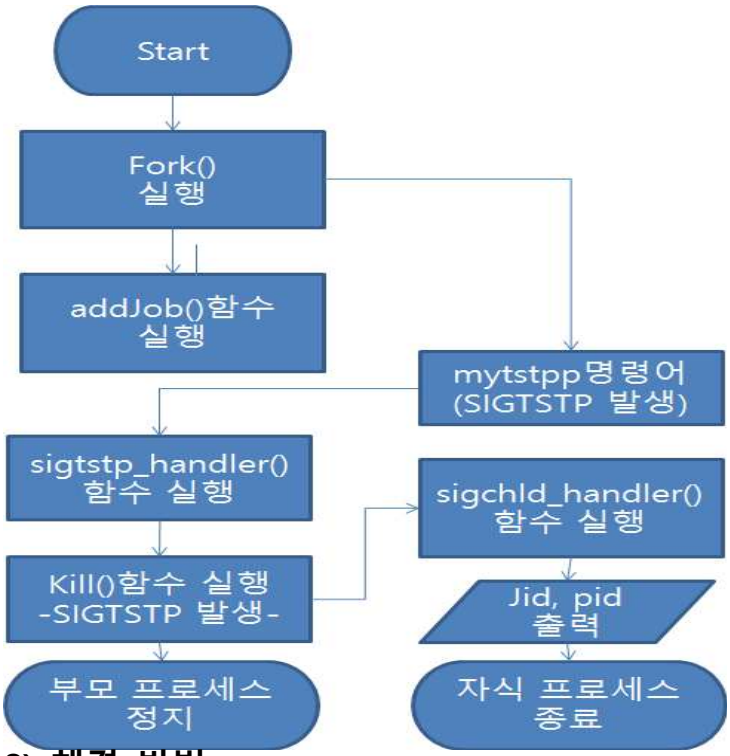
```
void sigint_handler(int sig)
{
    pid_t pid = fgpid(jobs);
    if(pid != 0){
        kill(-pid,sig);
    }
    return;
}
```

Trace 14

1) 정상 작동 모습

```
a201302482@host-192-168-0-5:~/shlab-handout$ ./sdriver -t 14 -s ./tsh
Running trace14.txt...
Success: The test and reference outputs for trace14.txt matched!
```

2) 플로우 차트



3) 해결 방법

Trace14은 Trace13과 똑같은 방식으로 구현을 하면 된다. fgpid()함수를 이용을 하여서 foreground로 실행이 되어지는 프로세스의 pid를 얻는다. 그 후 얻은 pid를 이용을 하여 foreground 작업에 대해서만 SIGTSTP가 처리가 되게 만든다.

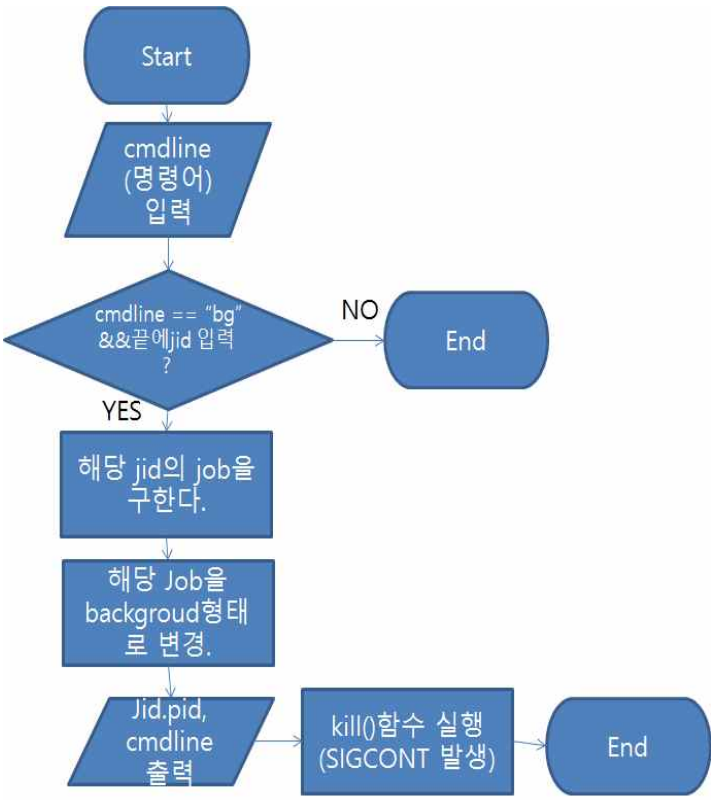
```
void sigtstp_handler(int sig)
{
    pid_t pid = fgpid(jobs);
    if(pid != 0)
    {
        kill(-pid,sig);
    }
    return ;
}
```

Trace 15 - 16

1) 정상 작동 모습

```
a201302482@host-192-168-0-5:~/shlab-handout$ ./sdriver -t 15 -s ./tsh
Running trace15.txt...
Success: The test and reference outputs for trace15.txt matched!
```

2) 플로우 차트



3) 해결 방법

Trace15는 built_in 명령어 bg를 구현을 하는 문제이다. cmdline으로 "bg"와 % jid를 입력을 하면 해당 jid의 job을 getjobjid()함수를 이용을하여 찾고 찾은 job의 상태를 background 상태로 만들어준다. 그 후 jid,pid,cmdline을 출력을 해주고 정지 되어있는 프로세스를 다시 시작을 시켜주는 SIGCONT를 발생을 시켜 재실행을 시켜준다.

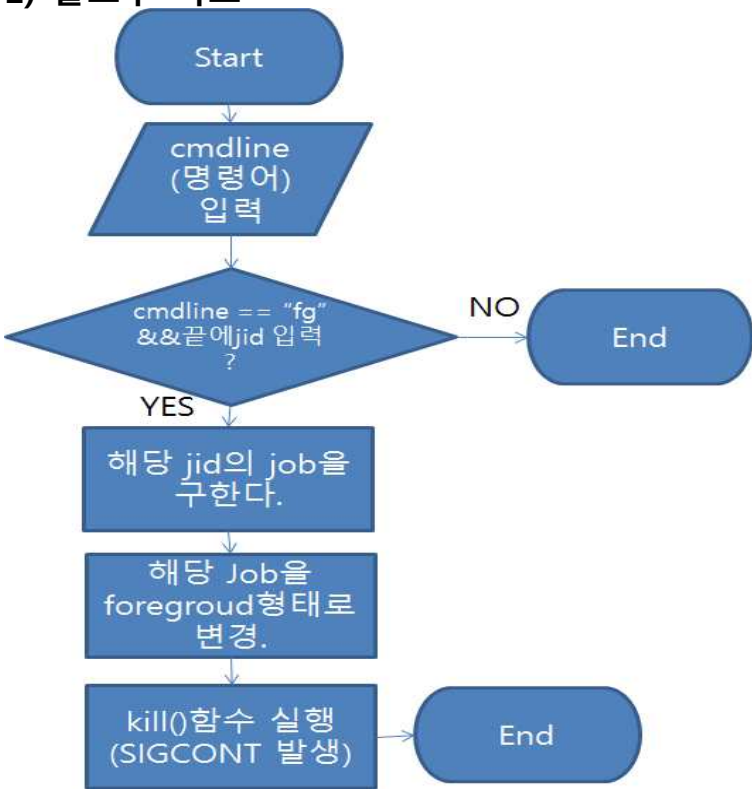
```
if(!strcmp(cmd,"bg")){
    givenjob = getjobjid(jobs,atoi(&argv[1][1]));
    givenjob->state = BG;
    printf("[%d] (%d) %s",givenjob->jid,givenjob->pid,givenjob->cmdline);
    kill(-givenjob->pid,SIGCONT);

    return 1;
}
```

Trace 17 - 18

1) 정상 작동 모습

```
a201302482@host-192-168-0-5:~/shlab-handout$ ./sdriver -t 16 -s ./tsh
Running trace16.txt...
Success: The test and reference outputs for trace16.txt matched!
```

2) 플로우 차트



3) 해결 방법

phase16은 built_in 명령어 fg를 구현을 하는 문제이다.. cmdline으로 "fg"와 % jid를 입력을 하면 해당 jid의 job을 getjobjid()함수를 이용을하여 찾고 찾은 job의 상태를 foreground 상태로 만들어준다. 그 후 kill()함수를 이용을하여서 찾은 job의 pid로 SIGCONT 시그널을 송신을 하여 다시 실행을 시켜준다.

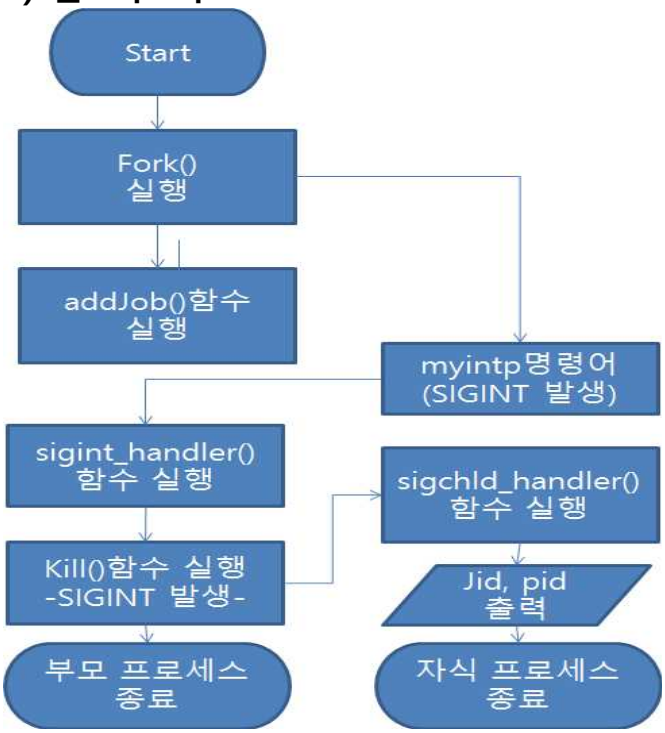
```
if(!strcmp(cmd,"fg")){
    givenjob = getjobjid(jobs,atoi(&argv[1][1]));
    givenjob->state = FG;
    kill(-givenjob->pid,SIGCONT);
    return 1;
}
```

Trace 19

1) 정상 작동 모습

```
a201302482@host-192-168-0-5:~/shlab-handout$ ./sdriver -t 19 -s ./tsh
Running trace19.txt...
Success: The test and reference outputs for trace19.txt matched!
```

2) 플로우 차트



3) 해결 방법

trace19는 모든 프로세스를 foreground 프로세스 그룹으로 설정을 하여 그룹에 SIGINT시그널을 보내는 것을 구현을 해야한다. 일단 모든 프로세스를 foreground프로세스 그룹으로 설정을 하기 위해서는 setpgid()함수를 이용을 한다. fork()를 수행을 하고 자식 프로세스가 수행이 될 때 setpgid(0,0)을 실행해 줌으로써 프로세스 그룹 아이디를 설정을 하고 자신이 속한 모든 그룹에 SIGINT를 보내기 위해서는 kill()함수를 사용을 할 때 pid 앞에 '-'을 붙여주면 프로세스 그룹에 SIGINT시그널을 보낼수 있다.

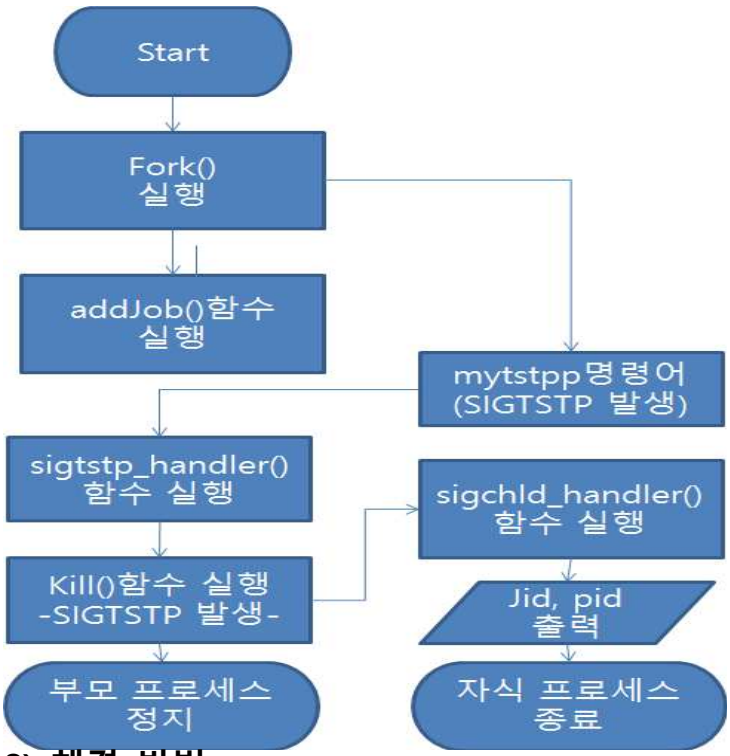
```
void sigint_handler(int sig)
{
    pid_t pid = fgpid(jobs);
    if(pid != 0){
        kill(-pid,sig);
    }
    return;
}
```

Trace 20

1) 정상 작동 모습

```
a201302482@host-192-168-0-5:~/shlab-handout$ ./sdriver -t 20 -s ./tsh
Running trace20.txt...
Success: The test and reference outputs for trace20.txt matched!
```

2) 플로우 차트



3) 해결 방법

trace20은 trace19와 비슷한 방식으로 구현을 하면 해결을 할 수 있다. trace20은 모든 프로세스를 foreground 프로세스 그룹으로 설정을 하여 그룹에 SIGTSTP시그널을 보내는 것을 구현을 해야한다. 일단 모든 프로세스를 foreground프로세스 그룹으로 설정을 하기 위해서는 setpgid()함수를 이용을 한다.eval에서 자식프로세스를 만들기 위해 fork()를 수행을 하고 자식 프로세스가 수행이 될 때 setpgid(0,0)을 실행해 줌으로써 프로세스 그룹 아이디를 설정을 한다. 자신이 속한 모든 그룹에 SIGTSTP를 보내기 위해서는 kill()함수를 사용을 할 때 pid 앞에 '-'을 붙여주면 프로세스 그룹에 SIGITSTP시그널을 보낼수 있다.

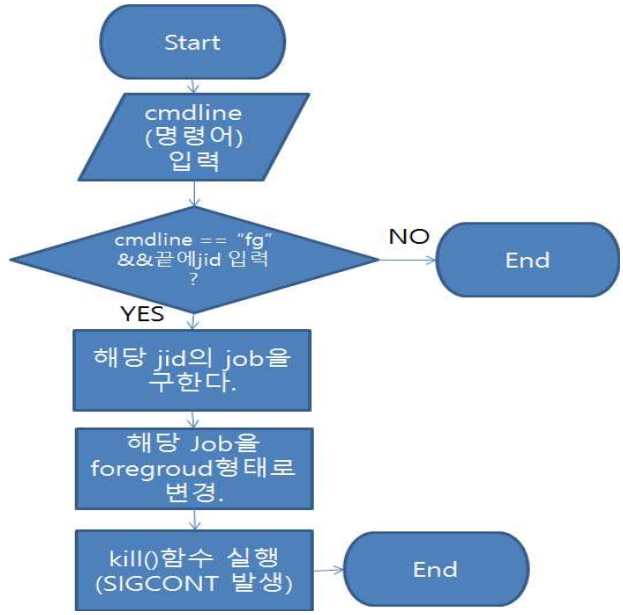
```
void sigtstp_handler(int sig)
{
    pid_t pid = fgpid(jobs);
    if(pid != 0)
    {
        kill(-pid,sig);
    }
    return ;
}
```

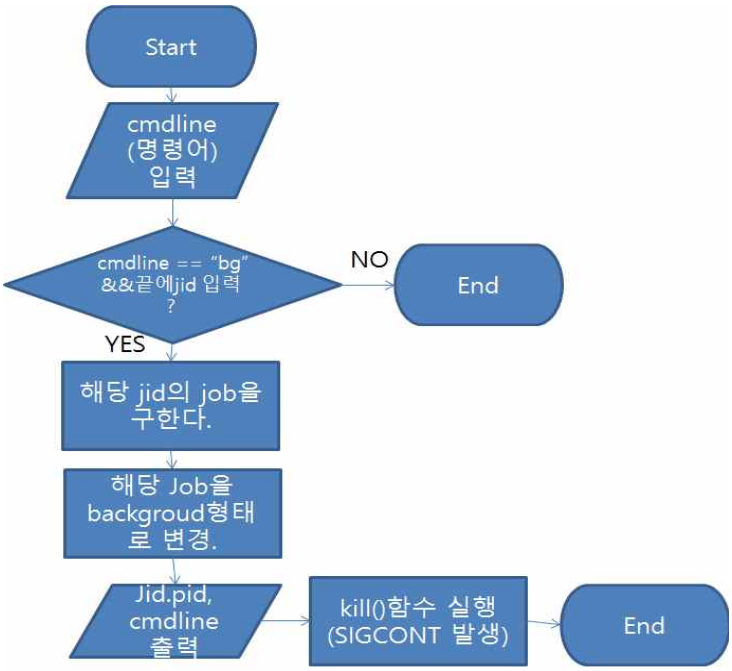
Trace 21

1) 정상 작동 모습

```
a201302482@host-192-168-0-5:~/shlab-handout$ ./sdriver -t 21 -s ./tsh
Running trace21.txt...
Success: The test and reference outputs for trace21.txt matched!
```

2) 플로우 차트





3) 해결 방법

trace21은 built_in 명령어 bg와 fg을 실행을 하였을 때 , 매개변수로 보낸 jid의 Job의 프로세스 그룹을 모두 재시작을 해주게 동작을 하게 만들어야 한다. 이것을 구현을 하기 위해서는 jid에 해당하는 job을 찾은후 원하는 상태로 바꾸어 준 후 kill()함수를 실행을 할 때 pid의 값을 음수로 넣어주면 해당 프로세스의 그룹에 SIGCONT 시그널을 송신을 할 수 있다.

```
if(!strcmp(cmd,"bg")){
    givenjob = getjobjid(jobs,atoi(&argv[1][1]));
    givenjob->state = BG;
    printf("[%d] (%d) %s",givenjob->jid,givenjob->pid,givenjob->cmdline);
    kill(-givenjob->pid,SIGCONT);

    return 1;
}
if(!strcmp(cmd,"fg")){
    givenjob = getjobjid(jobs,atoi(&argv[1][1]));
    givenjob->state = FG;
    kill(-givenjob->pid,SIGCONT);
    return 1;
}
```