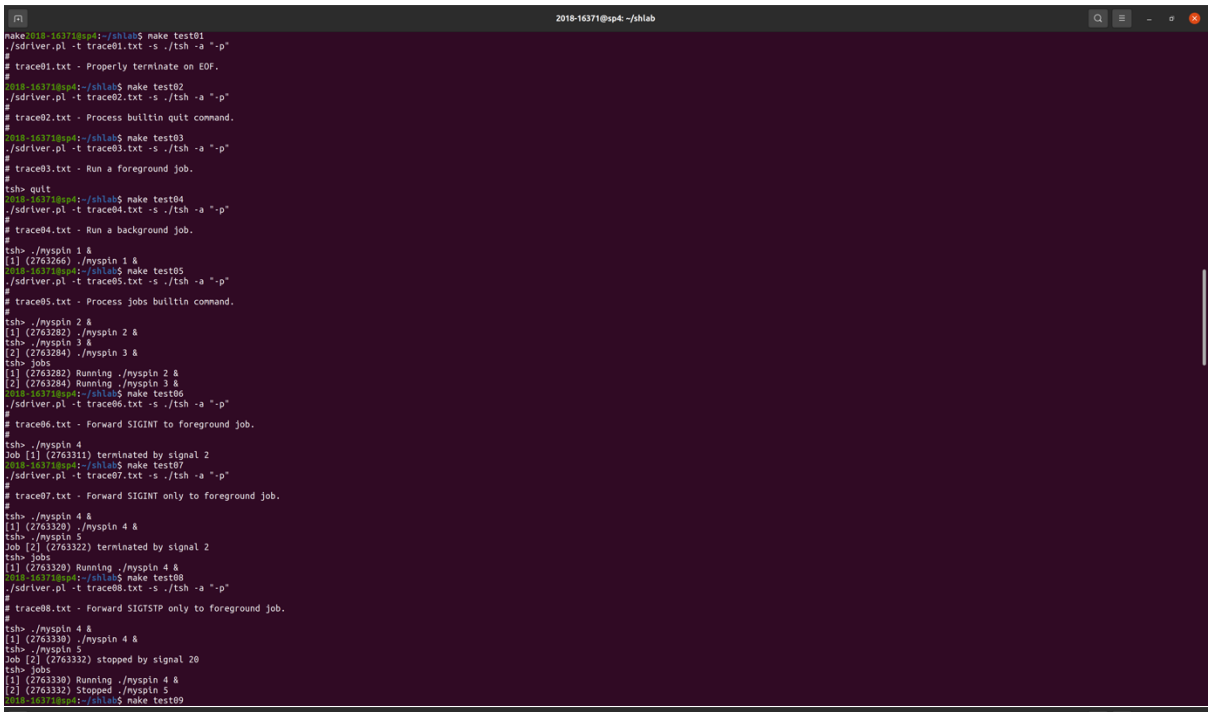
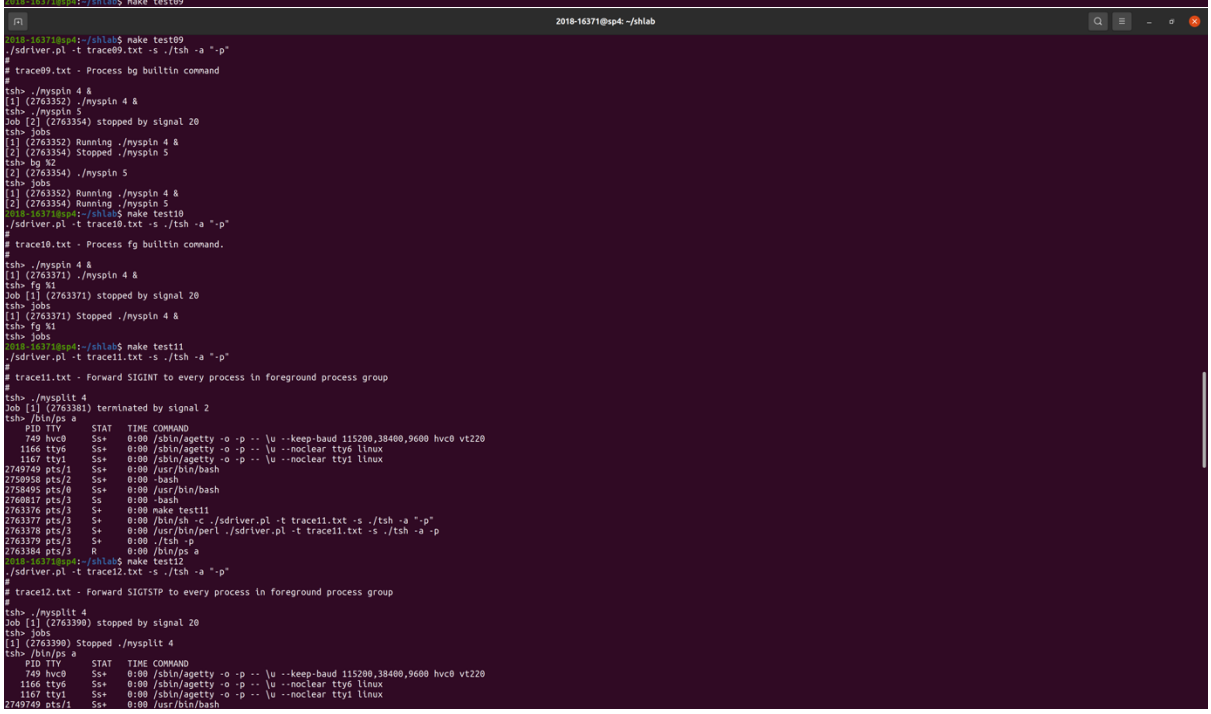
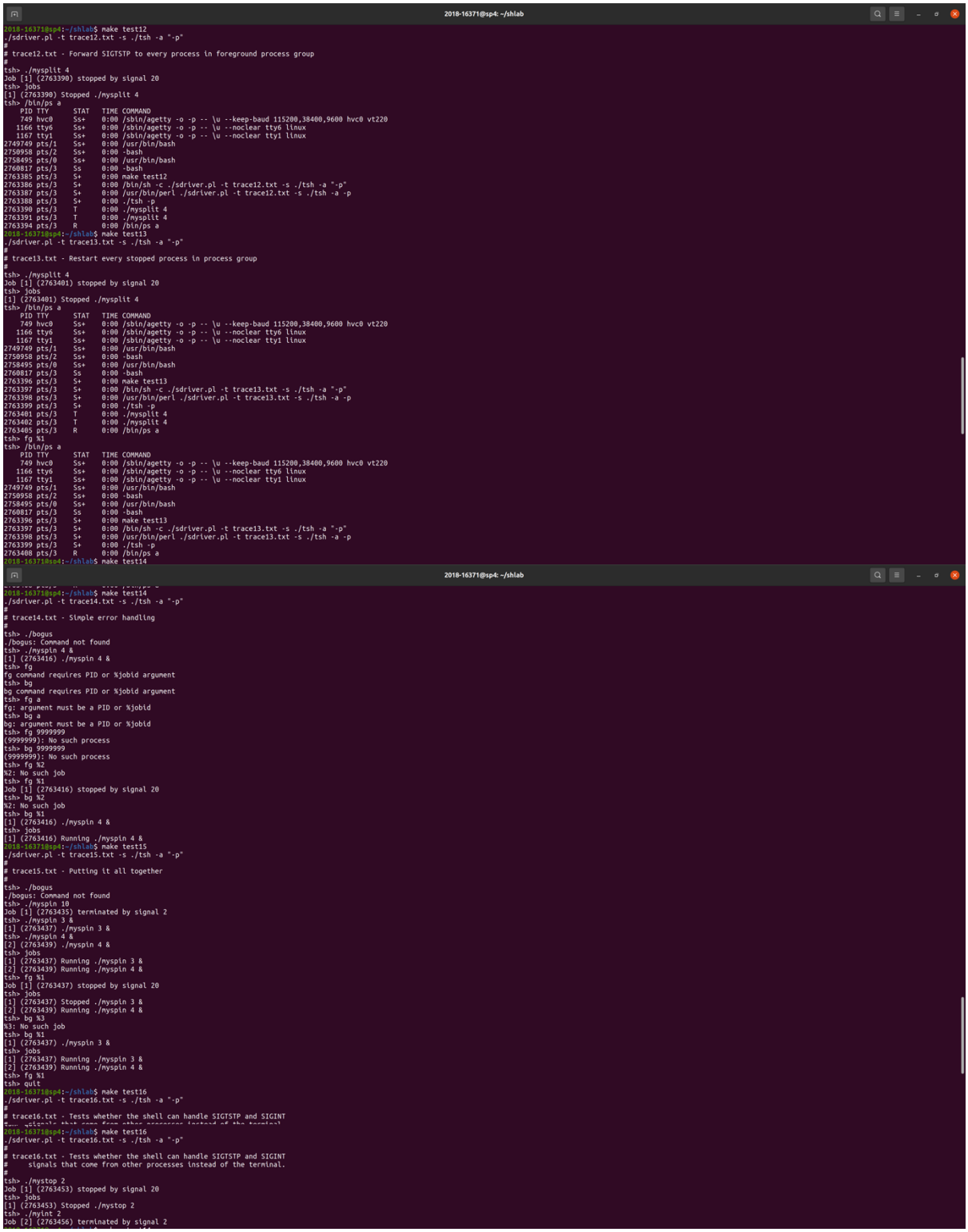
시스템 프로그래밍 Lab2 : tsh shell

2018-16371
자유전공학부 문보설

1. 실행 결과







2. 구현 방법

2.1 eval

eval에서는 먼저 인자로 전달받은 command line을 parseline으로 해체하고 요청이 background로 process를 실행하라는 것인지를 알아냈다. 다음으로 만약 요청이 builtin command의 일종인 경우 builtin command에서 수행하게 하고, 아닌 경우 새 process를

시작하라는 요청이므로 fork()를 수행했다. 그런데 fork()를 수행하기 전에 sigprocmask를 활용하여 sigchld를 블락해 주어야 한다. 만약 이것을 하지 않으면 부모가 fork() 다음 동작을 수행하기 전에 자식이 끝나버리고 sigchld를 보낼 수 있다. 그렇게 되면 부모가 addjob()을 통해 자식 process를 job 목록에 추가한 뒤에 자식은 영영 joblist에서 사라지지 않을 것이다. 이를 방지하기 위해 fork()이전에 sigprocmask를 해 줘야 하는 것이다. 자식 process는 자신의 pid를 group pid로 설정하고, execve를 하기 전에 부모로부터 copy한 시그널 마스크를 unblock하게 했다. 부모 process에서는 command line의 요청이 background process 실행인 경우 해당 설정으로 job을 추가하고, 시그널 마스크를 해제한 다음 실행한 background process의 정보를 출력하고 다른 입력을 받게 했다. 반면 command line의 요청이 foreground process 실행인 경우 해당 설정으로 job을 추가하고, 시그널 마스크를 해제한 다음 waitfg()를 불러서 foreground process group이 끝날 때까지 대기하게 했다.

2.2 builtin_cmd

tsh shell이 제공하는 builtin command는 jobs, quit, bg, fg가 있다. 인자로 전달받은 argv[0]를 통해 command 종류를 확인하고, builtin command인 경우 종류에 맞게 수행하고 1을 리턴하지만 그렇지 않은 경우 0을 리턴하게 했다. Command가 quit인 경우 바로 exit를 통해 shell을 종류한다. jobs인 경우에는 제공된 listjobs 함수를 통해 job 목록을 출력해 준다. fg나 bg의 경우 argv전체를 다시 do_bgfg함수로 넘겨 해당 함수에서 처리하게 했다.

2.3 do_bgfg

먼저 주어진 command 종류가 fg인지 bg인지 확인한다. Command 종류가 fg인 경우 먼저 atoi함수 등을 이용해 jid 혹은 pid를 받아오고(jid의 경우 %로 시작하는지 등의 과정이 더 필요했다), 제공된 getjobjid 혹은 getobjpid를 활용해 fg로 재 시작할 job을 받아왔다. 해당 job의 state를 FG로 바꾸고, kill을 사용해서 해당 job이 있는 프로세스 그룹에 SIGCONT 명령어를 보내서 재 시작했다. foreground에서 실행하라는 요청이므로 waitfg()를 불러서 process group이 끝날 때까지 대기했다. Command 종류가 bg인 경우에는 fg와 동일하게 재 시작할 job을 받아와서 state를 BG로 바꾸고, 동일하게 kill을 사용해서 job이 있는 프로세스 그룹을 재 시작했다. 이 때는 background에서 실행하라는 요청이므로 기다리지 않고 바로 리턴했다.

2.4 waitfg

인자로 받은 pid가 종료할 때까지 대기하는 함수이다. 스펙 문서의 힌트에서 많은 도움을 받아서 foreground group의 pid가 인자로 받은 pid인 동안 계속 sleep하도록 짰다. 이렇게 구현하면 나중에 foreground group이 종료했을 때 인자로 받은 pid와 forground group의 pid가 달라지고, 바로 리턴하는 방식으로 foreground group의 종료를 기다릴 수 있다. 시스템 콜에 해당하여 10,000 사이클 이상을 요구하는 wait를 여기저기서 호출하지 않아도 되는 것도 마음에 들었다.

2.5 sigchld_handler

자식 프로세스의 종료나 중지 시그널을 처리하고 종료된 자식 프로세스를 reap한다. waitpid함수에 pid를 -1로 주어서 아무거나 받을 수 있도록 하고, WUNTRACED와 WNOHANG옵션을 줘서 중단된 프로세스의 시그널을 받아오고 자식이 종료되지 않아도 바로 리턴할 수 있게 했다. 리턴된 pid 값이 0이면 그냥 아무 일도 없었던 것이라고 판단해서 처리하지 않았다. pid값이 0보다 작은데 errno가 ECHILD가 아닌(자식프로세스가 있는) 경우에는 에러가 있었던 것이기 때문에 에러처리를 하고, 0보다 큰 경우에는 종료 혹은 중단 이유를 찾아내서 그에 맞는 행동을 하게 했다. WIFEXITED(status)가 참인 경우, 자식 프로세스의 정상 종료에 해당하므로 job을 joblist에서 삭제한다. 그게 아니라 WIFSIGNALED(status)가 참인 경우에는 자식 프로세스가 비정상적으로 종료된 것이므로 일단 job을 삭제하지만 어떤 시그널로 어떤 프로세스가 종료했는지를 출력으로 알려준다. 그게 아니라 WIFSTOPPED(status)가 참인 경우에는 자식 프로세스가 중단된 것이므로 job status를 ST로 설정하고 어떤 시그널로 어떤 프로세스가 중단되었는지를 출력으로 알려줬다.

2.6 sigtstp_handler

ctrl+z를 받았을 때 foreground process group 중단해 준다. foreground process group을 제공된 fgpid()함수로 받아온 후, kill 함수를 통해 foreground process group에 SIGTSTP을 전달하는 방식으로 구현했다.

2.7 sigint_handler

ctrl+c를 받았을 때 foreground process group 종료해 준다. 마찬가지로 foreground process group을 제공된 fgpid()함수로 받아온 후, kill 함수를 통해 foreground process group에 SIGINT을 전달하는 방식으로 구현했다.

3. 어려웠던 점

우선 세가지 signal handler의 역할을 명확히 구분하지 못해서 시간이 걸렸다. 지금 구현에 따르면 sigint_handler와 sigtstp_handler는 interrupt를 받아서 child process에 전달하는 역할을, sigchld_handler는 chlild process로부터 signal을 받는 역할으로 핸들러가 두 종류로 나뉜다. 하지만 처음에는 이렇게 나누지 못하고 sigint_handler와 sigtstp_handler에서는 interrupt와 관련된 부모 프로세스와 자식 프로세스의 모든 작동을 다루려고 했다. 그러다 보니 sigchld_handler의 역할이 단순 프로세스의 정상 종료 인식으로 애매해졌고 전체적으로 분석하기 어려운 코드가 되었다. 무엇보다도 interrupt 이외의 비정상 종료와 비정상 중단을 처리하지 못한다는 치명적인 결함이 있었다. 이 부분을 처음부터 고려하지 못한 점이 아쉽다.

다음으로 fg와 bg를 구현하는 데에 엄청난 시간을 썼는데, 그 이유는 프로세스 중단이 프로세스 그룹 단위로 이루어진다는 점을 고려하지 못했기 때문이었다. 프로세스를 중단

할 때 해당 프로세스 그룹이 중단되기 때문에 재 시작할 때도 해당 프로세스 그룹을 재 시작해야 하는데, 처음에 프로세스 하나만 재 시작하려고 해서 자꾸 이상한 결과가 나왔다.

4. 알게된 점

   생각지 못했던 함수들이 시스템 콜이라는 것을 알게 되었고, 특히 커널 시그널 마스크를 만드는 것은 시스템 콜이 아니고 마스크를 씌우는 것만 시스템 콜이라는 것을 알게 되었다. 그리고 구현 이전에 세세한 부분을 신경 써서 구조를 짜야 한다는 것을 알게 되었다. do_fgbg를 구현할 때 아무 계획도 없이 그냥 구현했는데, 나중에 에러 메시지를 맞추려고 보니 fg인지 bg인지를 먼저 판단해야 한다는 것을 깨달아서 구조를 뜯어고쳐야 했다. jid나 pid는 모든 경우에 공유되는 부분이니까 코드를 적게 쓸 요량으로 먼저 jid와 pid, job을 가져왔는데 알고 보니 fg일때와 bg일때 에러메시지가 달라져서 bg와 fg를 먼저 구분하고 job을 가져오는 똑같은 코드를 두 경우에 중복해서 써야 했다. 이것을 깨닫고 이미 짠 코드의 구조를 다 고치고 있자니 후회가 밀려왔다. 앞으로는 과제를 할 때 좀 더 세부적인 부분을 신경 쓸 것이다.