



2021 Fall System Programming

Shell Lab 1

2021. 11. 01

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

❖ 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.

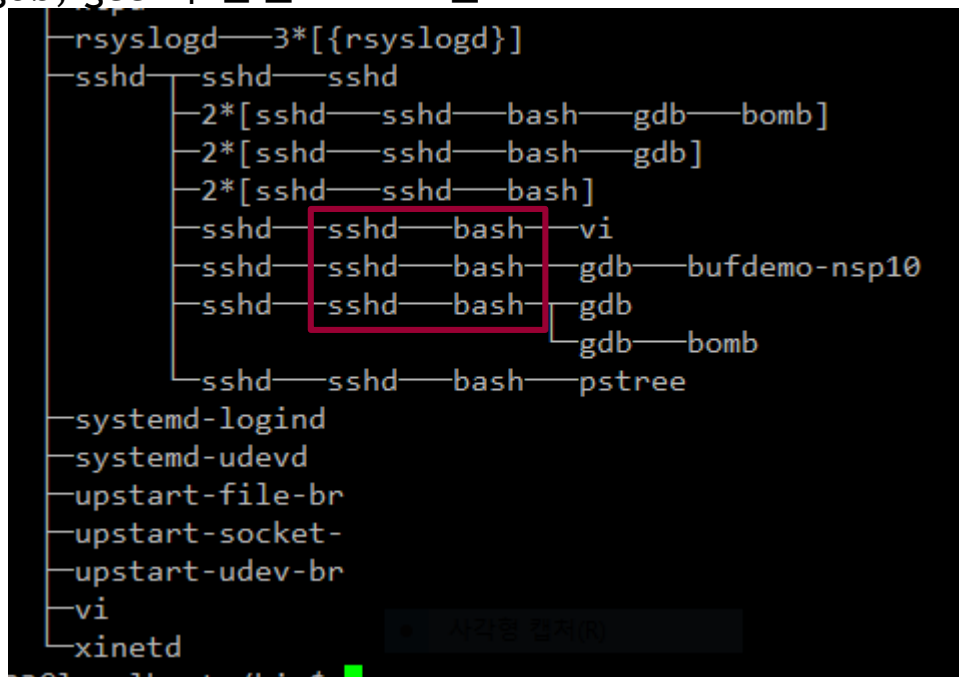
❖ 구현사항

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



Shell 이란?

- ❖ 사용자와 시스템간의 인터페이스 역할을 하는 프로그램
 - ◆ vi, gdb, gcc 와 같은 프로그램

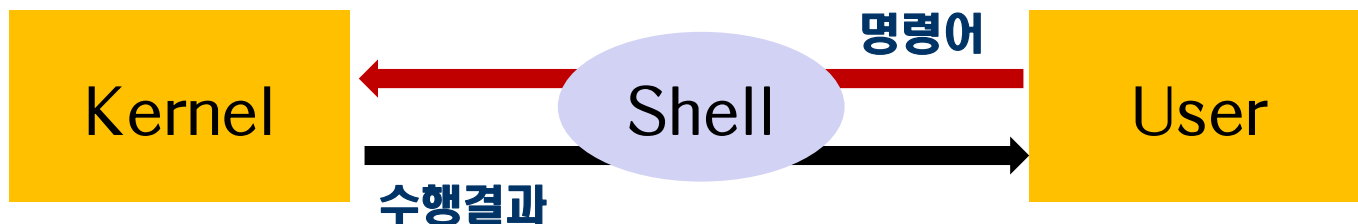


pstree 명령어 입력시 나오는 화면(한번 썩 입력해보세요)

- ◆ 실습을 위해 Putty로 접속을 하면, sshd 라는 원격접속 프로그램이 실행되고, **bash**라는 linux의 기본 shell이 실행된다.

Shell 이란?

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



- ❖ Shell의 기능
 - ◆ 명령어 해석기 기능
 - ❖ 사용자가 입력한 명령어를 해석하고 커널에 전달하는 역할
 - ❖ “ls” 명령어를 입력하면, “/bin/” 디렉토리 밑의 ls 프로그램을 실행시켜줌.
 - ❖ bash shell에서 vi를 입력하면, “/usr/bin/” 디렉토리 밑의 vi 프로그램을 찾아서 실행함.
 - ❖ “/bin/”, “/usr/bin/”과 같은 실행 프로그램의 위치는 “환경 변수”에 포함되어있음.
 - ◆ 프로그래밍 기능
 - ❖ 자체적인 프로그래밍 기능을 통해 프로그램 작성 가능
 - ❖ 셸 프로그램을 셸 스크립트라고 부름
 - ◆ 사용자 환경설정 기능
 - ❖ 초기화 파일을 이용해 사용자 환경을 설정



Shell Lab

- ❖ 프로세스의 관리와 시그널의 제어에 대해 이해하고, 작업의 제어를 지원하는 Unix Shell program을 작성하는 것을 목표로 한다.
- ❖ Shell Lab은 **trace00** 부터 **trace21(미정)** 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 : trace00 ~ 02**
 - ◆ Shell Lab 2주차 : trace..
 - ◆ Shell Lab 2주차 : trace..



Shell Lab - 준비

- ❖ Shell Lab 파일 복사 및 압축해제
 - ◆ `cp /home/sys01/sys01/week06/shlab-handout.tar.gz ~`
 - ◆ `tar xvzf ~/shlab-handout.tar.gz`



Shell Lab - 준비

❖ Shell Lab 파일 구성

파 일	설 명
Makefile	셸 프로그램의 컴파일 및 테스트
README	도움말
tsh.c	Shell 프로그램의 소스코드
tshref	Shell binary의 레퍼런스(다음의 실행함수로 정답을 실행할 수 있음!!)
sdriver	Shell에 각 trace 들을 실행하는 프로그램
trace{00-21}.txt	Shell 드라이버를 제어하는 22개의 trace
myspin.c, mysplit.c, mystop.c myint.c	trace 파일들에서 불러지는 C로 작성된 프로그램 (README 참조)



Shell Lab - 준비

❖ tsh.c 파일 내부 구성

함 수	설 명
eval	명령을 파싱 하거나 해석하는 메인 루틴
builtin_cmd	quit, fg, bg와 jobs 같은 built-in 명령어를 해석
waitfg	Foreground 작업이 완료될 때 까지 대기
sigchld_handler	SIGCHLD 시그널 핸들러
sigint_handler	SIGINT(ctrl-c) 시그널 핸들러
sigstp_handler	SIGTSTP(ctrl-z) 시그널 핸들러



Shell Lab – Trace (1주차)

- ❖ sdriver를 이용하여 Trace{00-02}를 테스트 할 수 있다.
 - ◆ 각 trace의 자세한 내용은 해당 파일을 열어서 확인할 수 있다.



Shell Lab – 시작 하기

❖ shell lab 소스파일 수정

- ❖ tsh.c 파일을 열어 맨 위쪽에 자신의 **학번**과 **이름**을 **기입**한다.
- ❖ tsh.c 파일 내부를 수정 또는 추가 작성하여 Shell Lab의 요구사항(각 trace 별 요구사항)을 해결해 나간다.

❖ shell lab 빌드

- ❖ make 명령을 통해 tsh.c을 컴파일 한다.
- ❖ 결과로 **tsh** 란 실행 파일이 생성된다.
 - ❖ **tsh** 은 본인이 작성하여 완성된 셸의 본체

❖ shell lab 실행 및 테스트

- ❖ 셸은 다음과 같이 실행한다.
 - ❖ **./tsh**
 - ❖ 참고로 **./tshref** 의 실행을 통해 정상적으로 완성된 셸을 경험할 수 있다.
- ❖ **'sdriver'**를 이용하여 생성된 셸이 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.

```
[b000000000@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

```
[b000000000@eslab shlab-handout]$ make clean
rm -f ./tsh ./myspin ./mysplit ./mystop ./myint *.o *
```



Shell Lab – Trace 검사

❖ sdriver 사용 방법

- ◆ `./sdriver -t <trace number> -s ./<shell name>`
 - ❖ ex) tsh 셸에서 trace00 검사
 - ❖ `./sdriver -t 00 -s ./tsh`
 - ◆ Shell name에 ./tshref를 하면 레퍼런스 셸을 확인한 것.
- ◆ `./sdriver.pl -h`를 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 number>	Trace 번호
-s <shell>	테스트 할 셸 프로그램



Shell Lab – Trace 검사

❖ 레퍼런스 코드 확인

- ◆ 완성되어있는 레퍼런스 셸을 통해 정상 동작하는 셸의 모습을 테스트하고 살펴볼 수 있다. (제대로 구현했다면 출력될 모습을 참고할 때 사용)
 - ❖ `./sdriver -t XX -s ./tshref`
- ◆ 본인이 구현한 코드를 확인하려면 make 한 후, `./tshref`가 아닌!! `./tsh`로 입력해야 한다.



Shell Lab – 셸 실행

❖ tsh 셸의 실행

- ♦ ./tsh를 통해 shlab을 실행할 수 있다.
- ♦ 참고용으로 만들어진 레퍼런스 셸 또한 같은 방식으로 실행 시킬 수 있다.

```
[b0000000000@eslab shlab-handout]$ ./tsh  
tsh> █
```

- ♦ trace01의 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.
- ❖ 셸의 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 – trace00

- ❖ **trace00** : EOF(End Of File)가 입력되면 종료.

```
sys03@localhost:~/workspace/shlab-handout$ ./sdriver -V -t 00 -s ./tsh
Running trace00.txt...
Success: The test and reference outputs for trace00.txt matched!
Test output:
#
# trace00.txt - Properly terminate on EOF.
#

Reference output:
#
# trace00.txt - Properly terminate on EOF.
#
```

- ♦ trace00은 EOF가 입력되면 셸이 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 – trace00

❖ trace00 수행 결과 확인

- ◆ 아래와 같이 sdriver를 이용하여 tsh의 trace00을 수행해본다.
- ◆ 이때 tshref의 trace00을 수행한 동작과 동일하면 성공이다.

```
sys03@localhost:~/workspace/shlab-handout$ ./sdriver -V -t 00 -s ./tsh
Running trace00.txt...
Success: The test and reference outputs for trace00.txt matched!
Test output:
#
# trace00.txt - Properly terminate on EOF.
#

Reference output:
#
# trace00.txt - Properly terminate on EOF.
#
```

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



Shell Lab – trace01

❖ trace01 : Built-in 명령어 'quit' 구현

```
sys03@localhost:~/workspace/shlab-handout$ ./sdriver -V -t 01 -s ./tsh
Running trace01.txt...
Success: The test and reference outputs for trace01.txt matched!
Test output:
#
# trace01.txt - Process builtin quit command.
#

Reference output:
#
# trace01.txt - Process builtin quit command.
#
```

- ◆ 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 – trace01

- ❖ `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 셸을 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 – trace01

- ❖ tsh.c 파일을 저장하고, make를 통해 빌드 한다.

```
sys00@2019sp:~/TestDir/shelllab/shlab-handout$ vi tsh.c
sys00@2019sp:~/TestDir/shelllab/shlab-handout$ make
```

- ❖ 수정된 tsh 셸을 sdriver를 통해 제대로 구현이 되었는지 테스트해본다.
 - ◆ tsh를 테스트한 동작결과와 동일하면 성공

```
sys00@2019sp:~/TestDir/shelllab/shlab-handout$ ./sdriver -V -t 01 -s ./tsh
Running trace01.txt...
Success: The test and reference outputs for trace01.txt matched!
Test output:
#
# trace01.txt - Process builtin quit command.
#
Reference output:
#
# trace01.txt - Process builtin quit command.
#
```

- ❖ 또한 직접 tsh 셸을 실행시키고 ‘quit’ 명령을 입력해본다.
 - ◆ 정상적으로 종료가 되는지 확인

```
sys00@2019sp:~/TestDir/shelllab/shlab-handout$ ./tsh
eslab_tsh> quit
sys00@2019sp:~/TestDir/shelllab/shlab-handout$ █
```



참조 - execve

- ❖ `execve(const char *path, const char *argv[], const char *envp[])`
 - ◆ 다른 프로그램을 실행하고 자신은 종료하는 함수
 - ◆ `#include <unistd.h>` 해야 사용 가능.
 - ◆ 현재 프로세스를 `execve` 함수로 호출한 프로그램으로 교체.
 - ❖ 호출한 프로그램의 텍스트, 데이터, bss, 스택이 호출된 프로그램의 것으로 교체됨.
 - ❖ PID와 열린 파일디스크립터 등은 호출한 프로그램것을 상속받음.
 - ◆ `*path`: 실행할 프로그램의 전체 경로를 입력
 - ❖ `"/bin/ls"`
 - ◆ `*argv[]`: `path`에 입력한 프로그램에 넘겨줄 인자를 배열 형태로 입력.
 - ❖ `char *arg[] = {"/bin/ls", "-a", "0"};`
 - ◆ `*envp[]`: 환경변수를 입력.

```
1 #include <unistd.h>
2 #include <stdio.h>
3
4 int main(void) {
5     char *arg[] = {"ls", "-l", 0};
6     char *env[] = {0};
7     execve("/bin/ls", arg, env);
8     return 0;
9 }
```

```
sys03@localhost:~/workspace$ ./test.out
total 32
-rw-r--r-- 1 sys03 sudo 171 Nov  4 15:40 fbprocess.c
-rwxr-xr-x 1 sys03 sudo 8576 Nov  4 15:40 fbprocess.out
-rw-r--r-- 1 sys03 sudo 153 Nov  4 17:39 test.c
-rwxr-xr-x 1 sys03 sudo 8520 Nov  4 17:39 test.out
sys03@localhost:~/workspace$
```



Shell Lab – trace02

❖ trace02 : Foreground 작업 형태로 프로그램 실행

```
sys03@localhost:~/workspace/shlab-handout$ ./sdriver -V -t 02 -s ./tsh
Running trace02.txt...
Success: The test and reference outputs for trace02.txt matched!
Test output:
#
# trace02.txt - Run a foreground job that prints an environment variable
#
# IMPORTANT: You must pass this trace before attempting any later
# traces. In order to synchronize with your child jobs, the driver
# relies on your shell properly setting the environment.
OSTYPE=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

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



Shell Lab – trace02

- ❖ 다음 코드를 참조하여 trace02을 해결하는 셸 코드를 구현해본다.

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

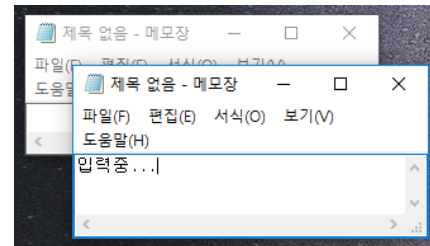
    parseline(cmdline, argv);

    if(!builtin_cmd(argv)){
        if(/*Child process 체크*/ ){ //Child Process인 경우, execve() 수행
            if( execve(argv[0], argv, environ) <0){
                printf("%s : Command not found\n\n", argv);
                exit(0);
            }
        }
        usleep(100000);
    }
    return 0 ;
}
```

- ◆ /* child process 체크 */ 부분을 작성하면 됨.
- ◆ **Hint**
 - ❖ fork() 함수를 통해 자식 프로세스를 생성하고, 실행되고 있는 프로세스가 자식 프로세스인지 확인.
 - ❖ Child proces인 경우 pid는 0.

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 PID
sys03@localhost:~/workspace$ result=5

[1]+  Done                  ./fbprocess.out
sys03@localhost:~/workspace$
  
```



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 만들지 마세요! (셸 테스트 후 항상 실시간 체크 바람)**



과제 유의사항

1. 이번 과제는 3주에 걸쳐서 진행을 할 예정이기 때문에 **3주차 과제 제출 시 한번에 제출해주셔야** 합니다. 한번에 수행하려고 한다면 강의도 따라올 수 없고, 양도 상당하기 때문에 해당 주차 과제는 미리 하시기 바랍니다.
2. 과제에서 **grace day**의 사용을 문의하시는 분들이 많아서 다시 설명합니다. **grace day**를 통해서 4일간 감점을 안하는 것은 4개의 lab 과제를 수행하면서 과제 제출 기간 내에 제출하지 못하였다면, **원래 늦은 날만큼 감점이 들어가지만, 4일 지각의 경우를 제외하고는 감점을 하지 않는 것입니다.**

ex. 2일 지각일 시 30% 감점이지만 grace day 2일 차감 & 100% 처리

1일 지각 (화요일 08:59까지) 15%

2일 지각 (수요일 08:59까지) 30%

3일 지각 (목요일 08:59까지) 45%

4일 지각 (금요일 00시부터) 0점 ← 과제 기간 만료, 제출 불가



과제(이번주 제출 아닙니다.)

1. Shell Lab

- I. trace00 ~ trace02에 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(-v 옵션 사용)
- 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]ShellLab01_학번.tar.gz
- II. 결과 보고서를 작성
 - 1) 파일명: [sys01]ShellLab01_학번.pdf
- III. I.과 II. 두개를 하나로 압축
 - 1) 파일명:[sys01]ShellLab01_학번_이름.zip