# 2016년 시스템 프로그래밍

## -Shell Lab-

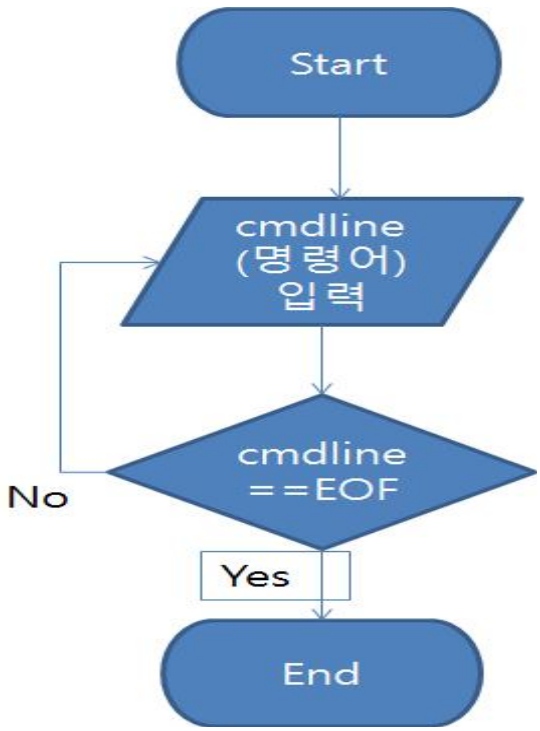
| 제출일자 | 2016.11.21. |
|---|---|
| 이 름 | 정윤수 |
| 학 번 | 201302482 |
| 분 반 | 00 |

# Trace 00

## 1) 정상 작동 모습

```
a201302482@localhost:~/shlab-handout$ ./sdriver -t 00 -s ./tsh
Running trace00.txt...
Success: The test and reference outputs for trace00.txt matched!
a201302482@localhost:~/shlab-handout$
```

## 2) 플로우 차트




## 3) 해결 방법

trace00은 EOF(End Of File)이 입력이 되면 쉘이 종료가 되게 하는 구조이다 그럼으로 if문으로 feof()가 들어오게 된 것을 감지 하면 exit(0) 함수로 바로 종료를 할수있게 하는 구조로 만들면 된다.

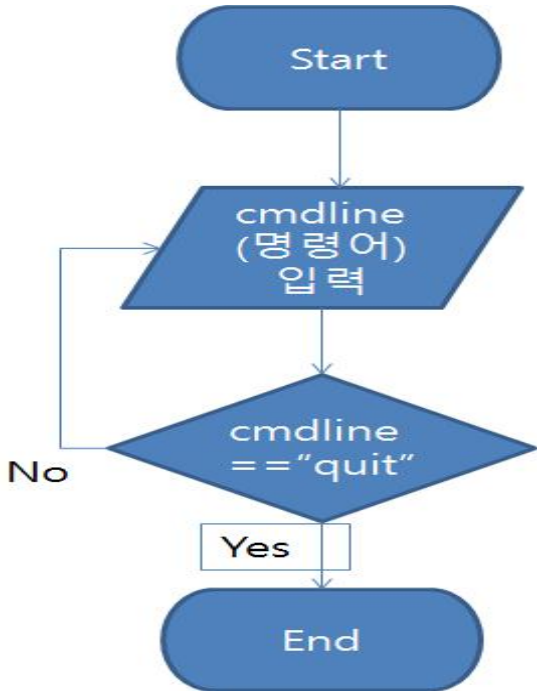
```
if(feof(stdin)){
    fflush(stdout);
    exit(0);
}
```

# Trace 01

## 1) 정상 작동 모습

```
a201302482@localhost:~/shlab-handout$ ./sdriver -t 01 -s ./tsh
Running trace01.txt...
Success: The test and reference outputs for trace01.txt matched!
a201302482@localhost:~/shlab-handout$
```

## 2) 플로우 차트




## 3) 해결 방법

Trace01은 “quit”가 입력이 되어지면 쉘을 종료를 시키는 것을 구현을 해야 한다. 그러므로 입력한 명령어를 처리를 하는 built_in 함수에서 quit가 입력이 되어지면 종료가 되게 구현을 해주어야 한다. built_in 함수에서는 입력한 명령어가 "quit”인지 strcmp()함수를 사용을 하여 비교를 한다. 입력한 명령어가 quit라면 exit(0)함수를 이용을 하여 쉘을 종료를 시킨다.

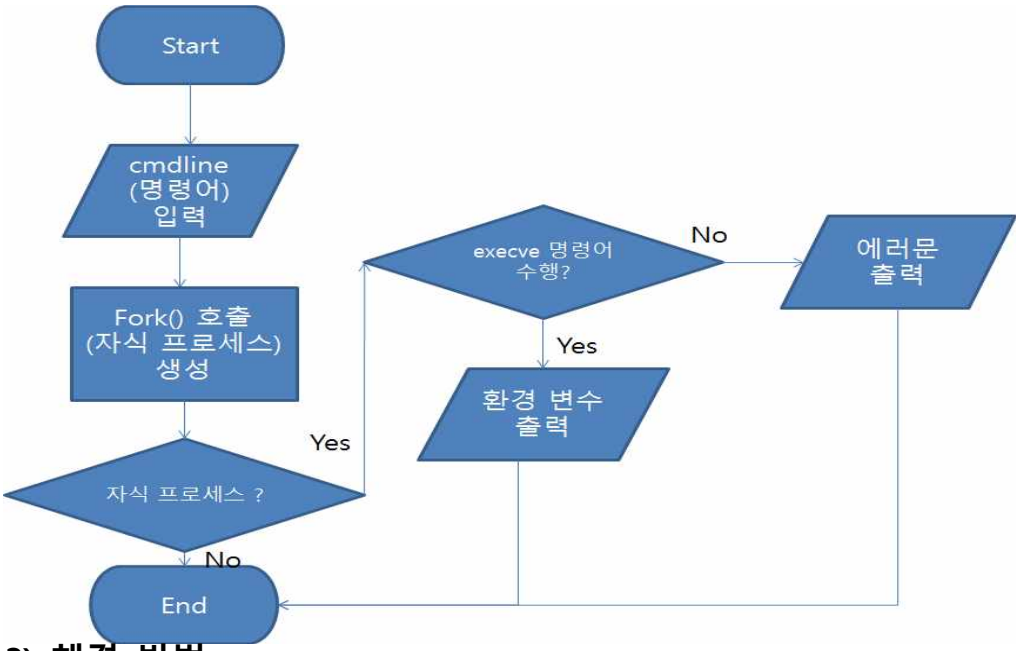
```
if(!strcmp(cmd,"quit")){
    exit(0);
}
```

# Trace 02

## 1) 정상 작동 모습

```
a201302482@localhost:~/shlab-handout$ ./sdriver -t 02 -s ./tsh
Running trace02.txt...
Success: The test and reference outputs for trace02.txt matched!
a201302482@localhost:~/shlab-handout$
```

## 2) 플로우 차트




## 3) 해결 방법

Trace02는 자식 프로세스를 생성을 하여 자식프로세스에서 새로운 프로그램을 실행을 하여 화면에 출력을 해주는 방법을 구현을 해야한다. 참고 자료에서는 myenv명령어를 이용을하여 결과를 확인을 한다. 명령어를 입력을 하면 fork()함수를 이용을 하여서 자식프로세스를 생성을 하게 되면. 부모 프로세스와 자식 프로세스가 존재 하게 된다.자식 프로세스는 프로세스 id로 0을 가지고 있다. 이것을 이용을하여 자식프로세스에서만 작업을 시킬수 있다. 먼저 evecve함수를 호출을하게 된면 evecve()함수는 새로운 프로그램을 실행을 하는 함수로 인자로 주어진 명령어를 새롭게 실행을 할수있게된다. Trace02에서는 인자로 환경변수를 출력을 하는 명령어를 넣었다. 이 수행이 정상적으로 되면 환경변수를 출력을 하고 정상적으로 수행을 하지 못하면 에러문을 출력을 하고 쉘을 종료를 하게 된다.

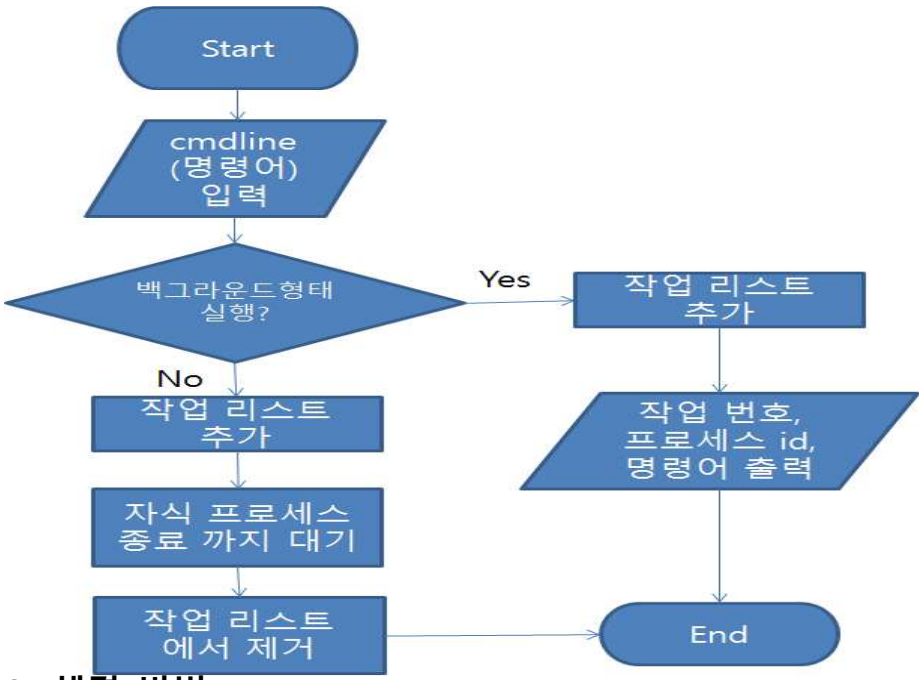
```
if(!builtin_cmd(argv)){
    if((pid = fork()) == 0){
        if((execve(argv[0],argv,environ) <0)){
            printf("%s : Command not found\n",argv);
            exit(0);
        }
    }
```

# Trace 05

## 1) 정상 작동 모습

```
a201302482@localhost:~/shlab-handout$ ./sdriver -t 05 -s ./tsh
Running trace05.txt...
Success: The test and reference outputs for trace05.txt matched!
a201302482@localhost:~/shlab-handout$
```

## 2) 플로우 차트




## 3) 해결 방법

Trace05는 명령어를 백그라운드 형태로 실행을 하면 그 명령어의 작업번호,프로세스id,명령어를 출력을 해주는 것을 구현을 하여야 한다. 백그라운드형태로 명령어를 실행을 할려면 명령어 뒤에 '&'을 붙여야 한다. 만약 명령어 끝에 '&'가 붙어있다면 pareseline()함수에서 이것을 감지하여 1을 반환을 해준다 이것을 이용을하여 어느 형태로 프로그램이 실행이 되었는지 알 수 있다. 프로세스가 foreground형태로 실행이 되었다면 작업리스트에 추가를 하고 자식 프로세스가 종료가 될 때 까지 기다린다. 자식프로세스가 종료가 되면 작업리스트에서 제거를 하게 된다. 백그라운드 형태로 실행이 되었다면 이 작업리스트에 추가를 하고 pid2jid() 함수를 이용을하여 작업번호를 출력을 하고, 프로세스id, 실행시킨 명령어를 출력을 한다.

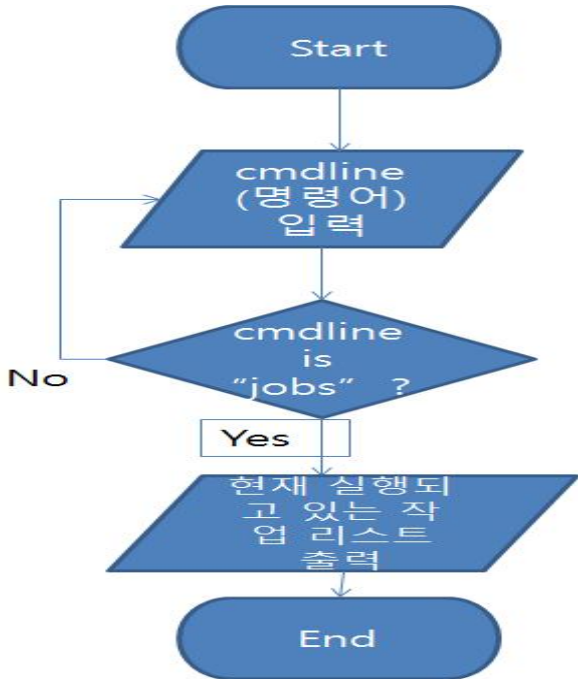
```
if(!state){
    addjob(jobs,pid,FG,cmdline);
    waitpid(pid,NULL,0);
    deletejob(jobs,pid);
}
else{
    addjob(jobs,pid,BG,cmdline);
    printf("(%d) (%d) %s",pid2jid(pid), pid, cmdline);

}
```

# Trace 07

## 1) 정상 작동 모습

```
a201302482@localhost:~/shlab-handout$ ./sdriver -t 07 -s ./tsh
Running trace07.txt...
Success: The test and reference outputs for trace07.txt matched!
a201302482@localhost:~/shlab-handout$
```

## 2) 플로우 차트




## 3) 해결 방법

Trace07은 jobs명령어를 구현을 해야하는 문제이다. jobs명령어를 실행하면 현재 실행되고 있는 작업의 리스트를 출력을 해준다. 앞서 "quit"를 구현을 할때랑 비슷한 방식으로 구현을 하면 된다. jobs명령어를 구현을 하기위해서는 built_in()함수에 jobs명령어가 입력이 되었을 때 하는일을 정해주어야 한다. strcmp()함수를 이용을하여 jobs명령어가 들어온 것을 알아차리면 listjobs()함수를 이용을 하여 현재 실행되고 있는 작업의 리스트를 출력을 출력을 해준다.

```
if(!strcmp(cmd,"jobs")){
    listjobs(jobs,1);
    return 1;
}
```

# Trace 03, 04, 06

## 1) 정상 작동 모습

```
a201302482@localhost:~/shlab-handout$ ./sdriver -t 03 -s ./tsh
Running trace03.txt...
Success: The test and reference outputs for trace03.txt matched!
a201302482@localhost:~/shlab-handout$ ./sdriver -t 04 -s ./tsh
Running trace04.txt...
Success: The test and reference outputs for trace04.txt matched!
```

```
a201302482@localhost:~/shlab-handout$ ./sdriver -t 06 -s ./tsh
Running trace06.txt...
Success: The test and reference outputs for trace06.txt matched!
a201302482@localhost:~/shlab-handout$
```

## 2) 해결 방법

Trace 03, 04, 06은 다른 것들을 구현을 하면 자동적으로 완료가 되는 문제들이다 그러므로 딱히 구현을 할 것은 없다.