2020년 2학기

Assignment #5

과목 : 컴퓨터SW시스템개론

담당 교수 : 김종

학과 : 컴퓨터공학과

학번 : 20190439

이름 : 오승훈

povis id : sho0927

|  |
| --- |
| < 명예서약(Honor code) >  “나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.” |

**Problem : Shell lab**

1. 문제의 개요
2. shell과 유사한 기능을 가진 tiny shell을 구현을 해보는 것을 목표로 한다.

이 문제에서는 구현을 해야할 함수가 총 7개가 존재한다.

void eval(char \*cmdline) : 이 함수는 shell의 main에서 받아지는 인자들을 어떻게 처리를 할 것인가에 대해서 평가를 해주는 함수라고 생각하면 된다.

int builtin\_cmd(char \*\*argv) : 이 함수는 shell의 기본적인 builtin 명령어인 quit, jobs, bg (filename), fg (filename) 명령어가 input으로 들어와질 때 제어를 해주는 함수이다.

void do\_bgfg(char \*\*argv) : 이 함수는 bg (filename), fg (filename)을 구현 시켜주는 함수이다.

void waitfg(pid\_t pid) : 이 함수는 부모 process가 자식 process가 foreground 상태에서 돌아갈 때 자식 process가 terminate 될 때 까지 기다리는 것을 구현시켜주는 함수이다.

void sigchld\_handler(int sig) : SIGCHILD에 대한 signal을 handling 해주는 함수이다.

void sigtstp\_handler(int sig) : SIGTSTP(ctrl-z) signals들이 발생했을 때 catch 해주는 함수이다.

void sigint\_handler(int sig) : SIGINT (ctrl-c) signals들이 발생했을 때 catch 해주는 함수이다.

1. 문제 해결 방법
2. int builtin\_cmd(char \*\*argv)

이 함수부터 설명을 하자면 built\_in 명령어들을 처리해주는 함수인데 argv[0]이 “quit” , “fg”, “bg”, “jobs”인지를 확인을 해주고 각 명령어마다 처리해야할 일을 진행하면 된다.

1. Quit의 경우 이 명령어는 process를 exit() 하라는 뜻이기 때문에 exit(0)을 해주면 된다.
2. Fg의 경우 do\_bgfg 함수를 call해서 fg에 대한 명령을 시행할 수 있도록 해주면 된다.
3. Bg의 경우 fg와 동일하게 do\_bgfg 함수를 call해서 bg에 대한 명령을 시행할 수 있도록 하면 된다.
4. Jobs의 경우 현재 진행중인 job들을 보는 것이고 이는 이미 구현이 되어 있기 때문에 listjobs(jobs)를 진행해주기만 하면 된다.
5. 현재 이 함수에서 built\_in 함수로 구현이 되어 있을 시 1을 반환해주면 되고, 안 되어 있을 시에는 0을 반환해주면 된다.
6. void waitfg(pid\_t pid)

이 함수의 경우 현재 pid가 존재하는 pid인지를 확인을 해줘야하고, 현재 기다리는 pid가 foreground에서 진행중일때까지 무한 loop를 걸어주면 된다. 그러므로 현재 pid의 job이 FG일때까지 while문을 돌리고, 그 후에는 FG가 아니게 되면 이때 return을 해주면 된다.

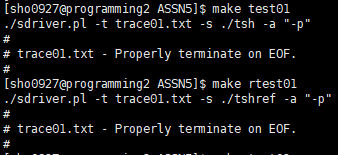
1. void do\_bgfg(char \*\*argv)
2. 이 함수에서는 먼저 argv[1] 인자가 NULL인지를 check 해줘야한다. 이는 bg (filename) or fg (filename)의 형식을 가지고 있기 때문에 filename에 대한 인자가 있지 않으면 명령어가 수행이 불가능해지기 때문이다.
3. argv[1]인자가 NULL이 아닌 경우 filename을 주는 형식이 두 가지이므로 이를 check 해야한다. 첫 번째 형식은 job id를 주는 형식이고, 이는 argv[1]의 첫번째 인자가 %로 시작한다고 했기 때문에 이것인지를 확인을 해주고, 맞다면 argv[1]의 두 번째 인자부터 나오는 값을 int 형식으로 해서 jid에 넣어주면된다. 그 후 jid 값이 0이라면 올바르지 않은 argument 형식이므로 error를 출력을 해주고, 아니라면 현재의 job의 주소를 갖는 pointer를 만들어준다.
4. 두 번째 형식은 pid를 주는 것이므로 pid의 값을 int 형식으로 변환을 시켜주고, 이때 변환이 안된 경우 0이 반환되므로 0이되면 argument error를 출력을 해주고, 아니라면 현재의 pid의 job의 주소를 갖는 pointer를 만들어준다.
5. 이후 fg 이든 bg이든 이 job을 다시 시작을 해야하기 때문에 kill 함수를 이용해 SIGCONT signal을 보내준다. 이 때 –를 사용해주는 이유는 tempt\_job->pid를 group pid로 갖는 모든 process를 다시 시작을 해줘야하기 때문이다. 그 후 bg job이면 job\_state를 BG으로 바꿔주기만 하면 되고, fg job이면 job\_state를 FG로 바꿔주고, 이 process가 끝날 때까지 waitfg를 해줘야한다.
6. void eval(char \*cmdline)
7. 이 함수가 거의 main dealing을 담당하는데 먼저 이 함수가 bg에서 돌아가는 함수이냐를 check 해주는 parseline() 함수를 이용하여 bg에서 진행되는지 아닌지를 check 해준다. 그 후 이 함수에서는 argv[0]이 NULL인지를 확인을 해줘야한다. 그리고, 만약 builtin\_cmd이면 builtin\_cmd 함수로 처리를 해주게된다.
8. 그 후 혹시 모르는 child의 빠른 signal pending을 막아주기 위해서 sigaddset(&mask, SIGCHLD)를 이용하여 mask에 SIGCHLD signal을 넣어준 후 sigprocmask 함수를 이용하여 SIGCHLD signal을 blocking해준다. 이를 구현함으로써 fork를 하여 함수가 child process를 만들었을 때 부모가 먼저 없어지는 상황인 고아 process 상황이나 zombie process 상황을 막아주기 위함이다.
9. 이후 fork를 진행하여 child process를 만들어 준 후 부모 process의 경우 bg이냐 fg이냐가 중요해지는데 일단 child process를 jobs라는 job\_table과 같은 역할을 하는 곳에 넣어주고, 그 후 blocking된 signal을 풀어준다. 그리고, fg job 인 경우 child process가 끝나기까지 wait을 해준다. Bg인 경우 그냥 진행을 하면 되기 때문에 printf만 해주면 된다.
10. child process의 경우 blocking된 SIGCHLD signal을 풀어준 후, setpgid를 이용하여 새로 시작하려는 process의 group pid를 본인것과 같게 setting을 해준다. 그리고 execve 함수를 이용하여 새로운 process를 시작하게 해준다. 만약 execve 함수가 제대로 진행이 되지 않으면 error 처리를 해주면 된다.
11. void sigchld\_handler(int sig)
12. 이 signal은 자식 process가 종료되었을 때 나오는 signal이기 때문에 먼저 모든 자식 process가 끝날 때까지 while문을 돌려줘야 한다는 것을 알 수 있다. 이는 자식 process가 여러 개 일수도 있기 때문이다.
13. 그 후 자식 process가 어떻게 끝났냐가 중요한 문제로 남게 되는데 WIFSTOPPED | WIFEXITED | WIFSIGNALED를 확인해주면된다.
14. WIFSTOPPED의 경우 child process가 정지된 경우이므로 현재의 job을 ST 상태로 변경을 해주고, print를 해주면 된다.
15. WIFEXITED의 경우 child process가 signal에 의해 정상 종료된 경우이기 때문에 job\_table에서 delete를 해주면 된다.
16. WIFSIGNALED의 경우 child process가 terminate된 경우이기 때문에 terminate되었다고 print를 해주고, 현재 job에서 삭제를 시켜주면 된다.
17. void sigtstp\_handler(int sig)

이 함수에서 진행해줘야 할 것은 너무 단순하다. 만약 이 현재 foreground에서 진행중인 process가 있는지를 확인해주고, 있다면 그 process에 signal을 kill 함수를 이용하여 보내주기만 하면된다.

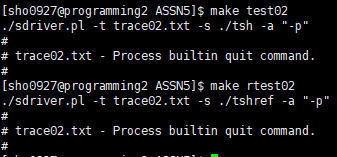
1. void sigint\_handler(int sig)

이 함수는 위의 sigtstp\_handler와 동일하게 만들어주면 된다. Signal이 다를 뿐이기 때문이다.

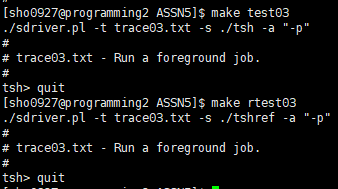
1. Trace를 이용한 test와 rtest 비교
2. Trace01



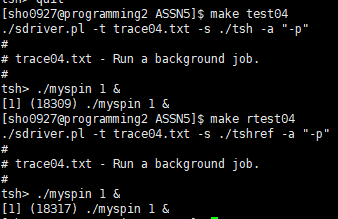
1. Trace02



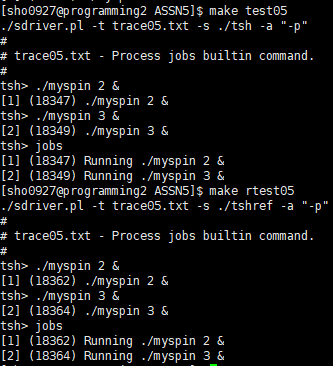
1. Trace03



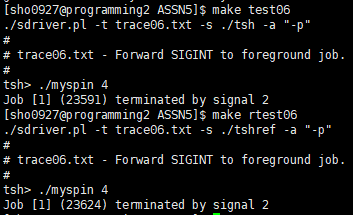
1. Trace04



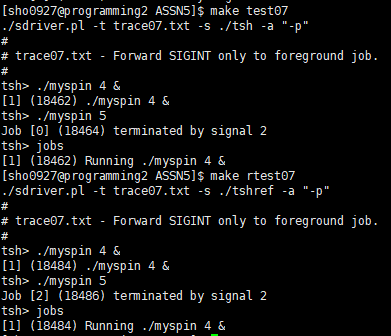
1. Trace05



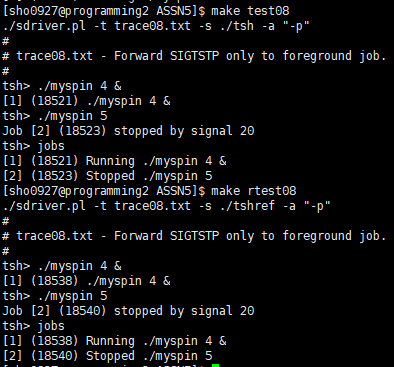
1. Trace06



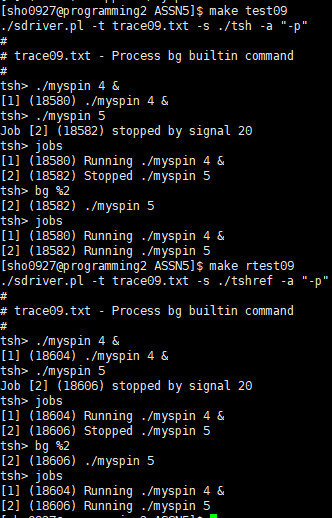
1. Trace07



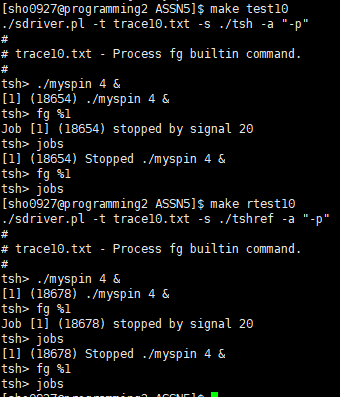
1. Trace08



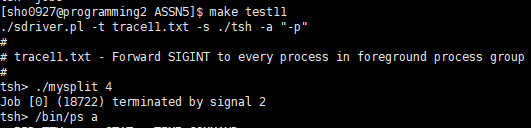
1. Trace09



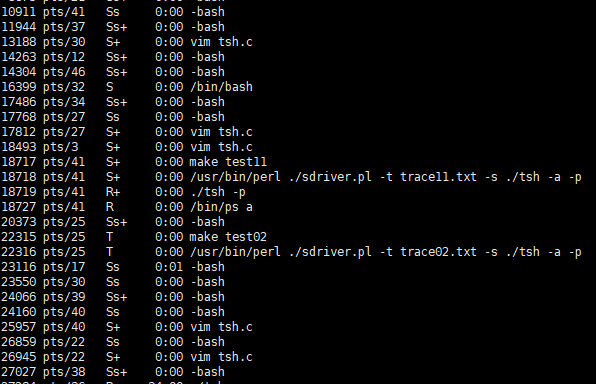
1. Trace10



1. Trace11
2. test

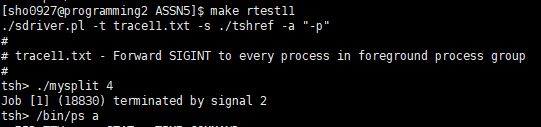


Test의 경우 18722가 signal 2에 의해 terminate 되었으므로 18722 pid를 가진 job이 있으면 안되고,

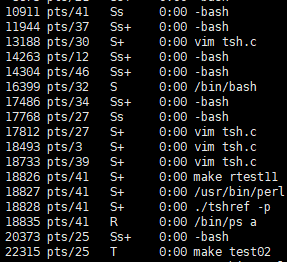


실제로도 존재하지 않는다는 것을 확인할 수 있다.

1. rtest

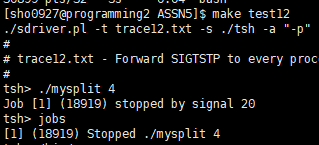


Rtest 역시 18830라는 pid를 가진 process는 존재하지 않아야하는데



실제로도 존재하지 않음을 확인할 수 있다.

1. Trace12
2. test

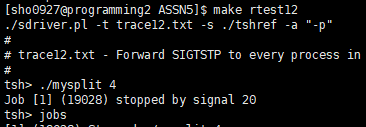


18919 pid를 가진 process는 stop이 되어 있어야함을 알 수 있고,

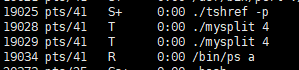


실제로도 stop이 되어 있음을 확인할 수 있다.

1. rtest

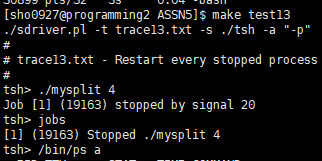


19028 pid를 가진 process는 stop이 되어 있어야하고,



실제로도 stop해있는 것을 확인할 수 있다.

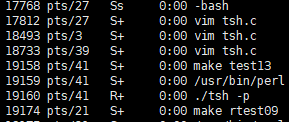
1. Trace13
2. test



19163이 stop이 되어 있어야 하고,

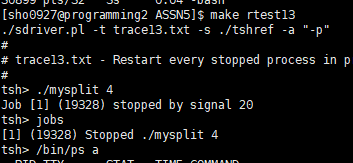


Stop이 되어있음을 확인할 수 있고, 이후 다시 fg에서 시작을 해주므로

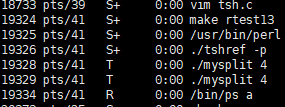


terminate되어 사라짐을 확인 할 수 있다.

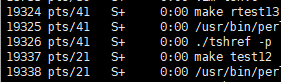
1. rtest



19328이 stop되어 있어야하고,

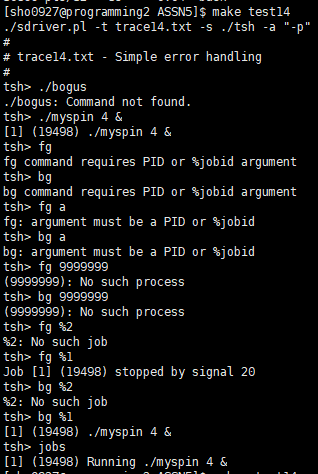


실제로도 멈춰있고, 그 후 다시 foreground에서 실행을 시키므로,

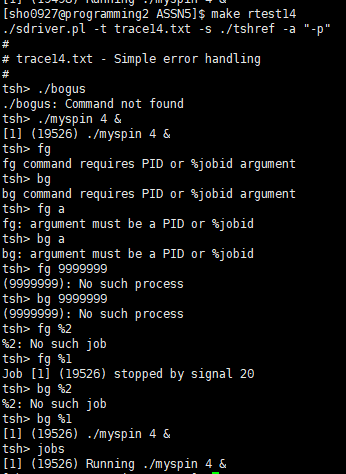


Terminate 되어 사라짐을 확인할 수 있다.

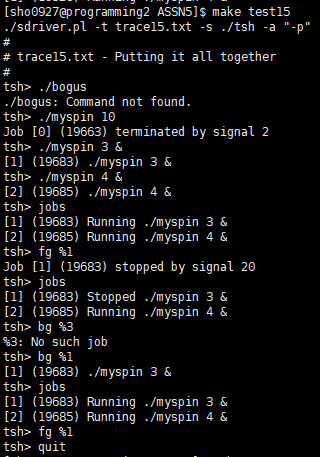
1. Trace14
2. test



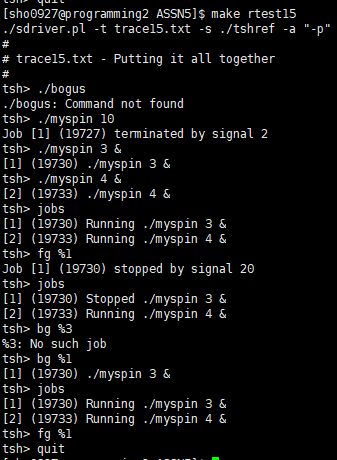
1. rtest



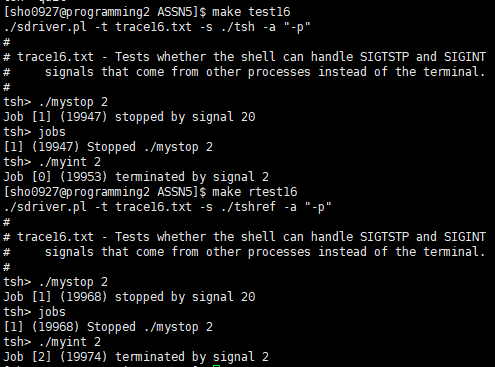
1. Trace15
2. test



1. rtest



1. Trace16



1. 토론

이번 shell을 구현을 해보면서 가장 신기했던 것은 SIGNAL이라는 함수들이다. 우리가 알던 절차 지향 프로그래밍을 지원하는 C언어에서는 결코 되지 않을 것 같던 일들이 event driven programming인 GUI 기반으로 하는 코딩인 QT에서 사용하는 connect와 같은 동작 역시 진행을 할 수 있다는 것을 깨닫게 해준 것 같고, 또한 수업에서 들을 때는 딱히 이해가 되지 않던 내용들이 확실히 코드로 진행을 해보니 실제 코드가 어떻게 진행이 되는지를 확인할 수 있었던 좋은 Lab 과제였다고 생각한다.

하지만 조금 아쉬웠던 점은 새로운 library를 이용하여 만들어진 함수들이 너무 많았던 것과 그 함수들을 찾아보느라 걸린 시간들이 많았다는 점인 것 같다. 그리고 교과서의 본문에 적혀있는 내용들을 미리 읽어보고 이 LAB을 진행했으면 훨씬 수월 했을 것 같다 라는 생각으로 이 LAB을 마무리한다.