**REPORT**



|  |  |
| --- | --- |
| **과목명** | 유닉스 프로그레밍 |
| **학과** | 컴퓨터공학과 |
| **학번** | 12161658 |
| **이름** | 조재민 |
| **제출일자** | 20.11.16 |

<프로젝트 1>

개요:

이번 3차 유닉스 프로그래밍 과제로는, 쉘을 구현하는 것으로 첫번째로 change directory와 background program이 정상적으로 작동하지 않는 것을 해결하는 과제 였습니다.

**요구사항:**

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

**Change Directory**:

기본 제공되는 template code 에서 ls, mkdir, ps 등 리눅스의 기본 명령어 들은 정상 작동 하지만 cd가 작동하지 않았습니다. 이유는 fork() 함수를 통해 자식 프로세스로 작동할때 change directory를 하면 부모 cwd (current working directory)와 자식의 cwd에 차이가 생기는 문제였습니다.

**텍스트이(가) 표시된 사진

자동 생성된 설명해결방안:** ‘cd’ 명령어를 main 함수 내에 fork() 함수 활용 전 input을 strcmp로 확인하여 change directory를 처리한다.

While문으로 shell programming이 끝나지 않고 계속 실행 되며, 매번 input을 받으면 change directory, 즉 cd 명령어가 입력됐는지를 확인합니다. 이는 fork() 함수와 if문으로 구별해놓아 부모, 자식간의 cwd가 달라지는 문제를 해결하였습니다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**구현된 change directory 함수 입니다.

Argc 가 1, 즉 인자 없는 cd 가 들어오면 home directory로 가도록 getenv 함수를 활용하였습니다.

Cd ~ 활용 역시 정상 작동 하도록 두번째 인자에 예외처리를 하였습니다.

두번째 인자로 들어온 directory가 정상적인 directory라면 해당 directory로 이동합니다.

**결과:**

텍스트이(가) 표시된 사진

자동 생성된 설명

기존 에러인 main: No Such File or Directory 가 아닌

Cd 명령어가 정상적으로 작동하는 것을 확인 하였습니다.

**Background**:

Command line에서 뒤에 & 이 붙으면 background에서 수행하여야 하므로, 고아 프로세스의 생성 원리를 이용하였습니다.

& 입력이 들어왔을때 해당 실행을 background 에서 한다는 것을 구분하기 위하여 background 변수를 생성하여 & 입력 시 해당 값을 바꿔 주었습니다. 또한 & 가 입력으로 들어오면 문자를 ‘\0’ 로 변경해 주었습니다.

그 다음 main 함수의 switch case 에서 background 값이 0, 즉 부모 프로세스라면 waitpid 함수를 호출하여 자식 프로세스가 종료 되었을때 거두어 주는 함수를 활용하였습니다. 이렇게 하여서 자식 프로세스가 좀비 프로세스가 되는것을 방지하였습니다.

**해결방안:**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

If문으로 입력에 & 들어왔는지 확인 및 문자열 변경

부모 프로세스라면 waitpid 함수를 호출

**결과:**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

Sleep 10 사용시 10초 대기 후 shell에 재입력 프로세스가 실행 하지 않고 background 에서 돌아가는 것을 확인 하였습니다.

**좀비 프로세스 생성 이유:**

좀비 프로세스란 fork, exec 를 활용해 자식 프로세스가 별도의 작업을 수행하고 부모 프로세스는 자식을 기다리지 않을 때의 자식 프로세스를 의미합니다. & 가 입력으로 들어왔을 경우 자식 프로세스가 background에서 수행 되는데 해당 프로세스를 별도의 프로세스로 구분 짓고 부모 프로세스를 계속해서 수행 하였기 때문에 해당 자식 프로세스의 수행 종료시 부모 프로세스가 waitpit 함수로 거두어 주어야 합니다. 그렇지 않으면 생성된 자식 프로세스가 종료 됐을경우 할당된 영역을 거두어준 부모가 없기 때문에 소위 ‘찌꺼기’ 처럼 남아 있을 것입니다.

**Exit 구현:**

기본 제공되는 template 코드에서 myshell 프로그램이 종료 될려면 강제 종료 입력인 ^c 입력만이 작동하였습니다. While 문으로 계속해서 실행되는 프로세스 이기 때문에 별도 종료가 없으면 계속해서 실행 되었습니다. 이 부분을 해결 하기위해 입력으로 exit 이 들어오면 프로세스를 종료하도록 설계 하였습니다.

해결방안:

Exit 입력이 들어왔는지 strcmp 함수를 활용하여 비요해 줍니다. 해당 값이 0 이라면, 즉 exit을 입력하였다면 프로세스를 종료 exit(1)을 해주었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

앞서 cd 입력과의 비교가 필요하기 때문에 else if 문으로 시작하였습니다. 또한 밑의 else 문은 현재 cd, exit 의 입력이 들어오지 않아 fork 를 호출하는 함수로 진입합니다.

**결과:**

텍스트이(가) 표시된 사진

자동 생성된 설명

Exit 입력시 정상적으로 메시지 출력 (myshell exit…) 과 함께 종료되었습니다.

<프로젝트 2>

개요:

이번 4차 유닉스 프로그래밍 과제 ‘쉘 프로젝트 #2’ 는 SIGCHLD를 활용하여 자식 프로세스에서 wait() 사용 시 프로세스가 온전하게 수행되도록 구현하는 것 과, ^C (SIGINT), ^\ (SIGQUIT) 사용시 쉘이 종료되지 않고, Foreground 프로세스 실행 시 SIGINT (^C)를 받으면 프로세스가 끝나도록 구현하는 과제입니다.

**요구사항: 다음 기능을 추가**

1. **SIGCHLD로 자식 프로세스 wait() 시 프로세스가 온전하게 수행되도록 구현**
2. **^C (SIGINT), ^\ (SIGQUIT) 사용시 쉘이 종료되지 않도록, Foreground 프로세스 실행 시 SIGINT를 받으면 프로세스가 끝나는 것을 구현**

구현:

1. SIGCHLD로 자식 프로세스 wait() 사용 시 프로세스가 온전하게 수행하도록 구현

이전 1차 쉘 구현에서 문제점인 좀비 프로세스 (Zombie Process) 의 생성되지 않도록 구현을 하며 signal을 활용하여 sigaction 구조체를 설정해주는 과정을 수행하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명Struct sigaction act; 로 act명의 sigaction 구조체를 선언한 뒤 해당 구조체를 설정하였습니다.

Memset을 활용하여 초기화를 해주고, sa\_handler 즉 시그널 발생시 childprocess\_handler 라는 함수를 시행하도록 설정하였습니다. 그 다음 구조체 내에 sa\_flags 변수를 SA\_RESTART 값으로 선언 해준 뒤 sigaction 함수로 선언된 sigaction 구조체를 인자로 넘겼습니다. SIGCHLD signal이 발생했을때 childprocess\_handler 함수가 실행됩니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Childprocess\_handler 내부의 코드입니다. Background로 실행되는 모든 자식 process를 정상적으로 종료될 때 까지 기다리기 위해 while 문과 waitpid 함수를 활용 하였습니다.

Waitpid 함수활용에 넘긴 인자로 -1 을 넘김으로 임의의 프로세스 (즉 모든 프로세스)를 대기하는 명령과, WNOHANG 으로 block 되지 않고 부모 프로세스가 계속 수행할 수 있도록 하였습니다.

결과:

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

이러한 구현으로 이제 좀비 프로세스가 생성되지 않습니다.

1. ^C (SIGINT), ^\ (SIGQUIT) 사용시 쉘이 종료되지 않도록, Foreground 프로세스 실행 시 SIGINT를 받으면 프로세스가 끝나는 것을 구현

Linux 기본 명령어로써 ^C, ^\ 등의 명령어를 활용하여 수행중인 process를 강제적으로 종료 시킬 수 있습니다. 이러한 강제종료를 막기위해 main에서 해당 시그널들을 무시하도록 설계하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Signal 함수를 호출하여 처리 하였습니다. 4가지 signal을 호출하였습니다.

SIGTSTP는 ‘^Z’, SIGQUIT 은 ‘^\’을, SIGINT는 ‘^C’를 위한 인자입니다. 그리고, SIGTSTP는 background process가 수행중 shell 터미널에 write를 시도하는 것 입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명fork로 자식 프로세스를 생성 하였을때 case 0, 즉 자식 프로세스라면 SIGQUIT과 SIGINT에 대해 SIG\_DFL을 활용하여 signal을 활성화 합니다. 그 후 Shell process와는 다른 자신의 process ID (PID)로 설정 해 주며, tcsetpgrp 함수를 통하여 터미널의 제어권을 foreground 프로세스일 경우 가져옵니다.

반대로 부모 프로세스라면 foreground일 때 자식 process를 기다리고 터미널 제어권을 다시 가져옵니다.

결과:

텍스트이(가) 표시된 사진

자동 생성된 설명

Shell 수행 중 ^C, ^\ 등의 명령어를 통한 강제 종료 수행을 거부하도록 설계하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

또한 ^C를 활용하여 foreground의 sleep을 강제 종료하며 ‘child process terminated’ 라는 종료 시그널 핸들러 함수를 호출하도록 설계하였습니다.

또한 좀비 프로세스가 생성되지 않습니다.