**유닉스 프로그래밍 프로젝트 보고서**

**12151489 조운택**

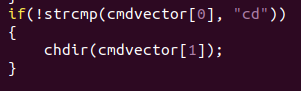
**제출일 : 2019-11-05**

**셀 구현 프로젝트 과제 #1**

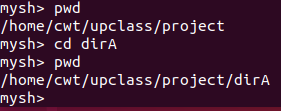
**1번 요구사항 : ‘cd’ 명령이 제대로 먹히지 않는 버그를 수정.**

**힌트에 보면 chdir()의 범위는 현재 프로세스에 제한이라는 글자가 있었다. 이를 토대로 생각해보니, main문에서 shell은 자식프로세스 한 개 만들고, 자식 프로세스를 exec 시스템 콜을 사용하여, Command로 바꾸는 구조이다. 하지만 cd를 자식프로세스에서 수행하면, 거기서는 정상으로 수행이 되지만, 자식프로세스가 종료 된 후, 부모 프로세스에서는 자식 프로세스에서의 cd에 대한 변경 사항이 적용이 안되는 문제점이 있었다. 이러한 문제 원인을 바탕으로, switch문 밖에서 만약에 cd라는 명령어가 들어온 경우에, switch문으로 가지 않고, 단순히 chdir을 수행한 뒤에 넘어가는 코드를 통해 문제점을 해결하였다.**

**구현코드**



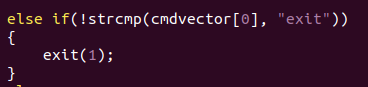
**결과창**



**2번 요구사항 : exit 명령을 구현하시오.**

**이것 또한 ch 명령과 같은 문제점을 가지고 있었다. 자식프로세스에서 exit를 수행하게 된다면 부모프로세스에게 영향을 주지 못하므로, 이것을 cd와 같이 main문의 switch문에서 밖으로 꺼내, 들어오는 명령어가 exit인지 확인하고, exit인 경우 switch문 밖에서 exit를 수행하였다.**

**구현코드**



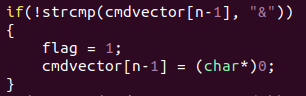
**결과창**

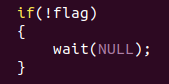


**3번 요구사항 : 백그라운드 실행을 구현**

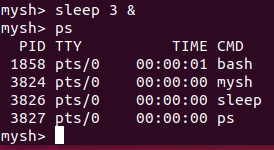
**백그라운드 실행의 가장 중점인 사항은 아마도 내가 실행하려고 했던 명령어는 백그라운드에서 실행이 되면서, 나는 포그라운드에서 내가 원하는 작업을 또 수행할 수 있다는 점이다. 이것은 쉽게말해, 백그라운드에서 sleep을 한 경우, 포그라운드에서도 프롬트가 뜬다는 것을 가장 쉬운 예시로 들 수 있다. 그렇다면 이를 수행하기 위해서는 &를 Command 마지막에 붙인 경우, 부모 process가 자식 process가 종료될 때 까지 wait를 하지 않으면 된다. 그렇기 때문에 flag를 세워두고, &가 온경우, 부모 process에서 wait를 하지 않은 식으로 구현하였다.**

**구현 코드**



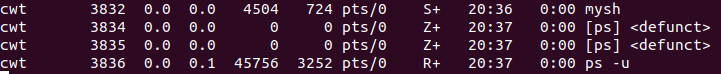


**결과창**



**[고찰 1] 좀비 프로세스가 생기는 이유에 대하여 보고서에 작성할 것.**

**위에서 쓴것과 마찬가지로, 부모 process가 자식 process가 끝날 때 까지 wait를 하지 않음으로 좀비 프로세스가 생기게 됩니다. 이런 경우에 왜 sleep의 zombie가 아닌, sleep이 끝난 후 ps를 하게 되면 이 ps가 좀비 프로세스로 바뀌는 지를 잘 모르겠습니다. 분명 이 ps는 백그라운드에서 돌린 것도 아닌데 백그라운드에서 돌림으로 부모 process가 wait를 해주지 않기 때문에 자식의 종료를 확인하지 못했다는 것인데, 이유를 잘 모르겠습니다.**

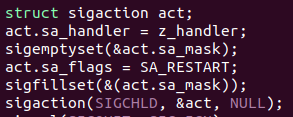
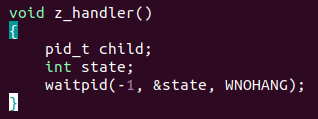


**[고찰 2] 백그라운드 기다리는 테스트 문제**

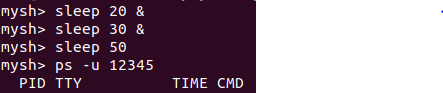
**이 Test 방법의 문제점을 잘 모르겠습니다. 백그라운드를 기다리지 않고 바로 쉘프롬프트가 잘 출력됩니다. 아마 문제점이라면 자식 프로세스에서 1개가 종료되기 전에 또 1개가 생성이 되고, 둘다 부모가 자식프로세스를 wait 안하고, 또 다른 자식프로세스가 만들어 진다면 이 새로운 자식은 zombie 프로세스가 됩니다.**

**셀 구현 프로젝트 과제 #2**

1. **SIGCHLD로 자식 프로세스 wait()시 프로세스가 온전하게 수행되도록 구현 : 이전 프로젝트 구현 #1에서는 백그라운드에서의 실행을 구현하기 위해 부모 프로세스에서 wait를 하지 않도록 막았습니다. 이로써 좀비 프로세스가 발생하게 되었고, 또한 백그라운드 뒤에 나오는 포그라운드에서의 실행도 방해를 받게 되었습니다. 이를 해결하기 위해 백그라운드에서 실행 한 경우에서, 만약 자식이 종료 될 때 SIGCHLD라는 시그널을 보내고, 이를 받게된다면, SIG\_HANDLER를 통해 미리 구현해둔 함수를 통해 waitpid를 수행해서, 자식 프로세스의 종료를 알렸습니다.**

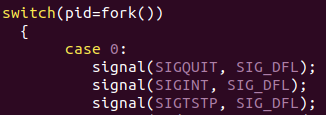
 

**테스트 사항 실행**



1. **쉘 프롬프트 상에서의 ^C, ^\ 사용시, 쉘은 종료되지 않도록 하면서 Foreground에선 SIGNAL을 받고 프로세스를 종료시키는 것을 구현해야 한다. 이것은 간단하게, 부모 Process만의 switch문 위의 코드를 수행하기 때문에, switch문 위에 SIGINT, SIGQUIT, SIGTSTP를 무시하는 코드를 넣고, 자식 프로세스에서는 각각을 DFL로 실행하는 signal 코드를 넣어준다면 쉘은 종료되지 않으면서, 자식 프로세스에서는 종료되는 효과를 얻을 수 있다.**

**-swtich문 위에 위치- -swtich문 안에 위치-**

**테스트 사항 실행**

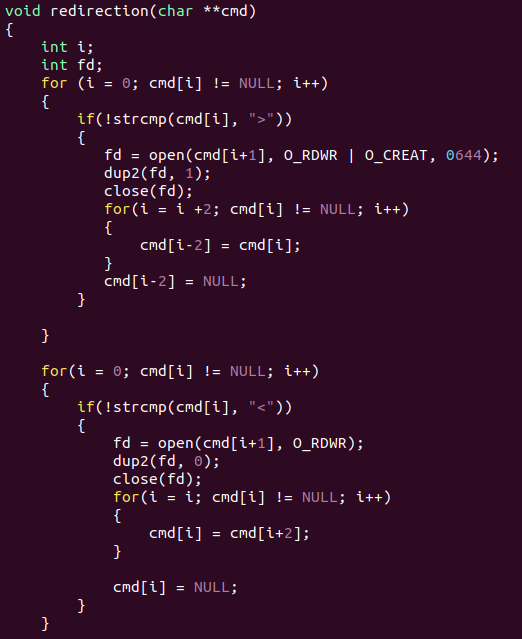
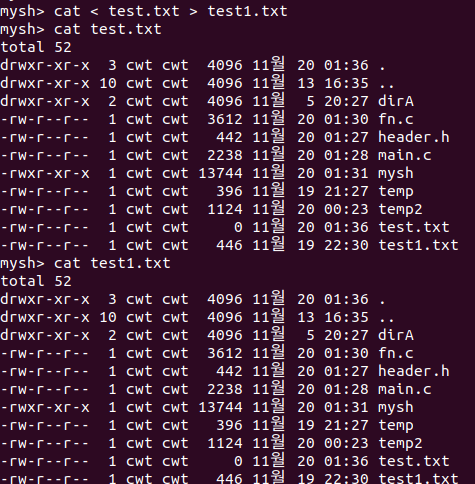


**셀 구현 프로젝트 과제 #3**

1. **Redirection 구현**

**Dup2를 통해 표준 입력, 표준 출력을 바꿔주는 역할을 수행하였고, 만약 한 문장에서 여러가지의 <,>등이 들어오는 문제를 해결하기 위해 for문을 돌렸습니다. 가장 주의해야할 것은 쓰고난 표준출력을 close해주는 과정이었습니다.**

**코드 결과**

1. **PIPE 구현**

**PIPE에 대해선 예저 코드를 많이 참고했습니다. 여기서 PIPE는 1개만 만들고서, fork를 실행하는 과정을 파이프의 개수 만큼 돌렸고, 마지막의 GP밑에 있는 P에서 실행할 명령을 for문 밖에 위치함으로써, 모든 자식들이 준비가 수행이 끝날때까지 자동으로 기다리게 되었습니다.**

**또 한, 중요한 점으로는 close로 pipe의 p[0]와 p[1]을 계속 Close해주는 과정이 있었습니다. 처음에 여기서 조금 헤매가지고 좀 오래 걸렸던거 같습니다.**

**코드**



**결과**