



# 2021 Fall System Programming

## Shell Lab 2

2021. 11. 08

유주원

clearlyhunch@gmail.com

Embedded System Lab.  
Dept. of Computer Science & Engineering  
Chungnam National University



## 실습 소개

### ❖ 과목 홈페이지

- ◆ 충남대학교 사이버캠퍼스 ( <http://e-learn.cnu.ac.kr/> )

### ❖ 연락처

- ◆ 유주원
- ◆ 공대 5호관 533호 임베디드 시스템 연구실
- ◆ clearlyhunch@gmail.com
  - ❖ Email 제목은 '[시스템프로그래밍][01][학번\_이름]용건' 으로 시작하도록 작성



# 개요

## ❖ 실습 명

- ◆ Shell Lab - 2

## ❖ 목표

- ◆ 작업 관리를 지원하는 간단한 Unix Shell 프로그램 구현과 이를 통한 프로세스의 제어와 시그널링(Signalling)의 개념 이해

## ❖ 과제 진행

- ◆ 수업시간에 배운 유닉스 지식을 활용하여 구현한다.

## ❖ 구현사항

- ◆ 기본적인 유닉스 셸의 구현
- ◆ 작업 관리(foreground / background)기능 및 셸 명령어 구현



## Shell Lab - 2

- ❖ 프로세스의 관리와 시그널의 제어에 대해 이해하고, 작업의 제어를 지원하는 Unix Shell program을 작성하는 것을 목표로 한다.
- ❖ Shell Lab은 **trace00** 부터 **trace12** 까지의 Trace를 모두 수행할 수 있도록 구현되어야 한다.
  - ◆ 테스트 프로그램 'sdriver' 를 통해 테스트할 수 있다.
- ❖ 총 3주에 걸쳐 진행되며, 각 주 별 수행 사항은 아래와 같다.
  - ◆ Shell Lab 1주차 : trace 00, 01, 02
  - ◆ **Shell Lab 2주차 : trace 03, 04, 05, 06, 07**
  - ◆ Shell Lab 3주차 : trace 08, 09, 10, 11, 12

CNU  
Embedded System  
Laboratory

trace00.txt	Properly terminate on EOF.
trace01.txt	Process built-in quit command.
trace02.txt	Run a foreground job that prints an environment variable
trace03.txt	Run a synchronizing foreground job without any arguments.
trace04.txt	Run a foreground job with arguments.
trace05.txt	Run a background job.
trace06.txt	Run a foreground job and a background job.
trace07.txt	Use the jobs built-in command.
trace08.txt	Send fatal SIGINT to foreground job.
trace09.txt	Send SIGTSTP to foreground job.
trace10.txt	Send fatal SIGTERM (15) to a background job.
trace11.txt	Child sends SIGINT to itself
trace12.txt	Child sends SIGTSTP to itself
trace13.txt	Forward SIGINT to foreground job only.
trace14.txt	Forward SIGTSTP to foreground job only.
trace15.txt	Process bg built-in command (one job)
trace16.txt	Process bg built-in command (two jobs)
trace17.txt	Process fg built-in command (one job)
trace18.txt	Process fg built-in command (two jobs)
trace19.txt	Forward SIGINT to every process in foreground process group
trace20.txt	Forward SIGTSTP to every process in foreground process group
trace21.txt	Restart every stopped process in process group

Shell Lab 02



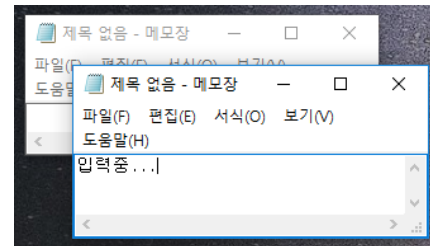
## Shell Lab – Trace 목록 (2주차)

- ❖ sdriver를 이용하여 Trace{03-07}를 테스트 할 수 있다.
  - ◆ 각 trace의 자세한 내용은 해당 파일을 열어서 확인할 수 있다.

Trace	설 명
trace00	EOF(End-Of-File)가 입력되면 종료
trace01	Built-in 명령어 'quit' 구현
trace02	Foreground 작업 형태로 프로그램 실행
trace03	Foreground 작업 형태로 매개변수가 없는 프로그램 실행
trace04	Foreground 작업 형태로 매개변수가 있는 프로그램 실행
trace05	Background 작업 형태로 프로그램 실행
trace06	동시에 foreground 작업 형태와 background 작업 형태로 프로그램을 실행
trace07	Built-in 명령어 'jobs' 구현

## 참조 - Foreground 와 Background

- ❖ 윈도우 환경에서 메모장을 2개 연달아 실행하면, 먼저 실행한 메모장은 제목이 회색으로 바뀌며 입력할 수 없다.
  - ◆ 검은색 제목의 **입력 가능한** 메모장이 Foreground process
  - ◆ 회색 제목의 **입력 불가능한** 메모장이 Background process



- ❖ 리눅스의 Process도 이와 같이 Foreground process와 Background process로 나뉜다.
  - ◆ 5초간 실행되는 프로그램을

### foreground 실행

```

sys03@localhost: ~/workspace
sys03@localhost:~/workspace$ ./fbprocess.out
[1] 22016
sys03@localhost:~/workspace$ result=5
sys03@localhost:~/workspace$ ^C
sys03@localhost:~/workspace$
  
```

### background 실행

```


sys03@localhost: ~/workspace
sys03@localhost:~/workspace$ ./fbprocess.out &
[1] 22016
sys03@localhost:~/workspace$ result=5
[1]+  Done                  ./fbprocess.out
sys03@localhost:~/workspace$
  
```



## Shell Lab – trace05

### ❖ trace05 : Background 작업 형태로 프로그램 실행

```
sys03@localhost:~/workspace/shlab-handout$ ./sdriver -V -t 05 -s ./tshref
Running trace05.txt...
Success: The test and reference outputs for trace05.txt matched!
Test output:
#
# trace05.txt - Run a background job.
#
tsh> ./myspin1 &
(1) (18045) ./myspin1 &
tsh> quit
```

```
1 
2 # trace05.txt - Run a background job.
3 #
4 /bin/echo -e tsh\076 ./myspin1 \046
5 NEXT
6 ./myspin1 &
7 NEXT
8
9 WAIT
10 SIGNAL
11
12 /bin/echo -e tsh\076 quit
13 NEXT
14 quit
```

trace05.txt

- ❖ 프로그램을 **background** 형태로 실행시키면 된다.
  - ❖ 해당 테스트를 살펴보면 echo를 foreground 형태로 실행한다.
  - ❖ 이때 실행되는 프로세스는 매개변수를 가질 수도 있고, 그렇지 않을 수도 있다.
  - ❖ 실행시키는 방법은 ./myspin1 & 으로 &(앰퍼샌드, 앤드기호)를 붙이면 백그라운드로 실행된다.





## Shell Lab – trace05

- ❖ Background 형태로 프로그램을 실행시키려면,
  - 셸에서 작업들을 관리하는 부분이 구현되어 있어야 한다.
    - ❖ Foreground 형태인 경우, 프로세스 하나만 실행되기 때문에 작업을 따로 관리할 필요가 없다.
    - ❖ 하지만 background의 경우, 여러 프로세스가 실행될 수 있으므로 작업에 대한 관리가 필수적이다.
  - 따라서 아래에 **제공된 자료구조를 이용하여 구현**하면 된다.

```
struct job_t {                /* The job struct */
    pid_t pid;                /* job PID */
    int jid;                  /* job ID [1, 2, ...] */
    int state;                /* UNDEF, BG, FG, or ST */
    char cmdline[MAXLINE];    /* command line */
};
struct job_t jobs[MAXJOBS]; /* The job list */
/* End global variables */
```

- 해당 자료구조는
  - ❖ 프로세스의 ID(PID), 작업의 ID(JID), 프로세스의 상태(State), 사용자가 입력한 명령 정보를 가지고 있다.
- 실행되고 있는 작업들은 이 자료구조 형태로 저장되며, jobs[MAXJOBS] 배열을 통해 관리된다.

참고 자료 : [http://www.hackerschool.org/HS\\_Boards/zboard.php?id=Free\\_Lectures&no=8032](http://www.hackerschool.org/HS_Boards/zboard.php?id=Free_Lectures&no=8032)



## Shell Lab – trace05

- ❖ 작업 관리와 관련된 함수들은 다음과 같다.

함 수	설 명
initjobs(struct job_t *jobs)	작업 리스트 초기화 (Main함수에서 이미 실행 중)
addjob (struct job_t *jobs, pid_t pid, int state, char *cmdline)	작업 리스트에 작업을 추가
deletejob (struct job_t *jobs, pid_t pid)	작업 리스트에서 작업을 제거
pid2jid(pid_t pid)	프로세스 ID를 작업 ID로 맵핑
listjobs(struct job_t *jobs, int output_fd)	작업 리스트를 출력

- ◆ 해당 함수에 대한 자세한 동작은 코드분석을 통해 알아낼 것!!



## Shell Lab – trace05

- ❖ 다음 코드 형태를 참조하여 eval( )함수를 구성하고, 셸을 구현해본다.

```
//addjob() 도우미 함수를 이용해서 joblist에 job을 추가한다
if(/*foreground job 체크 */)
{
    //.....//
}
else(/*background job 체크*/)
{
    //trace05가 요구하는 출력약식에 맞추어 출력
    //pid2jid() 도우미함수를 이용해서 양식에 맞추어 출력
}
```

- ◆ **foreground** 작업인 경우

- ❖ **자식 프로세스가 종료될 때까지 기다린다.** 자식 프로세스가 종료되면 작업 리스트에서 작업을 제거한다.

- ◆ **waitpid( )**함수를 활용하여, 자식 프로세스가 종료될 때까지 기다린다.

- ◆ **background** 작업인 경우

- ❖ 해당 작업의 정보를 출력하는 양식을 확인하고 **해당 양식에 맞추어 출력**할 수 있도록 프린트 문을 구성한다.



## Shell Lab – trace05

- ❖ 다음 코드 형태를 참조하여 eval( )함수를 구성하고, 셸을 구현해본다.

```
void eval(char *cmdline)
{
    char *argv[MAXARGS];
    pid_t pid;
    int bg;

    bg = parseline(cmdline, argv);

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

        addjob(jobs, pid, (bg == 1?BG: FG), cmdline);

        if(!bg) {
            //부모 프로세서가 자식 프로세서의 종료를 대기 후 처리
        }
        else {
            //백그라운드 작업 수행시 작업 내용 출력
            printf("(%d) (%d) %s", pid, bg, argv[0]);
        }
    }

    return;
}
```



## Shell Lab – trace05

- ❖ 다음 코드 형태를 참조하여 waitpid( )함수의 기능을 알아본다.

```
if(!bg) {  
    waitpid(pid, NULL, 0);  
    deletejob(jobs, pid);  
}
```

### ○ 인자 값

- pid : 자식 프로세스의 ID. 아래 설명 참조.
- status : wait() 함수와 같은 역할. (필요 없을 시 NULL)
- option : WNOHANG 입력 시 부모 프로세스는 기다리지 않도록 설정, 0 입력 시 wait()함수와 같이 자식 프로세스의 종료를 기다린다.

### ○ waitpid()에서 사용하는 pid인자값의 의미

- pid < -1 : pid의 절대값과 동일한 프로세스 그룹ID의 모든 자식 프로세스의 종료를 기다린다.
- pid == -1 : 모든 자식 프로세스의 종료를 기다린다. 만약 pid값이 -1이면 waitpid함수는 wait()함수와 동일하다.
- pid == 0 : 현재 프로세스의 프로세스 그룹 ID와 같은 프로세스 그룹 ID를 가지는 모든 자식 프로세스의 종료를 기다린다.
- pid > 0 : pid값에 해당하는 프로세스 ID를 가진 자식 프로세스의 종료를 기다린다.



## Shell Lab – trace07

### ❖ trace07 : Built-in 명령어 ‘jobs’ 구현

```
sys03@localhost:~/workspace/shlab-handout$ ./sdriver -V -t 07 -s ./tshref
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) (20717) ./myspin1 10 &
tsh> ./myspin2 10 &
(2) (20719) ./myspin2 10 &
tsh> jobs
(1) (20717) Running ./myspin1 10 &
(2) (20719) Running ./myspin2 10 &
```

```
1 █
2 # trace07.txt - Use the jobs builtin command.
3 #
4 /bin/echo -e tsh\076 ./myspin1 10 \046
5 NEXT
6 ./myspin1 10 &
7 NEXT
8
9 /bin/echo -e tsh\076 ./myspin2 10 \046
10 NEXT
11 ./myspin2 10 &
12 NEXT
13
14 WAIT
15 WAIT
16
17 /bin/echo -e tsh\076 jobs
18 NEXT
19 jobs
20 NEXT
21
22 quit
```

trace07.txt



## Shell Lab – trace07

### ❖ trace07

- 두 개의 background 작업을 실행한 후, built-in 명령어 ‘jobs’을 실행한다. 해당 명령어를 입력 받으면, **현재 실행되고 있는 작업의 리스트를 출력**해준다.
- 앞서 구현한 built-in 명령어 ‘quit’과 비슷한 방식으로 구현하면 된다.
  - ❖ 이때 작업 리스트를 출력하는데 사용하는 함수는 `listjobs( )`이다.
  - ❖ 해당 함수의 사용 방법은 소스코드 분석을 통해 알아낸다.

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

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

        return 1;
    }

    return 0;
}
```

Return 1은 builtin\_cmd인  
경우를 나타냄



## 주의 사항

- ❖ Shell Lab을 수행하다 보면, 부모 프로세스가 자식 프로세스보다 먼저 종료되어 자식 프로세스가 **좀비**가 되는 경우가 있다.

- ◆ 셸 종료 후, 'ps' 명령어를 입력

```
[b000000000@eslab shlab-handout]$ ps
  PID TTY          TIME CMD
 7714 pts/1        00:00:00 bash
 7805 pts/1        00:00:00 ps
```

- ◆ 위와 같이 tsh가 좀비가 되어 남아있는 것을 볼 수 있다. 따라서 해당 좀비 프로세스를 제거해야 한다.
  - ❖ **kill -9 <PID>**
  - ❖ ex) kill -9 29895
- ◆ **좀비 프로세스를 많이 만들면 감점! (셸 테스트 후 실시간 체크 바람)**





# 과제 (이번에 제출X)

## 1. Shell Lab

- I. trace 01, 02, 03, 04, 05, 06, 07 에 대한 코드 작성

## 2. Shell Lab 보고서

- I. 각 trace 별, tshref를 수행한 결과와 본인이 구현한 tsh와의 동작 일치를 증명
  - 1) sdriver 수행 결과와 tsh에서의 정상작동 모습 ( ./sdriver -V -t XX -s ./tsh )
- II. 각 trace 별, 플로우 차트
  - 1) 간단한 수행과정을 플로우차트로 나타내면 됨.
- III. 각 trace 별, 해결 방법에 대한 설명

## 3. 제출

- I. shlab-handout 디렉토리를 통째로 압축 ( tar -cvf [파일명.tar.gz] [폴더명] )
  - 1) 파일명: [sys01]ShellLab02\_학번.tar.gz
- II. 결과 보고서를 작성
  - 1) 파일명: [sys01]ShellLab02\_학번.pdf
- III. I.과 II. 두개를 하나로 압축
  - 1) 파일명:[sys01]ShellLab02\_학번\_이름.zip