

2021 시스템 프로그래밍

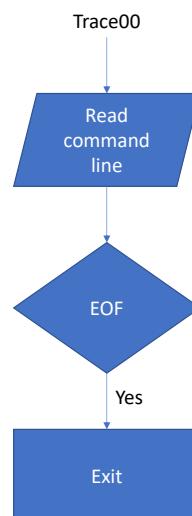
- Lab 03 -

제출일자	2020.11.21
분 반	01
이 름	조해창
학 번	201702897

Trace 00

```
b201702897@eslab-server:~/shlab-handout$ ./sdriver -V -t 00 -s ./tsh
Running trace00.txt...
Success: The test and reference outputs for trace00.txt matched!
Test output:
#
# trace00.txt - Properly terminate on EOF.
#
Reference output:
#
# trace00.txt - Properly terminate on EOF.
#
b201702897@eslab-server:~/shlab-handout$
```

Trace00 플로우 차트



Trace00 해결 방법 설명

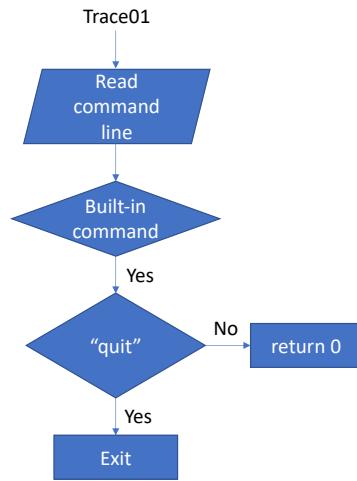
main 함수의 read while loop에서 아래 조건 문을 통해서 Ctrl + D 즉, EOF이 입력되면 shell이 종료되도록 구현되어있다.

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

Trace 01

```
b201702897@eslab-server:~/shlab-handout$ ./sdriver -V -t 01 -s ./tsh
Running trace01.txt...
Success: The test and reference outputs for trace01.txt matched!
Test output:
#
# trace01.txt - Process builtin quit command.
#
Reference output:
#
# trace01.txt - Process builtin quit command.
#
b201702897@eslab-server:~/shlab-handout$
```

Trace01 플로우 차트



Trace01 해결 방법 설명

eval 함수의 인자인 cmdline이 파싱된 값을 argv에 저장하여 builtin_cmd() 함수의 인자로 전달한다. 이때 만약 command line으로 A B CD 를 입력으로 받았다면 argv는 argv[0][0] = A, argv[1][0] = B, argv[2][0] = C, argv[2][1] = D 와 같이 초기화 된다.
 builtin_cmd()에서는 아래 코드 187 line의 조건문으로 “quit” 이면 exit(0), 아니면 0을 반환한다.

```
169 */
170 void eval(char * cmdline)
171 {
172     char * argv[MAXARGS];// command 저장
173
174     // 명령어를 parsesline을 통해 분리
175     parsesline(cmdline, argv);
176
177     // parsing된 명령어를 전달
178     builtin_cmd(argv);
179
180     return;
181 }
182
183 int builtin_cmd(char ** argv)
184 {
185     char * cmd = argv[0];
186
187     if (!strcmp(cmd, "quit")) { // quit command
188         exit(0);
189     }
190
191     return 0; // not a bulitin command
192 }
193
```

Trace (02, 03, 04)

```
b201702897@eslab-server:~/shlab-handout$ ./sdriver -V -t 02 -s ./tsh
Running trace02.txt...
Success: The test and reference outputs for trace02.txt matched!
Test output:
#
# trace02.txt - Run a foreground job that prints an environment variable
#
# IMPORTANT: You must pass this trace before attempting any later
# traces. In order to synchronize with your child jobs, the driver
# relies on your shell properly setting the environment.
OSTYPE=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin

Reference output:
#
# trace02.txt - Run a foreground job that prints an environment variable
#
# IMPORTANT: You must pass this trace before attempting any later
# traces. In order to synchronize with your child jobs, the driver
# relies on your shell properly setting the environment.
OSTYPE=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin

b201702897@eslab-server:~/shlab-handout$
```

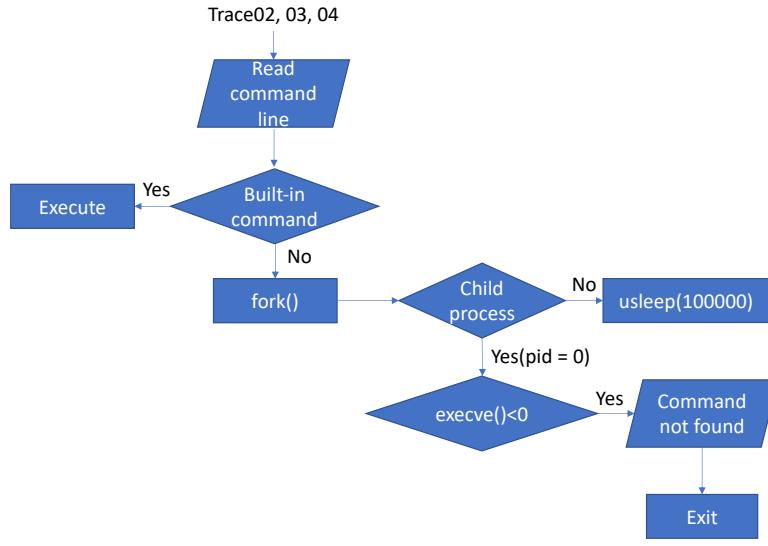
```
b201702897@eslab-server:~/shlab-handout$ ./sdriver -V -t 03 -s ./tsh
Running trace03.txt...
Success: The test and reference outputs for trace03.txt matched!
Test output:
#
# trace03.txt - Run a synchronizing foreground job without any arguments.
#
Reference output:
#
# trace03.txt - Run a synchronizing foreground job without any arguments.
#
b201702897@eslab-server:~/shlab-handout$ 
```

```
b201702897@eslab-server:~/shlab-handout$ ./sdriver -V -t 04 -s ./tsh
Running trace04.txt...
Success: The test and reference outputs for trace04.txt matched!
Test output:
#
# trace04.txt - Run a foreground job with arguments.
#
tsh> quit

Reference output:
#
# trace04.txt - Run a foreground job with arguments.
#
tsh> quit

b201702897@eslab-server:~/shlab-handout$ 
```

Trace (02, 03, 04) 플로우 차트



Trace (02, 03, 04) 해결 방법 설명

입력받은 command line 이 built-in 명령어가 아닌경우 fork()를 호출하여 자식 프로세스를 생성하고 pid를 초기화 해준다. 이때 pid가 0이면(자식 프로세스이면) execve()을 호출한다. execve 함수는 현재 프로세스를 첫번째 인자인 argv[0]으로 교체해준다. 따라서 built-in 명령어가 아닐경우 Foreground 작업 형태로 매개변수 있는 프로그램과 없는 프로그램을 실행할 수 있게된다.

```
tsh.c (~shlab-handout) - VIM
169 */
170 void eval(char * cmdline)
171 {
172     char * argv[MAXARGS];
173     pid_t pid;
174     parseline(cmdline, argv);
175
176     if (!builtin_cmd(argv)){
177         if ((pid = fork()) == 0){
178             if (execve(argv[0], argv, environ) < 0){
179                 printf("%s : Command not found\n\n", argv);
180                 exit(0);
181             }
182         }
183     }
184     usleep(100000);
185 }
186     return;
187 }
```

~/shlab-handout/tsh.c [utf-8,unix][c] 0,174/539 32%

Trace (05, 06)

```
b201702897@eslab-server:~/shlab-handout$ ./sdriver -V -t 05 -s ./tsh
Running trace05.txt...
Success: The test and reference outputs for trace05.txt matched!
Test output:
#
# trace05.txt - Run a background job.
#
tsh> ./myspin1 &
(1) (486639) ./myspin1 &
tsh> quit

Reference output:
#
# trace05.txt - Run a background job.
#
tsh> ./myspin1 &
(1) (486647) ./myspin1 &
tsh> quit

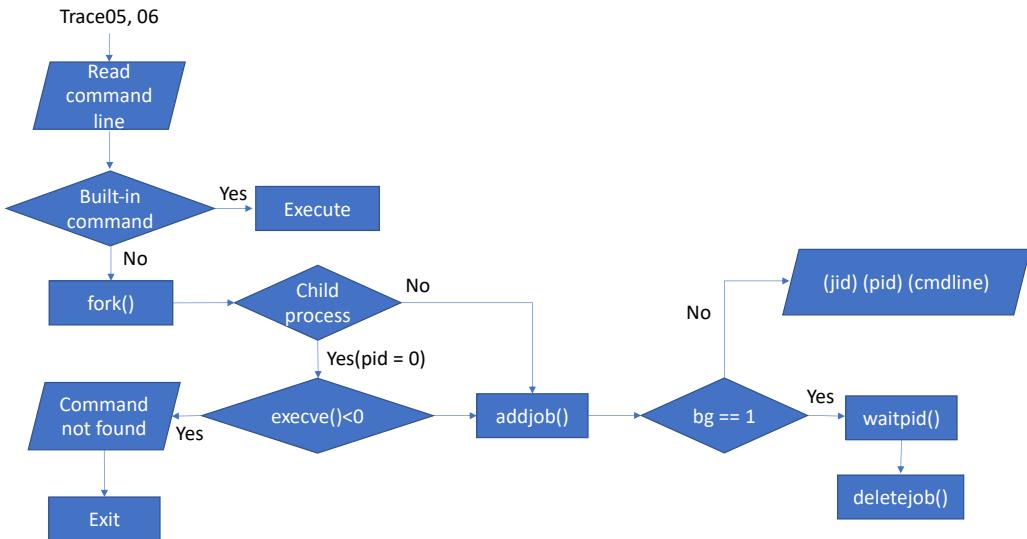
b201702897@eslab-server:~/shlab-handout$
```

```
b201702897@eslab-server:~/shlab-handout$ ./sdriver -V -t 06 -s ./tsh
Running trace06.txt...
Success: The test and reference outputs for trace06.txt matched!
Test output:
#
# trace06.txt - Run a foreground job and a background job.
#
tsh> ./myspin1 &
(1) (486671) ./myspin1 &
tsh> ./myspin2 1

Reference output:
#
# trace06.txt - Run a foreground job and a background job.
#
tsh> ./myspin1 &
(1) (486680) ./myspin1 &
tsh> ./myspin2 1

b201702897@eslab-server:~/shlab-handout$
```

Trace (05, 06) 플로우 차트



Trace (05, 06) 해결 방법 설명

176 line에서 parseline()은 입력받은 명령어 끝에 '&'가 포함되어 있으면 1을 아니면 0을 반환하므로 bg를 parseline()으로 초기화 한다.

fork()을 호출하고 pid가 0이 아니면(부모프로세스이면) addjob()을 호출하여 생성된 자식프로세스를 job list에 추가해준다.

addjob()은 인자로 job list, pid, state(BG, FG), cmdline을 인자로 받아서 인자로 받은 pid값을 가지는 자식프로세스를 job list에 추가해주는 함수이다.

185 line 조건문을 통해서 bg가 0이면(foreground process 이면) waitfg()를 호출한다. waitfg() 내부에서 waitpid()가 호출되어 자식프로세스가 종료될 때까지 기다리고, 종료되면 좀비를 제거하고 job list에서 자식 프로세스를 제거한다. bg가 1이면(background process 이면) 자식프로세스의 jid, pid 그리고 명령어를 출력한다.

```
tsh.c (~/shlab-handout) - VIM
170 void eval(char *cmdline)
171 {
172     char* argv[MAXARGS];
173     pid_t pid;
174     int bg;
175     bg = parseline(cmdline, argv);
176     if(!builtin_cmd(argv)) {
177         if((pid = fork()) == 0) {
178             if(execve(argv[0], argv, environ) < 0) {
179                 printf("%s : Command not found\n", argv);
180                 exit(0);
181             }
182         }
183         addjob(jobs, pid, (bg == 1 ? BG : FG), cmdline);
184         if(!bg){
185             waitfg(pid, 0);
186         }else{
187             printf("(%d) (%d) %s", pid2jid(pid), pid, cmdline);
188         }
189     }
190     return;
191 }
192 }
~/shlab-handout/tsh.c [utf-8,unix][c] 0,175/546 32%
```

```
tsh.c (~/shlab-handout) - VIM
204
205 void waitfg(pid_t pid, int output_fd)
206 {
207     pid = waitpid(pid, NULL, output_fd);
208     deletejob(jobs, pid);
209     return;
210 }
211
~/shlab-handout/tsh.c [utf-8,unix][c] 1,207/546 37%
```

Trace 07

```

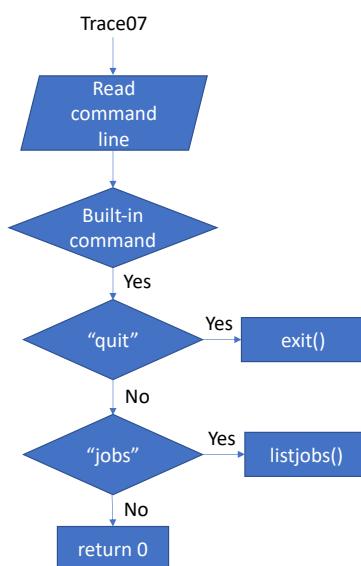
b201702897@eslab-server:~/shlab-handout$ ./sdriver -V -t 07 -s ./tsh
Running trace07.txt...
Success: The test and reference outputs for trace07.txt matched!
Test output:
#
# trace07.txt - Use the jobs builtin command.
#
tsh> ./myspin1 10 &
(1) (497551) ./myspin1 10 &
tsh> ./myspin2 10 &
(2) (497553) ./myspin2 10 &
tsh> jobs
(1) (497551) Running ./myspin1 10 &
(2) (497553) Running ./myspin2 10 &

Reference output:
#
# trace07.txt - Use the jobs builtin command.
#
tsh> ./myspin1 10 &
(1) (497561) ./myspin1 10 &
tsh> ./myspin2 10 &
(2) (497563) ./myspin2 10 &
tsh> jobs
(1) (497561) Running ./myspin1 10 &
(2) (497563) Running ./myspin2 10 &

b201702897@eslab-server:~/shlab-handout$ 

```

Trace07 플로우 차트



Trace07 해결 방법 설명

200 line의 조건문을 통해 입력받은 명령어가 “jobs”이면 listjob()를 호출한다.
listjob()은 인자로 job list, output_fd를 받는데 이때 output_fd는 write()의 인자로 들어가는 file descriptor로써 0(standard input), 1(standard output), 2(standard error) 값을 가진다.
447 line부터 job list를 순회하면서 pid가 0이 아니면 jid, pid를 buffer에 저장하고 write()를 호출한다. write()는 exceptional condition에서 -1을 리턴하기 때문에 리턴값이 0보다 작으면 에러 메세지를 출력한다.
456 line switch문에서 프로세스의 상태를 확인하여 BG(background), FG(foreground), ST(stopped)에 맞는 값을 buffer에 저장하여 write()를 호출한다.



```
193
194 int builtin_cmd(char **argv)
195 {
196     char *cmd = argv[0];
197
198     if(!strcmp(cmd, "quit")) {
199         exit(0);
200     }else if(!strcmp(cmd, "jobs")) {
201         listjobs(jobs, 0);
202         return 1;
203     }
204
205     return 0;
206 }
207
```

~/shlab-handout/tsh.c [utf-8,unix][c] 1,198/549 35%

```
tsh.c (~shlab-handout) - VIM  
 861  
442 void listjobs(struct job_t *jobs, int output_fd)  
443 {  
444     int i;  
445     char buf[MAXLINE];  
446  
447     for (i = 0; i < MAXJOBS; i++) {  
448         memset(buf, '\0', MAXLINE);  
449         if (jobs[i].pid != 0) {  
450             sprintf(buf, "(%d) (%d) ", jobs[i].jid, jobs[i].pid);  
451             if(write(output_fd, buf, strlen(buf)) < 0) {  
452                 fprintf(stderr, "Error writing to output file\n");  
453                 exit(1);  
454             }  
455             memset(buf, '\0', MAXLINE);  
456             switch (jobs[i].state) {  
457                 case BG:  
458                     sprintf(buf, "Running      ");  
459                     break;  
460                 case FG:  
461                     sprintf(buf, "Foreground   ");  
462                     break;  
463                 case ST:  
464                     sprintf(buf, "Stopped      ");  
465                     break;  
466                 default:  
467                     sprintf(buf, "listjobs: Internal error: job[%d].state=%d ",  
468                             i, jobs[i].state);  
469             }  
470             if(write(output_fd, buf, strlen(buf)) < 0) {  
471                 fprintf(stderr, "Error writing to output file\n");  
472                 exit(1);  
473             }  
474             memset(buf, '\0', MAXLINE);  
475             sprintf(buf, "%s", jobs[i].cmdline);  
476             if(write(output_fd, buf, strlen(buf)) < 0) {  
477                 fprintf(stderr, "Error writing to output file\n");  
478                 exit(1);  
479             }  
480         }  
481     }  
482     if(output_fd != STDOUT_FILENO)  
483         close(output_fd);  
~/shlab-handout/tsh.c [utf-8,unix][c] 1,447/549 86%
```

Trace 08

```

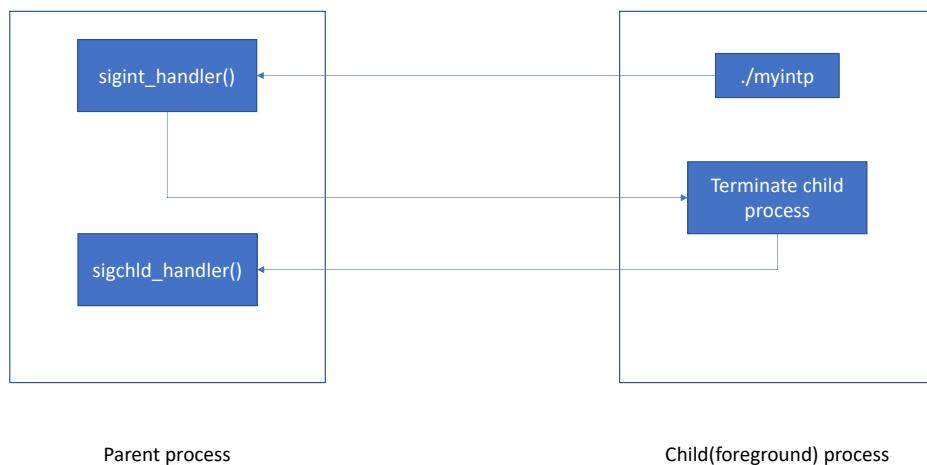
b201702897@eslab-server:~/shlab-handout$ ./sdriver -V -t 08 -s ./tsh
Running trace08.txt...
Success: The test and reference outputs for trace08.txt matched!
Test output:
#
# trace08.txt - Send fatal SIGINT to foreground job.
#
tsh> ./myintp
Job [1] (533193) terminated by signal 2
tsh> quit

Reference output:
#
# trace08.txt - Send fatal SIGINT to foreground job.
#
tsh> ./myintp
Job [1] (533201) terminated by signal 2
tsh> quit

b201702897@eslab-server:~/shlab-handout$ 

```

Trace08 플로우 차트



Trace08 해결 방법 설명

line 179 ~ 182에서 signal set의 mask를 0으로 초기화하여 SIGCHLD, SIGINT, SIGTSTP을 추가한다.

line 184에서 부모 프로세스에서 addjob()을 호출하기전에 자식 프로세스에서 SIGCHLD, SIGI T, SIGTSTP 시그널이 발생해서 delete()가 호출되는 race 현상을 방지하기 위해서 위 시그널

들을 block한다. 자식 프로세스를 실행하기전인 line 189, 부모 프로세스에서 addjob()을 호출한 뒤인 line 197에서 다시 prev를 이용하여 원래 mask로 복구시킨다(unblock한다.). line 199 ~ 204 에서 부모프로세스는 foreground job의 pid와 자식 프로세스의 id를 계속 비교 한다. 이때 foreground 작업이 종료되면 무한루프를 빠져나온다.

trace08에 사용되는 myintp.c의 line 23에서 kill()을 이용하여 부모프로세스에 SIGINT 신호를 보낸다. 이 신호는 sigint_handler()로 처리한다. 핸들러에서는 현재 foreground process(자식 프로세스)의 id로 pid를 초기화하고, kill()을 이용하여 pid에 SIGINT신호를 보내서 종료시킨다(좀비로 만든다.). 자식 프로세스가 좀비가 되면 부모 프로세스에게 SIGCHLD 신호를 보내고 이 신호는 sigchild_handler()로 처리한다.

line 249에서 시그널이 큐를 사용하지 않는 문제점을 해결하기위해서 무한루프를 돌면서 pid를 종료된 프로세스 id로 초기화한다. line 250 에서 만약 자식프로세스가 어떤 신호에 의해 종료되었다면 jid, pid 그리고 자식프로세스를 종료하도록한 신호를 출력하고 deletejob ()을 호출한다.

```
● ● ● tsh.c + (~shlab-handout) - VIM 7981
170 void eval(char *cmdline)
171 {
172     char* argv[MAXARGS];
173     pid_t pid;
174     int bg;
175     sigset_t mask, prev;
176
177     bg = parseline(cmdline, argv);
178
179     sigemptyset(&mask);
180     sigaddset(&mask, SIGCHLD);
181     sigaddset(&mask, SIGINT);
182     sigaddset(&mask, SIGTSTP);
183
184     sigprocmask(SIG_BLOCK, &mask, &prev); // Block signal
185
186     if(!builtin_cmd(argv)) {
187         // Child
188         if((pid = fork()) == 0) {
189             sigprocmask(SIG_SETMASK, &prev, NULL); // Unblock signal
190             if(execl(argv[0], argv, environ) < 0) {
191                 printf("%s : Command not found\n", argv);
192                 exit(0);
193             }
194         }
195         // Parent
196         addjob(jobs, pid, (bg == 1 ? BG : FG), cmdline);
197         sigprocmask(SIG_SETMASK, &prev, NULL); // Unblock signal
198         if(!bg){ // Foreground
199             while(1){
200                 if(pid != fgpid(jobs))
201                     break;
202                 else
203                     sleep(1);
204             }
205         }else{ // Background
206             printf("(%d) (%d) %s", pid2jid(pid), pid, cmdline);
207         }
208     }
209     return;
210 }
211
```

~shlab-handout/tsh.c [utf-8,unix][+][c] 23,205/585 31%

A screenshot of a terminal window titled "myintp.c (~/shlab-handout) - VIM". The code in the buffer is:

```
8 #include <sys/types.h>
9 #include <signal.h>
10 #include <stdlib.h>
11 #include "config.h"
12
13 void sigalrm_handler()
14 {
15     exit(0);
16 }
17
18 int main()
19 {
20     signal(SIGALRM, sigalrm_handler);
21     alarm(JOB_TIMEOUT);
22
23     if (kill(getppid(), SIGINT) < 0) {
24         perror("kill");
25         exit(1);
26     }
27
28     while(1);
29     exit(0);
30 }
```

The status bar at the bottom shows the file path as "~/shlab-handout/myintp.c [utf-8,unix][c]" and the current position as "0,27/30 Bot".

A screenshot of a terminal window titled "tsh.c + (~/shlab-handout) - VIM". The code in the buffer is:

```
268 void sigint_handler(int sig)
269 {
270     pid_t pid = fgpid(jobs);
271     if(pid != 0){
272         kill(pid, sig);
273     }
274     return;
275 }
```

The status bar at the bottom shows the file path as "~/shlab-handout/tsh.c [utf-8,unix][+][c]" and the current position as "1,271/585 46%".

```
tsh.c (~/shlab-handout) - VIM
243 */
244 void sigchld_handler(int sig)
245 {
246     int child_status = 0;
247     pid_t pid;
248     while((pid = waitpid(-1, &child_status, 0)) > 0){
249         if(WIFSIGNALED(child_status)) {
250             printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(pid), pid, WTERMSIG(child_status));
251             deletejob(jobs, pid);
252         }else{
253             deletejob(jobs, pid);
254         }
255     }
256 }
257 return;
258 }
259
260 /*
~/shlab-handout/tsh.c [utf-8,unix][c] 1,248/582 42%
```

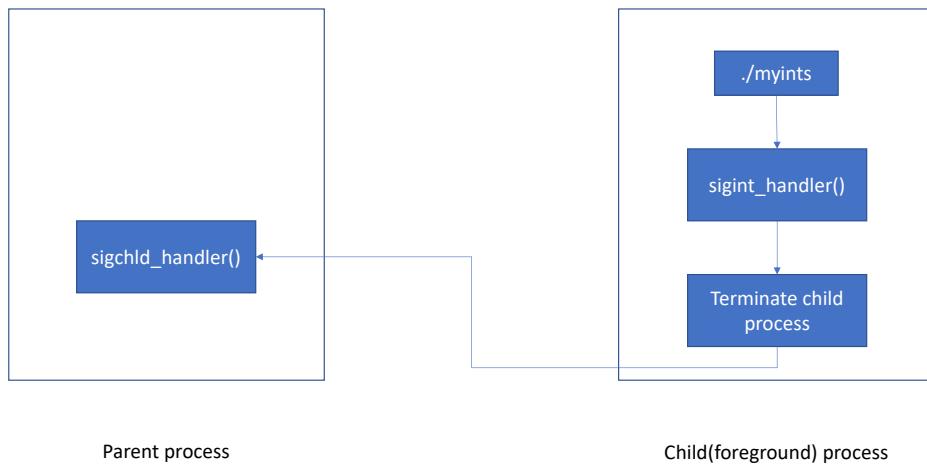
Trace 11

```
b201702897@eslab-server:~/shlab-handout$ ./sdriver -V -t 11 -s ./tsh
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] (547977) terminated by signal 2
tsh> quit

Reference output:
#
# trace11.txt - Child sends SIGINT to itself
#
tsh> ./myints
Job [1] (547985) terminated by signal 2
tsh> quit

b201702897@eslab-server:~/shlab-handout$
```

Trace11 플로우 차트



Trace11 해결 방법 설명

trace11에 사용되는 myints.c의 line 21에서 kill()을 이용하여 자신의 프로세스에 SIGINT 신호를 보낸다. 자식 프로세스는 부모 프로세스와 동일한 context를 가지기 때문에 다음의 메커니즘은 trace08과 동일하게 sigint_handler()와 sigchld_handler()가 순차적으로 실행된다.

```
4 #include <stdio.h>
5 #include <unistd.h>
6 #include <sys/types.h>
7 #include <signal.h>
8 #include <stdlib.h>
9 #include "config.h"
10
11 void sigalrm_handler()
12 {
13     exit(0);
14 }
15
16 int main()
17 {
18     signal(SIGALRM, sigalrm_handler);
19     alarm(JOB_TIMEOUT);
20
21     if (kill(getpid(), SIGINT) < 0) {
22         perror("kill");
23         exit(1);
24     }
25
26     while(1);
27     exit(0);
28 }
```