**2022-2 CSED211 Lab10 Report**

학번 : 20210479

이름 : 이주현

**명예서약 (Honor Code)**

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

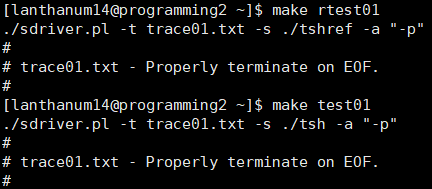
1. **코드 설명**

아래의 함수 설명은 최종 구현을 바탕으로 하였으며, tsh.c에서 직접 구현한 함수를 사용한 경우에는 bold 처리하여 두었다.

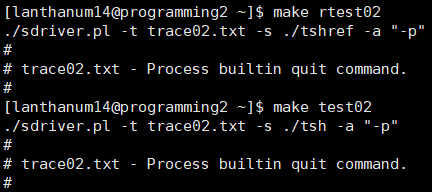
* + **eval(char \*cmdline)**
    - 우선 eval 함수에서는 입력받은 command line을 parseline()의 매개변수로 넘겨 명령을 분석한다. 이때 명령을 입력하지 않아 argv[0]인 경우에 대해서는 ignore하고 프로그램을 종료할 수 있도록 예외 처리를 해주었다.
    - 만약 이 명령이 background에서 실행되는 것이라면 ifBg 변수를 통해 그 여부를 저장하도록 했다. 그리고 builtin\_cmd() 함수의 반환값을 통해 builtin command를 입력한 것인지를 확인해 주었다.
    - 만일 builtin command가 아닐 시, SIGCHLD를 block한 뒤, fork() 함수를 수행해주었다. 수행 중 만일 child process인 경우 setgpid(0, 0)을 호출해 자신이 새로운 프로세스 그룹의 리더가 된다. 이후 sigprocmask를 이용해 block을 해제하고, execve()를 호출한 후 자식 프로세스를 실행시킨다. 이때 해당하는 command가 없는 경우에도 오류 메시지를 출력하고 프로그램을 종료하도록 했다.
    - 만약 builtin command가 아니고, foreground에서 수행해야 하는 job인 경우에는 job의 state를 FG로 바꾸고, addjob()을 호출한 뒤 SIHCHLD를 unblock해 주었다. 이후 **waitfg()** 함수를 호출해 process가 foreground가 아닐 때까지 기다리도록 해 주었다.
    - 만약 builtin command가 아니고 background에서 수행해야 하는 job인 경우에는 state를 BG로 하여 addjob()을 호출해 주었다. 이때는 background job이라는 특성 상 다른 command를 받는 것이 가능하기 때문에 따로 다른 함수를 호출해주지 않았다.
  + **builtin\_cmd(char \*\*argv)**
    - parameter로 넘겨받은 argv을 가지고 strcmp() 함수를 호출해 입력받은 command가 builtin인지 확인해 주었다.
    - argv[0]의 내용에 따라, ‘quit’이면 프로그램을 종료하고, ‘fg’, ‘bg’인 경우 **do\_bgfg()**를 호출하고, ‘jobs’인 경우에는 listjobs(jobs)를 호출해 주었다.
    - 입력한 command가 ‘fg’, ‘bg’, ‘jobs’인 경우 그 반환값은 1이며, ‘quit’이나 그 외의 command인 경우 반환값은 0이다.
  + **do\_bgfg(char \*\*argv)**
    - 먼저 실제 동작에 앞서 ‘bg’ / ‘fg’ 이후에 나오는 argument가 존재하지 않거나 형식에 맞지 않는 경우에는 에러 메시지를 출력하고 return하도록 각각 예외처리 해주었다.
    - 만약 입력받은 argument가 jid인 경우, getjobjid()를 이용해 job\_t\* 형태의 job을 얻어 주었다. 이때 입력받은 jid에 해당하는 job이 없을 경우에는 error message를 출력하고 return할 수 있도록 예외 처리를 해 주었다.
    - 만약 입력받은 argument가 pid인 경우 getjobpid()를 이용해 job\_t\* 형태의 pointer를 얻어주었다. 이때 입력받은 pid에 해당하는 job이 없을 경우에는 error message를 출력하고 return할 수 있도록 해 주었다.
    - 만약 argv[0]이 ‘bg’일 경우 state를 BG로 설정하여 job의 상태를 출력하고, argv[0]이 ‘fg’인 경우에는 state를 FG로 설정해 주었다. 이후 두 동작 모두에서 해당 process group에 kill 함수를 통해 SIGCONT를 전달해 주었다.
  + **waitfg(pid\_t pid)**
    - foreground의 job인 경우 **waitfg()**를 실행하면, while문을 돌면서 sleep()함수로 호출한 만큼의 시간 간격을 두고 foreground의 pid값이 parameter로 받은 pid와 같은지 확인한다. 만약 두 값이 같으면 while문을 지속하며, 같지 않은 경우 해당 job이 완료되었음을 의미하므로 return한다.
  + **sigchld\_handler(int sig)**
    - waitpid()의 옵션을 “ WNOHANG | WUNTRACED “로 두어 자식 프로세스가 중단 혹은 종료되었을 경우 아래의 세 가지 중 하나의 코드를 따르도록 했다.
    - WIFEXITED: deletejob()을 호출해 주었다.
    - WIFSTOPPED: 몇 번째 시그널에 의해 종료된 것인지 출력한 후, waitpid()의 반환값으로 받아온 pid를 이용해 getjobpid()를 호출하고, 해당하는 job의 state를 ‘ST’로 바꾸어 주었다.
    - WIFSIGNALED: 몇 번째 시그널에 의해 종료된 것인지 출력한 후, waitpid()에서 반환받은 pid를 parameter로 넘겨 deletejob()으로 해당 job을 삭제해 주었다.
  + **sigtstp\_handler(int sig)**
    - foreground가 아닐 경우 pid 값이 0일 것이므로 그냥 그대로 반환해 주었다. 이것을 if 조건문으로 구현해 주었다.
    - foreground인 경우에는 해당되는 pid의 progress group에 signal을 보내 주었다.
  + **sigint\_handler(int sig)**
    - foreground가 아닐 경우 pid 값이 0일 것이므로 이 경우 그냥 return하도록 만들어 주었다.
    - 만약 foreground인 경우에는 pid가 속한 process group에 signal을 보내 주었다.

1. **parse{0-16}.txt 수행 결과**

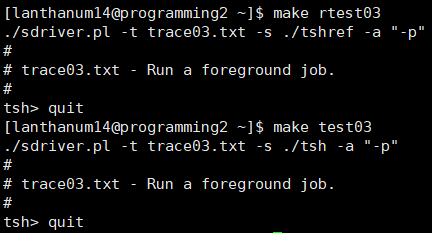
* 모든 결과는 tshref 🡪 tsh 순서로 출력하였다.
* **parse1.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.



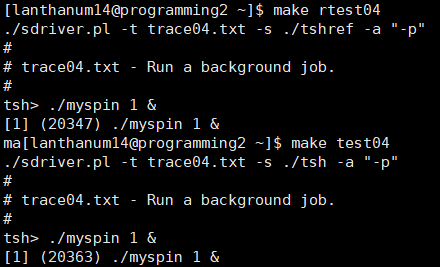
* **parse2.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.



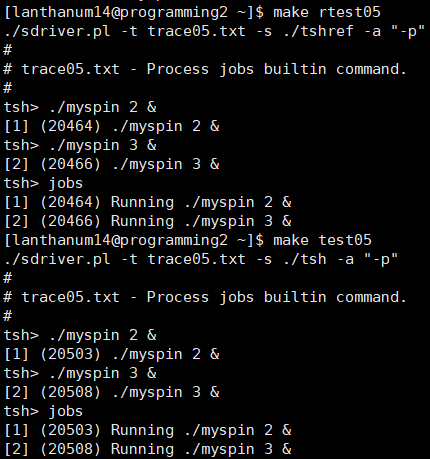
* **parse3.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.



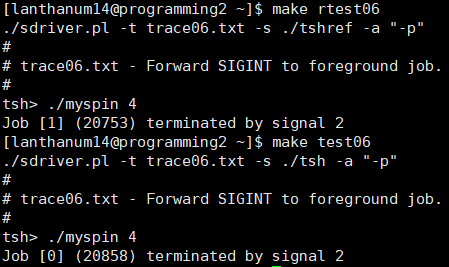
* **parse4.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.



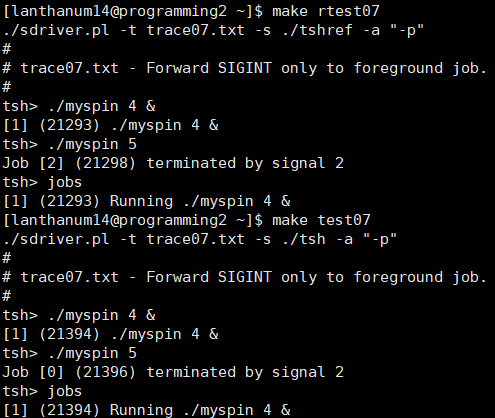
* **parse5.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.



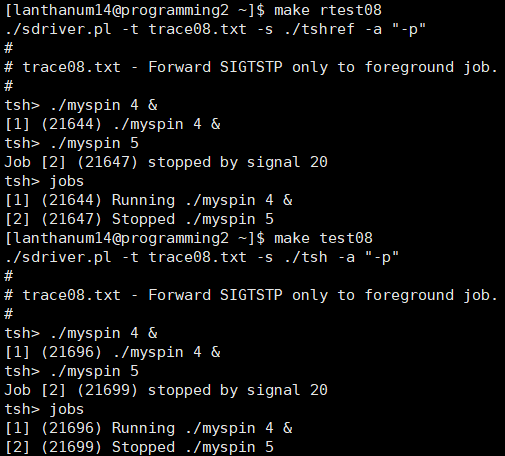
* **parse6.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.



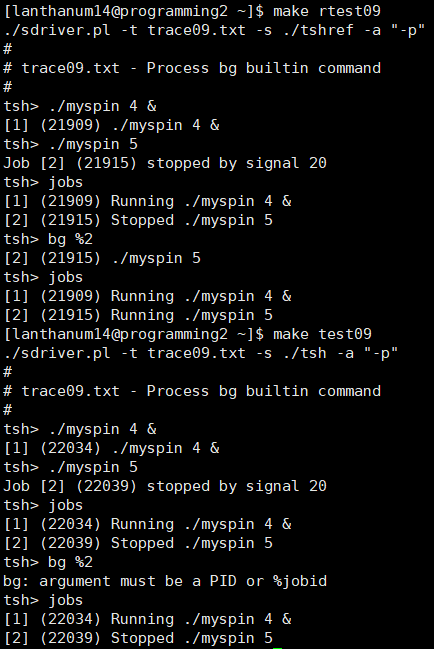
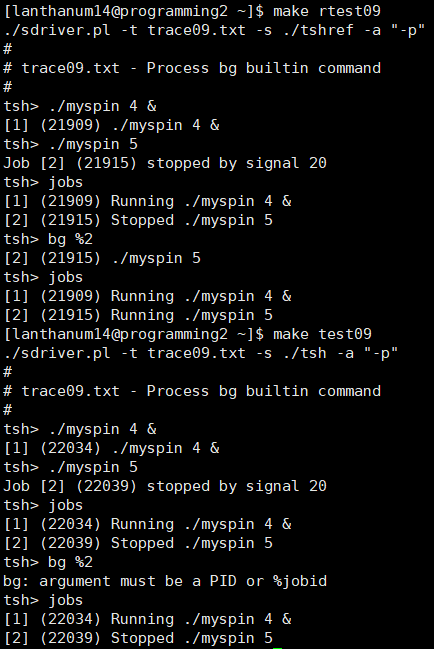
* **parse7.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.



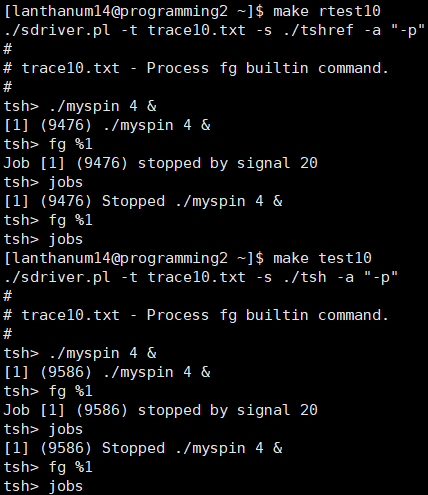
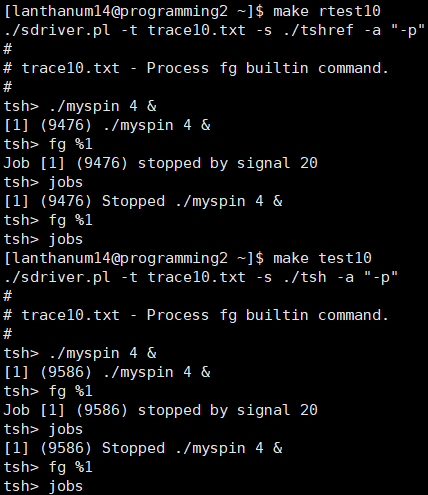
* **parse8.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.



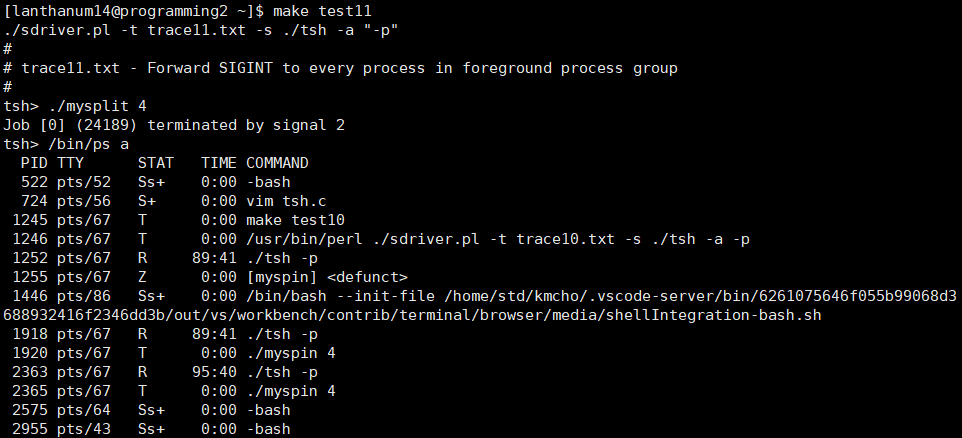
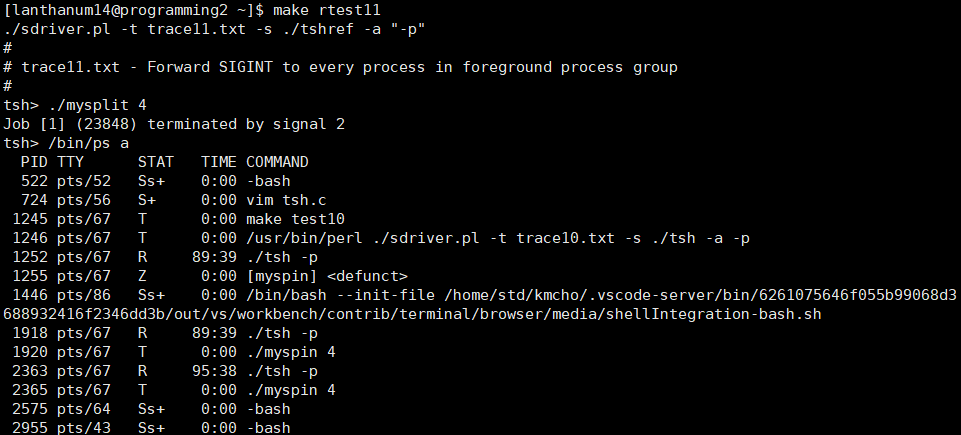
* **parse9.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.

* **parse10.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.

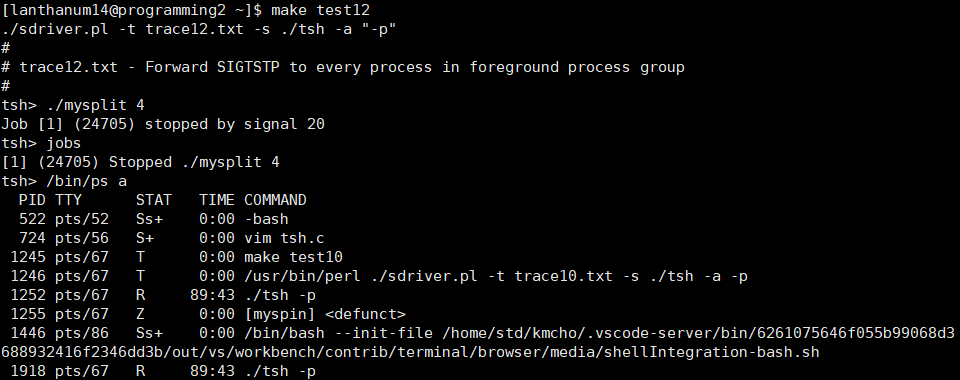
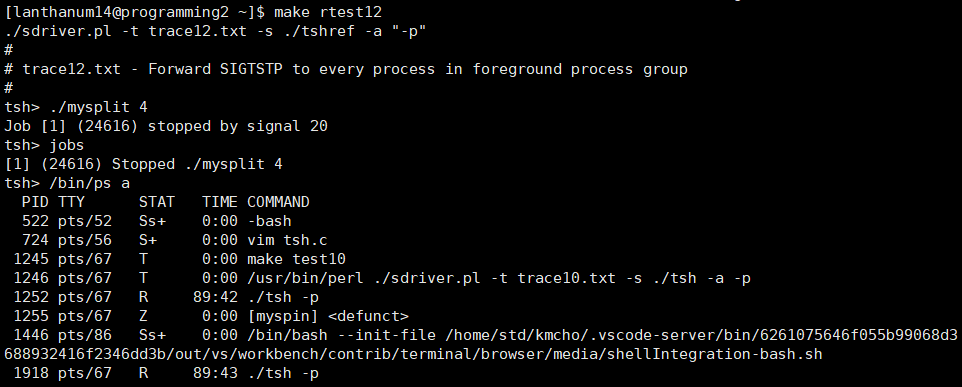
 

* **parse11.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.



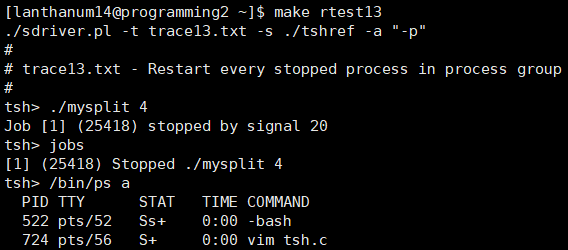
(캡쳐본 이하 출력 내용 생략)

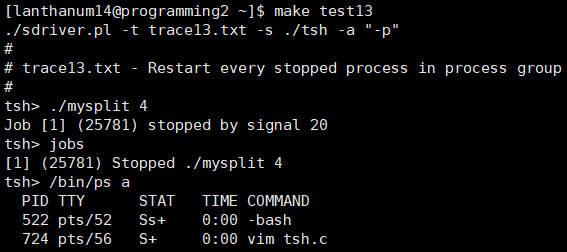
* **parse12.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.



(캡쳐본 이하 출력 내용 생략)

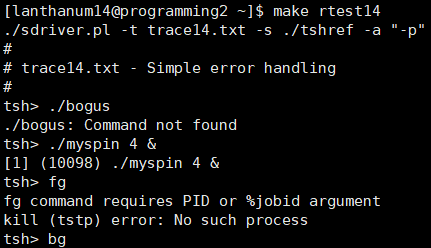
* **parse13.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.

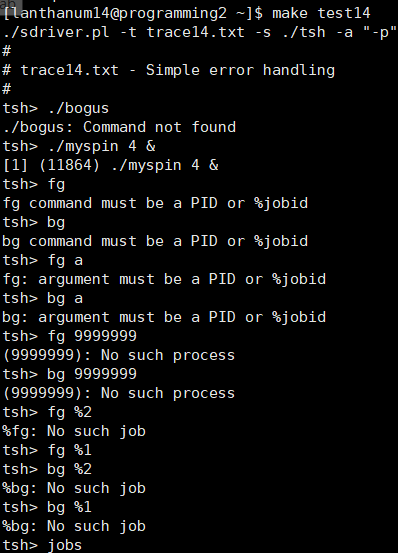




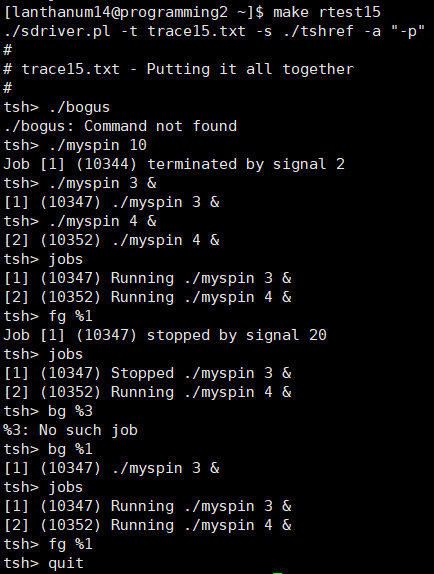
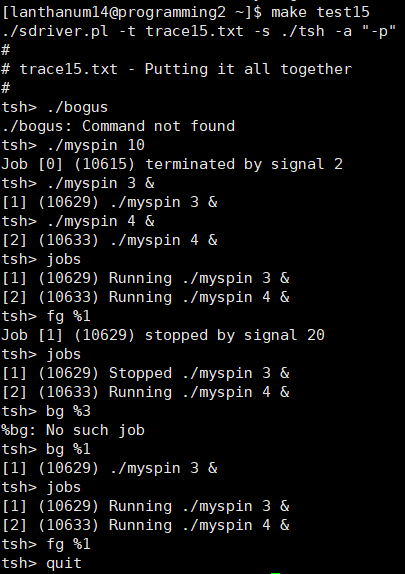
(캡쳐본 이하 출력 내용 생략)

* **parse14.txt**: 아래와 같이 문제가 생겼음을 알 수 있다. 추측 가능한 원인으로는 command를 입력함에 있어 계속하여 pid, jid를 입력하지 않았다는 것과, 존재하지 않는 job을 계속해서 부른 것을 꼽을 수 있을 것 같다.

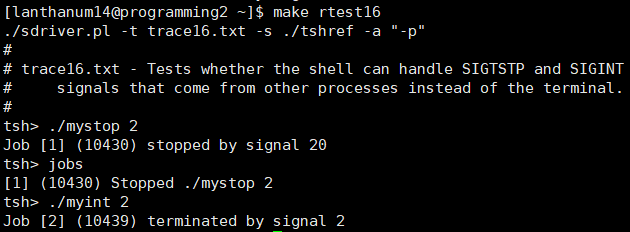


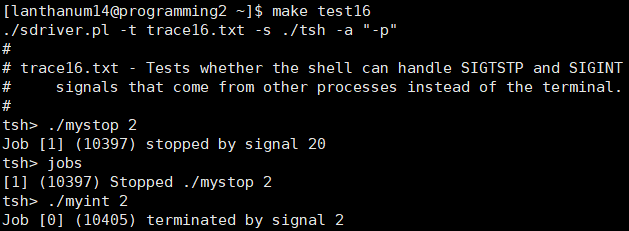


* **parse15.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.

* **parse16.txt**: 아래와 같이 동일하게 출력되는 것을 확인할 수 있다.





1. **Lab10에서 배운 점**

* Lab 10 Shell Lab을 해결하면서 process들을 컨트롤하는 방법과 signaling 방법을 익힐 수 있었다. 그리고 이해한 바를 바탕으로 간단한 shell을 구현해 보며 shell의 개념과 역할, 기능 등을 복습할 수 있었다.