2016314827 김규연

PA 3 코드 설명

시그널 설정

1. SIGINT는 무시된다.

2. SIGTSTP를 받으면, SIGCHLD 시그널이 부모 프로세스에게 전달된다.

3. SIGCHLD 시그널을 받으면, waitpid를 통해 자식을 회수하고, WIFSTOPPED가 TRUE인 경우에는 해당 자식의 프로세스 그룹에 SIGKILL 시그널을 보냈다.

사용자 입력 처리

사용자가 입력한 input이 주어졌으면, cmdchk함수에 넣는다.

cmdchk 함수는 다음과 같은 일을 수행한다.

1. 올바른 파이프 형식의 입력이 아니면 Command not found를 출력한다.

2. 올바른 입력이라면, 해당 명령어에 파이프가 존재하는지, 존재한다면 몇 개 존재하는지 찾아낸다.

명령어 처리

1. 파이프가 존재하지 않을 때

redirection을 검사한다. 어떤 타입의 redirection 인지 검사한다. 만약 정의되지 않은 redirection이라면, Command not found를 출력한다.

올바른 redirection이 쓰였다면,

무슨 명령어(cd, rm, head, tail… 따위)가 쓰였는지 알아낸다. 정의되지 않은 명령어에 대해선 Command not found를 출력하고 끝낸다.

해당 명령어에 대한 argument들을 args라는 배열에 잘 쪼개어 순서대로 저장한다.

또한, input redirection, output redirection에 해당하는 파일 이름을 각각 filein, fileout에 저장한다. 만약 filein이나 fileout이 NULL이라면, redirection이 필요없다는 뜻이다.

해당 명령어를 찾아가 실행시킨다.

cmd_type2는 직접 구현했고, fork로 자식프로세스를 실행하여 처리가 진행된다.

cmd_type3(mv, rm, cd)는 파이프에 입력되지 않는다는 점을 감안하여 fork로 자식 프로세스를 생성하지 않았다.

cmd_type4에서는 pwd의 경우 fork로 자식프로세스를 생성했다. 그러나, exit는 자식 프로세스를 생성하지 않았다.

2. 파이프가 존재할 때

일단 파이프를 기준으로 쪼개진 명령어를 순서대로 저장하고, 그 순서대로 실행시킨다.

2-1. 첫 번째 파이프 명령어일 경우

input redirection이 있는지 검사하고, 다음 파이프에 전달할 stdout을 dup2와 pipe를 통해 전달하였다. 그리고 pipe read부분은 닫지 않고, 다음 명령어가 사용할 수 있도록 file descriptor을 별도로 저장하였다.

2-2. 중간 파이프 명령어일 경우

이전 명령어에서 pipe read부분을 전달 받아 stdin을 대체한다. stdout을 처리하는 건 첫 번째 파이프 명령어와 동일하다.

2-3. 마지막 파이프 명령어일 경우

output redirection이 있는지 검사하고, 전달받은 이전 pipe의 read 부분을 최종적으로 닫는다.