**REPORT**

**| Unix Programming**

**| Shell Project 1**

**| Deadline : 2018/11/20**

**| 컴퓨터정보공학과**

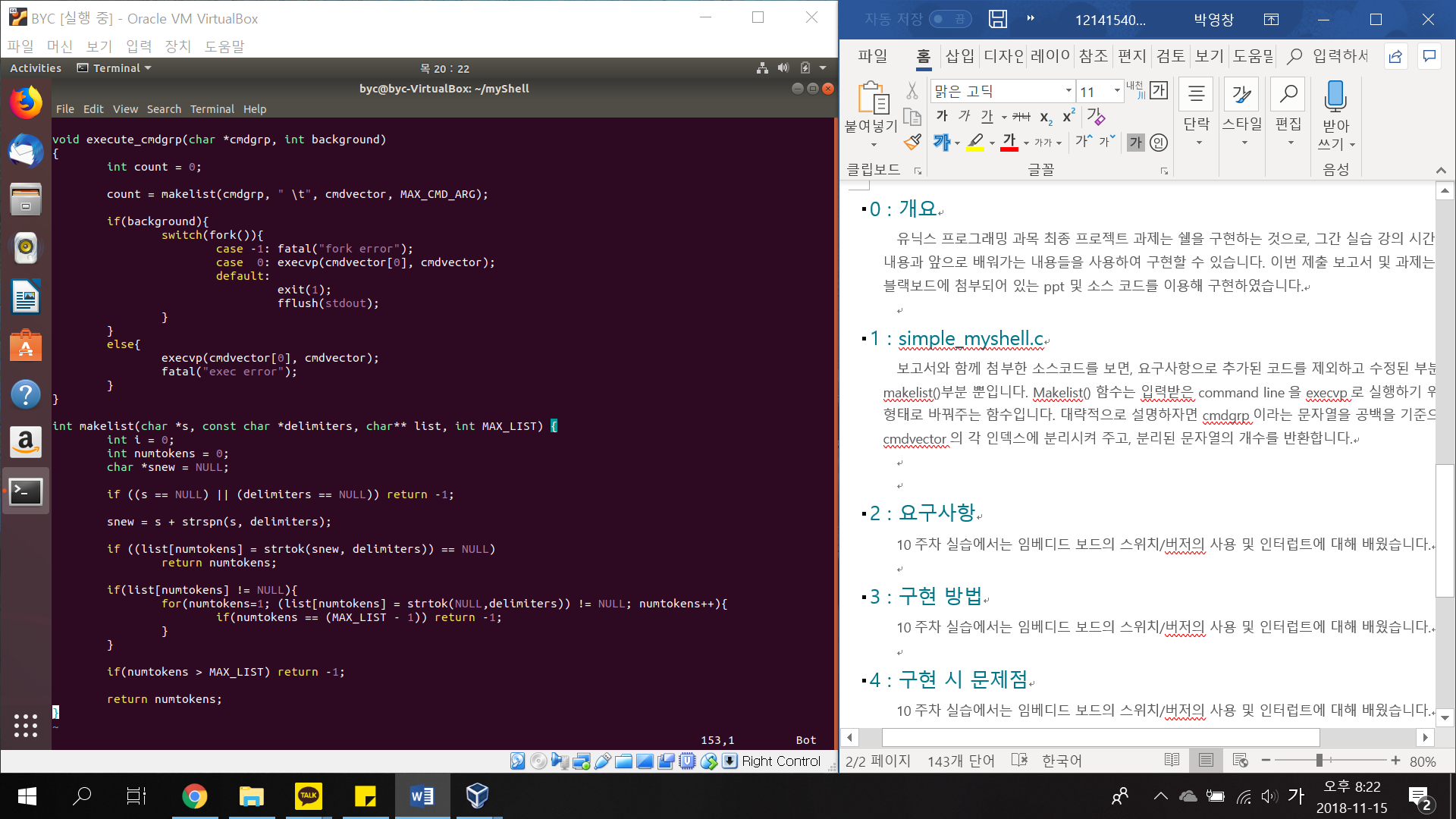
**| 12141540 박영창**

# [0] : 개요

유닉스 프로그래밍 과목 최종 프로젝트 과제는 쉘을 구현하는 것으로, 그간 실습 강의 시간에 배웠던 내용과 앞으로 배워가는 내용들을 사용하여 구현할 수 있습니다. 이번 제출 보고서 및 과제는 대부분 블랙보드에 첨부되어 있는 ppt 및 소스 코드를 이용해 구현하였습니다.

# [1] : simple\_myshell.c

보고서와 함께 첨 부한 소스코드를 보면, 요구사항으로 추가된 코드를 제외하고 수정된 부분은 makelist()부분 뿐입니다. Makelist() 함수는 입력받은 command line을 execvp로 실행하기 위해 vector 형태로 바꿔주는 함수입니다. 대략적으로 설명하자면 cmdgrp이라는 문자열을 공백을 기준으로 cmdvector의 각 인덱스에 분리시켜 주고, 분리된 문자열의 개수를 반환합니다.



# [2] : 요구사항

이번 주차 과제의 요구사항은 myshell에 다음 기능을 추가하는 것이었습니다.

1. **‘cd’ 명령이 제대로 작동하지 않는 버그를 수정**
2. **‘exit’ 명령을 구현**
3. **백그라운드 실행을 구현**

# [3] : 구현 방법 및 문제점

**1) ‘cd’ 명령이 제대로 작동하지 않는 버그를 수정**

**접근**

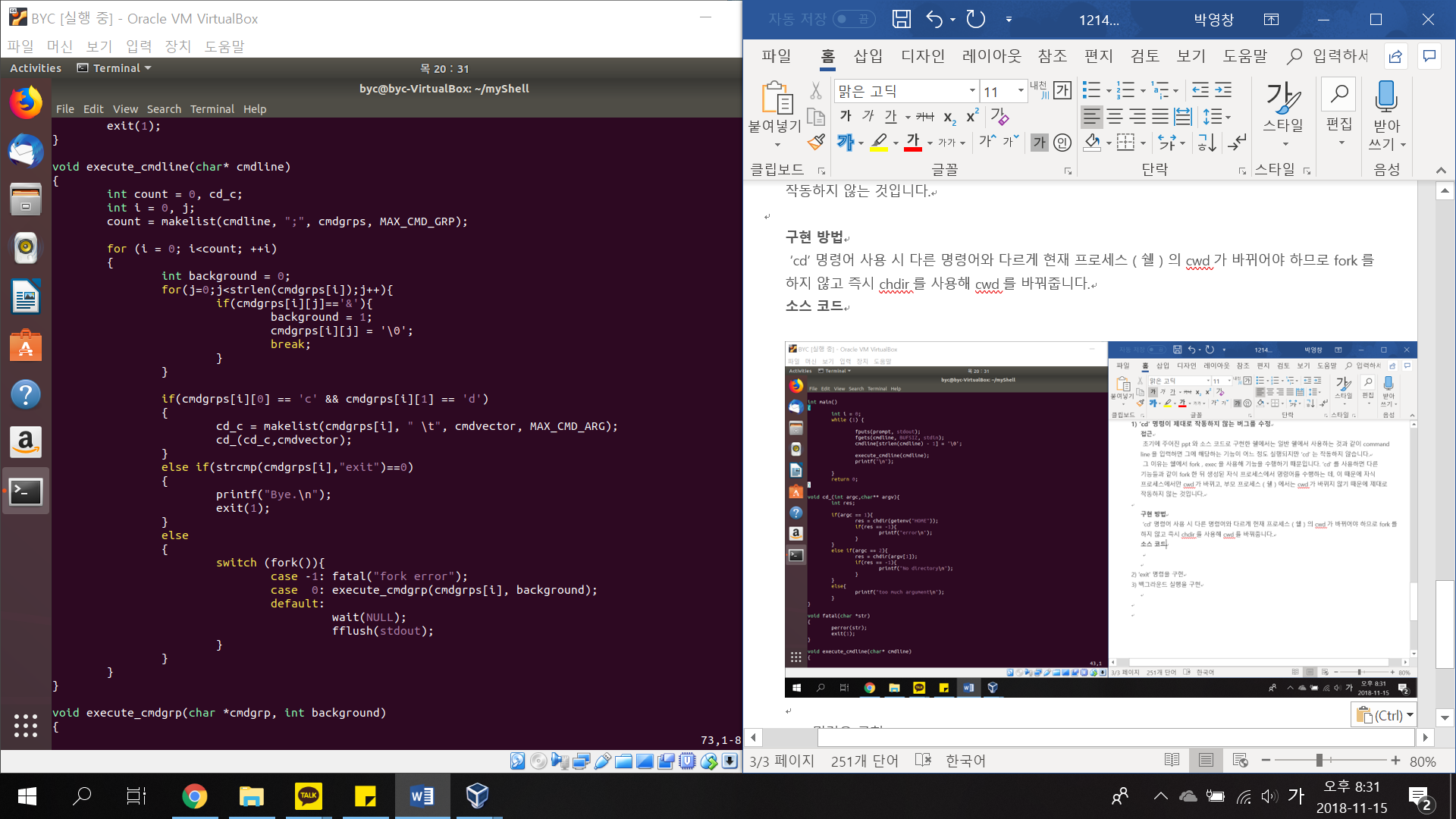
초기에 주어진 ppt와 소스 코드로 구현한 쉘에서는 일반 쉘에서 사용하는 것과 같이 command line을 입력하면 그에 해당하는 기능이 어느 정도 실행되지만 ‘cd’ 는 작동하지 않습니다.

그 이유는 쉘에서 fork , exec을 사용해 기능을 수행하기 때문입니다. ‘cd’ 를 사용하면 다른 기능들과 같이 fork한 뒤 생성된 자식 프로세스에서 명령어를 수행하는 데, 이 때문에 자식 프로세스에서만 cwd가 바뀌고, 부모 프로세스 ( 쉘 ) 에서는 cwd가 바뀌지 않기 때문에 제대로 작동하지 않는 것이었습니다.

**구현 방법**

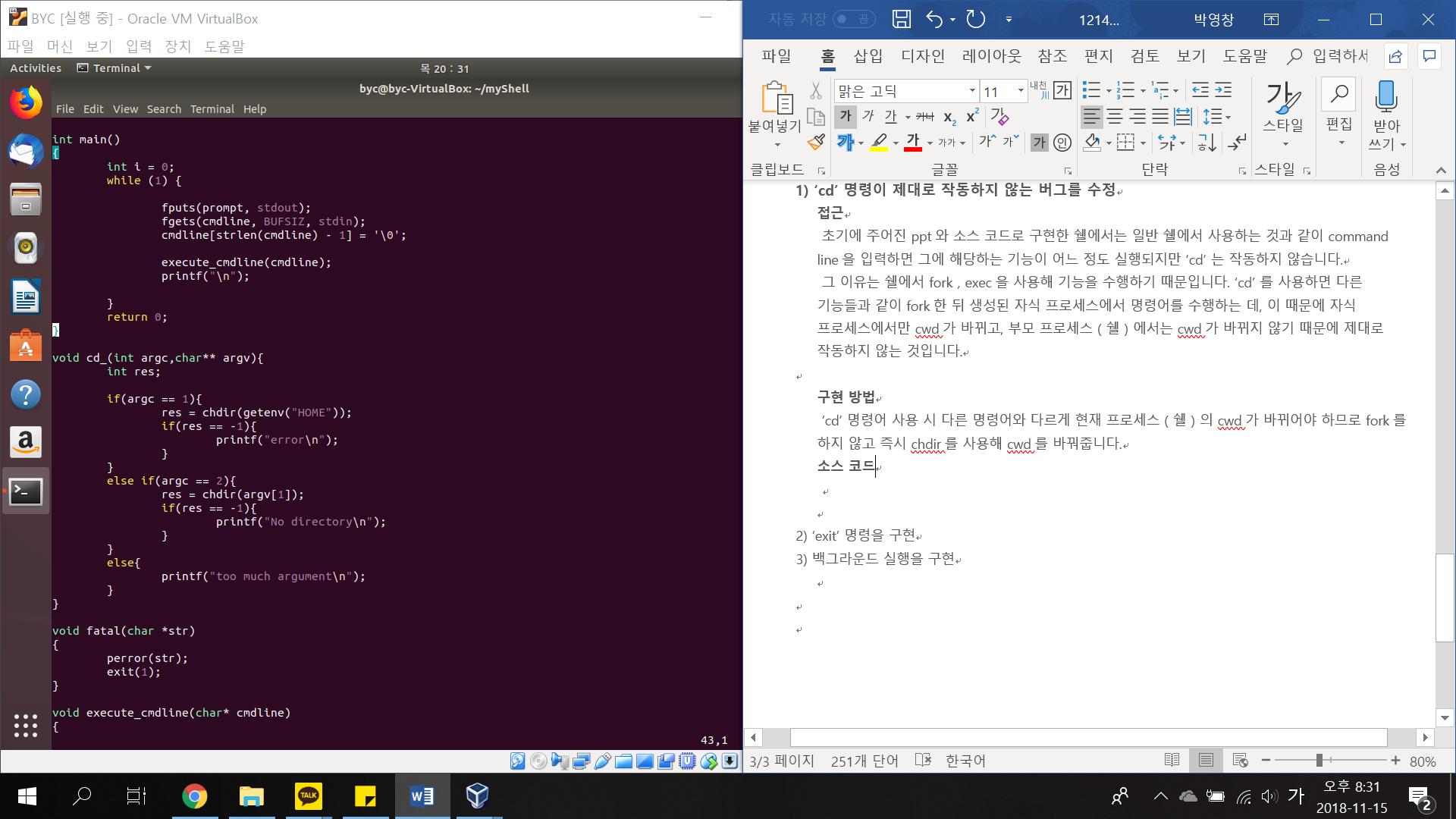
‘cd’ 명령어 사용 시 다른 명령어와 다르게 현재 프로세스 ( 쉘 ) 의 cwd가 바뀌어야 하므로 fork를 하지 않고 즉시 chdir를 사용해 cwd를 바꿔줍니다.

**소스 코드**

command line이 cd로 시작하는지

cd 명령어라면 makelist 호출 후

cd\_ 함수 호출



cd 뒤에 아무것도 오지 않은 경우

HOME 디렉토리로 이동한다

cd 뒤에 이동할 디렉토리가 입력된 경우

해당 디렉토리로 이동한다

**문제점**

1. HOME 디렉토리로 이동 시 쉘 코드가 있는 디렉토리가 아닌 전체 환경의 HOME으로 이동
2. ‘cd’ 명령어 사용 시 cd\_를 실행해야 하지만 command의 첫 번째와 두 번째 문자만 확인해 기능을 수행하므로 잘못된 명령어 입력 시 오류가 발생할 수 있음

**2) ‘exit’ 명령을 구현**

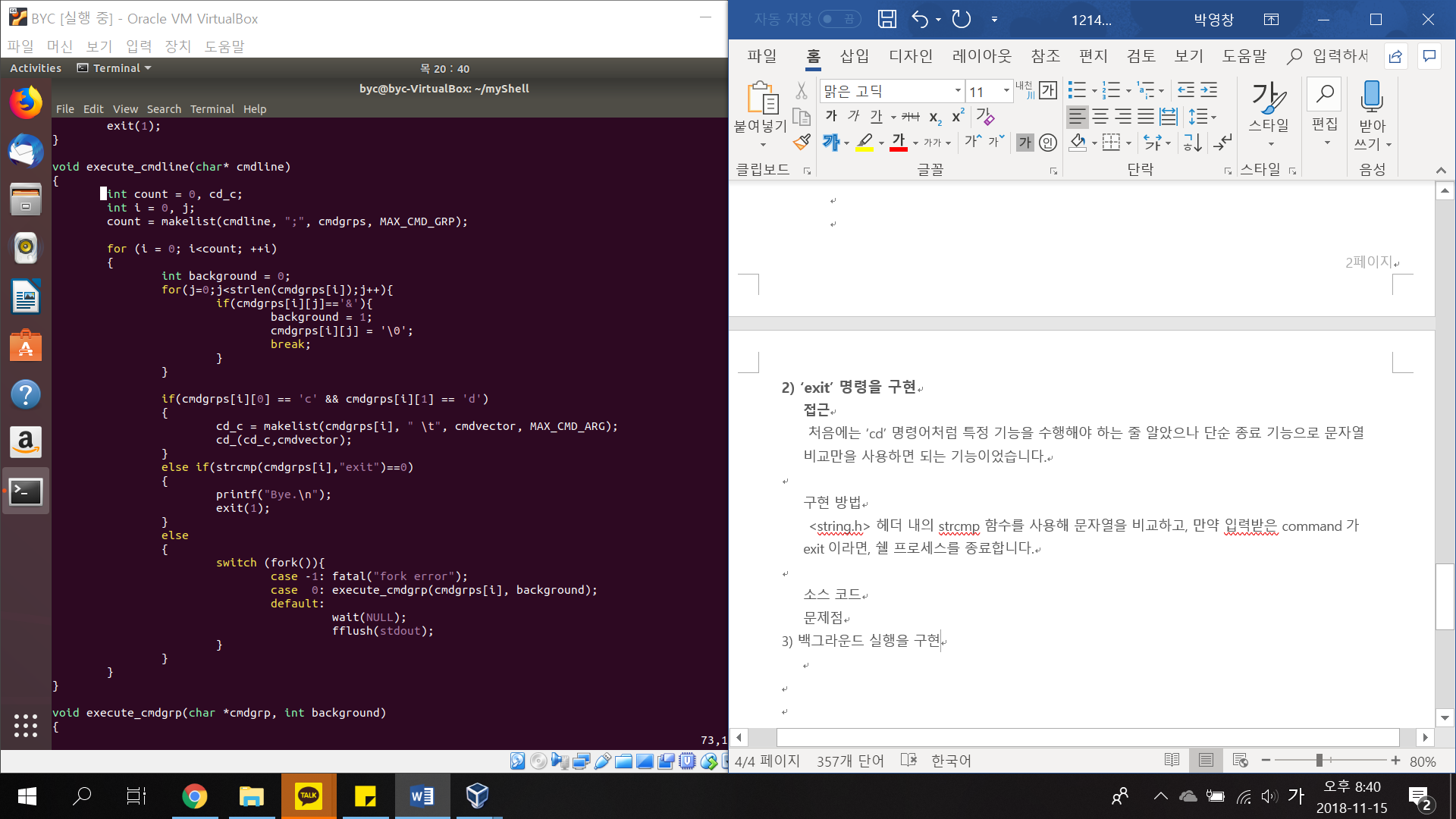
**접근**

처음에는 ‘cd’ 명령어처럼 특정 기능을 수행해야 하는 줄 알았으나 단순 종료 기능으로 문자열 비교만을 사용하면 되는 기능이었습니다.

**구현 방법**

<string.h> 헤더 내의 strcmp 함수를 사용해 문자열을 비교하고, 만약 입력받은 command가 exit이라면, 쉘 프로세스를 종료합니다.

**소스 코드**

else if인 이유는 exit과 비교 이전에 ‘cd’ 명령어인지 비교했기 때문입니다.

**3) 백그라운드 실행을 구현**

**접근**

Command line의 뒤에 &가 있다면 해당 기능을 백그라운드에서 실행해야 하므로 고아 프로세스의 생성 원리를 이용해 쉘 프로세스는 해당 기능 때문에 방해 받지 않으며 실행되도록 접근했습니다.

**구현 방법**

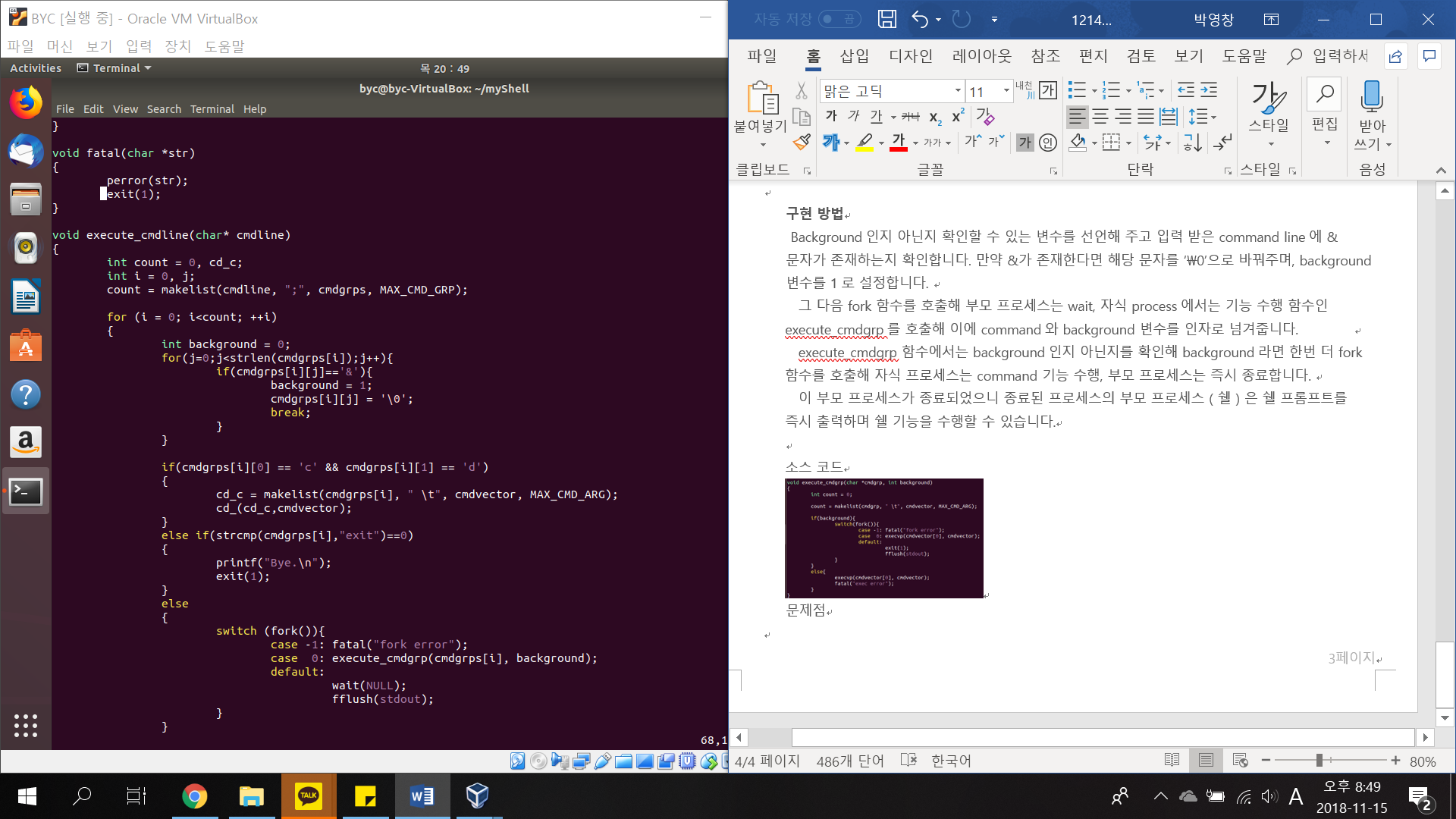
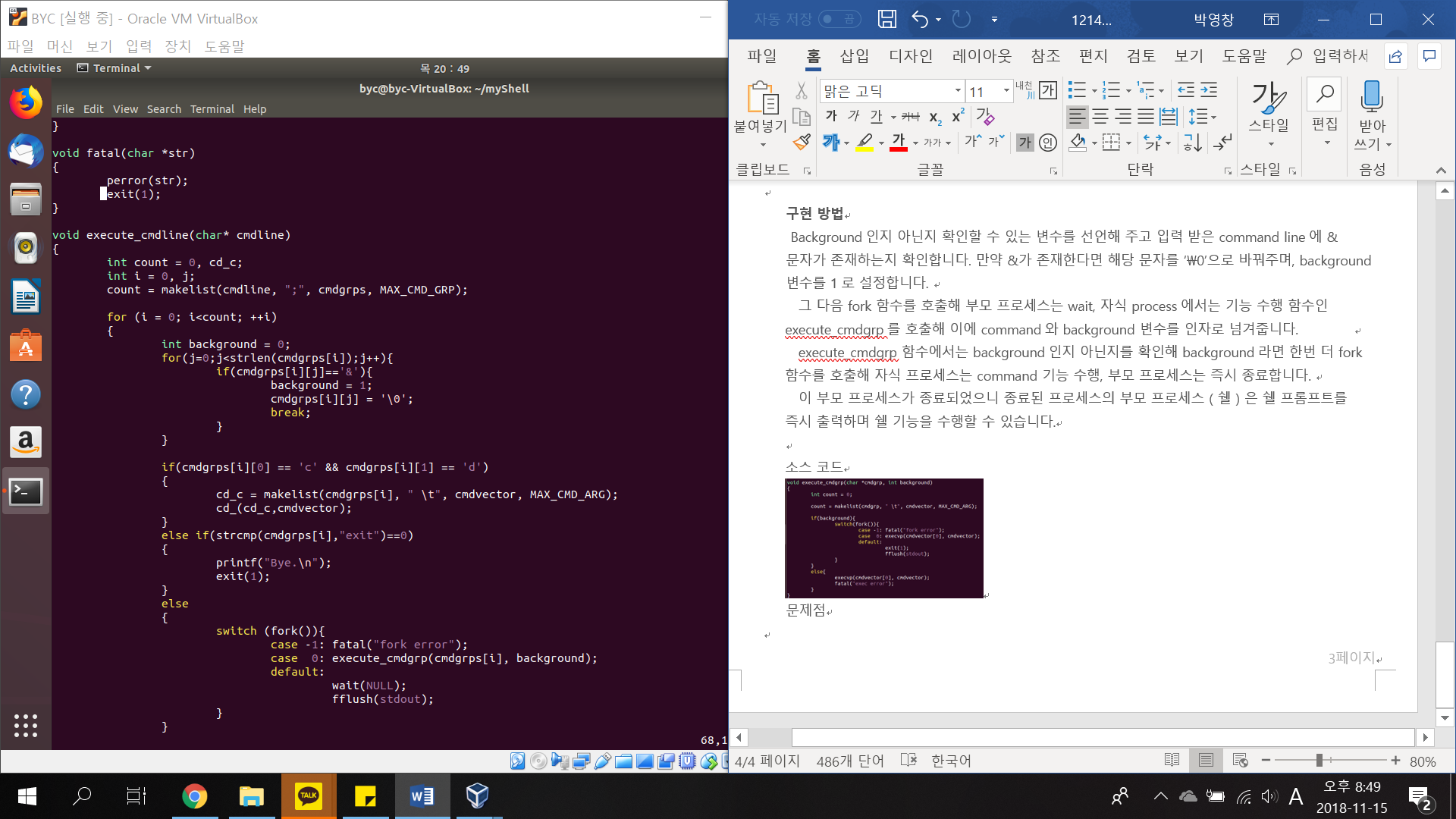
Background인지 아닌지 확인할 수 있는 변수를 선언해 주고 입력 받은 command line에 & 문자가 존재하는지 확인합니다. 만약 &가 존재한다면 해당 문자를 ‘\0’으로 바꿔주며, background 변수를 1로 설정합니다.

그 다음 fork 함수를 호출해 부모 프로세스는 wait, 자식 process에서는 기능 수행 함수인 execute\_cmdgrp를 호출해 이에 command와 background 변수를 인자로 넘겨줍니다.

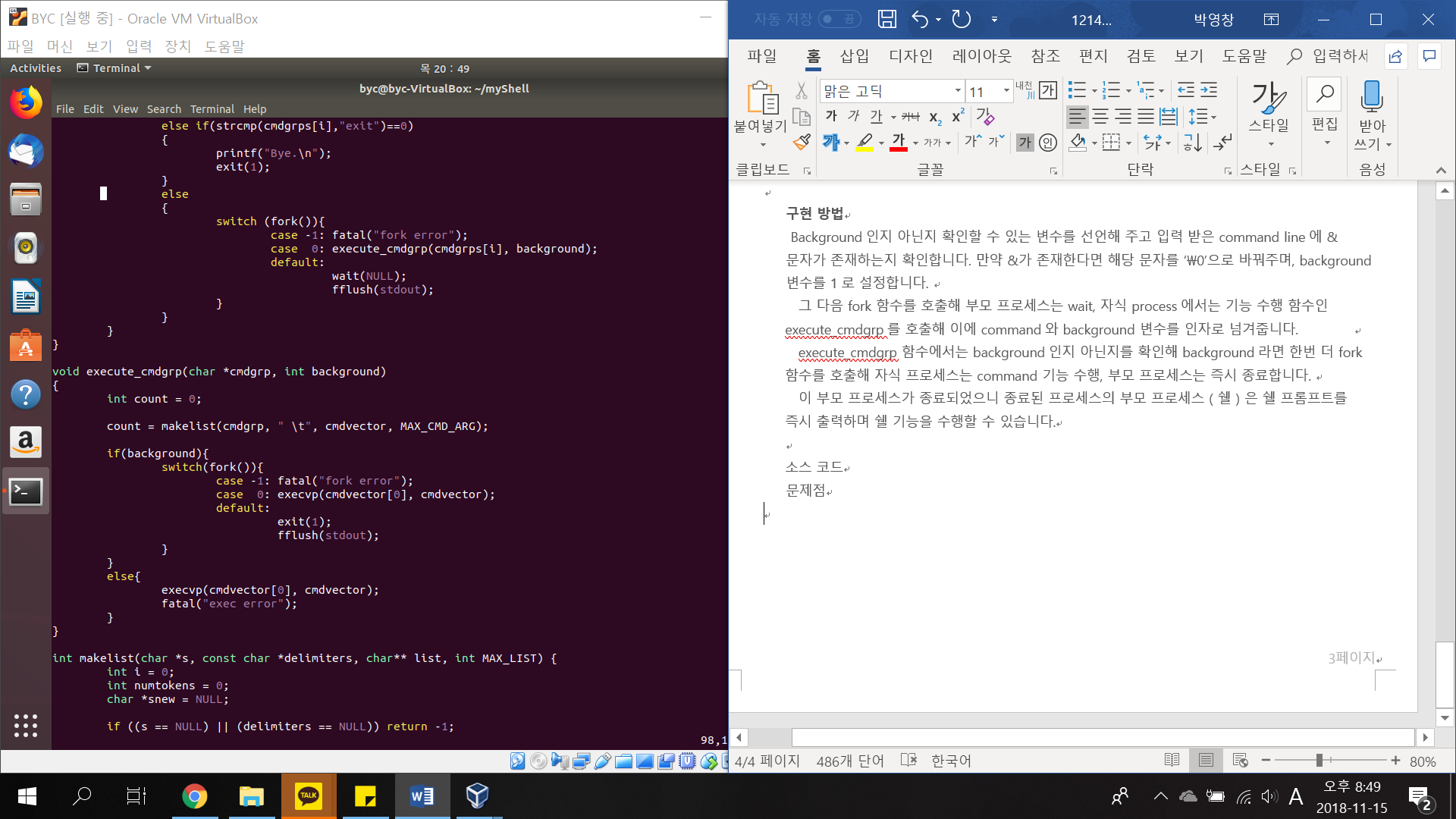
execute\_cmdgrp 함수에서는 background인지 아닌지를 확인해 background라면 한번 더 fork 함수를 호출해 자식 프로세스는 command 기능 수행, 부모 프로세스는 즉시 종료합니다.

이 부모 프로세스가 종료되었으니 종료된 프로세스의 부모 프로세스 ( 쉘 ) 은 쉘 프롬프트를 즉시 출력하며 쉘 기능을 수행할 수 있습니다.

**소스 코드**

& 여부에 따라 background 변수를 설정해주고, 쉘 프로세스에서 fork해 기능을 수행



background가 1로 들어왔다면, 한번 더 fork를 호출해 자식 프로세스에서 exec, 부모 프로세스는exit해 고아 프로세스로 만든다.

# [4] : 고찰

**1) 좀비 프로세스가 생성되는 이유**

좀비 프로세스란 fork , exec를 사용해 자식 프로세스는 특정 기능을 수행하고 부모 프로세스는 자식을 기다리지 않을 때의 자식 프로세스를 의미합니다. 저는 쉘을 구현할 때 고아 프로세스를 아이디어로 접근해 fork를 두 번 호출하였는데, 블랙보드에 올라와 있는 예제를 입력하여도 좀비 프로세스가 생성되지 않았습니다.

좀비 프로세스가 생성되지 않는 이유는 이 때문인 것 같습니다. Fork를 두 번 호출하므로 현재 쉘 프로세스에서는 자식 프로세스가 바로 종료되기 때문에 기다릴 자식 프로세스가 존재하지 않고, 자식의 자식 프로세스가 기능을 수행하므로 기능을 수행하는 프로세스가 고아 프로세스가 되어 바로 최상위 프로세스의 자식 프로세스로 들어가기 때문에 좀비 프로세스가 발생하지 않는다고 생각합니다.

**2) 백그라운드를 기다리는 데에 발생하는 문제점**

이 문제점은 블랙보드에 올라와 있는 예제를 입력해도 어떤 문제점인지 찾을 수가 없었습니다…

