UNIX project

1차 REPORT

DEADLINE: 2019/11/08

컴퓨터공학과

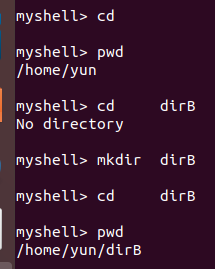
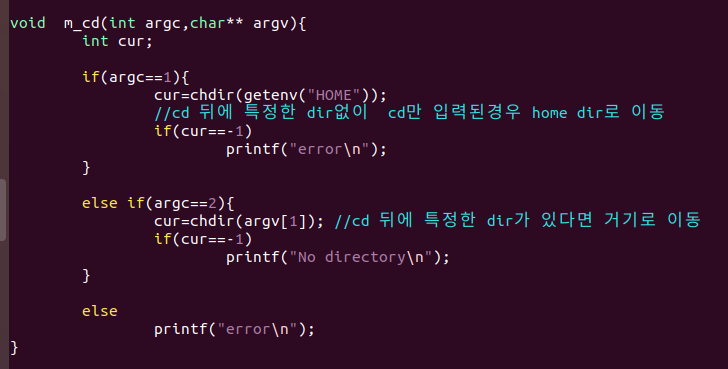
12161558 김혜윤

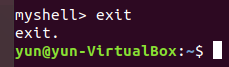
1. 목표 #1
   1. chdir() 처리, exit() 처리
   2. 백그라운드 실행 (명령 뒤에 & 표시)
2. 요구사향
   1. makeargv 함수는 makelist 함수를 이용하여 구현
      1. 쉘 작성시 명령을 실행하기 위해 execv 함수를 이용할 것임
      2. 따라서 사용자로부터 입력 받은 명령 라인을 vector 형태로 변환 해야함
      3. makelist() 는 명령라인을 vector 형태로 변환하는 함수
   2. 다음 기능을 추가
      1. cd 명령이 제대로 먹히지 않는 버그를 수정
      2. exit 명령을 구현
      3. 백그라운드 실행을 구현
3. 구현방법 및 문제점
   1. makelist 함수변경
      * + 주어진 코드에 변경한 부분, cmdzip 이라는 문자열을 공백을 기준으로 cmdvector의 index에 분리시켜 주고, 분리된 문자열의 개수(numtokens)를 반환함.



* 1. ‘cd’ 명령이 제대로 먹히지 않는 버그를 수정
     + - 주어진 소스 코드에서는 일반 쉘에서 사용하는 것과 같이 command line을 입력하면 해당하는 기능이 어느 정도 되지만 ‘cd’ 는 수행되지 않음. shell에서 fork, exec 을 사용하여 기능을 수행하기 때문. cd를 사용하면 다른 기능들과 같이 fork 한 뒤 생성된 자식 process에서 명령어를 실행해서 자식 process에서만 cwd가 바뀌고, 부모 process 즉, shell 에서는 cwd가 변경되지 않기 때문에 작동이 안됨.
       - 구현:

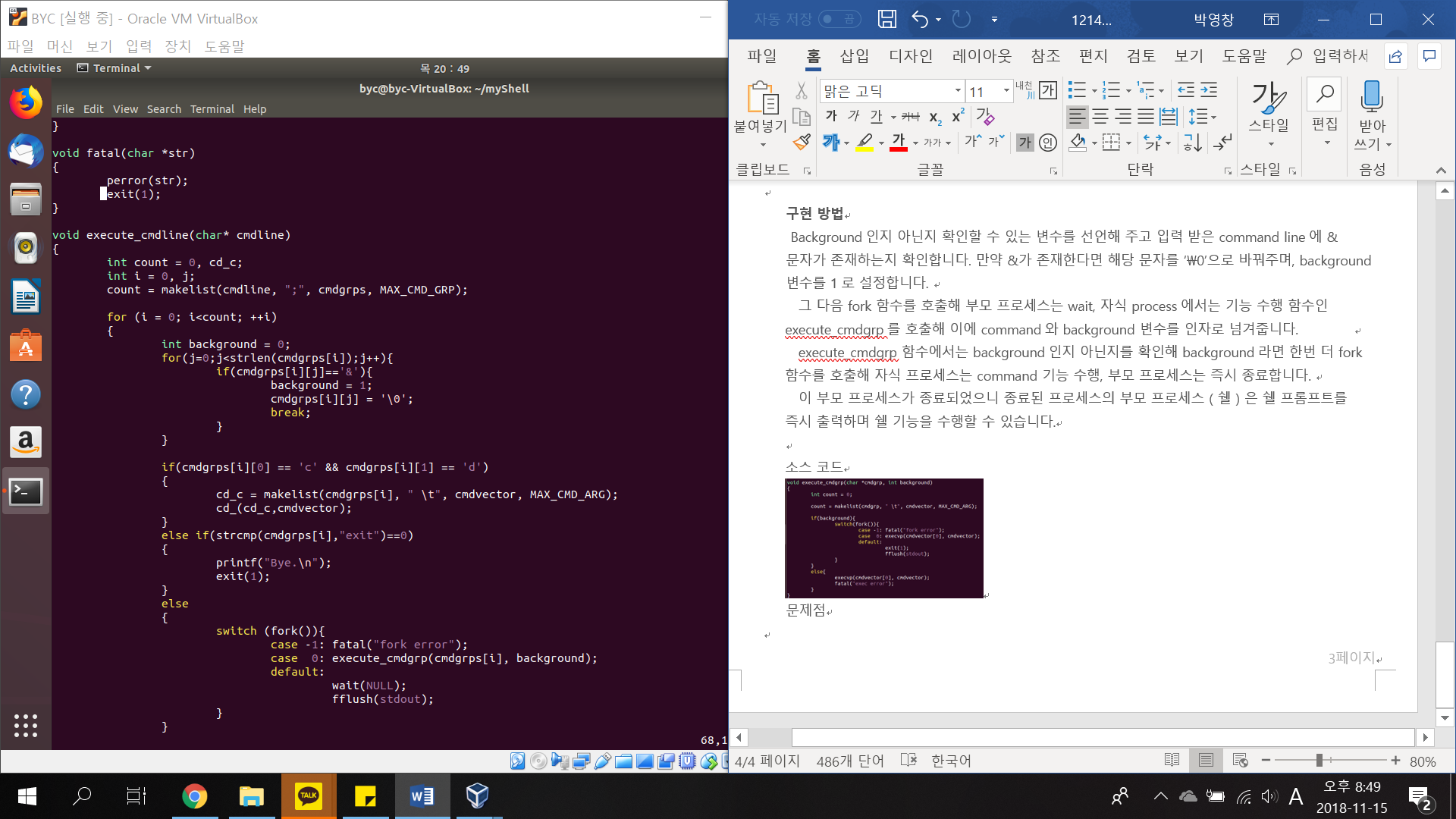
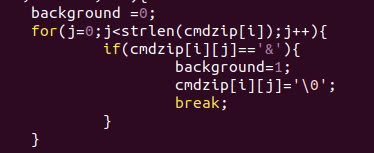
cd 명령어 사용시 다른 명령어와 다르게 현재 프로세스의 cwd가 바뀌어야 하므로 fork를 하지 않고 즉시 chdir를 사용해서 cwd을 변경

<- 결과화면

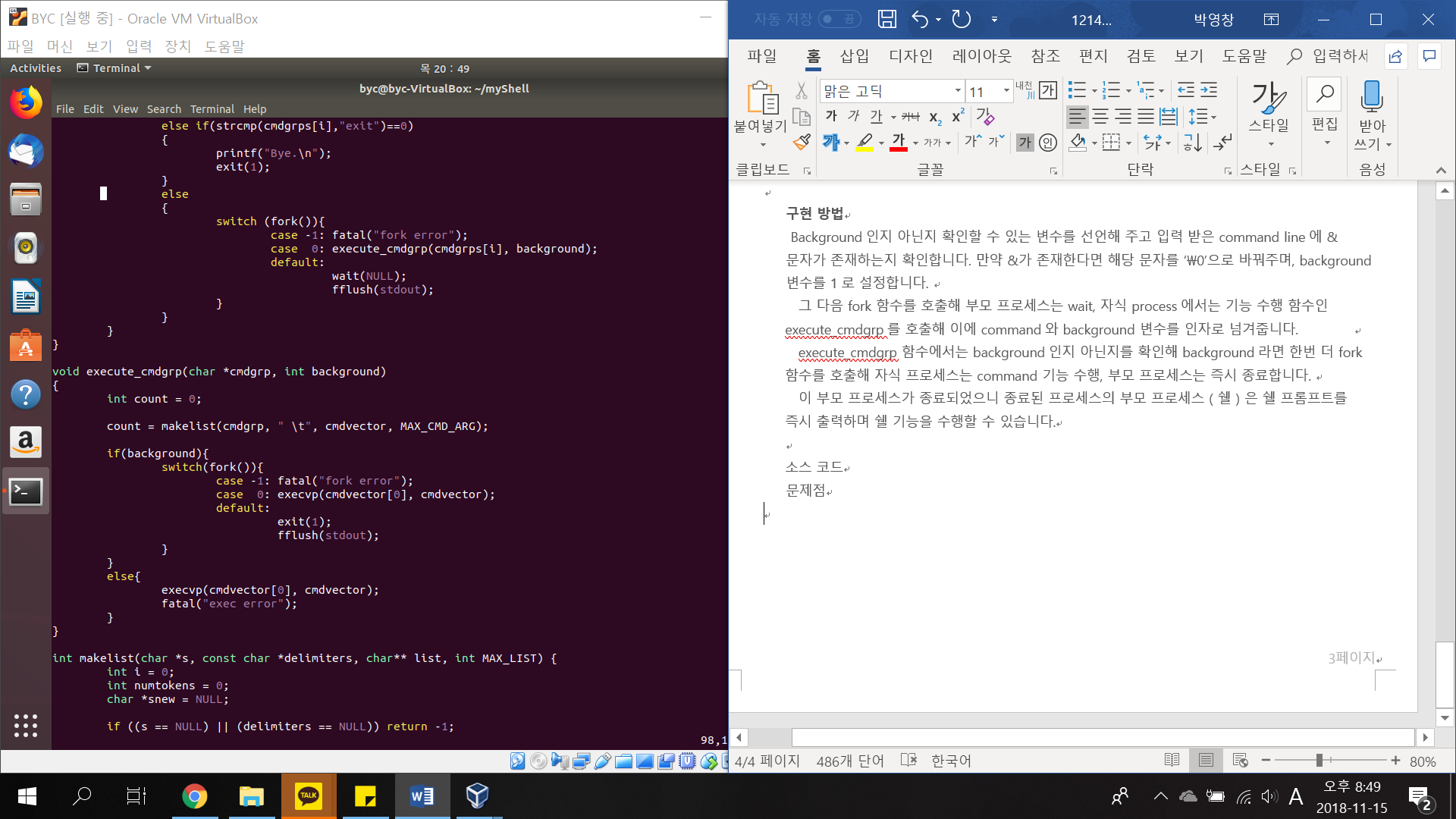
* 1. ‘exit’ 명령 구현
     + - <string.h> 내의 strcmp 함수를 사용하여 문자열을 비교. 입력받은 cmdline이 exit라면 종료’ 
       - <- 결과화면
  2. Background 실행을 구현
     + - Cmdline의 뒤에 &가 있다면 해당 기능을 background에서 실행해야 하므로 orphan process의 생성 원리를 이용해 shell process바 방해 받지 않으며 실행되도록 접근
       - 구현

Background인지 확인할 수 있는 변수를 이용해 cmdline에 & 문자가 존재하는지 확인해줌. 존재하면 해당 문자를 ‘\0’ 로 바꿔주고 변수를 1로 설정

그 다음 fork함수를 호출해 부모 process는 wait, 자식 process에서는 기능 수행 함수인 execute\_cmdzip를 호출해 이에 cmd 와 background 변수를 인자로 넘겨줌. execute\_cmdzip함수에서는 background 기능 수행, 부모 process는 즉시 종료. 이 부모 process가 종료되었으니 종료된 process의 부모 process, 즉 shell 은 shell prompt를 즉시 출력하며 shell 기능 수행.



* & 여부에 따라 background 변수를 설정해주고 fork를 하고 수행

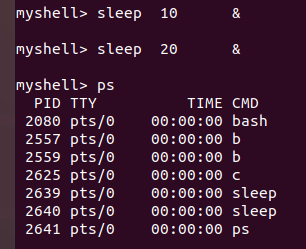


background 변수 값이 1이라면 한번 더 fork를 호출해 자식 process에서 exec, 부모 process는 exit해 orphan process로 만듬.

1. 고찰
   1. 좀비 프로세스가 생기는 이유?
      1. 좀비 프로세스란 fork, exec를 사용해 자식 process는 특정 기능을 수행하고 부모 process는 자식을 기다리지 않을 때의 자식 process임. Myshell을 구현할 때 orphan process를 이용하여 구현하여 좀비 process가 생성되지 않음.

Fork를 두 번 호출하므로 현제 쉘에서는 자식 process가 바로 종료되기 때문에 기다릴 자식 process가 존재하지 않고, 자식의 자식 process가 기능을 수행하는 process가 orphan process가 되어 바로 최상위 process의 자식으로 들어가기 때문에 좀비 process가 발생하지 않다고 유추.

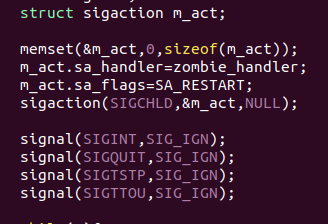
* 1. 백그라운드를 기다리는 데에 발생하는 문제점



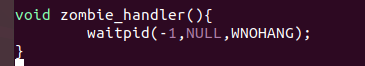
어떠한 문제인지 찾을 수가 없음.

2차 REPOPRT

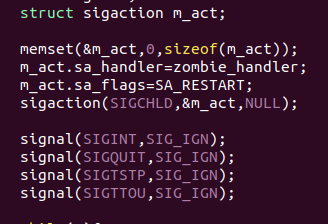
DEADLINE : 2019-11-15

1. 요구사항
   1. SIGHLD로 자식 프로세스 wait() 시 프로세스가 온전하게 수행되도록 구현
   2. ^C(SIGINT), ^\(SIGQUIT) 사용 시 쉘이 종료되지 않도록, Foreground 프로세스 실행 시 SIGINT를 받으면 프로세스가 끝나는 것을 구현
2. 구현방법
   1. SIGCHLD로 자식 프로세스 wait() 사용시 프로세스가 온전하게 수행하도록 구현
      1. Sigaction 구조체에 sa\_handler를 구현한 zombie\_handler를 설정하고, sa\_handler를 SA\_RESTART로 설정해 준 뒤에, sigaction 함수를 통해 SIGCHLD signal 과 앞에서 설정한 sigaction 구조체를 인자로 줌으로써, SIGCHLD signal이 발생 했을 때, zombie\_handler 함수가 실행되고, 일부 system call이 signal을 통해 재시작 할 수 있도록 설정함. Zombie\_handler의 내용에는 waitpid 함수를 사용함. 첫번째 인자로 -1 을 줌으로써 임의의 process를 기다리고, 두번째 인자를 사용하지 않으며 세번째 인자로 WNOHANG을 줌으로 자식 프로세스를 확인만 하고 복귀해 부모 프로세스가 block 되지 않도록 함
      2. 

Sigaction 구조체 act에 값 설정 후 sigaction 함수에 SIGCHLD와 함께 인자로 넘겨줌.

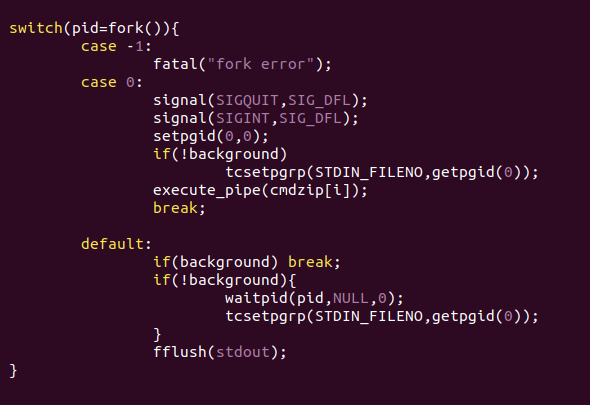
* + 1. 15

임의의 프로세스를 기다림.

* 1. ^C(SIGINT), ^\(SIGQUIT) 사용 시 쉘이 종료되지 않도록, foreground 프로세스 실행 시 SIGINT를 받으면 프로세스가 끝나는 것을 구현
     1. Signal 함수를 통해 쉘 프로세스 (main) 에서 무시하고자 하는 singal들을 SIG\_IGN을 인자로 줘서 무시하는 것 만으로 쉘은 signal에 의해 종료되지 않는 문제가 발생함. 하지만, 이 설정 때문에 foreground 프로세스 또한 signal 에 의해 종료되지 않음. 즉, fork하는 switch 문 부분에서 자식 프로세스라면 signal함수를 통해 SIG\_DFL을 인자로 줘서 다시 signal을 활성시키면 자식 프로세스를 singal로 종료할 수 있음. 처리해 주어야 하는 부분이 하나 남았는데, foreground 제어권을 자식 프로세스가 갖도록 넘겨주어야 함. 이는 background 프로세스가 아닐 경우에 무조건 제어권을 넘겨주면 되고, 부모 프로세스는 자식을 기다리고 난 뒤 다시 제어권을 가져와야 함.
     2. 

Main 함수 내에서 signal 함수를 호출해 처리

SIGINT 는 ^C, SIGQUIT는 ^\, SIGTSTP는 ^Z를 SIGTTOU는 백그라운드 프로세스가 터미널에 write시도함.

* + 1. 

Fork를 통해 switch 문 진입

자식 프로세스

SIGQUIT/SIGINT에 대해 SIG\_DFL을 사용해 signal을 활성화 함.

Shell process와 다른 PGID로 설정함.

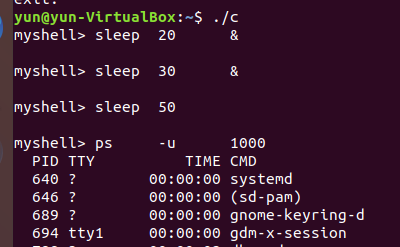
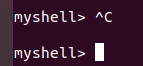
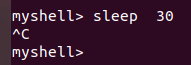
Foreground 프로세스일 경우 제어권을

얻어 옴.

부모 프로세스

Background 라면 기다리지 않음

Foreground 라면 pid를 기다리고 난 뒤 제어권을 얻어 옴

1. 테스트
   1. 
      1. sleep 50 이 sleep 20& 과 sleep 30 & 에 방해 받지 않고 수행됨. Ps를 통해 좀비 process없음을 확인 함.
   2. 
      1. 무시되고, 쉘이 종료되지 않음
   3. 
      1. 제어키(^C)를 받아 foreground 프로세스 sleep이 죽음을 확인함.

3차 report

Deadline : 2019-11-22

1. 목표
   1. 쉘의 주요 기능인 redirection 처리 기능과 pipe 처리 기능 구현
   2. Redirection

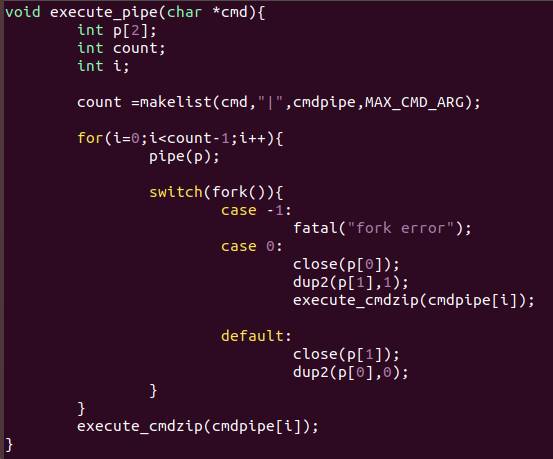
* 표준 입출력 등을 사용자 지정 위치로 우회할 수 있는 기능을 말함. 방향 지정, 방향 변경
* < : 뒤에 오는 파일 내용을 입력으로 앞의 명령어를 수행함
* > : 뒤에 오는 파일 내용에 앞의 명령어의 결과를 출력
  1. Pipe
* 프로세스간 통신을 가능하게 해주는 버퍼
* | 를 통해 사용
* | : 앞과 뒤 프로세스를 연결 가능
* 어떠한 명령어의 표준 출력을 또 다른 명령어의 표준입력으로 연결가능
* | 앞의 명령 결과가 뒤의 명령어의 입력으로 들어감

1. 요구사항
   1. Redirection 처리 기능 구현
   2. Pipe 처리 기능 구현
2. 구현 방법
   1. Redirection 처리 기능 구현

* execute\_cmdzip 함수에서 execvp 함수를 통해 명령어를 수행하기전에 redirection 이라는 함수를 호출하여 redirection 여부를 확인함과 동시에 입출력 위치 지정. Redirection 이라는 함수는 인자로 실행하기 직전의 명령어를 인자로 받아 그 안에 < 또는 > 이 있는지 확인하고, 있다면 바로 다음에 오는 문자열을 특정 문자를 기준으로 자르는 함수 strtok으로 찾아냄. 이 문자열은 입출력 위치를 지정할 대상이 될 파일명으로, 파일을 open 후 dup2 함수를 호출해 입출력 위치를 지정함. < 또는 > 를 널 값으로 변경.



* if ‘<’ : strtok 함수를 이용해 file 이름을 찾아 open하고, 해당 fd를 표준 입력으로 함.
* If ‘>’ : strtok 함수를 이용해 file 이름을 찾아 open 하고, 해당 fd를 표준 출력으로 함.
  1. Pipe 처리 기능 구현
* Execute\_pipe 함수가 실행 되었을 때, ‘|’ 를 기준으로 makelist 함수를 호출해 문자열을 분리하여 저장
* 그 다음 생성된 문자열의 개수보다 1 적은 횟수의 반복문을 실행하는 이유는 반복문 내에서 이미 마지막 명령어로의 입력을 모두 준비했기 때문이고, 더 이상 준비할 입력이 없기 때문
* fork시 자식 프로세스는 부모 프로세스의 open file descriptor table을 상속받게 하기 위해서 반복문 내에서는 pipe 호출 후에 fork를 호출함
* 이를 상속한 후, 자식 프로세스에서는 pipe 중 입력 부분에 해당하는 p[0] 를 닫아버리고, dup2를 호출해 출력부분인 p[1] 을 표준 출력으로 지정함. 그다음 execute\_cmdzip 을 호출해 명령어를 수행함.
* 반면, 부모 프로세스에서는 pipe 중 출력 부분에 해당하는 p[1]을 닫아버리고, dup2를 호출해 입력부분인 p[0]를 표준 입력으로 지정함
* 위 과정은 ‘|’ 앞에 오는 명령어를 자식 프로세스에서 실행하되 표준 출력 위치로 지정하고, 부모 프로세스에서는 표준 입력으로 이를 받아드려 앞 뒤 프로세스를 서로 연결. 따라서 execute\_cmdzip 은 앞에서 받아온 입력으로 마지막 함수를 실행함.

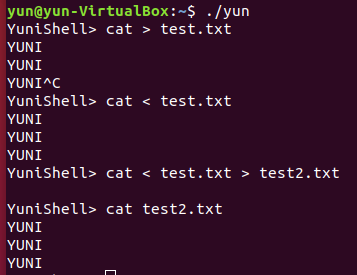


* ‘|’ case 0 는 자식 프로세스

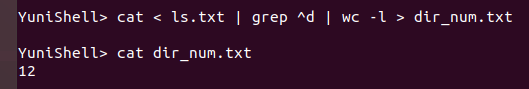
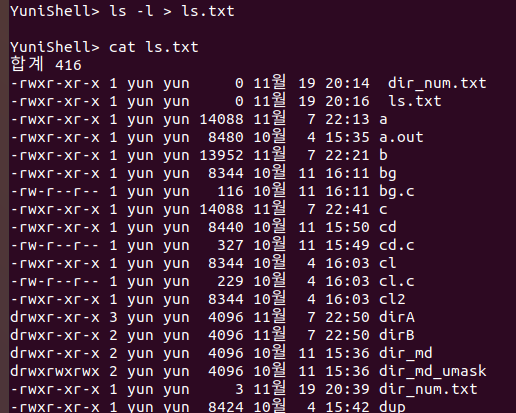
Default는 부모 프로세스



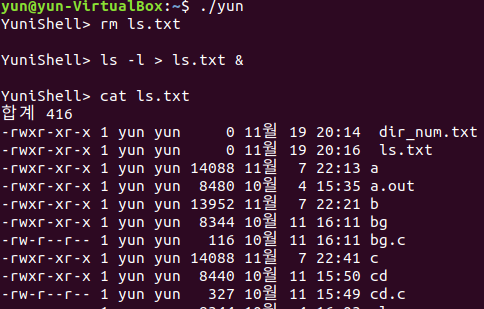
1. 테스트
   1. Redirection 확인



* Cat > 으로 test.txt 파일을 작성하고, cat < test.txt 로 출력
* Cat < test.txt > test2.txt 로 새로운 test2.txt 란 파일에 test.txt 내용를 입력 후 cat test2.txt 로 출력 확인
  1. Redirection, Pipe 동시 사용 확인



* ls -l 의 내용을 redirection ‘>’로 ls.txt 파일에 입력하고, cat ls.txt 로 출력하여 확인.
* 그 후 cat < 으로 ls.txt의 내용 중 pipe 를 이용해 directory만 선별하고, wc -l 로 라인의 개수를 입력으로 받고, ‘>’를 이용해 dir\_num.txt에 출력.
  1. Redirection, background 동시 사용 확인



* 기존의 ls.txt 파일을 삭제하고 ls -l > ls.txt & 명령어로 background로 redirection 실행 후, cat ls.txt을 사용해 확인해 보니 처리가 됨을 확인함.