2018 시스템 프로그래밍
- Lab 07 -

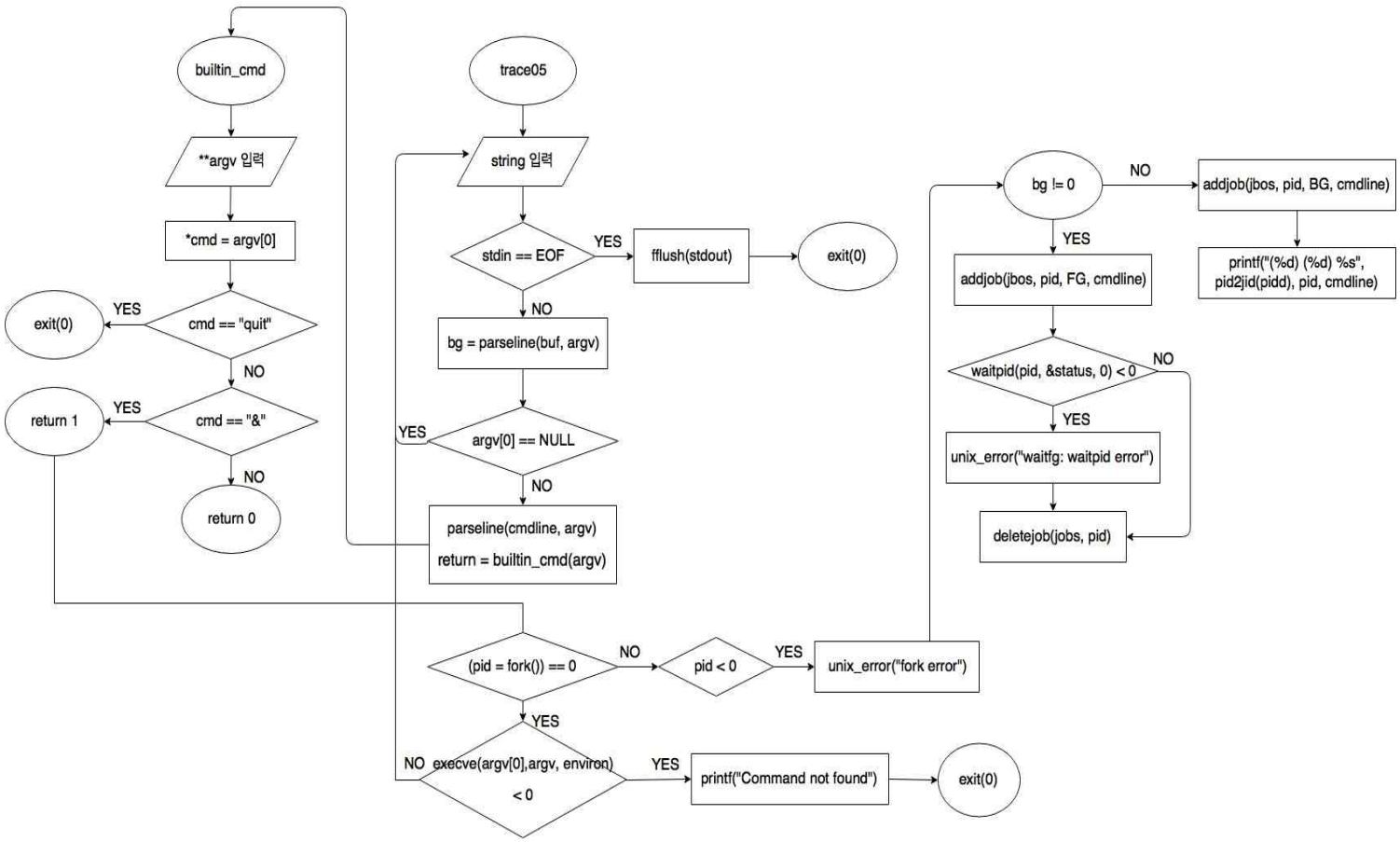
제출일자	2018.11.13
분 반	00
이 름	노효근
학 번	201502049

Trace 05

```
a2015020049@2018-sp:~/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) (23427) ./myspin1 &
tsh> quit

Reference output:
#
# trace05.txt - Run a background job.
#
tsh> ./myspin1 &
(1) (23435) ./myspin1 &
tsh> quit
```

각 trace 별 플로우 차트



trace 해결 방법 설명

```
171 void eval(char *cmdline)
172 {
173     char *argv[MAXARGS];
174     pid_t pid;
175     int bg;
176     // ? ? ? ? parseline? ? ? ? ?
177     bg = parseline(cmdline,argv);
178
179     if(!builtin_cmd(argv)){
180         if((pid=fork())==0){
181             if((execve(argv[0], argv, environ)<0)){
182                 printf("%s : Command not found\n", argv);
183                 exit(0);
184             }
185         }
186
187     addjob(jobs, pid, (bg ==1?BG: FG), cmdline);
188
189     if(!bg){ // foreground? ? ?
190         waitfg(pid, 0);
191     }
192     else{ // background ? ?
193         printf("(%d) (%d) %s",pid2jid(pid),pid ,cmdline);
194     }
195     }
196     return;
197 }
198
212 void waitfg(pid_t pid, int output_fd)
213 {
214     pid = waitpid(pid, NULL,output_fd);
215     deletejob(jobs,pid);
216     return;
217 }
218
```

Trace5는 Background 작업 형태로 프로그램을 실행하는 과정이므로 따라서 쉘에서 작업들을 따로 관리하는 부분 구현이 필요하다.

```
42 struct job_t {              /* The job struct */
43     pid_t pid;              /* job PID */
44     int jid;                /* job ID [1, 2, ...] */
45     int state;              /* UNDEF, BG, FG, or ST */
46     char cmdline[MAXLINE];  /* command line */
47 };
48 struct job_t jobs[MAXJOBS]; /* The job list */
49
```

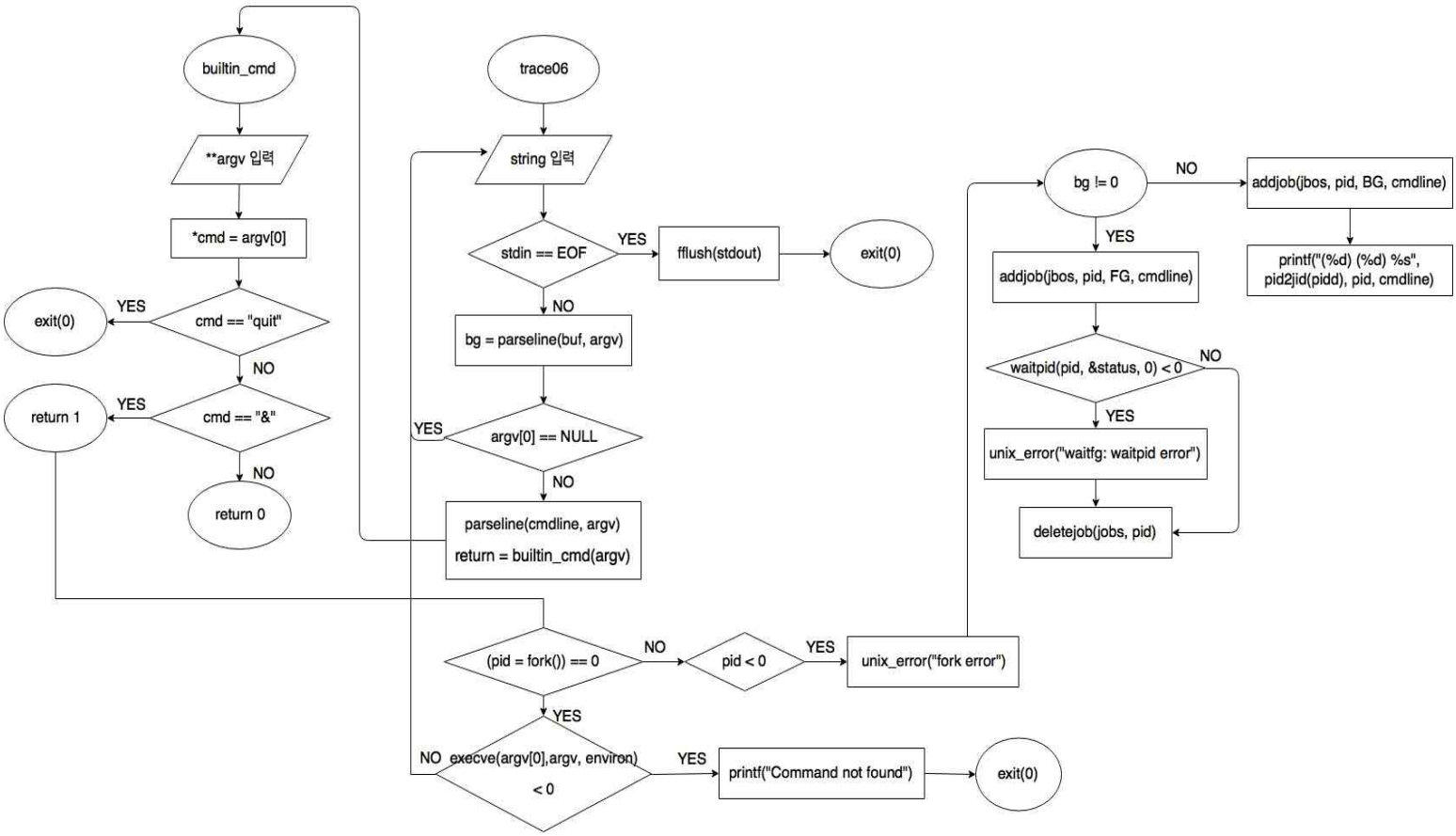
위와 같은 자료구조를 이용하여 구현을 실시한다. job_t 구조체는 프로세스의 ID(pid), 작업의 id(jid), 프로세스의 상태(state), 사용자가 입력한 명령 정보(cmdline)를 가지고 있다. 실행되고 있는 작업들은 모두 job_t구조체에 저장되며, jobs[MAXJOBS]배열을 통해 관리된다.
foreground로 실행시키려면 명령어로 &를 입력해야하므로 addjobs 함수 조건으로 이 부분을 처리한다. 이후 waitpid() 함수를 통해 foreground가 자식프로세서가 종료될 때 까지 기다리도록 한다. 작업이 종료되면 deletejob() 함수를 통해 작업을 삭제하도록 한다.
이후 출력양식을 맞춰 printf("(%d) (%d) %s", pid2jid(pid), pid, cmdline); 의 코드를 추가한다.

Trace 06

```
a2015020490@2018-sp:~/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) (24094) ./myspin1 &
tsh> ./myspin2 1

Reference output:
#
# trace06.txt - Run a foreground job and a background job.
#
tsh> ./myspin1 &
(1) (24105) ./myspin1 &
tsh> ./myspin2 1
```

각 trace 별 플로우 차트



trace 해결 방법 설명

```
171 void eval(char *cmdline)
172 {
173     char *argv[MAXARGS];
174     pid_t pid;
175     int bg;
176     // 명 령 어 parseline을 통 해 분 석
177     bg = parseline(cmdline,argv);
178
179     if(!builtin_cmd(argv)){
180         if((pid=fork())==0){
181             if((execve(argv[0], argv, environ)<0)){
182                 printf("%s : Command not found\n", argv);
183                 exit(0);
184             }
185         }
186
187     addjob(jobs, pid, (bg ==1?BG: FG), cmdline);
188
189     if(!bg){ // foreground일 경 우
190         waitfg(pid, 0);
191     }
192     else{ // background 일 경
193         printf("(%d) (%d) %s",pid2jid(pid),pid ,cmdline);
194     }
195     }
196     return;
197 }
198
212 void waitfg(pid_t pid, int output_fd)
213 {
214     pid = waitpid(pid, NULL,output_fd);
215     deletejob(jobs,pid);
216     return;
217 }
218
```

Trace6는 Foreground와 Background에서 명령어를 실행시키는 단계이다.
소스코드의 구성은 Trace 6과 동일하게 구성한다.

```
 42 struct job_t {              /* The job struct */
 43     pid_t pid;              /* job PID */
 44     int jid;                /* job ID [1, 2, ...] */
 45     int state;              /* UNDEF, BG, FG, or ST */
 46     char cmdline[MAXLINE];  /* command line */
 47 };
 48 struct job_t jobs[MAXJOBS]; /* The job list */
 49
```

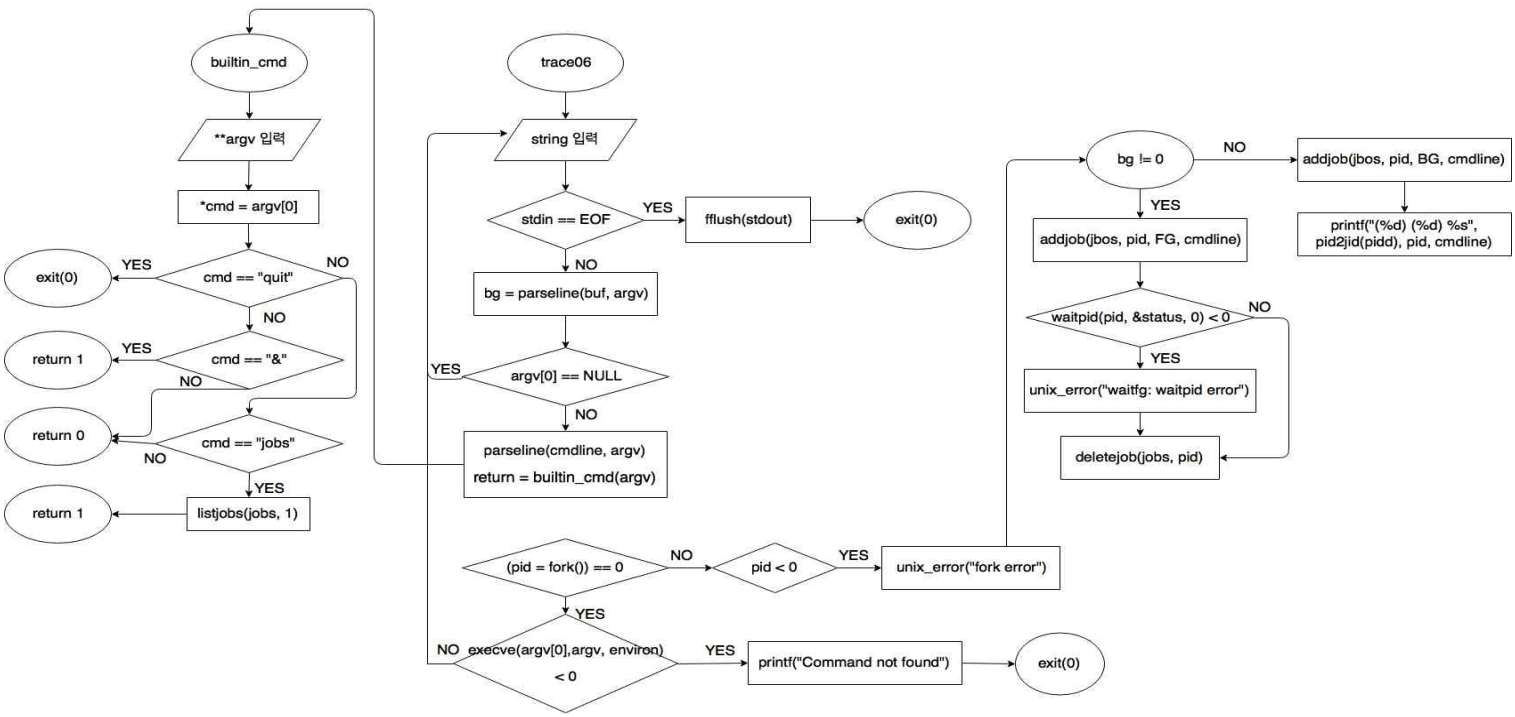
위와 같은 자료구조를 이용하여 구현을 실시한다. job_t 구조체는 프로세스의 ID(pid), 작업의 id(jid), 프로세스의 상태(state), 사용자가 입력한 명령 정보(cmdline)를 가지고 있다. 실행되고 있는 작업들은 모두 job_t구조체에 저장되며, jobs[MAXJOBS]배열을 통해 관리된다.
foreground로 실행시키려면 명령어로 &를 입력해야하므로 addjobs 함수 조건으로 이 부분을 처리한다. 이후 waitpid() 함수를 통해 foreground가 자식프로세서가 종료될 때 까지 기다리도록 한다. 작업이 종료되면 deletejob() 함수를 통해 작업을 삭제하도록 한다.
이후 출력양식을 맞춰 printf("(%d) (%d) %s", pid2jid(pid), pid, cmdline); 의 코드를 추가한다.

Trace 07

```
a201502049@2018-sp:~/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) (24610) ./myspin1 10 &
tsh> ./myspin2 10 &
(2) (24612) ./myspin2 10 &
tsh> jobs
(1) (24610) Running    ./myspin1 10 &
(2) (24612) Running    ./myspin2 10 &

Reference output:
#
# trace07.txt - Use the jobs builtin command.
#
tsh> ./myspin1 10 &
(1) (24620) ./myspin1 10 &
tsh> ./myspin2 10 &
(2) (24622) ./myspin2 10 &
tsh> jobs
(1) (24620) Running ./myspin1 10 &
(2) (24622) Running ./myspin2 10 &
```

각 trace 별 플로우 차트



trace 해결 방법 설명

```
199 int builtin_cmd(char **argv)
200 {
201     char *cmd = argv[0];
202     if(!strcmp(cmd, "quit")){
203         exit(0);
204     }
205     else if(!strcmp(cmd, "jobs")){
206         listjobs(jobs, 1);
207         return 1;
208     }
209     return 0;
210 }
```

Trace7는 Background 작업을 실행하고, builtin 명령어 'jobs'을 실행하게 해주는 단계이다.
명령어 입력 시, 현재 실행되는 작업의 리스트를 출력해준다.
작업리스트를 출력하는 함수 listjobs()를 사용하여, 구조체를 넣어주고 write 함수에서 쓰이는 output_fd로 값은 1로 한다.