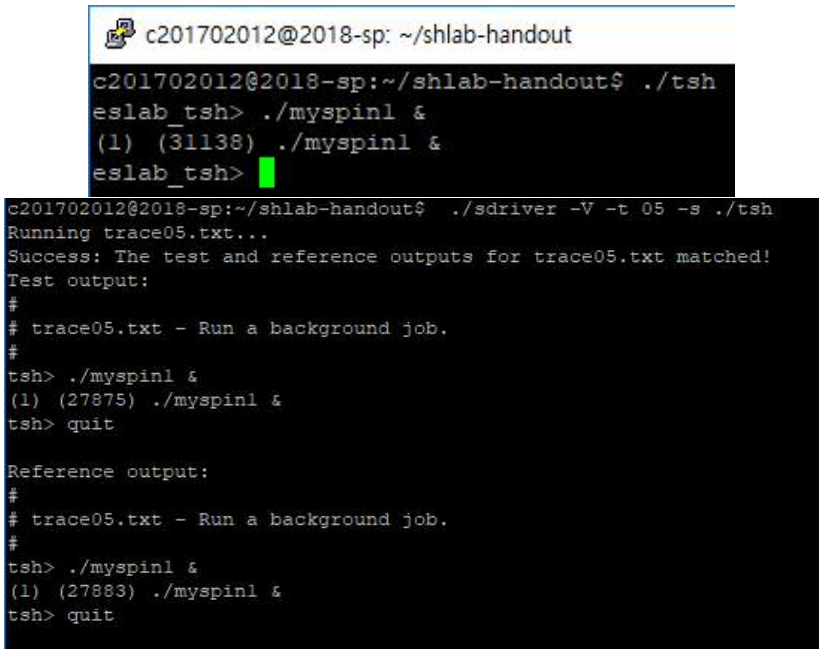
2018 시스템 프로그래밍
- Lab 07 -

제출일자	2018.11.20
분 반	02
이 름	박상현
학 번	201702012

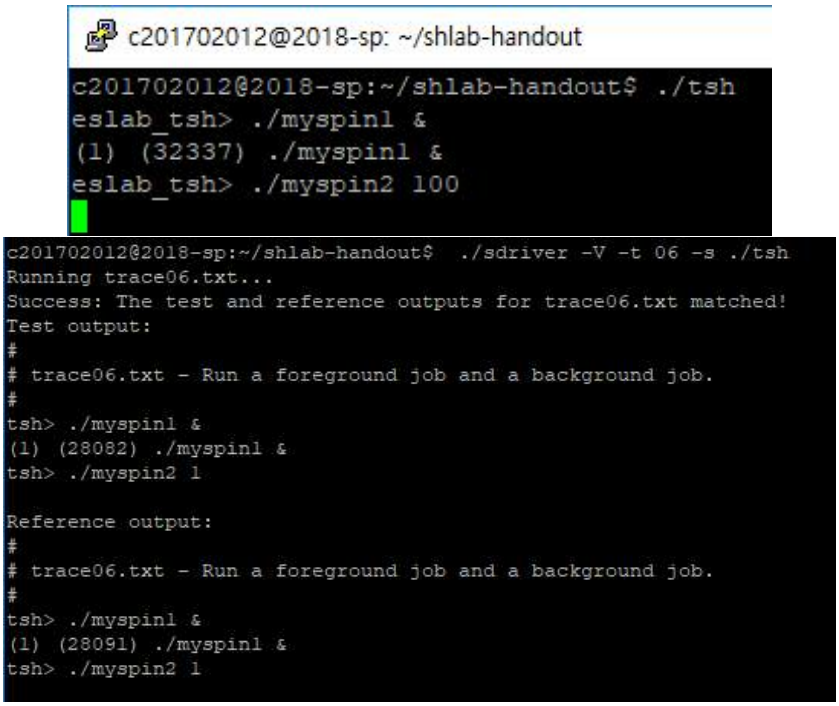
Trace	설 명
trace05	Background 작업 형태로 프로그램 실행

Trace	설 명
trace06	동시에 foreground 작업 형태와 background 작업 형태로 프로그램을 실행

trace05, 06 [화면 캡쳐] - 결과 화면



위 : trace05, 아래 : trace06



trace05, 06 [과정 설명] - 설명 & flowchart진행 과정

/***eval** - Evaluate the command line that the user has just typed in
If the user has requested a built-in command (quit, jobs, bg or fg)
then execute it immediately. **Otherwise, fork a child process and
run the job in the context of the child. If the job is running in
the foreground, wait for it to terminate and then return.**
*/

***** trace05와 trace06을 동시에 작성하였습니다. *****

eval() 에서는 파싱한 command가 built-in command인지 확인하고, 아니면 fork()를 통해 자식을 실행하고 자식에게 일(job)을 시킨다.
이 때, job이 FG에서 실행중이면, 그 job이 끝날 때까지 기다린다. 이 때 사용하는 함수가 waitfg(pid, bg)이다.

```
void waitfg(pid_t pid, int output_fd)
{
    pid = waitpid(pid, NULL, output_fd);
    deletejob(jobs, pid);
    return;
}
```

waitfg(pid, bg)는 waitpid()함수를 호출하여 job(자식 프로세스)이 끝날때까지 기다리며, 끝난 job을 jobs[]에서 삭제한다. 이 때 deletejob(jobs, pid)을 이용하는데,

```
/* deletejob - Delete a job whose PID=pid from the job list */
int deletejob(struct job_t *jobs, pid_t pid)
{
    int i;

    if (pid < 1)
        return 0;

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

이와 같이 jobs[]에서 해당 pid의 job을 찾아서 삭제를 하는 함수임을 알 수 있다.
여기까지가 실행한 job이 foreground인 경우다.

다음으로는 job이 background에서 실행될 때를 알아보겠다.
job이 background에서 실행될 경우에는 해당 job의 jid와 pid와 cmdline을 각각 출력해준다.

위의 evel()코드에서는 bg에 parseline의 리턴을 넘겨주었다. parseline은 다음과

```
    /* should the job run in the background? */
    if ((bg = (*argv[argc-1] == '&')) != 0)
        argv[--argc] = NULL;

    return bg;

}
```

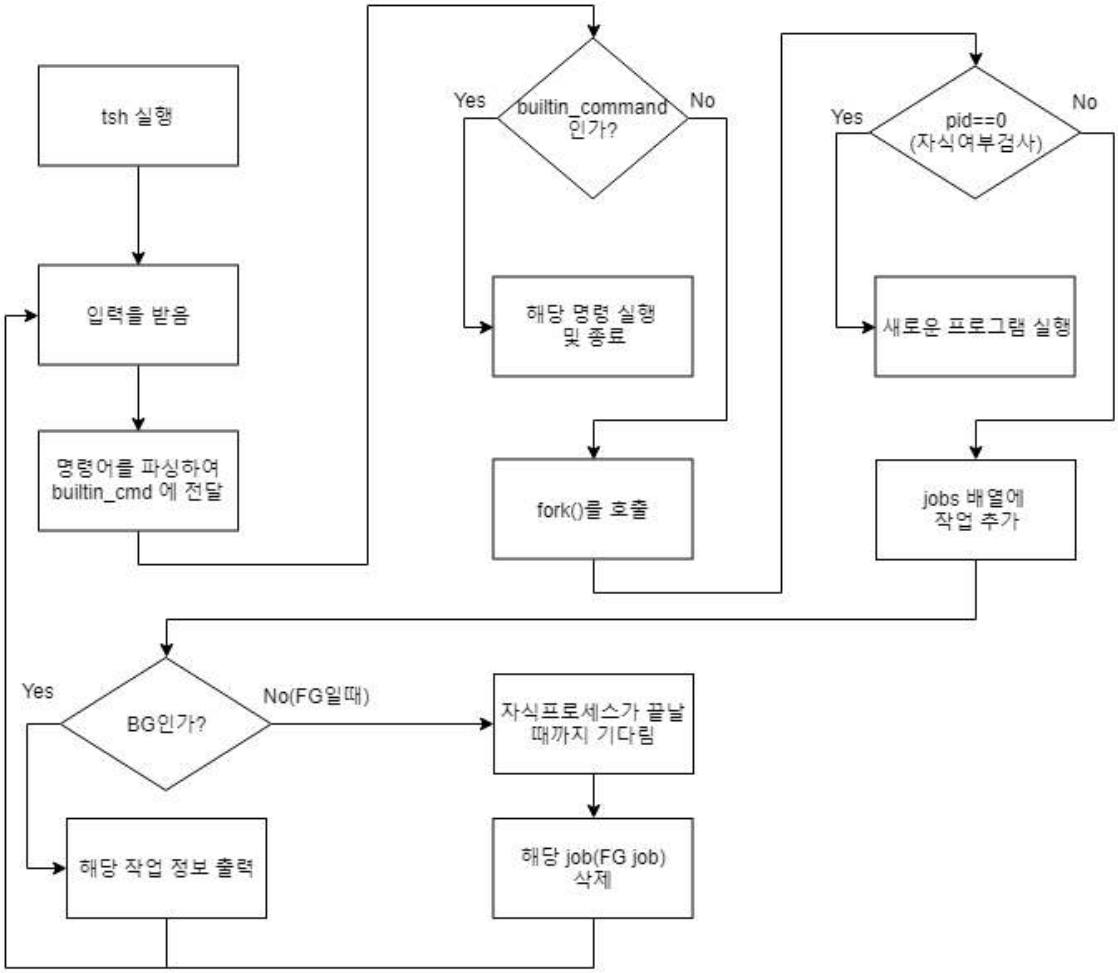
같이 마지막 문자에 & 가 있으면 bg작업으로 인식하여 마지막 인덱스에 null을 넣고 size를 줄여준다. 또한, 궁극적으로 bg에 1을 리턴해준다.
즉, & 가 마지막 문자에 오면, bg는 1이 된다.

```
void eval(char *cmdline) //     ////////////////////////////////////////////////////////eval/////////
{

    char *argv[MAXARGS];    //  command 저장
    pid_t pid;
    int bg;

    bg = parseline(cmdline, argv);

    if(feof(stdin)){ /* trace00    End of file (ctrl - d) */
        fflush(stdout);
        exit(0);
    }

    if(!builtin_cmd(argv)){ /* trace02, trace05 - 07 */
        pid = fork();
        if( pid  == 0/* Child Process 체크  */ ){    //  //  Child Process 인  경우 , execve() 수행
            if((execve(argv[0], argv, environ) < 0 )){
                printf("%s : command not found\n", argv);
                exit(0);
            }
        }

        addjob(jobs, pid, (bg == 1 ? BG :  FG), cmdline);   //  add a job to array jobs[]

        if(!bg){ /*foreground job*/

            waitfg(pid, bg);

        }
        else{    /*background job*/

            //  백그라운드 작업 수행시 작업 내용 출력

            //       jid  pid, cmdline
            printf("([%d]) (%d) %s", pid2jid(pid), pid, cmdline);
        }

    }

    return;
}
```

코드를 분석해보면, 우선 pid == 0이면, 즉 자식이면 execve()를 호출하여 프로그램을 실행한다.
0이 아니면, addjob()을 통해 jobs[]에 추가한다. 그러고나서 FG인지 BG인지를 판별한다.

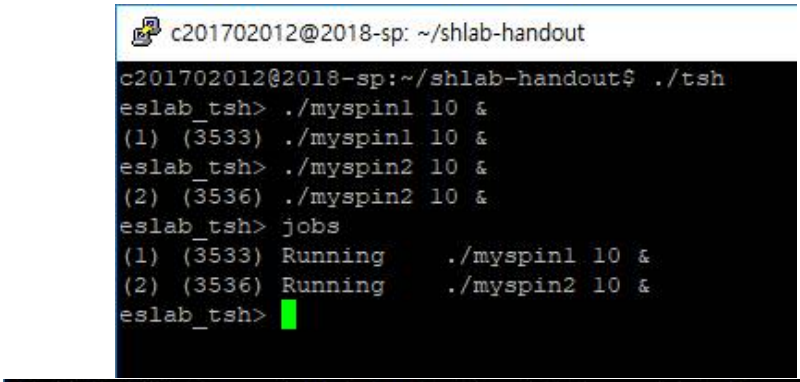
위의 설명대로 FG이면 실행된 자식프로세스가 끝날때까지 기다리는 코드를 짜준다.
반대로, FG가 아니면, 즉 BG이면, 알맞은 포맷으로 출력 해준다.

한가지 궁금한 점은, waitfg(pid, 1)을 해주면 에러가 발생하여 waitfg(pid, bg)로 변경하였는데 에러가 해결되었습니다. output_fd 매개변수의 정확한 용도가 궁금합니다.


tsh 실행
입력을 받음
명령어를 파싱하여
builtin_cmd 에 전달
builtin_command
인가?
Yes
No
해당 명령 실행
및 종료
fork()를 호출
pid==0
(자식여부검사)
Yes
No
새로운 프로그램 실행
jobs 배열에
작업 추가
BG인가?
Yes
No(FG일때)
해당 작업 정보 출력
자식프로세스가 끝날
때까지 기다림
해당 job(FG job)
삭제

Trace	설 명
trace07	Built-in 명령어 'jobs'구현

trace07 [화면 캡쳐] - 결과 화면




```
c201702012@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) (28262) ./myspin1 10 &
tsh> ./myspin2 10 &
(2) (28264) ./myspin2 10 &
tsh> jobs
(1) (28262) Running    ./myspin1 10 &
(2) (28264) Running    ./myspin2 10 &

Reference output:
#
# trace07.txt - Use the jobs builtin command.
#
tsh> ./myspin1 10 &
(1) (28272) ./myspin1 10 &
tsh> ./myspin2 10 &
(2) (28274) ./myspin2 10 &
tsh> jobs
(1) (28272) Running ./myspin1 10 &
(2) (28274) Running ./myspin2 10 &

c201702012@2018-sp:~/shlab-handout$
```

trace07 [과정 설명] - 설명 & flowchart진행 과정

trace07은 built-in command인 "jobs"를 구현하는 과정입니다.
이는 이전에 구현한 built-in command인 "quit"와 유사하였습니다.

```c
int builtin_cmd(char **argv)    //      ////////////////////////////////////////////////builtin//////
{
    char *cmd = argv[0];

    if(!strcmp(cmd, "quit")){ /* trace01  quit command */
        exit(0);
    }
    else if(!strcmp(cmd, "jobs")){  /* trace07 jobs command */
        listjobs(jobs, 1);
        return 1;
    }

    return 0;   /* not a builtin command */
}
```

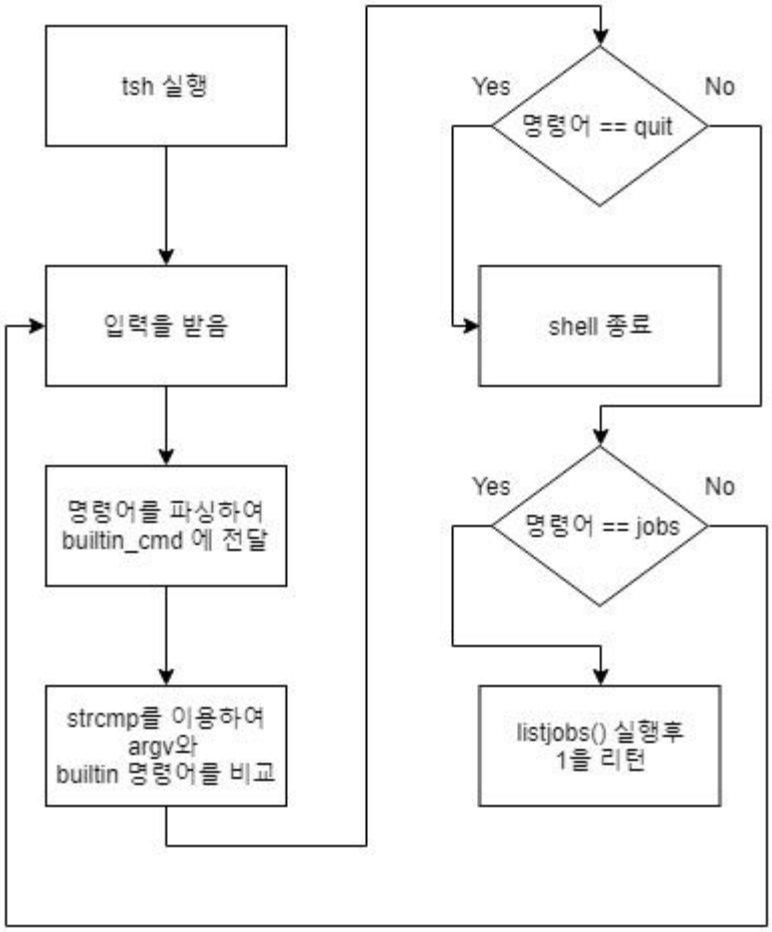
여기서 쓰이는 함수를 살펴보자.

listjobs라는 함수가 사용된다.

```c
/* listjobs - Print the job list */
void listjobs(struct job_t *jobs, int output_fd)
{
    int i;
    char buf[MAXLINE];

    for (i = 0; i < MAXJOBS; i++) {
        memset(buf, '\0', MAXLINE);
        if (jobs[i].pid != 0) {
            sprintf(buf, "(%d) (%d) ", jobs[i].jid, jobs[i].pid);
            if(write(output_fd, buf, strlen(buf)) < 0) {
                fprintf(stderr, "Error writing to output file\n");
                exit(1);
            }
            memset(buf, '\0', MAXLINE);
            switch (jobs[i].state) {
                case BG:
                    sprintf(buf, "Running    ");
                    break;
                case FG:
                    sprintf(buf, "Foreground ");
                    break;
                case ST:
                    sprintf(buf, "Stopped    ");
                    break;
                default:
                    sprintf(buf, "listjobs: Internal error: job[%d].state=%d ",
                            i, jobs[i].state);
            }
            if(write(output_fd, buf, strlen(buf)) < 0) {
                fprintf(stderr, "Error writing to output file\n");
                exit(1);
            }
            memset(buf, '\0', MAXLINE);
            sprintf(buf, "%s", jobs[i].cmdline);
            if(write(output_fd, buf, strlen(buf)) < 0) {
                fprintf(stderr, "Error writing to output file\n");
                exit(1);
            }
        }
    }
    if(output_fd != STDOUT_FILENO)
        close(output_fd);
}
```

이 함수는 현재 진행중인 모든 job들을 각각의 알맞은 형태로 출력해주는 함수이다. jobs 배열의 작업들을 for문을 이용하여 하나씩 그 job의 상태들에 맞는 format으로 출력한다.
가령 BG job은 Running , FG job은 Foreground , ST job은 Stopped 라고 출력한다. 그 후에는 입력받은 cmdline을 %s 로 출력한다.

또 하나의 주목할 점은, return값이 1이라는 것이다. 이는 built-in command라는 뜻이다. 앞의 eval() 함수에서 if(!builtin_cmd(argv))에서도 확인할 수 있다.



```
170 void eval(char *cmdline) //     ////////////////////////////////////////////////////////eval/////////
171 {
172
173     char *argv[MAXARGS];    //  command 저장
174     pid_t pid;
175     int bg;
176
177     bg = parseline(cmdline, argv);
178
179     if(feof(stdin)){ /* trace00    End of file (ctrl - d) */
180         fflush(stdout);
181         exit(0);
182     }
183
184     if(!builtin_cmd(argv)){ /* trace02, trace05 - 07 */
185         pid = fork();
186         if( pid  == 0/* Child Process 체크 */ ){    //  //  Child Process 인 경우 , execve() 수행
187             if((execve(argv[0], argv, environ) < 0 )){
188                 printf("%s : command not found\n", argv);
189                 exit(0);
190             }
191         }
192
193         addjob(jobs, pid, (bg == 1 ? BG :  FG), cmdline);   //  add a job to array jobs[]
194
195         if(!bg){ /*foreground job*/
196
197             waitfg(pid, BG);
198
199         }
200         else{    /*background job*/
201
202             //  백그라운드 작업 수행시 작업 내용 출력
203
204             //       jid  pid, cmdline
205             printf("(%d) (%d) %s", pid2jid(pid), pid, cmdline);
206         }
207
208     }
209
210     return;
211 }
```

위 : 최종 eval(), 아래 : 최종 builtin_cmd

```
213 int builtin_cmd(char **argv)    //      ////////////////////////////////////////////////builtin//////
214 {
215     char *cmd = argv[0];
216
217     if(!strcmp(cmd, "quit")){ /* trace01  quit command */
218         exit(0);
219     }
220     else if(!strcmp(cmd, "jobs")){  /* trace07 jobs command */
221         listjobs(jobs, 1);
222         return 1;
223     }
224
225     return 0;   /* not a builtin command */
226 }
227
228 void waitfg(pid_t pid, int output_fd)
229 {
230     pid = waitpid(pid, NULL, output_fd);
231     deletejob(jobs, pid);
232     return;
233 }
```