2018 시스템 프로그래밍
- Lab 08 -

제출일자	2018.11.20
분 반	01
이 름	이규봉
학 번	201502085

<Shell Lab 2주차 – tsh 8,9,10,11,12>

tsh 8

=> SIGINT 시그널이 발생되었을 때, Foreground 작업을 종료하면 됩니다. 즉 Foreground 프로세스들에게 sinint_handler의 인자로 들어온 sig를 전달해주고, 이 핸들러를 설치해주면 됩니다.

```
/* These are the ones you will need to implement */
Signal(SIGINT,  sigint_handler);   /* ctrl-c */
Signal(SIGTSTP, sigtstp_handler);  /* ctrl-z */
Signal(SIGCHLD, sigchld_handler);  /* Terminated or stopped child */
Signal(SIGTTIN, SIG_IGN);
Signal(SIGTTOU, SIG_IGN);
```

그런데, tsh.c의 경우 이미 main에 핸들러 설치 코드가 있으므로, 핸들러 코드만 작성해주면 됩니다.

foreground 프로세스의 pid_t를 얻기 위해 아래 함수를 사용했습니다.

```
/* fgpid - Return PID of current foreground job, 0 if no such job */
pid_t fgpid(struct job_t *jobs) {
    int i;

    for (i = 0; i < MAXJOBS; i++)
        if (jobs[i].state == FG)
            return jobs[i].pid;
    return 0;
}
```

아래는 제가 작성한 sigint_handler입니다. 위 fgpid를 이용하여 jobs 중 FG 프로세스인 것을 찾고 pid가 0이 아니라면 sig (SIGINT)를 보냅니다. myintp 가 실행되면, 자식 프로세스에서 커널로, 커널에서 부모 프로세스로 SIGINT가 보내지고, sigint_handler가 이 시그널을 처리하게 됩니다. 즉, kill(pid,sig)가 실행되고, 자식 프로세스가 종료되면서 부모 프로세스에게 SIGCHLD가 전해지며, 부모 프로세스의 SIGCHLD Handler가 이를 처리하게 됩니다.

```
void sigint_handler(int sig)
{
    pid_t pid = fgpid(jobs);

    if (pid != 0) kill(pid,sig);

    return;
}
```

아래는 제가 작성해 본 Sigchld_handler입니다. while문의 조건에서 자식 프로세스가 종료되기를 기다리며, 자식프로세스가 시그널에 의해 종료되면 (WIFSIGNALED) 정해진 문장을 출력하고, deletejob 함수를 호출해 해당 job을 job_t 자료구조에서 지웁니다.

```
void sigchld_handler(int sig)
{

    int child_status = 0;
    pid_t pid;

    while((pid = waitpid(-1, &child_status, 0 )) > 0){

        if(WIFSIGNALED(child_status)) {
            printf("Job [%d] (%d) terminated by signal %d\n",pid2jid(pid),pid,WTERMSIG(child_status));
            deletejob(jobs,pid);
        }

        else {
            deletejob(jobs,pid);
        }
    }
    return;
}
```

아래는 eval 함수입니다.
mask를 sigemptyset으로 초기화 하고 sigaddset으로 블록하려는 시그널들을 mask에 등록했습니다. 그리고 sigprocmask로 부모프로세스에서 sigchld, sigint, sigtstp를 블록하고 자식 프로세스에선 언블록했습니다.
addjob이 끝난 후 부모프로세스에서도 mask를 언블록해 주었습니다.
(eval은 trace12까지 이 코드를 그대로 사용했습니다.)

```
void eval(char *cmdline)
{
    /*if(feof(stdin)){
        fflush(stdout);
        exit(0);
    }*/

    char *argv[MAXARGS];
    int bg = parseline(cmdline, argv);
    pid_t pid;
    sigset_t mask;
    // sigset_t prev_mask;

    sigemptyset(&mask);
    sigaddset(&mask, SIGCHLD);
    sigaddset(&mask, SIGINT);
    sigaddset(&mask, SIGTSTP);

    sigprocmask(SIG_BLOCK, &mask, NULL);

    if(!builtin_cmd(argv)){

        if((pid=fork())==0){
            sigprocmask(SIG_UNBLOCK, &mask , NULL);
            if((execve(argv[0],argv,environ) < 0)){
                printf("%s: Command not found\n", argv);
                exit(0);
            }
        }
        addjob(jobs,pid,(bg ==1? BG:FG), cmdline);
        sigprocmask(SIG_UNBLOCK, &mask, NULL);

        if(!bg) waitfg(pid, STDOUT_FILENO);
        else printf("(%d) (%d) %s",pid2jid(pid),pid,cmdline);
    }
    return;
}
```

tsh 9

tsh 9는 SIGTSTP 시그널이 발생했을 때 Foreground 프로세스를 Stop 상태로 만들면 됩니다. 그렇게 하기 위해 아래의 getjobpid 함수를 이용했습니다.

```
/* getjobpid  - Find a job (by PID) on the job list */
struct job_t *getjobpid(struct job_t *jobs, pid_t pid) {
    int i;

    if (pid < 1)
        return NULL;
    for (i = 0; i < MAXJOBS; i++)
        if (jobs[i].pid == pid)
            return &jobs[i];
    return NULL;
}
```

아래는 sigtstp 함수입니다. tsh 8과 마찬가지로 fgpid로 jobs를 받아 pid에 넘기고, pid가 0이 아닌 프로세스 (부모 프로세스) 에 kill 함수를 이용해 시그널을 보냅니다.

```
void sigtstp_handler(int sig)
{
    pid_t pid = fgpid(jobs);

    if(pid!=0) kill(pid,sig);

    return;
}
```

아래는 업데이트한, sigchld 핸들러입니다. WIFSTOPPED 조건을 추가하여, Stop된 자식 프로세스의 상태를 ST로 만들고 printf로 화면에 프로세스가 정지되었다는 stopped 문구를 출력하도록 했습니다.

이 때, waitpid의 option (세 번째 인자)를 디폴트 (0)로 놓으면, 자식프로세스가 중지된 상태에서 프로그램이 멈추게 됩니다. 따라서, 자식 프로세스가 종료되어 있지 않으면 바로 리턴하게 하는 WNOHANG과 멈춤상태와 종료상태에서 모두 pid를 리턴하게 해주는 옵션인 WUNTRACE를 함께 사용하였습니다.

옵션을 이렇게 바꾸어도 tsh08 역시 통과할 수 있었습니다.

```
void sigchld_handler(int sig)
{

    int child_status = 0;
    pid_t pid;

    while((pid = waitpid(-1, &child_status, WNOHANG | WUNTRACED  )) > 0){

        if(WIFSIGNALED(child_status)) {
            printf("Job [%d] (%d) terminated by signal %d\n",pid2jid(pid),pid,WTERMSIG(child_status));
            deletejob(jobs,pid);
        }

        else if(WIFSTOPPED(child_status)){
            getjobpid(jobs,pid)->state = ST;
            printf("Job [%d] (%d) stopped by signal %d\n",pid2jid(pid),pid,WSTOPSIG(child_status));
        }

        else {
            deletejob(jobs,pid);
        }
    }
    return;
}

/*
 * sigint_handler - The kernel sends a SIGINT to the shell whenver the
 *    user types ctrl-c at the keyboard.  Catch it and send it along
 *
```

tsh 10

tsh 10은 자식프로세스가 정상적으로 종료되었을 때의 경우를 처리해주면 됩니다. 따라서, 앞에서 추가한 코드들에 sigchild_handler에 WIFEXITED 일 때 deletejob(jobs,pid) 하는 분기를 추가해주기만 하면 정상적으로 통과할 수 있습니다.

```
void sigchld_handler(int sig)
{

    int child_status = 0;
    pid_t pid;

    while((pid = waitpid(-1, &child_status, WNOHANG | WUNTRACED  )) > 0){

        if(WIFSIGNALED(child_status)) {
            printf("Job [%d] (%d) terminated by signal %d\n",pid2jid(pid),pid,WTERMSIG(child_status));
            deletejob(jobs,pid);
        }

        else if(WIFSTOPPED(child_status)){
            getjobpid(jobs,pid)->state = ST;
            printf("Job [%d] (%d) stopped by signal %d\n",pid2jid(pid),pid,WSTOPSIG(child_status));
        }

        else if(WIFEXITED(child_status)){
            deletejob(jobs,pid);
        }
    }
    return;
}
```

tsh 11

trace11에서 사용하는 myints.c는 자식프로세스가 스스로에게 SIGINT를 kill 하는 구조로 되어 있습니다.
따라서, sigint 핸들러가 부모 프로세스에게 sigchld 시그널을 보내고 자식 프로세스가 종료되며,
부모 프로세스가 sigchld 시그널을 받게 됩니다. sigchld 핸들러에서 WIFSIGNALED(child_status)가 참이 되므로,
tsh08과 같은 분기로 통과하게 됩니다.

따라서, tsp08를 정상적으로 통과하면 11도 자동으로 통과됩니다.

```c
/*
 * myints.c - Sends a SIGINT to itself
 */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <stdlib.h>
#include "config.h"

void sigalrm_handler()
{
    exit(0);
}

int main()
{
    signal(SIGALRM, sigalrm_handler);
    alarm(JOB_TIMEOUT);

    if (kill(getpid(), SIGINT) < 0) {
    perror("kill");
    exit(1);
    }

    while(1);
    exit(0);
}
```

tsh 12

trace12에서 사용하는 mytstps.c 는 자식프로세스가 스스로에게 SIGTSTP를 kill 하는 구조로 되어 있습니다.
따라서, sigtpts 핸들러가 부모 프로세스에게 sigchld 시그널을 보내고 자식 프로세스가 중지되며,
부모 프로세스가 sigchld 시그널을 받게 됩니다. sigchld 핸들러에서 WIFSTOPPED(child_status)가 참이 되므로,
tsh09와 같은 분기로 통과하게 됩니다.

따라서, tsp09를 정상적으로 통과하면 12도 자동으로 통과됩니다.

```c
/*
 * mytstps.c - Sends a SIGTSTP to itself, terminates when restarted.
 */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <stdlib.h>

int main()
{
    if (kill(getpid(), SIGTSTP) < 0) {
    perror("kill");
    exit(1);
    }
    exit(0);
}
```

<sdriver Test>

※ 위가 tsh, 아래가 tshref의 sdriver 실행결과입니다.

tsh 08

```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -t 08 -s ./tsh -V
Running trace08.txt...
Success: The test and reference outputs for trace08.txt matched!
Test output:
#
# trace08.txt - Send fatal SIGINT to foreground job.
#
tsh> ./myintp
Job [1] (29292) terminated by signal 2
tsh> quit

Reference output:
#
# trace08.txt - Send fatal SIGINT to foreground job.
#
tsh> ./myintp
Job [1] (29300) terminated by signal 2
tsh> quit

b201502085@2018-sp:~/Shelllab/shlab-handout$
```

```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -t 08 -s ./tshref -V
Running trace08.txt...
Success: The test and reference outputs for trace08.txt matched!
Test output:
#
# trace08.txt - Send fatal SIGINT to foreground job.
#
tsh> ./myintp
Job [1] (14421) terminated by signal 2
tsh> quit

Reference output:
#
# trace08.txt - Send fatal SIGINT to foreground job.
#
tsh> ./myintp
Job [1] (14429) terminated by signal 2
tsh> quit

b201502085@2018-sp:~/Shelllab/shlab-handout$
```

tsh 09

```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -t 09 -s ./tsh -V
Running trace09.txt...
Success: The test and reference outputs for trace09.txt matched!
Test output:
#
# trace09.txt - Send SIGTSTP to foreground job.
#
tsh> ./mytstpp
Job [1] (29325) stopped by signal 20
tsh> jobs
(1) (29325) Stopped    ./mytstpp

Reference output:
#
# trace09.txt - Send SIGTSTP to foreground job.
#
tsh> ./mytstpp
Job [1] (29333) stopped by signal 20
tsh> jobs
(1) (29333) Stopped ./mytstpp
```

```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -t 09 -s ./tshref -V
Running trace09.txt...
Success: The test and reference outputs for trace09.txt matched!
Test output:
#
# trace09.txt - Send SIGTSTP to foreground job.
#
tsh> ./mytstpp
Job [1] (14599) stopped by signal 20
tsh> jobs
(1) (14599) Stopped ./mytstpp

Reference output:
#
# trace09.txt - Send SIGTSTP to foreground job.
#
tsh> ./mytstpp
Job [1] (14607) stopped by signal 20
tsh> jobs
(1) (14607) Stopped ./mytstpp
```

tsh 10

```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -t 10 -s ./tsh -V
Running trace10.txt...
Success: The test and reference outputs for trace10.txt matched!
Test output:
#
# trace10.txt - Send fatal SIGTERM (15) to a background job.
#
tsh> ./myspin1 5 &
(1) (29506) ./myspin1 5 &
tsh> /bin/kill myspin1
kill: failed to parse argument: 'myspin1'
tsh> quit

Reference output:
#
# trace10.txt - Send fatal SIGTERM (15) to a background job.
#
tsh> ./myspin1 5 &
(1) (29516) ./myspin1 5 &
tsh> /bin/kill myspin1
kill: failed to parse argument: 'myspin1'
tsh> quit

b201502085@2018-sp:~/Shelllab/shlab-handout$
```

```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -t 10 -s ./tshref -V
Running trace10.txt...
Success: The test and reference outputs for trace10.txt matched!
Test output:
#
# trace10.txt - Send fatal SIGTERM (15) to a background job.
#
tsh> ./myspin1 5 &
(1) (14633) ./myspin1 5 &
tsh> /bin/kill myspin1
kill: failed to parse argument: 'myspin1'
tsh> quit

Reference output:
#
# trace10.txt - Send fatal SIGTERM (15) to a background job.
#
tsh> ./myspin1 5 &
(1) (14643) ./myspin1 5 &
tsh> /bin/kill myspin1
kill: failed to parse argument: 'myspin1'
tsh> quit

b201502085@2018-sp:~/Shelllab/shlab-handout$
```

tsh 11

```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -t 11 -s ./tsh -V
Running trace11.txt...
Success: The test and reference outputs for trace11.txt matched!
Test output:
#
# trace11.txt - Child sends SIGINT to itself
#
tsh> ./myints
Job [1] (29688) terminated by signal 2
tsh> quit

Reference output:
#
# trace11.txt - Child sends SIGINT to itself
#
tsh> ./myints
Job [1] (29696) terminated by signal 2
tsh> quit

b201502085@2018-sp:~/Shelllab/shlab-handout$
```

```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -t 11 -s ./tshref -V
Running trace11.txt...
Success: The test and reference outputs for trace11.txt matched!
Test output:
#
# trace11.txt - Child sends SIGINT to itself
#
tsh> ./myints
Job [1] (14674) terminated by signal 2
tsh> quit

Reference output:
#
# trace11.txt - Child sends SIGINT to itself
#
tsh> ./myints
Job [1] (14682) terminated by signal 2
tsh> quit

b201502085@2018-sp:~/Shelllab/shlab-handout$
```

tsh 12

```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -t 12 -s ./tsh -V
Running trace12.txt...
Success: The test and reference outputs for trace12.txt matched!
Test output:
#
# trace12.txt - Child sends SIGTSTP to itself
#
tsh> ./mytstps
Job [1] (29723) stopped by signal 20
tsh> jobs
(1) (29723) Stopped    ./mytstps

Reference output:
#
# trace12.txt - Child sends SIGTSTP to itself
#
tsh> ./mytstps
Job [1] (29731) stopped by signal 20
tsh> jobs
(1) (29731) Stopped ./mytstps

b201502085@2018-sp:~/Shelllab/shlab-handout$
```

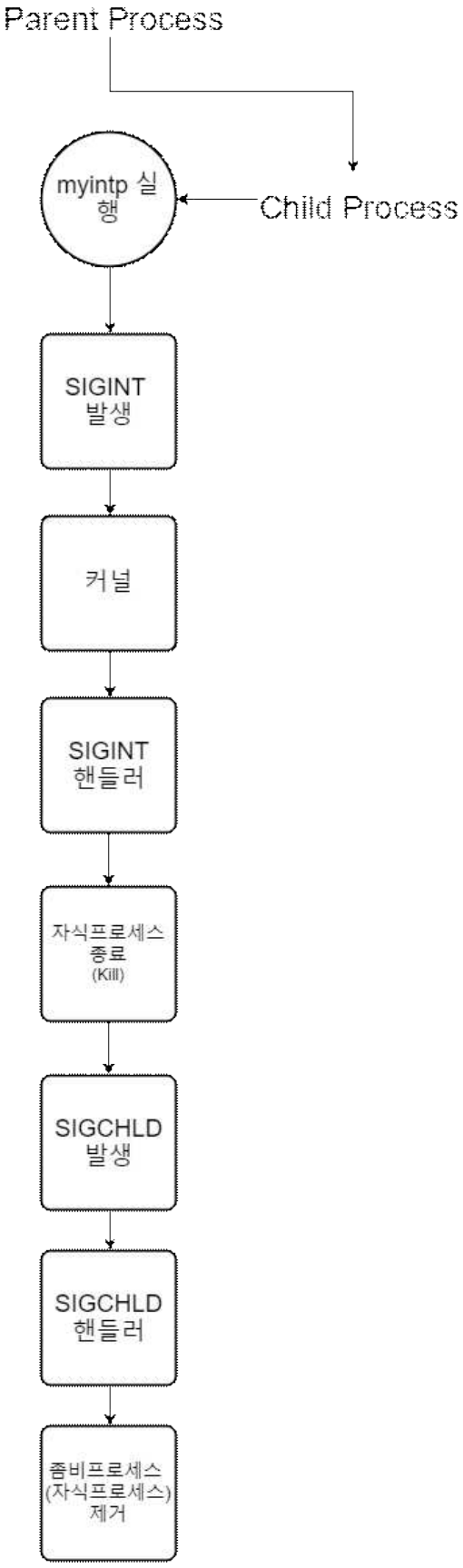
```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -t 12 -s ./tshref -V
Running trace12.txt...
Success: The test and reference outputs for trace12.txt matched!
Test output:
#
# trace12.txt - Child sends SIGTSTP to itself
#
tsh> ./mytstps
Job [1] (14849) stopped by signal 20
tsh> jobs
(1) (14849) Stopped ./mytstps

Reference output:
#
# trace12.txt - Child sends SIGTSTP to itself
#
tsh> ./mytstps
Job [1] (14857) stopped by signal 20
tsh> jobs
(1) (14857) Stopped ./mytstps

b201502085@2018-sp:~/Shelllab/shlab-handout$
```

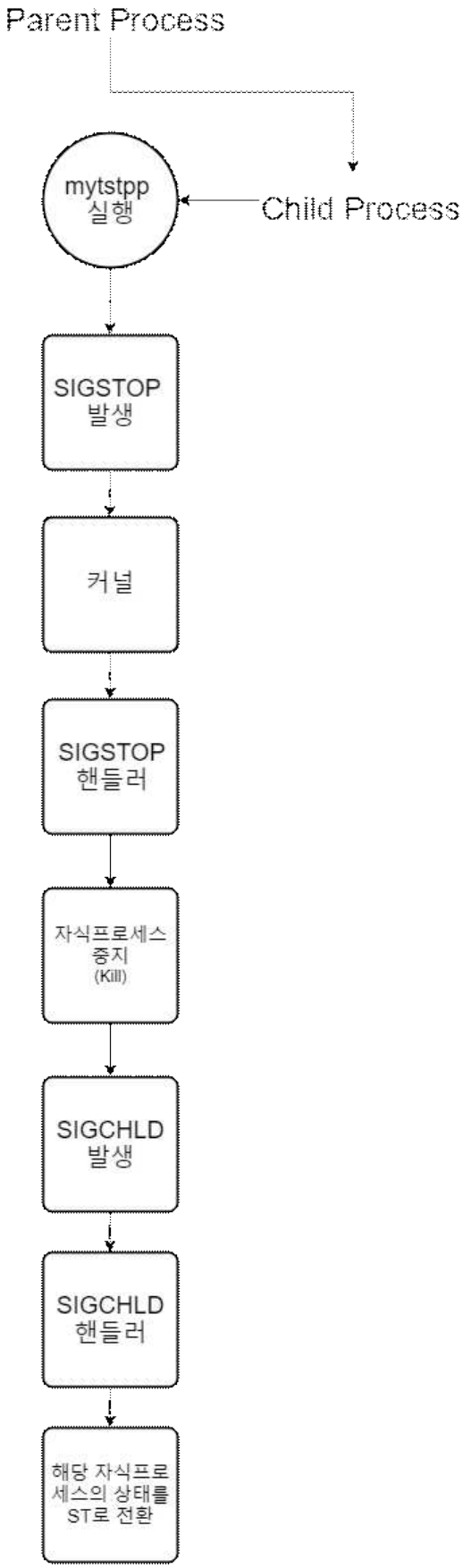
정상적으로 tsh08부터 tsh12까지 통과할 수 있었습니다.

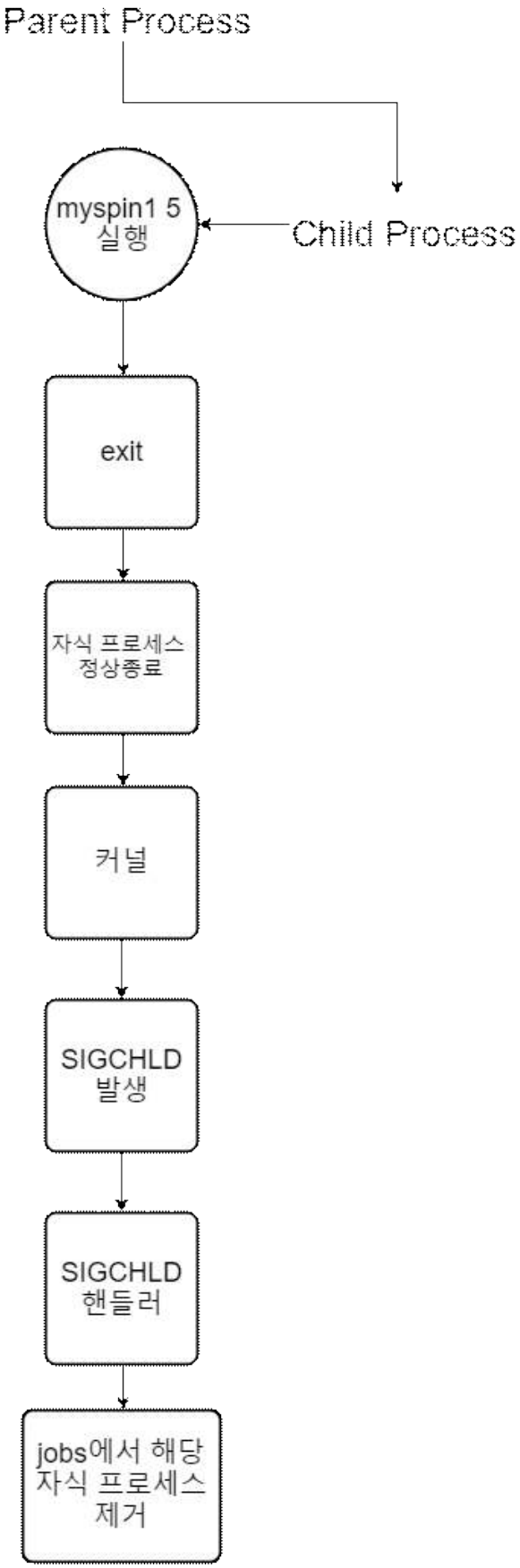
<Flow chart>

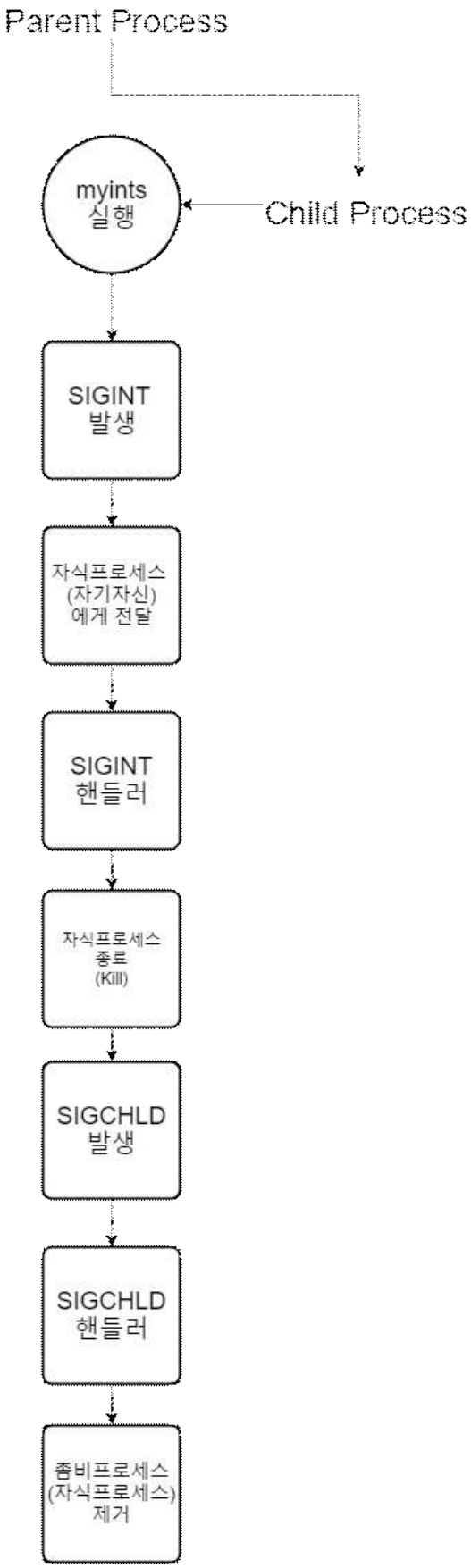
제가 이해한 프로그램의 흐름을, 프로그램 실행에서부터 정리해보았습니다. 제가 이해한 흐름을 그대로 적었기 때문에 틀린 부분이 있을 수 있습니다.

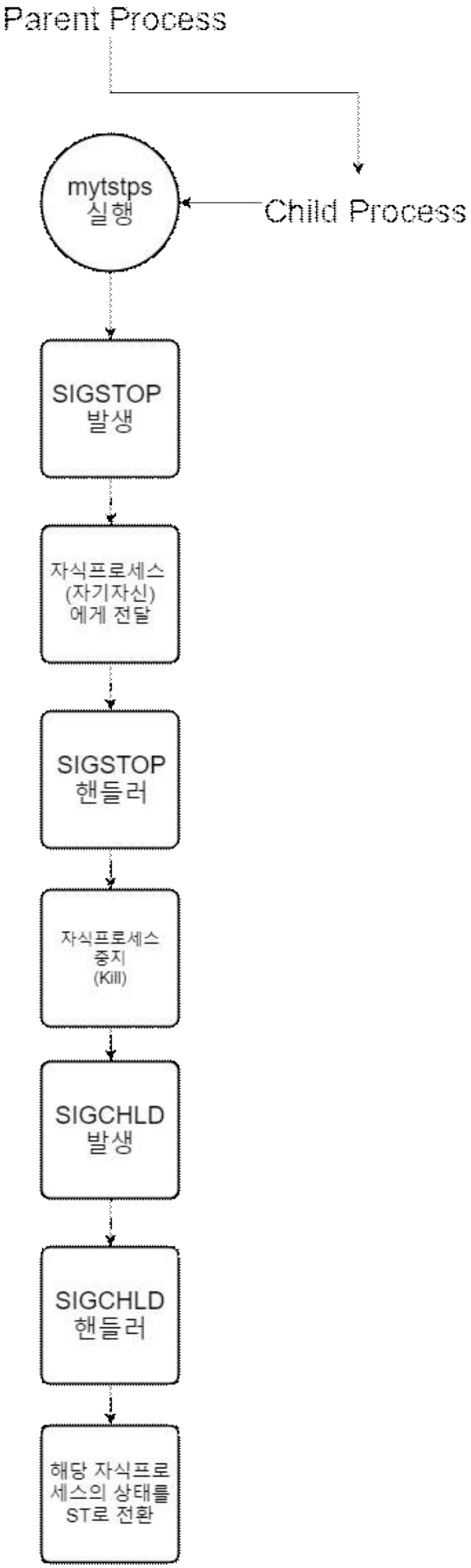
tsh 8




Parent Process
mytstpp
실행
Child Process
SIGSTOP
발생
커널
SIGSTOP
핸들러
자식프로세스
중지
(Kill)
SIGCHLD
발생
SIGCHLD
핸들러
해당 자식프로
세스의 상태를
ST로 전환


Parent Process
myspin1 5
실행
Child Process
exit
자식 프로세스
정상종료
커널
SIGCHLD
발생
SIGCHLD
핸들러
jobs에서 해당
자식 프로세스
제거


Parent Process
myints
실행
Child Process
SIGINT
발생
자식프로세스
(자기자신)
에게 전달
SIGINT
핸들러
자식프로세스
종료
(Kill)
SIGCHLD
발생
SIGCHLD
핸들러
좀비프로세스
(자식프로세스)
제거


Parent Process
mytstps
실행
Child Process
SIGSTOP
발생
자식프로세스
(자기자신)
에게 전달
SIGSTOP
핸들러
자식프로세스
중지
(Kill)
SIGCHLD
발생
SIGCHLD
핸들러
해당 자식프로
세스의 상태를
ST로 전환