eval 함수

먼저 유저가 입력한 커맨드를 parseline을 이용하여 띄어쓰기 단위로 파싱하여 argv배열에 저장한다. 이때 back/fore ground를 의미하는 parseline의 리턴값을 변수 asked_bg에 저장해준다.

그리고 empty line을 먼저 예외로 처리해준다. (argv[0]이 NULL이라는 말은 결국 command로 입력된 것이 없다는 뜻이므로)

그리고 builtin_cmd함수에 argv배열을 넣어 builtin command인지 체크한다. builtin command라면 builtin_cmd 함수 내부에서 execute될 것이므로 builtin command가 아닌 경우에 대해서 eval에서 다뤄준다.

그리고 힌트를 참고하여 SIGCHLD 시그널을 block해준후 child process를 fork하여 리턴값을 pid에 저장한다. pid가 음수라는 것은 fork에 실패했다는 것이므로 에러메시지를 출력하고 리턴한다.

pid가 0이라는 것은 child process라는 뜻이므로, unblock해준뒤 새로운 프로세스 그룹을 만들어준다. 그리고 execvp함수를 통해 command를 execute해준다. 이때 리턴값이 음수라는 것은 execvp가 실패했다는 것이므로 레퍼런스 파일처럼 "Command not found."라는 메시지를 출력해주고 exit(0)으로 끝낸다. return을 사용하면 종료가 되지 않고 fork로 인해 그다음 커맨드들을 입력했을 때 2번 수행되므로 exit(0)으로 끝내주어야 함에 주의한다. (ex. trace14에서 버그 발생)

pid가 0이 아니면 parent process라는 뜻이다. 일단 asked_bg가 1이면 background, asked_bg가 0이면 foreground이므로 이 2가지 케이스로 나눠준다. 그래서 background인 경우는 job 리스트에 BG로서 프로세스를 추가하고 unblock해준다. 그리고 레퍼런스 파일처럼 정보를 출력해준다. foreground경우는 job 리스트에 FG로서 프로세스를 추가하고 unblock해준다. 그리고 terminate될 때까지 기다린 후 리턴해준다.

builtin_cmd 함수

builtin command인 quit, fg, bg, jobs를 처리해야한다.

quit가 입력되면 exit 함수를 통해 shell을 종료한다.

fg 또는 bg가 입력되면 do_bgfg함수를 수행해주고, builtin command가 맞으므로 1을 리턴한다.

jobs가 입력되면 listjobs함수를 통해 모든 백그라운드 job을 출력하고, builtin command가 맞으므로 1을 리턴한다.

위 4개의 builtin command가 아닌 경우는 아무것도 하지 않고 0을 리턴해준다.

do_bgfg 함수

bg, fg command를 처리해주는 함수이다. 함수를 크게 예외처리, id에 대응하는 job 찾기, command 수행 3개의 단계로 나누었다.

먼저 예외처리 단계이다. 이 단계에서는 2가지 예외를 처리해준다. 첫번째는 ID가 입력되지 않은 경우이다. 입력으로 bg 또는 fg가 들어오고 뒤에 ID를 입력해주지 않은 케이스를 처리해준다. 두번째는 ID가 아닌 다른 것을 입력한 경우이다. PID가 입력되면 숫자, JID가 입력되면 %숫자가 입력되므로 ID의 첫번째 글자가 숫자 또는 %가 아니라면 잘못된 입력이 들어왔다는 뜻이다. 따라서 이 경우를 처리해준다.

두번째는 id에 대응하는 job 찾기 단계이다. ID 문자열을 파싱하여 PID인지 JID인지로 나누어 처리해준다. JID의 경우는 %숫자 형식으로 입력되므로 문자열의 첫번째 문자가 %이면 JID %가 아니면 PID임을 알 수 있다. 따라서 이것을 조건으로 하여 2가지 케이스로 분류해준다. 그리고 이 ID에 해당하는 job을 getjobpid함수 또는 getjobjid함수를 활용하여 job변수에 저장해준다. 이때 만약 job변수가 NULL이라면 ID에 해당하는 job이 존재하지 않는다는 것이므로 예외로 처리해준다.

마지막 단계는 command 수행 단계이다. 이 단계도 fg와 bg 두가지 커맨드에 대해 케이스를 나누어 진행해준다. 먼저 fg command가 입력된 경우에는 job을 foreground로 다시 실행시켜야 하므로 state를 FG로 설정해주고 kill 함수를 통해 SIGCONT 시그널을 보내 job을 재실행시킨다. 그리고 terminate될때까지 기다린 후 리턴해준다. bg command가 입력된 경우에는 job을 background로 다시 실행시켜야 하므로 state를 BG로 설정해주고 kill 함수를 통해 SIGCONT 시그널을 보내 job을 재실행시키고 writeup 파일에 나와있는 것처럼 정보를 출력해준뒤 리턴해준다.

waitfg 함수

함수의 argument로 들어오는 pid에 대응되는 process가 끝날때까지 기다리게 만드는 함수이다. 이 함수에서도 예외처리를 먼저 해주어야한다. pid에 대응되는 job이 없는 경우를 예외처리해준다.

그리고 pid에 대응되는 process가 foreground process가 아닐때까지 계속 루프속에서 sleep을 반복하며 가둬둔다. 즉 foreground에서 실행중인 job이 terminate될때까지 기다린다.

sigchld_handler 함수

sigchld_handler는 SIGCHLD 시그널을 받았을때 수행된다. 즉, child process가 terminate되거나 시그널로 인해 멈추게 된 경우를 처리한다. 핵심은 실행중인 child process가 terminate되는 것을 기다리지 않는다는 것이다. 이를 통해 waitpid 함수가 필요함을 알 수 있다.

waitpid 함수의 리턴값을 pid에 저장하고, pid가 0보다 큰 경우에만 루프를 돌게 된다. 결국 모든

가능한 좀비 children을 reap해야 하므로 첫번째 argument를 -1로 두어 임의의 child process를 기다리게 만든다. 그리고 세번째 argument를 WNOHANG | WUNTRACED로 둠으로써 terminate된 process가 없고, 여전히 실행중인 경우는 terminate되는 것을 기다리지 않고 즉시 0을 리턴하고, stop된 경우 stop된 상태라는 것이 status에 저장되도록 하였다. 즉, 루프안에는 terminate되었거나 stop된 child process만 들어올 수 있게 된다. 따라서 이번에도 각 케이스별로 처리해주면 된다.

waitpid가 리턴되면서 status에 그 child process의 상태가 저장되므로 종료 상태 정보 매크로 함수들을 이용하여 케이스를 나눠주었다.

첫번째는 terminate된 경우이다. 이 경우는 끝이므로 job list에서 pid에 해당하는 job을 제거해주면 끝이다. 그리고 추가적으로 특정 시그널에 의해 terminate된 경우는 어떤 시그널에 의해 terminate된 것인지 정보를 출력해준다.

두번째는 stop된 경우이다. 이 경우는 pid에 해당하는 job의 state를 ST, 즉 정지 상태로 바꿔주고 어떤 시그널에 의해 stop된 것인지 정보를 출력해주면 된다.

sigint_handler 함수

sigint_handler는 SIGINT 시그널을 받았을때 수행된다. 이 함수는 SIGINT 시그널을 foreground job에 전달해주어야 하므로 fgpid함수를 통해 foreground job의 PID를 얻어온다.

그리고 얻어온 PID가 0인 경우, 즉 foreground job이 없는 경우를 예외로 처리하여 바로 리턴해준다.

예외처리를 통과하면, kill함수를 통해 SIGINT 시그널을 전달해주면 끝이다.

sigtstp_handler 함수

sigtstp_handler는 SIGTSTP 시그널을 받았을때 수행된다. 함수의 작동 방식은 sigint_handler와 완벽하게 동일하고, 단지 다른점은 kill함수를 통해 SIGTSTP 시그널을 전달해준다는 것이다.