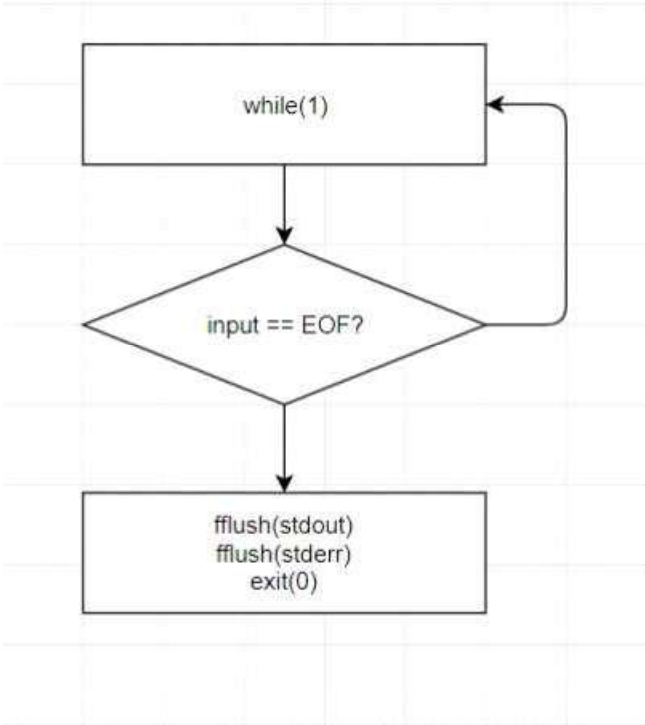
Trace 00

```
a201402439@2018-sp:~/shelllab/shlab-handout$ ./sdriver -t 00 -s ./tsh
Running trace00.txt...
Success: The test and reference outputs for trace00.txt matched!
a201402439@2018-sp:~/shelllab/shlab-handout$ ./sdriver -t 00 -s ./tshref
Running trace00.txt...
Success: The test and reference outputs for trace00.txt matched!
```

trace00 플로우 차트



trace00 해결 방법 설명

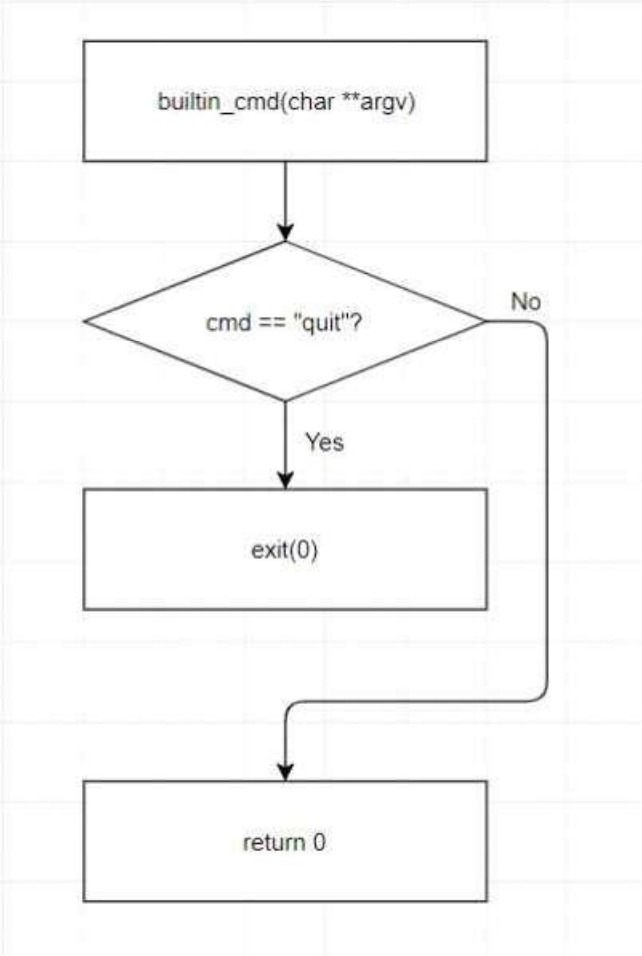
```
157         if (feof(stdin)) {
158             fflush(stdout);
159             fflush(stderr);
160             exit(0);
161         }
```

EOF가 입력이되면, 출력버퍼를 비우고 exit(0) 하도록 main()에 구현이 되어있다.

Trace 01

```
a2014024390@2018-sp:~/shelllab/shlab-handout$ vi tsh.c
a2014024390@2018-sp:~/shelllab/shlab-handout$ ./sdriver -t 01 -s ./tsh
Running trace01.txt...
Success: The test and reference outputs for trace01.txt matched!
a2014024390@2018-sp:~/shelllab/shlab-handout$ ./sdriver -t 01 -s ./tshref
Running trace01.txt...
Success: The test and reference outputs for trace01.txt matched!
```

trace01 플로우 차트



trace01 해결 방법 설명

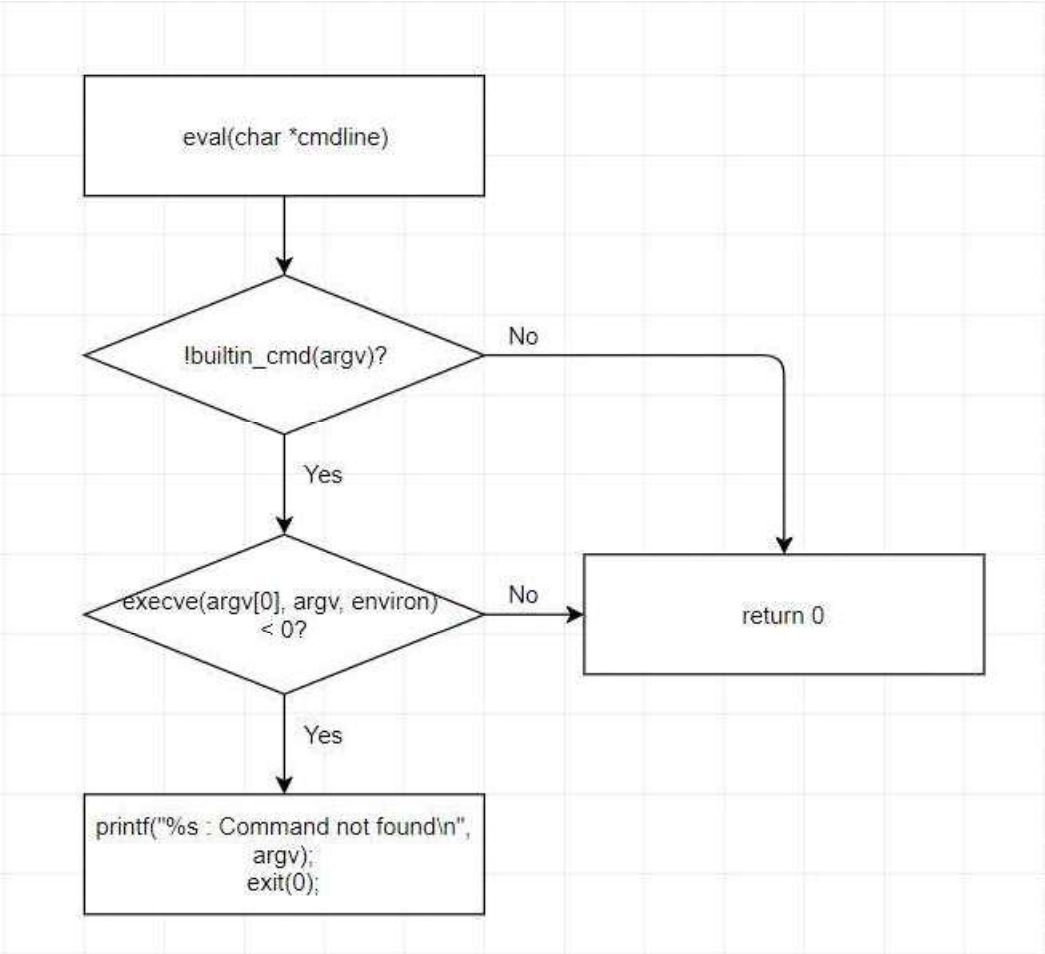
```
183 void eval(char *cmdline)
184 {
185     char *argv[MAXARGS];//command의 수
186     //명령어를 parseline 통해 분리
187     parseline(cmdline, argv);
188     //parsing은 명령어 인식
189     builtin_cmd(argv);
190
191     return;
192 }
193
194 int builtin_cmd(char **argv)
195 {
196     char *cmd = argv[0];
197
198     if(!strcmp(cmd, "quit")){//quit command
199         exit(0);
200     }
201
202     return 0;//not a builtin command
203 }
```

우선 eval함수 내에 입력받은 커맨드를 저장하는 문자형 배열 argv를 선언해준다.
그리고 입력받은 문자열을 parseline을 통해 분리하고, 파싱한 명령어를 builtin_cmd함수에 전달해준 다음 return 한다.
builtin_cmd 함수는 전달된 인자를 저장하는 char 포인터 cmd에 저장하고,
입력받은 문자열이 "quit"이면 exit(0)하고, 아니면 return 0;하고 종료한다.

Trace 02

```
a201402439@2018-sp:~/shelllab/shlab-handout$ ./sdriver -t 02 -s ./tsh
Running trace02.txt...
Success: The test and reference outputs for trace02.txt matched!
a201402439@2018-sp:~/shelllab/shlab-handout$ ./sdriver -t 02 -s ./tshref
Running trace02.txt...
Success: The test and reference outputs for trace02.txt matched!
```

trace02 플로우 차트



trace02 해결 방법 설명

```
183 void eval(char *cmdline)
184 {
185     char *argv[MAXARGS];//command가 들
186     pid_t pid;
187
188     //명령어를 parseline 통해 분리
189     parseline(cmdline, argv);
190     if(!builtin_cmd(argv)){
191         if((pid = fork()) == 0){
192             if(execve(argv[0], argv, environ) < 0){
193                 printf("%s : Command not found\n", argv);
194                 exit(0);
195             }
196         }
197     }
198     return;
199 }
200
201     //parsing된 명령어 실행
202     //builtin_cmd(argv);
203
204     //return;
205 }
206
207 int builtin_cmd(char **argv)
208 {
209     char *cmd = argv[0];
210
211     if(!strcmp(cmd, "quit")){//quit command
212         exit(0);
213     }
214
215     return 0;//not a builtin command
216 }
```

eval함수를 조금 수정했다.

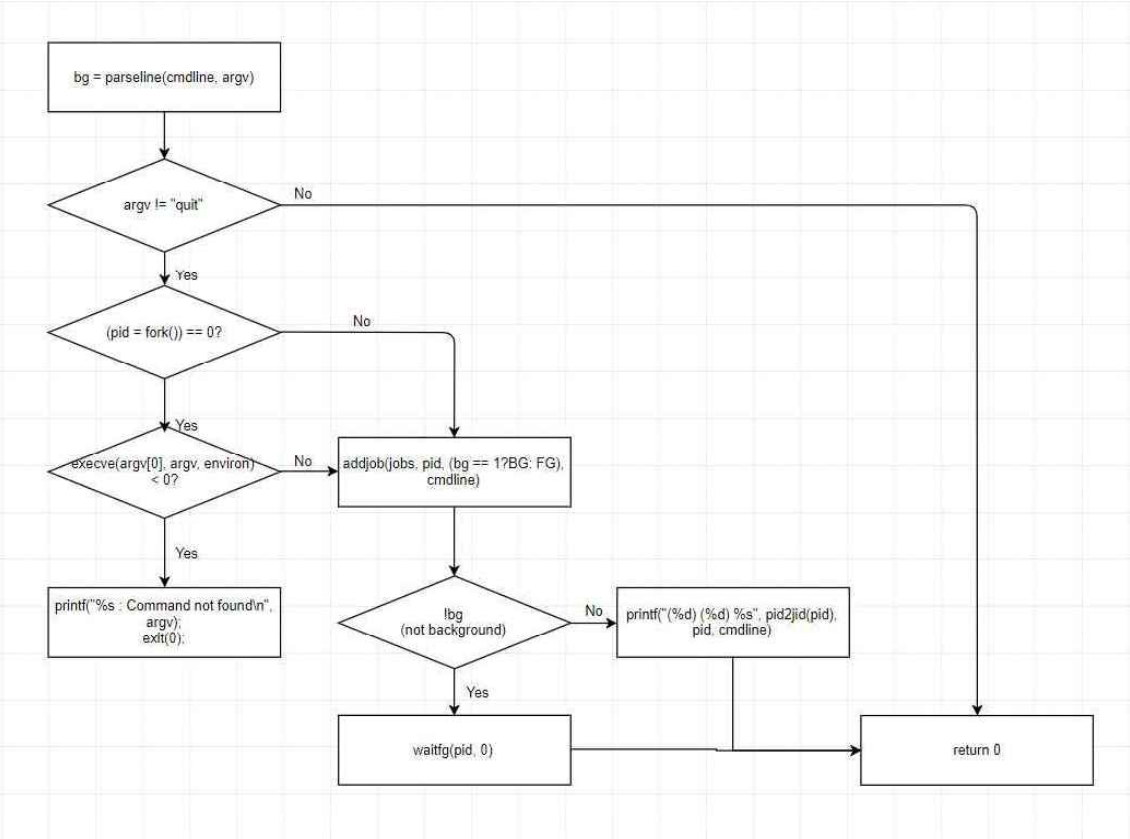
우선 프로세스id를 저장할 변수 pid를 선언하고, 입력받은 문자열을 파싱한 뒤, 만약 입력받은 문자열이 "quit"이 아니라면(quit이 아니면 builtin_cmd 함수에서 0을 리턴하므로), fork()를 한 뒤 리턴값을 pid에 저장하고 만약 pid가 0이면(fork()한 프로세스가 자식프로세스면) execve함수를 실행해서 현재 프로세스를 argv[0]으로 교체해준다. 만약 execve가 정상적으로 실행되지 않았으면(음수값 리턴) "[명령어] : Command not found"를 출력하고 exit(0)한다.

그 이외의 경우에는 return 0;으로 정상 종료한다.

Trace 05

```
a2014024390@2018-sp:~/shelllab/shlab-handout$ ./sdriver -t 05 -s ./tsh
Running trace05.txt...
Success: The test and reference outputs for trace05.txt matched!
```

trace05 플로우차트



trace 05 해결 방법

```
183 void eval(char *cmdline)
184 {
185     char *argv[MAXARGS]; //command저장
186     pid_t pid;
187     int bg;
188
189     //명령어를 parseline 통해 분석
190     bg = parseline(cmdline, argv);
191     if(!builtin_cmd(argv)){
192         if((pid = fork()) == 0){
193             if(execve(argv[0], argv, environ) < 0){
194                 printf("%s : Command not found\n", argv);
195                 exit(0);
196
197             }
198         }
199         addjob(jobs, pid, (bg == 1?BG: FG), cmdline);//cmdline을 job에 argv로
200         if(!bg){
201             //자식프로세스가 작업을 수행할 때까지 부모프로세스는 기다려야 하므로 wait 해준다
202             waitfg(pid, 0);
203         }
204         else{
205             //백그라운드 작업을 수행할 때 작업 내용 출력
206             printf("(%d) (%d) %s",pid2jid(pid) , pid, cmdline);
207         }
208     }
209     return;
210
211
212     //parsing한 결과에 따른
213     //builtin_cmd(argv);
214
215     //return;
216 }
```

```
229 void waitfg(pid_t pid, int output_fd)
230 {
231     pid = waitpid(pid, NULL, output_fd);
232     deletejob(jobs, pid);
233     return;
234 }
```

우선 builtin_cmd함수가 "quit" 이외의 인자를 받게 되면 0을 리턴하므로, quit이 아니면 fork()를 실행하고 pid값을 fork()의 리턴값으로 정한다. 만약 fork()된 프로세스가 자식프로세스면(pid == 0) execve < 0 인지 확인하고(execve < 0이면 에러 발생) 에러 문구를 출력한 후 exit(0)한다.

만약 fork()된 프로세스가 자식프로세스가 아니면, addjob()해주고, 프로세스가 백그라운드 프로세스가 아니라면 프로세스를 wait 시켜준다.

만약 프로세스가 백그라운드 프로세스라면, 프로세스의 jid와 pid, 명령어를 출력한다

Trace 06

```
a201402439@2018-sp:~/shelllab/shlab-handout$ ./sdriver -t 06 -s ./tsh
Running trace06.txt...
Success: The test and reference outputs for trace06.txt matched!
```

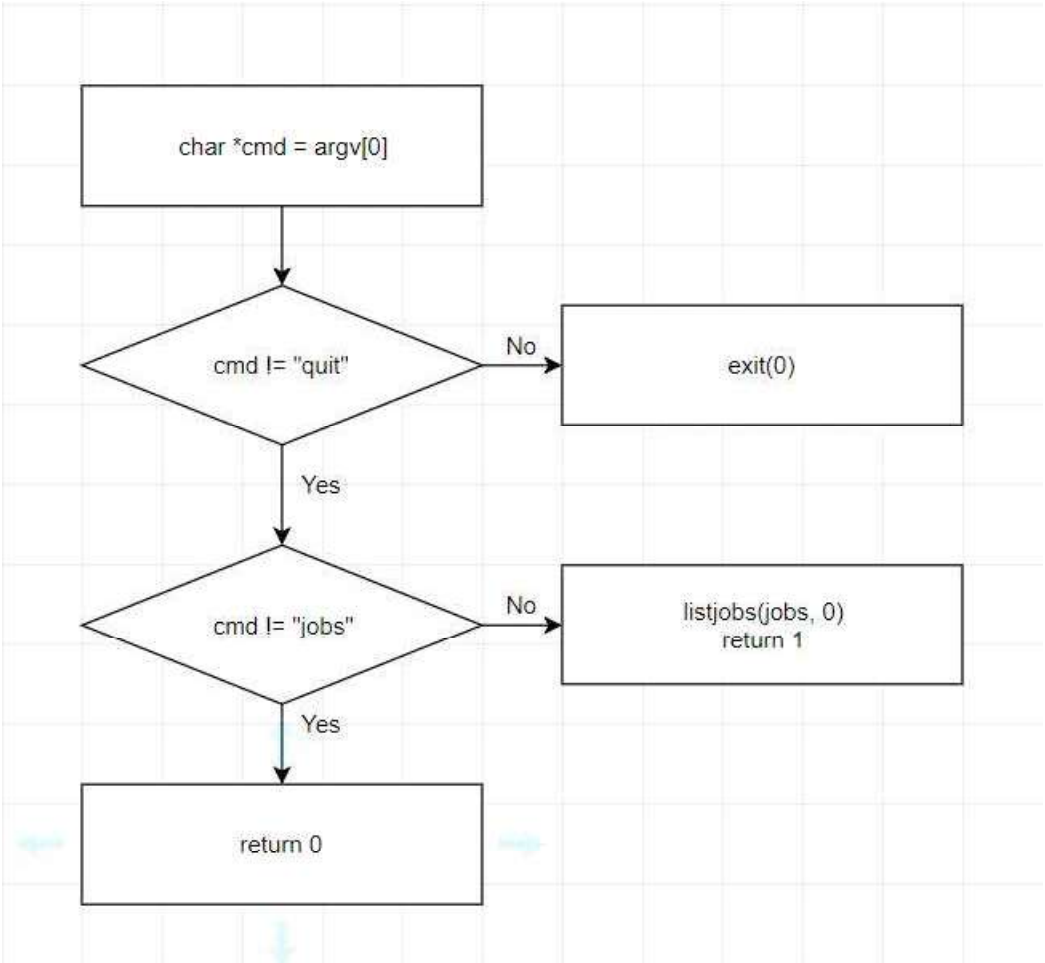
trace 06 해결 방법

trace05에서 작성한 코드로 trace06까지 해결이 가능하였다.
background 실행 시 foreground도 실행되지만, foreground의 자식이 종료될때까지 wait하게 된다

Trace 07

```
a201402439@2018-sp:~/shelllab/shlab-handout$ ./sdriver -t 07 -s ./tsh
Running trace07.txt...
Success: The test and reference outputs for trace07.txt matched!
```

trace 07 플로우차트



trace 07 해결 방법

```
218 int builtin_cmd(char **argv)
219 {
220     char *cmd = argv[0];
221
222     if(!strcmp(cmd, "quit")){//quit command
223         exit(0);
224     }
225     else if(!strcmp(cmd, "jobs")){
226         listjobs(jobs, 0);
227         return 1;
228     }
229
230     return 0;//not a builtin command
231 }
```

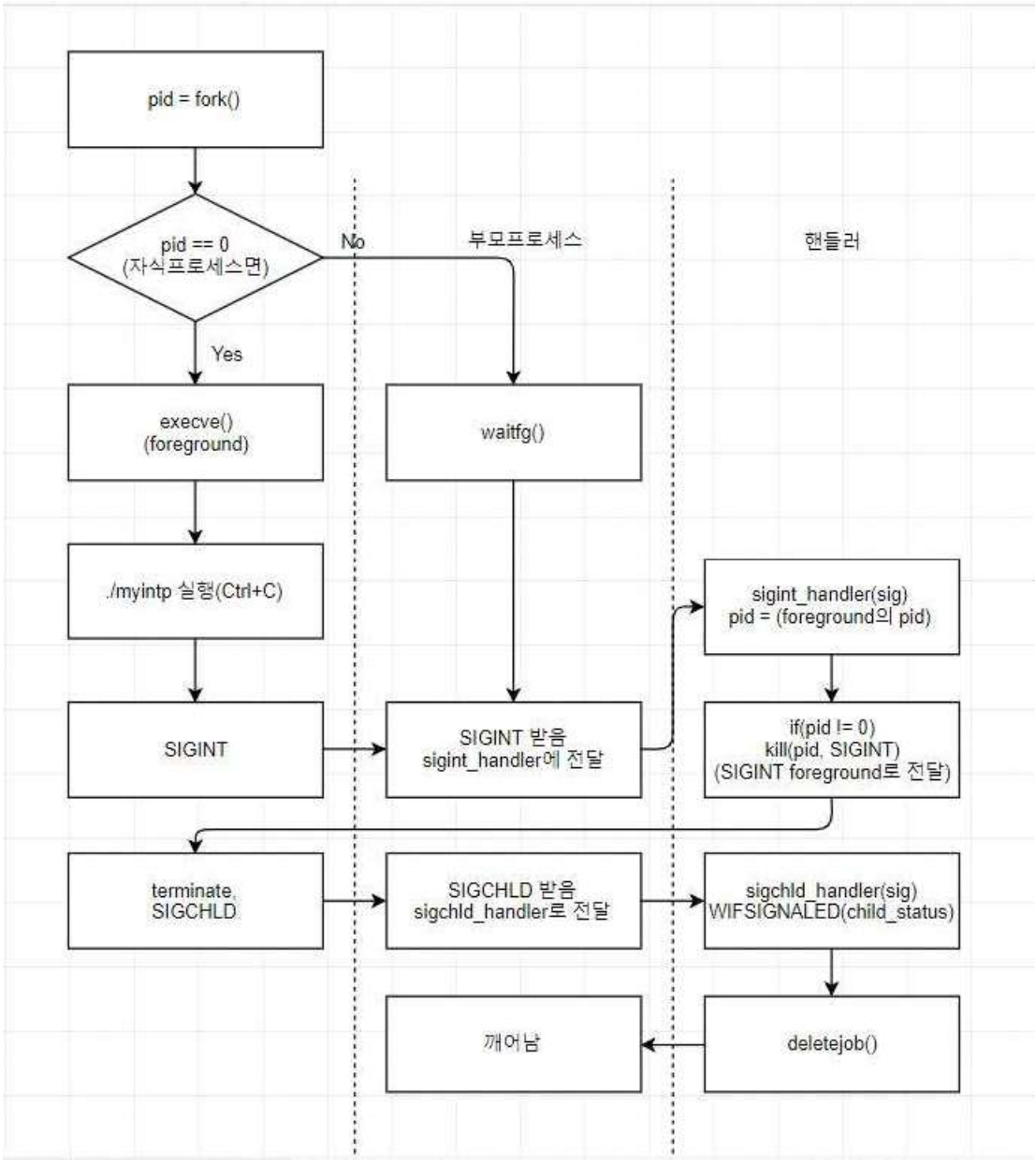
우선 입력받은 명령어가 “quit”이면 exit(0) 한다.
"jobs"이면 listjobs()함수로 현재 작업 리스트를 출력하고 return 1 로 종료한다.
그 둘 이외의 명령어는 return 0 으로 정상 종료한다.

Trace 08

```
a201402439@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] (24123) terminated by signal 2
tsh> quit

Reference output:
#
# trace08.txt - Send fatal SIGINT to foreground job.
#
tsh> ./myintp
Job [1] (24131) terminated by signal 2
tsh> quit
```

trace 08 플로우 차트



trace 08 해결 방법

```
183 void eval(char *cmdline)
184 {
185     char *argv[MAXARGS];
186     pid_t pid;
187     int bg;
188     sigset_t mask;
189
190
191     bg = parseline(cmdline, argv);
192
193     sigemptyset(&mask);
194     sigaddset(&mask, SIGCHLD);
195     sigaddset(&mask, SIGINT);
196     sigaddset(&mask, SIGTSTP);
197
198     sigprocmask(SIG_BLOCK, &mask, NULL);
199
200
201     if(!builtin_cmd(argv)){
202         if((pid = fork()) == 0){
203             sigprocmask(SIG_UNBLOCK, &mask, NULL);
204             if(execve(argv[0], argv, environ) < 0){
205                 printf("%s : Command not found\n", argv);
206                 exit(0);
207
208             }
209         }
210
211         addjob(jobs, pid, (bg == 1?BG: FG), cmdline);
212         sigprocmask(SIG_UNBLOCK, &mask, NULL);
213         if(!bg){
214
215             waitfg(pid, STDOUT_FILENO);
216         }
217         else{
218
219             printf("(%d) (%d) %s",pid2jid(pid) , pid, cmdline);
220         }
221     }
222     return;
223
```

```
285 void sigchld_handler(int sig)
286 {
287     int child_status = 0;
288     pid_t pid;
289 
290     //printf("SIGCHLD Handler!!\n");
291     while((pid = waitpid(-1, &child_status, WNOHANG|WUNTRACED)) > 0){
292         if(WIFSIGNALED(child_status)){
293             printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(pid), pid, WTERMSIG
    (child_status));
294             deletejob(jobs, pid);
295         }
296         else if(WIFSTOPPED(child_status)) {
297             //printf("WIFSTOPPED\n");
298             getjobpid(jobs, pid) -> state = ST;
299             printf("Job [%d] (%d) stopped by signal %d\n", pid2jid(pid), pid, WSTOPSIG(ch
    ild_status));
300             //struct job_t *j = getjobpid(jobs, pid);
301             //j -> state = ST;
302         }
303         else{
304             deletejob(jobs, pid);
305         }
306     }
307     return;
308 }
309 
310 /*
311  * sigint_handler - The kernel sends a SIGINT to the shell whenver the
312  *    user types ctrl-c at the keyboard.  Catch it and send it along
313  *    to the foreground job.
314  */
315 void sigint_handler(int sig)
316 {
317     //printf("sigint handler!!\n");
318     pid_t pid = fgpid(jobs);
319     if(pid != 0){
320         kill(pid, SIGINT);
321     }
322 
323     return;
324 }
```

우선 eval 함수에서, 시그널 셋인 mask에 SIGCHLD, SIGINT, SIGTSTP를 추가한다.
그 후 sigprocmask(SIG_BLOCK, &mask, NULL)을 통해서 mask에 추가한 시그널들을 블록해준다. 이는 addjob하기 이전에 시그널이 발생하여 deletejob이 되지 않게 막기 위해 미리 블록해주는 것이다.
그다음 fork()하는데 만약 자식프로세스이면 블록해준 시그널들을 해제해주고, execve(argv[0], argv, environ)으로 실행해줘서 자식프로세스를 포그라운드 프로세스로 만들어준다.
./myintp(Ctrl+C) 실행을 통해 SIGINT가 발생하면 SIGINT는 커널을 통해 자식프로세스에서 부모프로세스로 전달된다. 부모프로세스는 SIGINT를 sigint_handler로 전달을 해주고, sigint_handler는 포그라운드 프로세스에 SIGINT를 전달하여 kill한다.
자식이 죽게되면 SIGCHLD가 발생하여 부모에게 전달된다. 부모는 SIGCHLD를 sigchld_handler에 전달해준다.
sigchld_handler를 보면 우선 waitpid함수에서 모든 자식프로세스를 기다리고, WNOHANG은 자식이 종료된 것이 없으면 즉시 0을 받는다. 0을 받으면 wait()함수처럼 자식프로세스의 종료를 기다리게 된다.
만약 자식이 외부 신호에 의해 종료되었다면(WIFSIGNALED) 종료된 자식의 jid와 pid를 출력하고 deletejob한다.

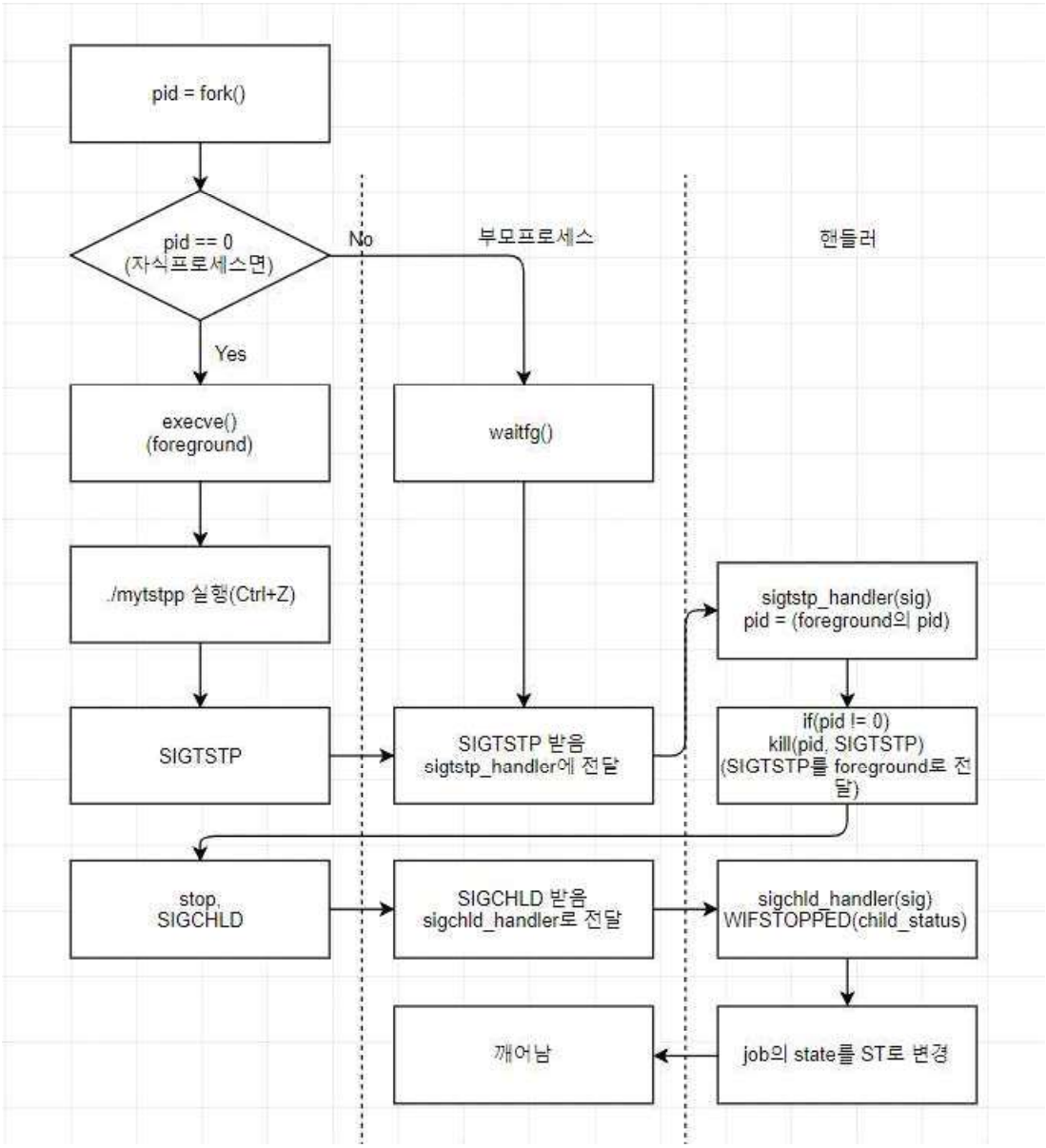
종료되었는데도 deletejob을 해주는 이유는 프로세스는 죽었지만, 프로세스를 포함하던 job는 아직 남아있으므로, deletejob을 해주어야 완전히 죽게된다.

Trace 09

```
a201402439@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] (25003) stopped by signal 20
tsh> jobs
(1) (25003) Stopped    ./mytstpp

Reference output:
#
# trace09.txt - Send SIGTSTP to foreground job.
#
tsh> ./mytstpp
Job [1] (25011) stopped by signal 20
tsh> jobs
(1) (25011) Stopped ./mytstpp
```

trace 09 플로우 차트



trace09 해결 방법

```
328 void sigtstp_handler(int sig)
329 {
330     //printf("sigtstp handler!!\n");
331     pid_t pid = fgpid(jobs);
332     if(pid != 0){
333         kill(pid, SIGTSTP);
334     }
335
336     return;
337
338 }
```

우선 execve() 과정까지는 trace08과 같다.

./mytstpp(Ctrl+Z) 실행을 통해 SIGTSTP가 발생되고, SIGTSTP는 커널을 통해 부모프로세스로

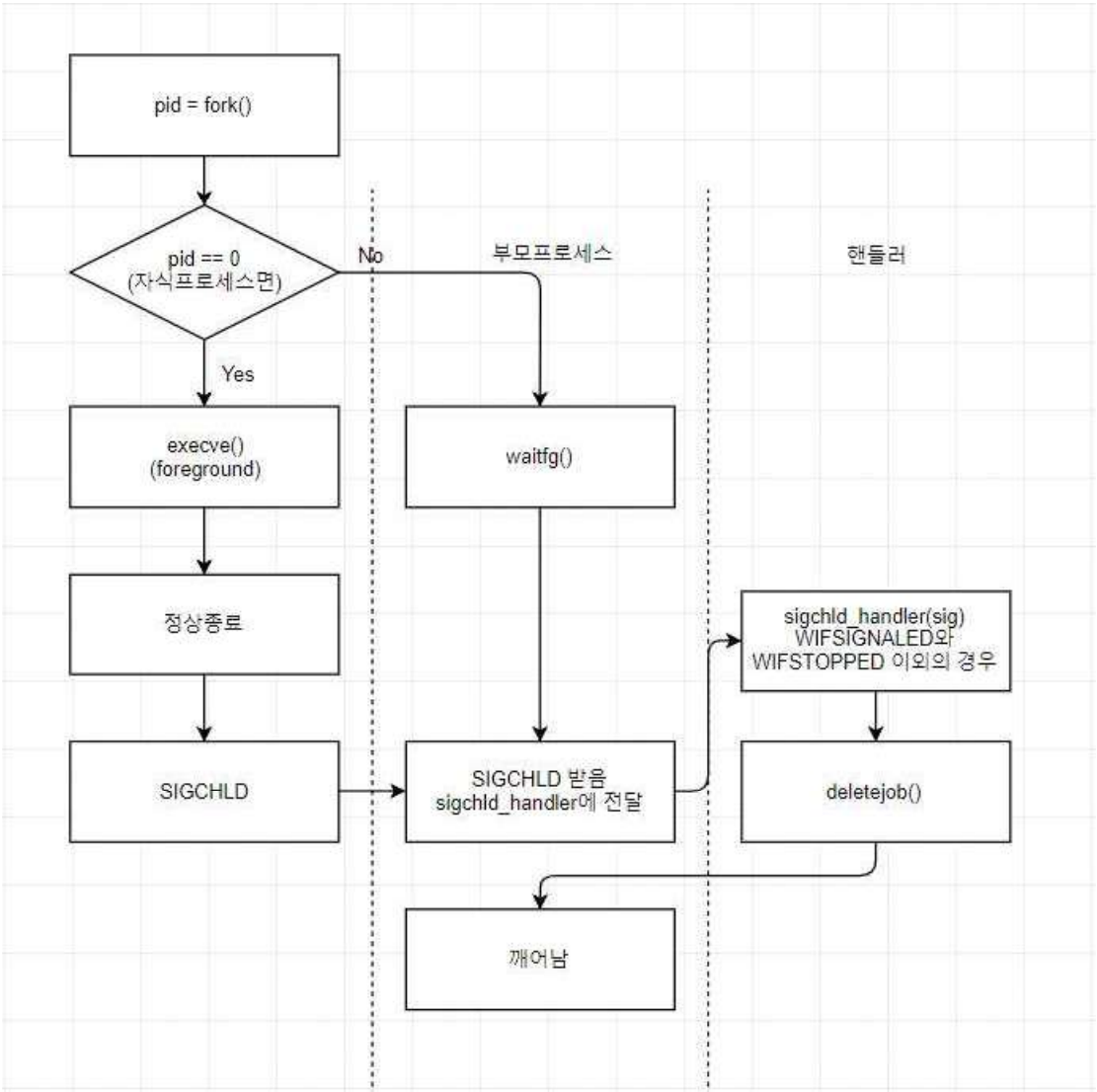
전달된다. 부모프로세스는 SIGTSTP를 sigtstp_handler로 전달해주고, sigtstp_handler는 포그라운드 프로세스에 SIGTSTP를 전달하여 정지시킨다. 포그라운드 프로세스가 정지되면 SIGCHLD가 발생하여 부모프로세스에게 전달되고, 부모프로세스는 SIGCHLD를 sigchld_handler에 전달한다. sigchld_handler의 waitpid()에서 우선 모든 자식프로세스를 기다리고, WUNTRACED는 중단된 자식프로세스의 상태를 받는다.
반환의 원인이 된 자식프로세스가 현재 정지되어 있다면(WIFSTOPPED), 해당 job의 state를 ST로 바꾸고 정지된 job을 출력한다.

Trace 10

```
a201402439@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) (25331) ./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) (25341) ./myspin1 5 &
tsh> /bin/kill myspin1
kill: failed to parse argument: 'myspin1'
tsh> quit
```

trace 10 플로우 차트



trace10 해결 방법

execve() 과정까지는 trace08과 같다.
자식프로세스가 정상 종료되고 SIGCHLD가 발생한다.
SIGCHLD는 부모를 통해 sigchld_handler로 가게 된다.
WIFSIGNALED(외부 신호에 의한 비정상적 종료), WIFSTOPPED(정지) 이외의 상태이므로,
else문으로 가서 deletejob()을 수행하여 완전히 제거하게 된다.

Trace 11

```
a201402439@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] (25514) terminated by signal 2
tsh> quit

Reference output:
#
# trace11.txt - Child sends SIGINT to itself
#
tsh> ./myints
Job [1] (25522) terminated by signal 2
tsh> quit
```

trace11 해결 방법

sigint_handler 함수 내에는 목표한 pid에 시그널을 전달하기 위해 kill함수를 쓰는데, 자식이 스스로에게 SIGINT를 전송하면, sigint_handler를 거쳐 kill에 의해 자신 스스로에게 SIGINT 전달이 되므로 통과된다.

Trace 12

```
a201402439@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] (25815) stopped by signal 20
tsh> jobs
(1) (25815) Stopped    ./mytstps

Reference output:
#
# trace12.txt - Child sends SIGTSTP to itself
#
tsh> ./mytstps
Job [1] (25837) stopped by signal 20
tsh> jobs
(1) (25837) Stopped ./mytstps
```

trace12 해결 방법

trace11과 마찬가지로, 스스로에게 SIGTSTP 시그널을 전송하면 sigtstp_handler를 거쳐 kill함수에 의해 SIGTSTP를 전달받고 처리된다.