시스템 프로그래밍

- Shell Lab -

2014. 11. 13.

박시형

sihyeong@cnu.ac.kr

Embedded System Lab. Computer Engineering Dept.

Chungnam National University

개 요

❖ 실습 명

- Shell Lab

❖ 목표

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

❖ 과제 진행

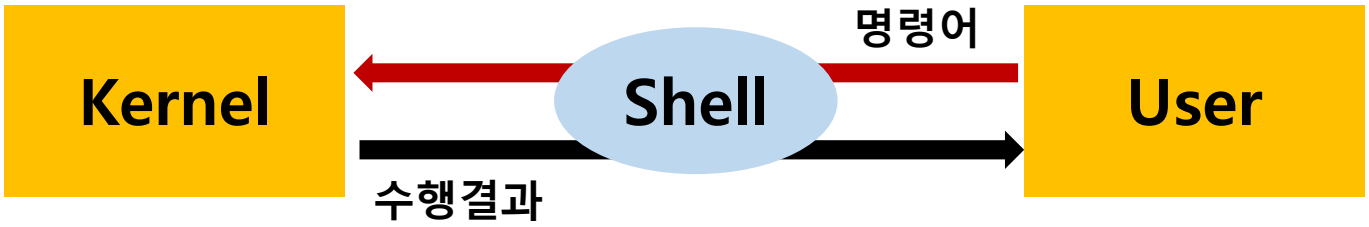
- 수업시간에 배운 유닉스 지식을 활용하여 구현한다.

❖ 구현사항

- 기본적인 유닉스 쉘의 구현
- 작업 관리(foreground / background)기능 및 쉘 명령어 구현

Shell이란?

❖ 쉘(Shell)은 사용자와 Kernel을 연결시켜주는 인터페이스 역할을 한다.



❖ Shell의 기능

- 명령어 해석기 기능
  - 사용자가 입력한 명령어를 해석하고 커널에 전달하는 역할
  - 명령어 해석기(interpreter) 또는 번역기(translator)
- 프로그래밍 기능
  - 자체적인 프로그래밍 기능을 통해 프로그램 작성 가능
  - 쉘 프로그램을 쉘 스크립트라고 부름
- 사용자 환경설정 기능
  - 초기화 파일을 이용해 사용자 환경을 설정

❖ 이러한 shell의 기능을 Shell Lab을 통해 구현해본다.

Shell Lab

- 프로세스의 관리와 시그널의 제어에 대해 이해하고, 작업의 제어를 지원하는 Unix Shell program을 작성하는 것을 목표로 한다.

- Shell Lab은 trace01 부터 trace16 까지의 Trace를 모두 수행할 수 있도록 구현되어야 한다.
  - 테스트 프로그램 'sdriver' 를 통해 테스트할 수 있다.

- 총 2주에 걸쳐 진행되며, 각 주 별 수행 사항은 아래와 같다.
  - Shell Lab 1주차 : trace01 ~ 05
  - Shell Lab 2주차 : trace06 ~ 16

Shell Lab - 준비

Shell Lab 파일 복사 및 압축해제

- /home/syspro/shlab-handout.tar.gz

Shell Lab 파일 구성

파 일	설 명
Makefile	쉘 프로그램의 컴파일 및 테스트
README	도움말
tsh.c	Shell 프로그램의 소스코드
tshref	Shell binary의 레퍼런스
sdriver.pl	Shell에 각 trace 들을 실행하는 프로그램
trace{01-16}.txt	Shell 드라이버를 제어하는 15개의 trace
tshref.out	15개의 trace 에 대한 레퍼런스 쉘의 출력 예제
myspin.c, mysplit.c, mystop.c myint.c	trace 파일들에서 불려지는 C로 작성된 프로그램 (README 참조)

Shell Lab – 함수 설명

❖ tsh.c 파일 내부 구성

함 수	설 명
eval	명령을 파싱 하거나 해석하는 메인 루틴
builtin_cmd	quit, fg, ba와 작업s 같은 built-in 명령어를 해석
do_bgfg	Built-in 명령어인 bg 와 fg 명령어 구현
waitfg	foreground 작업이 완료될 때 까지 대기
sigchld_handler	SIGCHILD 시그널 핸들러
sigint_handler	SIGINT(ctrl-c) 시그널 핸들러
sigtstp_handler	SIGTSTP(ctrl-z) 시그널 핸들러

Shell Lab – Trace (1주차)

- sdriver를 이용하여 Trace{01-05}를 테스트 할 수 있다.
  - 각 trace의 자세한 내용은 해당 파일을 열어서 확인할 수 있다.

Trace	설 명
trace01	EOF(End-Of-File)가 입력되면 종료
trace02	Built-in 명령어 'quit' 구현
trace03	Foreground 작업 형태로 프로그램 실행
trace04	Background 작업 형태로 프로그램 실행
trace05	Built-in 명령어 'jobs'구현

Shell Lab – Trace (2주차)

❖ **sdriver를 이용하여 Trace{06-16}를 테스트 할 수 있다.**

Trace	설 명
trace06	Foreground 작업에 SIGINT 전송
trace07	Foreground 작업에만 SIGINT 전송
trace08	Foreground 작업에만 SIGTSTP 전송
trace09	Built-in 명령어 'bg' 수행
trace10	Built-in command 'fg' 수행
trace11	모든 foreground 프로세스 그룹의 모든 프로세스에게 SIGINT 전송
trace12	모든 foreground 프로세스 그룹의 모든 프로세스에게 SIGTSTP 전송
trace13	프로세스 그룹의 모든 정지된 프로세스를 다시 시작
trace14	간단한 에러 처리
trace15	trace01~14 전부 수행
trace16	쉘이 SIGTSTP 와 SIGINT를 처리할 수 있는지 테스트

Shell Lab – 시작 하기

- shell lab 소스파일 수정
  - tsh.c 파일을 열어 맨 위쪽에 자신의 학번과 이름을 기입한다.
  - tsh.c 파일 내부를 수정 또는 추가 작성하여 Shell Lab의 요구사항(각 trace 별 요구사항)을 해결해 나간다.

- shell lab 빌드 (p10)
  - make 명령을 통해 tsh.c을 컴파일 한다.
  - 결과로 tsh 란 실행 파일이 생성된다.
    - tsh 은 본인이 작성하여 완성된 쉘의 본체

- shell lab 실행 및 테스트
  - 쉘은 다음과 같이 실행한다.
    - ./tsh
    - 참고로 ./tshref 의 실행을 통해 정상적으로 완성된 쉘을 경험할 수 있다.
  - 'sdriver.pl'를 이용하여 생성된 쉘이 제 기능을 하는지 검사할 수 있음
    - 소스 수정 후에 항상 make한 뒤 검사

Shell Lab – 컴파일

❖ shlab 빌드

- 작성한 tsh.c를 컴파일 하여 tsh 쉘을 빌드 한다.
- 미리 작성된 Makefile이 존재하기 때문에 make 명령으로 간단히 컴파일 할 수 있다.

```
[bOOOOOOOOOO@eslab shlab-handout]$ make
gcc -Wall -O2    tsh.c   -o tsh
gcc -Wall -O2    myspin.c   -o myspin
gcc -Wall -O2    mysplit.c   -o mysplit
gcc -Wall -O2    mystop.c   -o mystop
gcc -Wall -O2    myint.c   -o myint
```

- make clean 명령

```
[bOOOOOOOOOO@eslab shlab-handout]$ make clean
rm -f ./tsh ./myspin ./mysplit ./mystop ./myint *.o *~
```

Shell Lab – Trace 검사

❖ sdriver 사용 방법

- ./sdriver.pl -t <trace> -s ./<shell name> -a <arg>
  - ex) tsh 쉘에서 trace01 검사
  - ./sdriver.pl –t trace01.txt –s ./tsh –a “-p”
  - (-a "p" : 쉘이 프롬포트를 나타내지 않는다.)

- ./sdriver.pl -h를 통해 사용 방법과 사용 가능한 옵션을 확인 할 수 있다.

옵 션	설 명
-h	도움말 출력
-v	자세한 동작 과정에 대한 출력
-t <trace>	Trace 파일
-s <shell>	테스트 할 쉘 프로그램
-a <args>	쉘 인자

Shell Lab – Trace 검사

❖ make 명령을 통해서도 테스트 가능

- make <trace>

```
[b000000000@eslab shlab-handout]$ make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
```

❖ 레퍼런스 코드 확인

- 완성되어있는 레퍼런스 쉘을 통해 정상 동작하는 쉘의 모습을 테스트하고 살펴볼 수 있다.
  - ./sderver.pl –t traceXX.txt –s ./tshref –a “-p”
  - make rtestXX

```
[b000000000@eslab shlab-handout]$ make rtest01
./sdriver.pl -t trace01.txt -s ./tshref -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
```

Shell Lab – 쉘 실행

- tsh 쉘의 실행
  - ./tsh를 통해 shlab을 실행할 수 있다.
  - 참고용으로 만들어진 레퍼런스 쉘 또한 같은 방식으로 실행 시킬 수 있다.

```
[bOOOOOOOOO@eslab shlab-handout]$ ./tsh
tsh>
```

  - trace02의 구현 이전에는 'ctrl + d'를 통해 빠져 나와야 한다.

Shell Lab – 작성

- tsh.c 내부의 각 함수들을 작성한다.
  - tsh.c 파일 상단에 자신의 학번과 이름을 필수로 기입한다.

- 쉘의 구성에 있어 핵심이 되는 함수는 eval( )이다.
  - 메인은 eval( )함수 / built-in 명령은 builtin_cmd( )함수 / … /
    - 나머지 도우미 함수들을 사용하여 구성하면 된다.

```
/*
 * 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.  Note:
 * each child process must have a unique process group ID so that our
 * background children don't receive SIGINT (SIGTSTP) from the kernel
 * when we type ctrl-c (ctrl-z) at the keyboard.
*/
void eval(char *cmdline)
{
    return;
}
```

Shell Lab - trace01

❖ **trace01 : EOF(End Of File)가 입력되면 종료.**

```
[b000000000@eslab shlab-handout]$ ./sdriver.pl -v -t trace01.txt -s ./tshref -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
./sdriver.pl: Closing output end of pipe to child 7471
./sdriver.pl: Waiting for child 7471
./sdriver.pl: Child 7471 reaped
./sdriver.pl: Reading data from child 7471
./sdriver.pl: Shell terminated
```

tshref에서 trace01의 동작

```
#
# trace01.txt - Properly terminate on EOF.
#
CLOSE
WAIT
```

trace01.txt

- **trace01은 EOF가 입력되면 쉘이 종료되도록 tsh.c를 내용을 구성하면 된다.**
  - **EOF는 'ctrl + d'를 입력했을 때를 의미한다.**
- **하지만 해당 기능은 main( )에 구현되어 있다.**
  - **따라서 구현하지 않은 tsh도 tshref의 동작과 동일하다.**

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

Shell Lab - trace01

❖ trace01 수행 결과 확인

- 아래와 같이 sdriver를 이용하여 tsh의 trace01을 수행해본다.
- 이때 tshref의 trace01을 수행한 동작과 동일하면 성공이다.

```
[bOOOOOOOOO@eslab shlab-handout]$ ./sdriver.pl -v -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
./sdriver.pl: Closing output end of pipe to child 8197
./sdriver.pl: Waiting for child 8197
./sdriver.pl: Child 8197 reaped
./sdriver.pl: Reading data from child 8197
./sdriver.pl: Shell terminated
```

- 다른 방법으로는 tsh 쉘을 실행시킨 뒤 'ctrl + d'를 입력하였을 때, 종료되는지 확인하는 방법이 있다.
  - 먼저 tshref에서의 동작을 먼저 살펴보는 것이 중요하다.

Shell Lab - trace02

❖ **trace02 : Built-in 명령어 'quit' 구현**

```
[b000000000@eslab shlab-handout]$ ./sdriver.pl -v -t trace02.txt -s ./tshref -a "-p"
#
# trace02.txt - Process builtin quit command.
#
./sdriver.pl: Sending :quit: to child 2089
./sdriver.pl: Waiting for child 2089
./sdriver.pl: Child 2089 reaped
./sdriver.pl: Reading data from child 2089
./sdriver.pl: Shell terminated
```

tshref에서 trace02의 동작

```
#
# trace02.txt - Process builtin quit command.
#
quit
WAIT
```

trace02.txt

- 쉘의 명령어 입력 창에서 'quit'을 입력하면, 쉘이 종료되도록 구현하면 된다.
- built-in 명령어를 구현하는 방법은 다음과 같다.
  - 1. eval( )함수에서 입력 받은 명령어를 파싱한다.
  - 2. 파싱된 명령어를 builtin_cmd( )함수로 전달한다.
  - 3. 해당 명령어가 "quit"인 경우 쉘을 종료할 수 있도록 builtin_cmd( )함수를 구성한다.
- 이 구현과정을 따라 해보면서 built-in 명령 구현하는 방법을 익혀본다.

Shell Lab - trace02

❖ eval( ) 함수에서 입력 받은 명령어를 파싱하고 builtin_cmd( )함수로 전달한다.

```
void eval(char *cmdline)
{
    char *argv[MAXARGS];        // command 저장

    // 명령어를 parseline을 통해 분리
    parseline(cmdline, argv);
    // parsing된 명령어를 전달
    builtin_cmd(argv);

    return;
}
```

ex) tsh > A B CD
argv[0][0] = A
argv[1][0] = B
argv[2][0] = C
argv[2][1] = D

❖ 해당 명령어가 'quit'인 경우 쉘을 종료할 수 있도록 builtin_cmd( )함수를 구성한다.

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

    if (!strcmp(cmd, "quit")){ /* quit command */
        exit(0);
    }

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

Shell Lab - trace02

- tsh.c 파일을 저장하고, make를 통해 빌드 한다.

```
[b000000000@eslab shlab-handout]$ vi tsh.c
[b000000000@eslab shlab-handout]$ make
gcc -Wall -O2    tsh.c   -o tsh
```

- 수정된 tsh 쉘을 sdriver를 통해 제대로 구현이 되었는지 테스트해본다.
  - tshref를 테스트한 동작결과와 동일하면 성공

```
[b000000000@eslab shlab-handout]$ ./sdriver.pl -v -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
./sdriver.pl: Sending :quit: to child 2456
./sdriver.pl: Waiting for child 2456
./sdriver.pl: Child 2456 reaped
./sdriver.pl: Reading data from child 2456
./sdriver.pl: Shell terminated
```

- 또한 직접 tsh 쉘을 실행시키고 'quit' 명령을 입력해본다.
  - 정상적으로 종료가 되는지 확인

```
[b000000000@eslab shlab-handout]$ ./tsh
tsh> quit
[b000000000@eslab shlab-handout]$
```

Shell Lab - trace03

❖ **trace03 : Foreground 작업 형태로 프로그램 실행**

```
[b000000000@eslab shlab-handout]$ ./sdriver.pl -v -t trace03.txt -s ./tshref -a "-p"
#
# trace03.txt - Run a foreground job.
#
./sdriver.pl: Sending :/bin/echo tsh> quit: to child 2794
./sdriver.pl: Sending :quit: to child 2794
./sdriver.pl: Reading data from child 2794
tsh> quit
./sdriver.pl: Shell terminated
```

tshref에서 trace03의 동작

```
#
# trace03.txt - Run a foreground job.
#
/bin/echo tsh> quit
quit
```

trace03.txt

- 프로그램을 foreground 형태로 실행시키면 된다.
  - 이때 실행되는 프로세스는 매개변수를 가질 수도 있고, 그렇지 않을 수도 있다.
  - 해당 테스트를 살펴보면 echo를 foreground 형태로 실행하는 것을 볼 수 있다.

- 쉘에서 새로운 프로그램을 실행시키기 위한 방법은 다음과 같다.
  - fork( )를 통해 자식 프로세스를 생성
  - 자식 프로세스에서 execve( )를 이용해 새로운 프로그램 실행

Shell Lab - trace03

❖ 다음 코드를 참조하여 trace03을 해결하는 쉘 코드를 구현해본다.

```
void eval(char *cmdline)
{
    char *argv[MAXARGS];          // command 저장
    pid_t pid;                    // process ID

    parseline(cmdline, argv);

    if (!builtin_cmd(argv)){
        if( /* 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);
            }
        }
    }
    return;
}
```

- **Hint**
  - fork를 통해 자식 프로세스를 생성하고, 실행되고 있는 프로세스가 자식 프로세스인지 확인.

- **execve( )사용법을 교과서에서 참고하고, 설명을 보고서에 첨부**

Shell Lab - trace04

❖ **trace04 : Background 작업 형태로 프로그램 실행**

```
[b000000000@eslab shlab-handout]$ ./sdriver.pl -v -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
./sdriver.pl: Sending :/bin/echo -e tsh> ./myspin 1 \046: to child 2803
./sdriver.pl: Sending :./myspin 1 &: to child 2803
./sdriver.pl: Reading data from child 2803
tsh> ./myspin 1 &
[1] (2805) ./myspin 1 &
./sdriver.pl: Shell terminated
```

tshref에서 trace04의 동작

```
#
# trace04.txt - Run a background job.
#
/bin/echo -e tsh> ./myspin 1 \046
./myspin 1 &
```

trace04.txt

- 프로그램을 background 형태로 실행시키면 된다.
  - 해당 테스트를 살펴보면 echo를 foreground 형태로 실행한다.
  - 이때 실행되는 프로세스는 매개변수를 가질 수도 있고, 그렇지 않을 수도 있다.

Shell Lab - trace04

- Background 형태로 프로그램을 실행시키려면,
  - 쉘에서 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 구현하면 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.

Shell Lab - trace04

❖ 작업 관리와 관련된 함수들은 다음과 같다.

함 수	설 명
initjobs(struct job_t *jobs)	작업 리스트 초기화
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)	작업 리스트를 출력

- 해당 함수에 대한 자세한 동작은 코드분석을 통해 알아낼 것

Shell Lab - trace04

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

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

- **foreground 작업**인 경우
  - **자식 프로세스가 종료될 때까지 기다린다**. 자식 프로세스가 종료되면 작업 리스트에서 작업을 제거한다.
    - waitpid( )함수를 활용하여, 자식 프로세스가 종료될 때까지 기다린다.
- **background 작업**인 경우
  - 해당 작업의 정보를 출력하는 양식을 확인하고 **해당 양식에 맞추어 출력**할 수 있도록 프린트 문을 구성한다.

Shell Lab - trace05

❖ **trace05 : Built-in 명령어 'jobs' 구현**

```
[b000000000@eslab shlab-handout]$ ./sdriver.pl -v -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
./sdriver.pl: Sending :/bin/echo -e tsh> ./myspin 2 \046: to child 2814
./sdriver.pl: Sending :./myspin 2 &: to child 2814
./sdriver.pl: Ignoring blank line
./sdriver.pl: Sending :/bin/echo -e tsh> ./myspin 3 \046: to child 2814
./sdriver.pl: Sending :./myspin 3 &: to child 2814
./sdriver.pl: Ignoring blank line
./sdriver.pl: Sending :/bin/echo tsh> jobs: to child 2814
./sdriver.pl: Sending :jobs: to child 2814
./sdriver.pl: Reading data from child 2814
tsh> ./myspin 2 &
[1] (2816) ./myspin 2 &
tsh> ./myspin 3 &
[2] (2818) ./myspin 3 &
tsh> jobs
[1] (2816) Running ./myspin 2 &
[2] (2818) Running ./myspin 3 &
./sdriver.pl: Shell terminated
```

tshref에서 trace05의 동작

```
#
# trace05.txt - Process jobs builtin command.
#
/bin/echo -e tsh> ./myspin 2 \046
./myspin 2 &

/bin/echo -e tsh> ./myspin 3 \046
./myspin 3 &

/bin/echo tsh> jobs
jobs
```

trace05.txt

Shell Lab - trace05

- trace05
  - 두 개의 background 작업을 실행한 후, built-in 명령어 'jobs'을 실행한다. 해당 명령어를 입력 받으면, 현재 실행되고 있는 작업의 리스트를 출력해준다.
  - 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.

Shell Lab - 예외처리

- 시스템 콜 호출 시, 예외가 전송하면 unix_error 함수(tsh.c에 구현되어 있음)를 이용하여 오류 번호에 따른 오류 메시지를 출력할 수 있도록 한다.
  - ex) fork( )로 프로세스 생성에 실패한 경우의 예외처리

```
/* Create a child process */
if ((pid = fork()) < 0)
    unix_error("fork error");
```

- 쉘 프로그램에서 예외가 발생할 때, app_error 함수(tsh.c에 구현되어 있음)를 이용하여 단순한 에러 메시지를 출력할 수 있도록 한다.
  - ex) tsh.c의 main 함수에 구현되어 있는 쉘 커멘드라인에 대한 예외처리

```
if ((fgets(cmdline, MAXLINE, stdin) == NULL) && ferror(stdin))
        app_error("fgets error");
```

주의 사항

- Shell Lab을 수행하다 보면, 부모 프로세스가 자식 프로세스보다 먼저 종료되어 자식 프로세스가 좀비가 되는 경우가 있다.
  - 쉘 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! (쉘 테스트 후 실시간 체크 바람)

제출 사항

❖ 제출 내용

- 본인이 수행한 shlab-handout 디렉토리를 통째로 압축
  - 압축파일명 : [sys02]shell_학번.tar.gz
- trace01~05에 대한 보고서 작성
  - 각 trace 별, 해결 방법에 대한 설명
  - tshref를 수행한 결과와 본인이 구현한 tsh와의 동작 일치를 증명
    - sdriver 수행결과와 tsh에서의 정상동작 모습 (-v 옵션 사용)
  - 각 trace 별, 플로우차트
    - 간단한 수행과정을 플로우차트로 나타내면 됨.
- 압축파일과 보고서를 조교메일로 제출 (보고서는 서면으로도 제출)
  - sihyeong@cnu.ac.kr
  - 제목 : [sys02]HW07_학번_이름

❖ 제출 일자

- 메일 제출 : 2014년 11월 19일 23시 59분 59초
- 서면 제출 : 2014년 11월 20일 수업시간