20210815\_JaeHyeonJeong

**eval**

커맨드가 올바르게 받았는지 판단하고 나서, builtin\_cmd를 이용하여 builtin인지 확인한다. Builtin 커맨드의 경우 이전 과정에서 해당 커맨드를 실행할 수 있다. 반면 builtin 커맨드가 아닐 경우 foreground부터 살펴보면, state를 fg로 한다. 이때 addjob을 이용하여 SIGCHLD를 언블락하고, waitfg를 사용하여 기존 pid의 작업이 exit될때 까지 wait한다. 이어서 background의 경우 state를 bg로 하고, SIGCHLD를 블락하고, fork를 진행한다. Child process는 setpgid를 이용하고, sigprocmask로 블락을 해제하고 child process를 실행하고, 오류가 있을 경우 메세지를 띄워주고 return한다.

**builtin cmd**

argv[0]에 대해서 살펴본다. 먼저 fg,bg,jobs에 대해 살펴보면 fg,bg는 do\_bgfg를 콜한다. 이어서 jobs는 listjobs를 콜한다. 이 세가지의 경우 모두 return 1을 하게 되고 나머지의 경우는 builtin 커맨드가 아니기 때문에 return 0를 한다. 여기서 만약 quit가 나온다면 program을 끄게 된다. 이후 builtin 커맨드는 해당 명령을 수행 후 return 1을 해준다. 결국 정리해보면 quit,jobs,fg,bg 는 return 1이었고, 나머지는 return 0를 한다.

**do bgfg**

%가 붙은 jid혹은 숫자로 된 PID를 받아서 상응하는 job을 가져온다. 이때 jid는 getjobjid로, pid는 getjobpid로 받아온다. 이때, 상응하는 것이 없거나 적절하지 않을 때 error 메세지를 띄워주고 return 한다. Argv[0]에 대해서 fg 커맨드는 job의 state를 FG로 변경하고, foreground 작업이 exit까지 wait하고 kill 함수로 SIGCONT를 보낸다. 반면, bg 커맨드는 job의 state를 BG로 변경하고 job을 띄워준다.

**waitfg**

current foreground job이 exit해야만 Foreground job은 next 커맨드를 전달 받는다. 해당 함수는 pid를 arguement로 전달받고 current foreground의 pid와 비교하고 같지 않을 경우 return한다. 같은 지 비교하는 시간의 interval을 위해 sleep function을 콜한다.

**sigchld handler**

WNOHANG|WUNTRACE와 Waitpid를 사용하여 모든 Child Process가 스톱 혹은 exitㅎ했다면 return 0나 특정 Child PID의 값으로 return하여 pid와 getjobpid를 이용하여 job을 구한다. Exit이나 스톱한 Child는 WIFEXITED WIFSIGNALED WIFSTOPPED를 나타내는데 WIFEXITED는 deletejob을 call했다. WIFSTOPPED의 경우 job의 state를 ST로 바꾸고 어느 시그널로 스톱했는지 보여주고, WIFSIGNALED는 어느 시그널에 의해 스톱했는지 보여주며 job을 삭제하고, 결국 zombie process 문제를 해결한다.

**sigint handler**

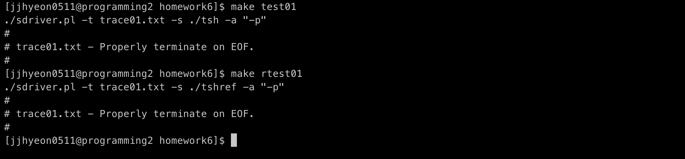
pid=0(foreground x)는 return하고, pid>0(foreground)는 process group에 시그널을 보내므로 결국 ctrl-z에 해당하는 SIGSTP를 다룬다고 할 수 있다.

**sigtstp handler**

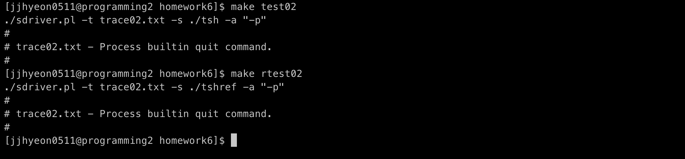
pid=0(foreground x)는 return하고, pid>0(foreground)는 process group에 시그널을 보내므로 결국 ctrl-z에 해당하는 SIGSTP를 다룬다고 할 수 있다.

다음은 test와 rtest의 실행 결과를 분석해보았다.

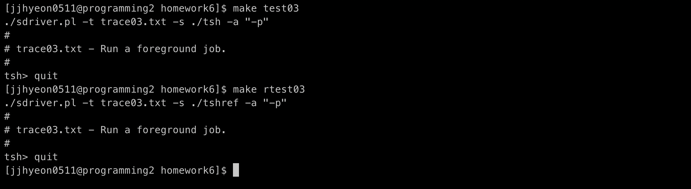
01



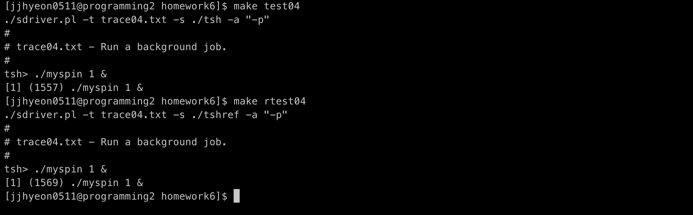
02



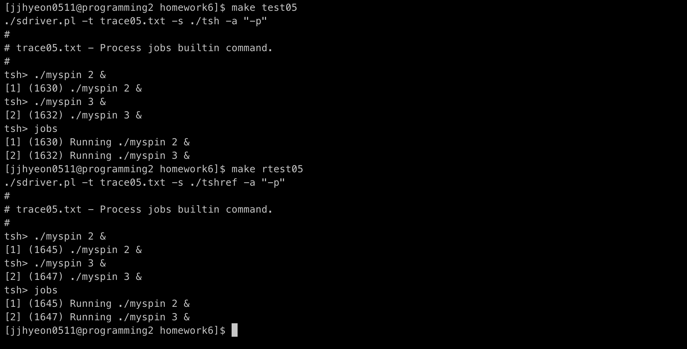
03



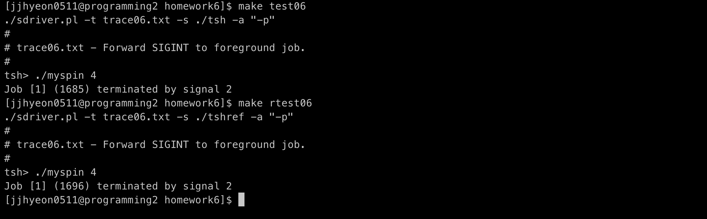
04



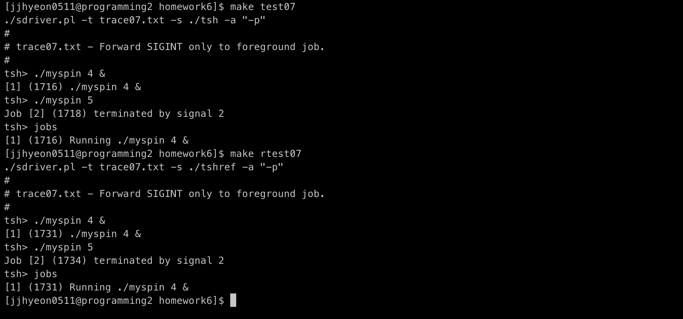
05



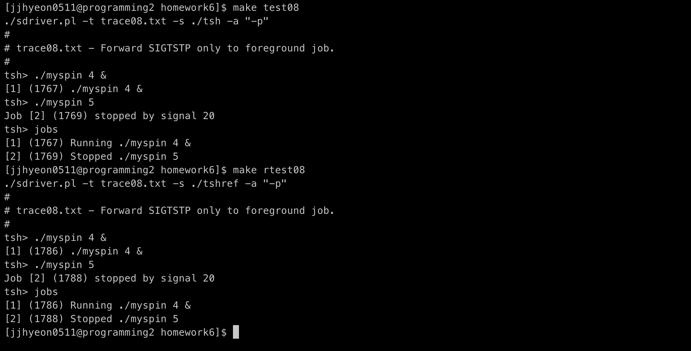
06



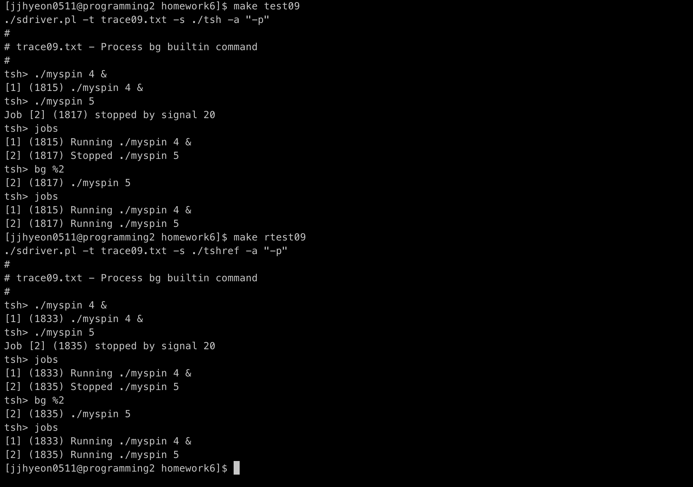
07



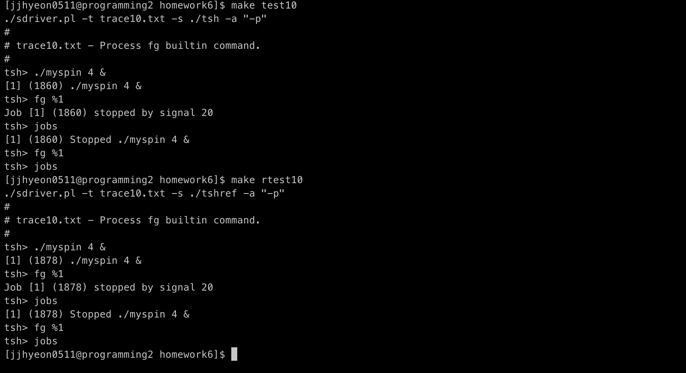
08



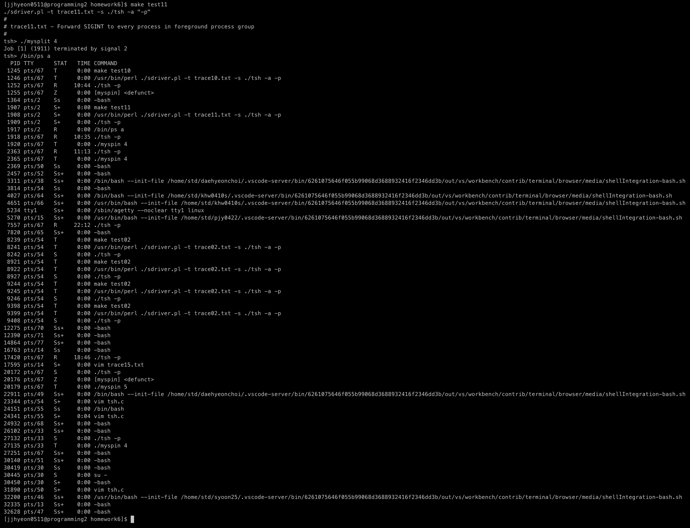
09



10

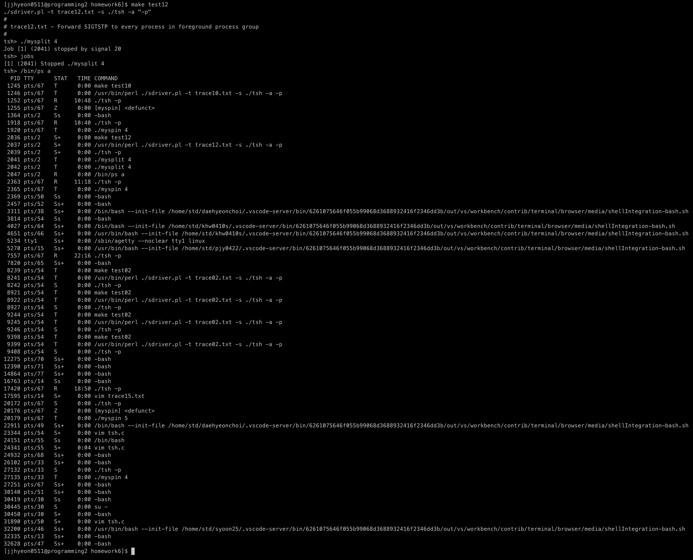


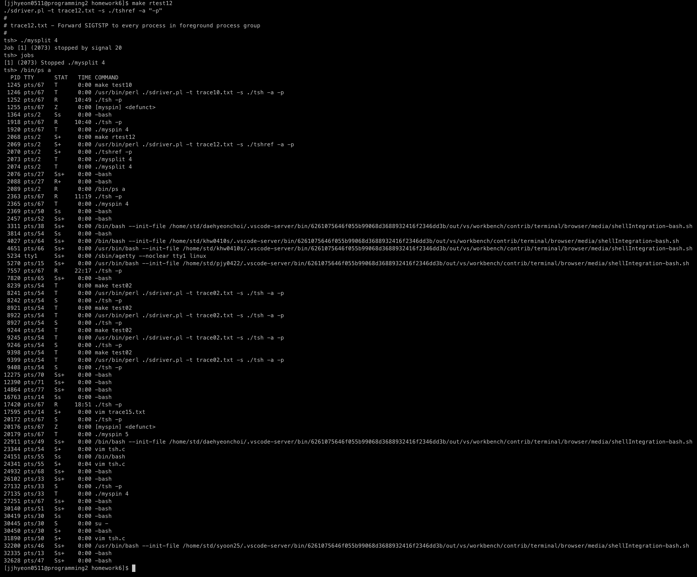
11



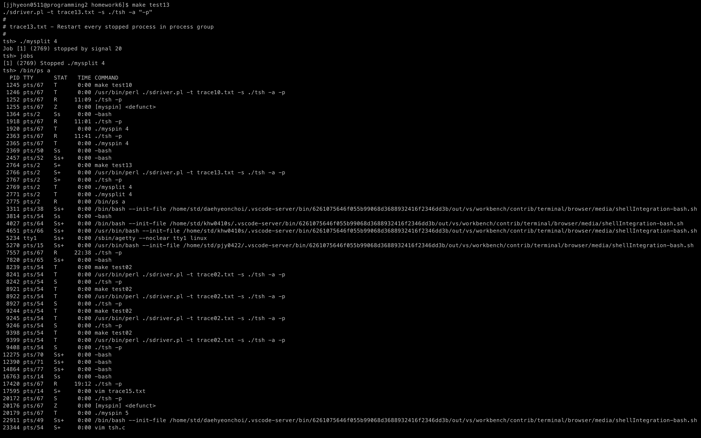


12





13

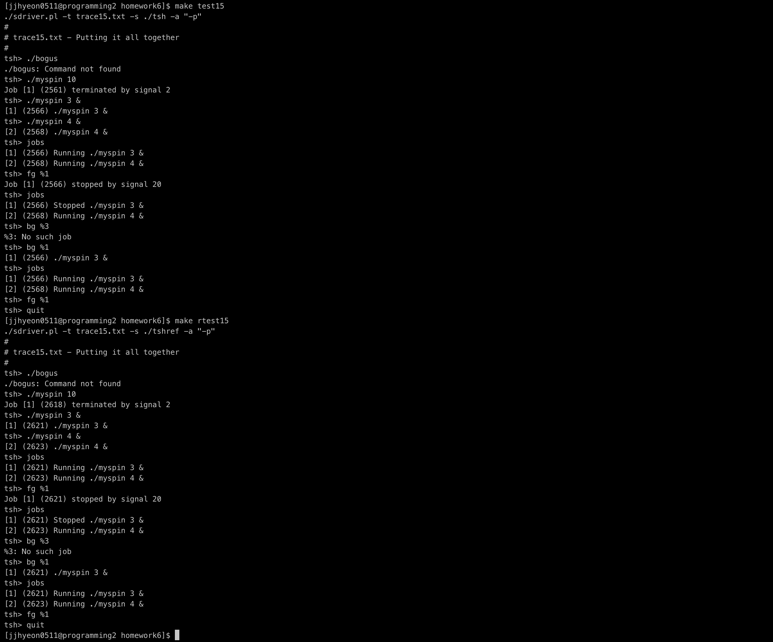




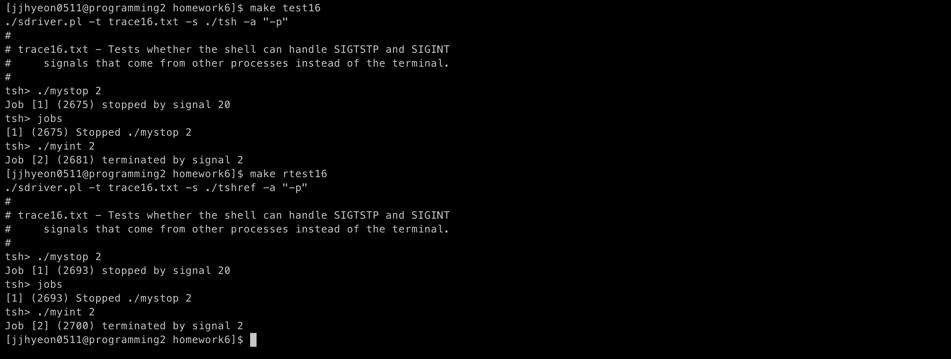
14



15



16



이 처럼 모두 정상적으로 결과가 나오는 것을 확인할 수 있다.

이번 LAB과제를 통해 쉘에 대한 이해와 시그널에 대해 깊게 공부할 수 있었다. Shell을 구성하면서 프로세스를 컨트롤 하고, 시그널링을 다루면서 조금 헷갈리는 부분도 많았지만, 기본 함수들을 이용하여 eval, builtin\_cmd, do\_bgfg, wait\_fg 및 sigchld, sigint, sigstp 핸들러를 다룰 수 있었다. 사용자 컨트롤 조합으로 중간에 시그널을 보낼 수 있다는 것을 배웠고, 이번 과제에서 다루진 않았지만 다양한 시그널로 어떤 결과가 나오는지 공부해볼 수 있었다.