운영체제 REPORT

**-HW 1-**

**<Simple Shell>**

**모바일 시스템 공학과**

**32151181**

**김진화**

**CONTENTS**

**1. 주제**

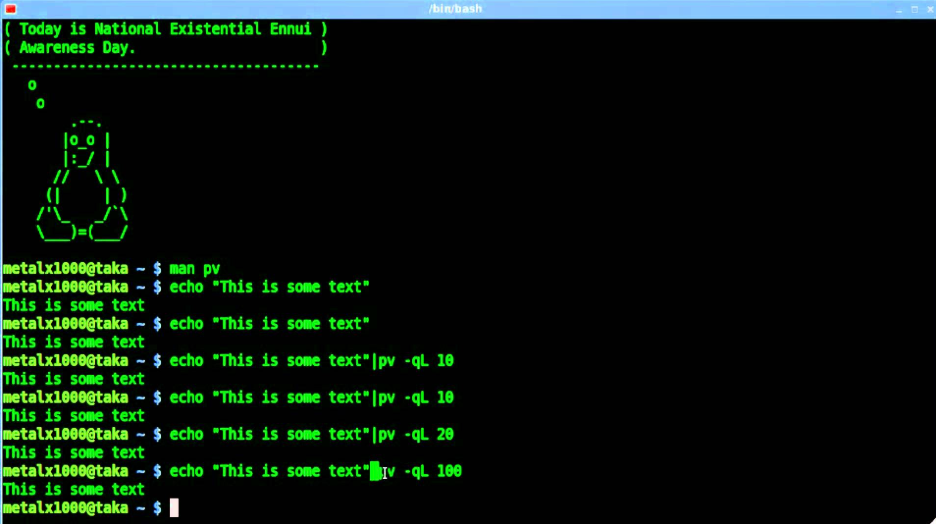
**2. 알고리즘**

**3. 소스코드 및 부연설명**

**4. 결과 및 느낀점**

1. **주제**
2. **Simple Shell**

HW0의 주제는 Simple shell 을 구현하는 것이다. 사용자로부터 프로그램의 이름을 string input으로 받아, 사용자가 원하는 기능을 Execution path를 기준으로 실행시키는 shell을 구현하는 것이다.



1. **Shell 이란?**

Shell은 운영체제상에서 다양한 운영체제 기능과 서비스를 구현하는 인터페이스를 제공하는 프로그램이다. 또한 사용자가 입력하는 명령어를 기계어로 번역하여 커널에 전달하는 역할을 하는 **명령어 해석기**를 말한다.

1. **Fork 개념**

Fork란 프로세스가 자기 자신을 복제하는 동작이다. 이는 일반적으로 시스템 호출의 일종이며, 커널 안에서 구현 된다. Fork 는 유닉스 계열 운영 체제에서 프로세스를 만드는 주된 방식이다. 복제의 대상을 부모 프로세스라 하고 그 결과물을 자식 프로세스라 한다.

1. **Fork 가 필요한 이유**

프로그램에서 exec를 호출하게 되면 exec로 호출되는 프로그램이 현재 메모리에 올라와 있는 프로그램을 덮으면서 로딩이 되기 때문에 현재 메모리에 상주하고 있는 이후 프로그램이 무시되 버린다. 따라서 별도의 메모리 공간을 할당하고, 그 할당된 공간에서 exec를 실행하게 되면 다른 메모리 공간에서 실행되고 있는 원래의 프로세스는 계속해서 실행 될 수 있다. 이러 한 일을 fork 가 해주기 때문에 fork가 필요하다.

1. **알고리즘**

**\*Shell 의 원리는 다음과 같다.**

1. 사용자로부터 명령어를 받는다. (ex) echo $PATH , ls , quit
2. String input으로 받은 명령어를 여러 개의 string으로 parsing 한다.

(ex) echo // $PATH

1. PATH environment variable 을 읽는다.

* getenv(char\* ) 함수를 통해 환경변수를 읽는다.

(ex) 환경변수를 얻고 싶은 경우 getenv(“PATH”) 를 통해 얻음

1. 얻은 PATH를 ‘:’ 를 기준으로 tokenizing 하고, 각각의 디렉토리에 사용자가 원하는 파일이 있는지 확인한다.

(ex) PATH=/bin:/usr/bin:/usr/local/bin 일 때,

‘:’ 기준으로 tokenizing 을 하면 경로는

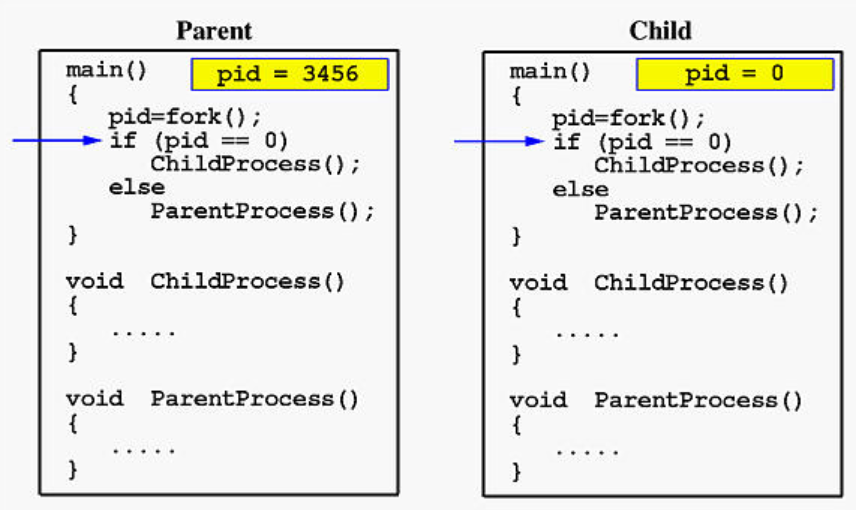
/bin , /usr/bin, /usr/local/bin 과 같고 각 경로에 사용자가 입력한 파일이 있는지 확인한다. 확인하는 경로는 다음과 같다.

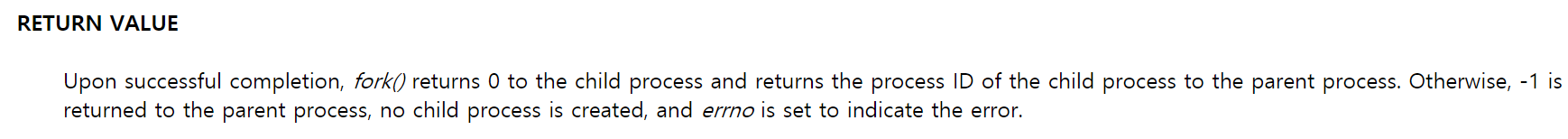
/bin/ls, /usr/bin/ls, /usr/local/bin/ls를 순차적으로 확인한다.

**🡪stat()** 함수를 사용하여 파일이 있는지 없는 지 확인할 수 있다.

1. 파일이 있을 경우 Fork 함

🡪**fork()**함수 사용함



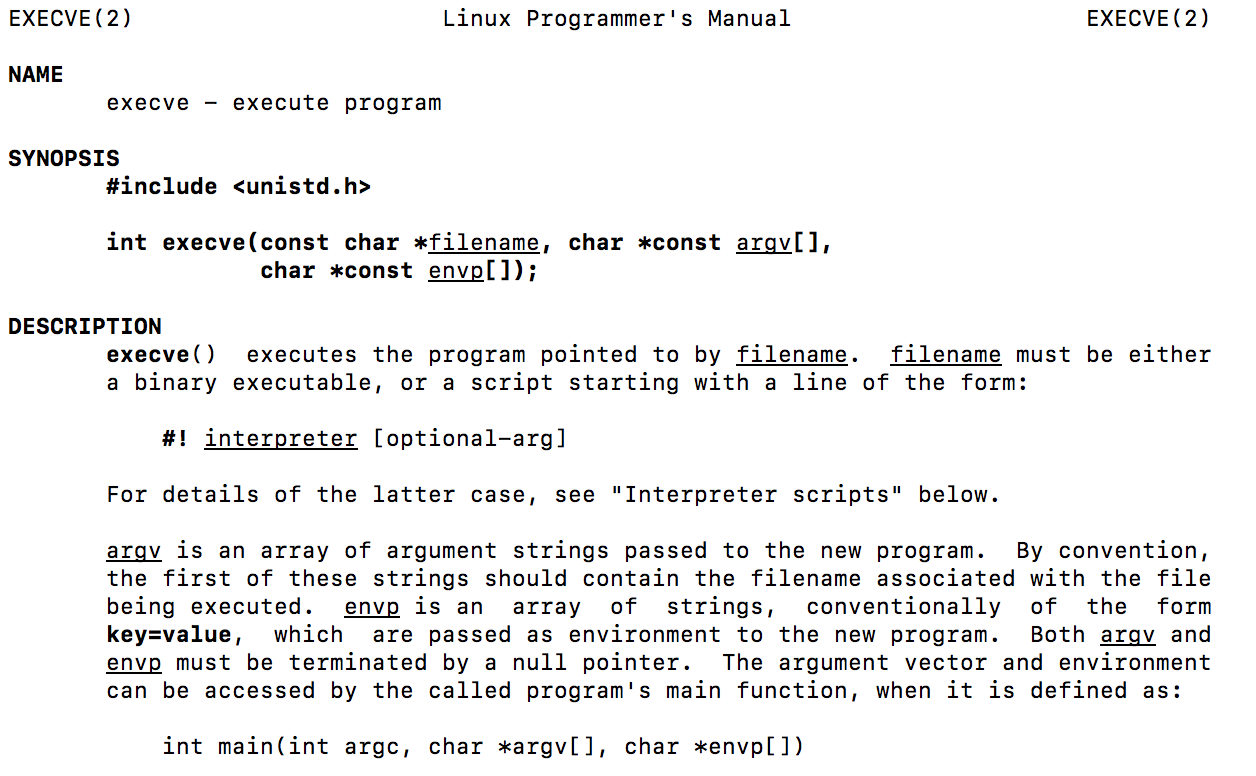


* Parent: Child process가 끝날 때까지 기다린다.
* Child: 파일이 있을 경우 parameter들을 가지고 함수를 실행시켜준다.

🡪 **execve()** system call 사용

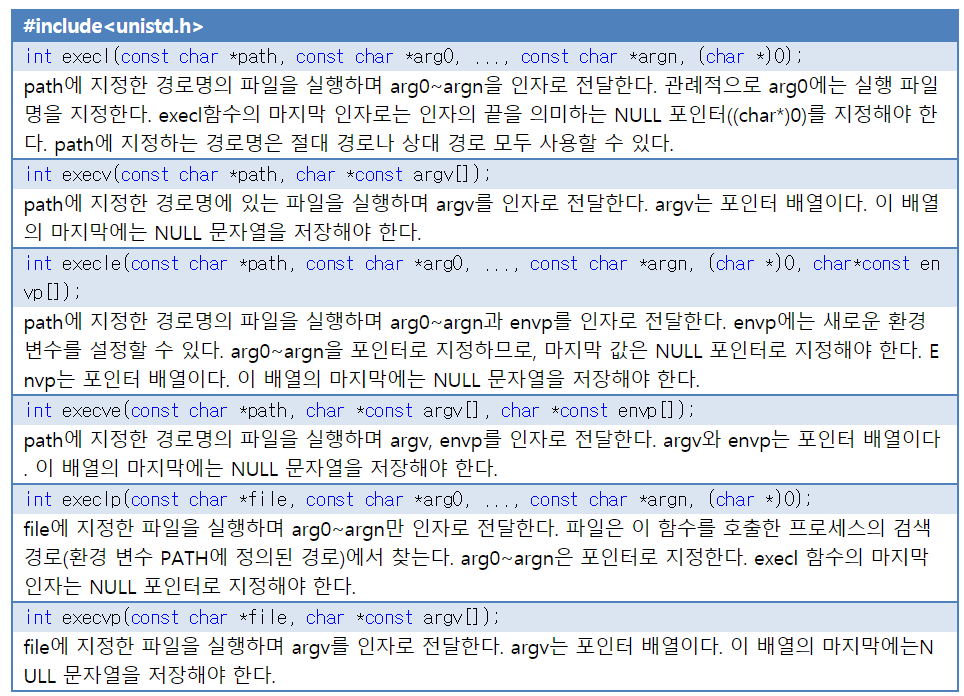
Parameter: 파일이 들어있는 디렉토리 string (ex)/usr/bin/ls

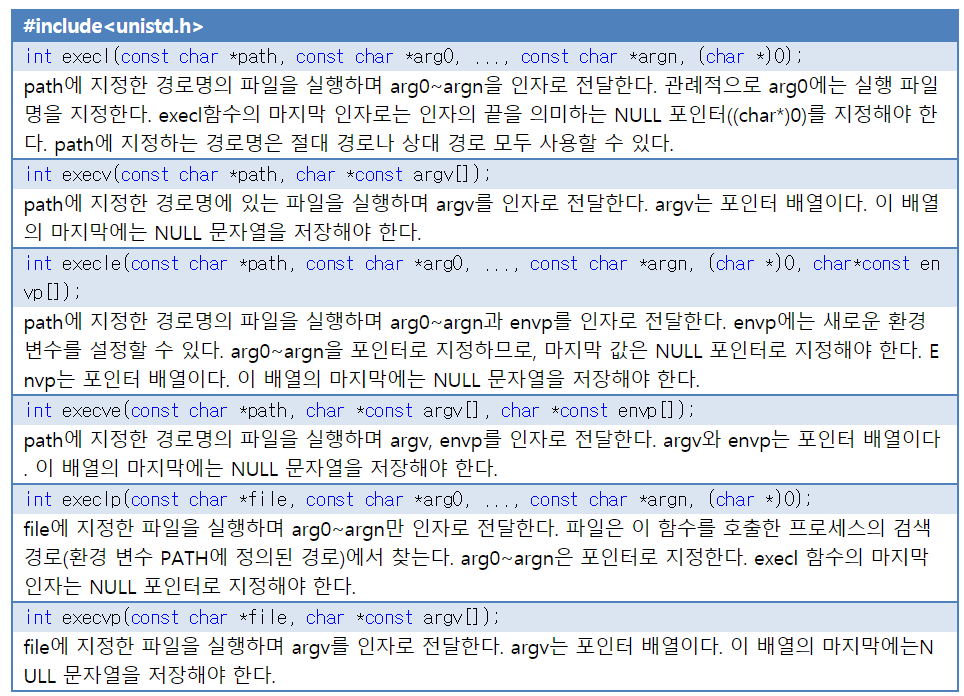
사용자가 입력한 command line pointer



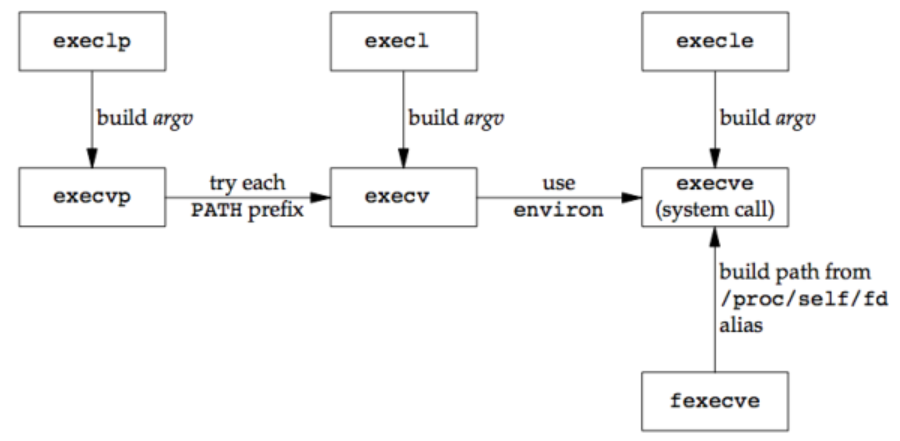
**\*exec 계열 함수들**

exec 함수 군은 path나 file에 지정한 명령이나 실행 파일을 실행한다. 각 함수별로 경로명까지 지정하거나 단순히 실행 파일명만 지정하는 등 차이가 있고 파라미터를 전달하는 형태에도 차이가 있다. exec 함수 는 다음과 같다.



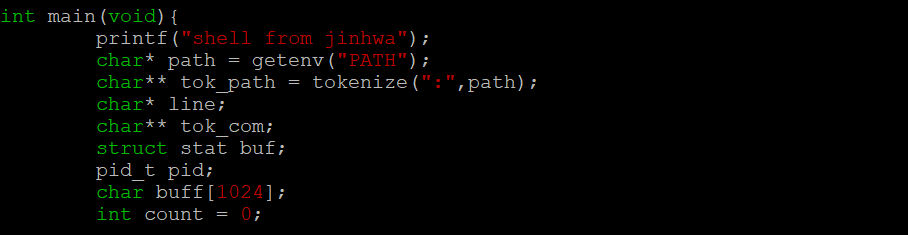


exec 함수군의 관계는 다음과 같으며, execve()가 아닌 **execvp()** 함수를 사용할 경우 PATH를 직접 찾아 주기 때문에 PATH 정보를 넘길 필요가 없다.



1. **소스코드 및 부연설명**
2. **sh\_main.c** : execve() 함수로 구현한 코드
3. main함수

-초기 변수 정의 & 초기화



getenv 함수를 통해 환경변수를 가져와 path 변수에 저장함.

이 후에 path 변수를 tokenizing 하여 사용자가 입력한 command 의 파일이 있는지 확인해야 하므로 미리 tokenizing 해 둠. ‘:’ 기준으로 tokenizing 해야 하므로 ‘:’ 를 파라미터로 넘겨 줌.

-while 문 (shell 기능)

현재 작업 디렉토리를 받아와 보여줘 기존 shell 과 같이 보이도록 함

read\_command() 함수로 사용자로부터 명령어를 받아 옴.

사용자의 명령어는 스페이스로 구분되어 있기 때문에 “ “를 파라미터로 넘겨 줌.

“quit “ 는 파일이 없기 때문에 코드 내부적으로 정의해줘야 함.

tok\_com[0] 에는 명령어 함수 명이 저장 되어 있기 때문에 이를 strncmp함수를 통해 비교 해 줌. 만약 quit 일 경우는 while 문을 빠져 나와야 하므로 break를 걸어 줌.

알고리즘 3)에서 설명한 것 과 같이 tokenizing 된 path 들 각각에 명령어가 있는지 확인 해야 함. 이 때, /bin/ls, /usr/bin/ls 과 같은 형태로 만들기 위해 strcat 함수를 사용하여 경로와 ‘/’ 와 명령어 함수를 이어 줌.

stat 함수를 사용하여 파일이 있는지 확인함.

파일이 있는 경우 0을 리턴 함.



ret 가 0 인 경우, 즉 파일이 있는 경우 fork 함.

pid 가 0인 경우 : 자식 프로세스

자식프로세스는 execve()를 통해 파일을 실행시킴.

pid 가 0이 아닌 경우 (-1 인 에러 상황 제외) : 부모 프로세스

wait() 함수를 통해 자식 프로세스가 끝날 때까지 기다림.

모든 경로를 확인한 후, 파일이 없다면 파일이 없다는 문구를 출력함.

1. tokenize 함수



파라미터로 tokenizing 할 기준 string과 (ex) “:” , ” “), tokenizing 할 문장을 받음.

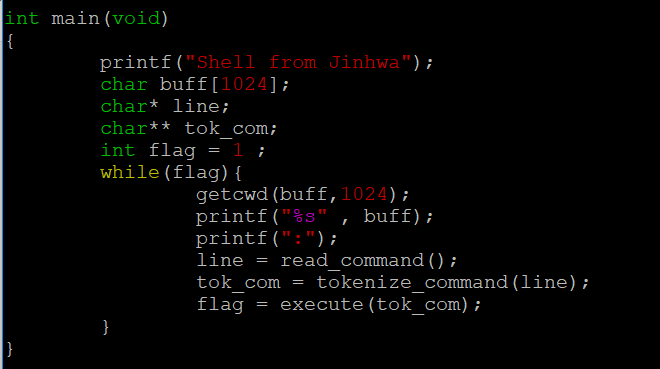
만약 $가 있다면 (ex) $PATH environment path로 대체해줘야 함. 따라서 string에 $가 있는 경우 & tok\_num 이 2 이상인 경우 (Null pointer 오류를 피하기 위해) getenv()로 환경변수를 받아와 대체 해 줌.

기준이 “:” 인 경우는 환경 변수 tokenizing 인 경우임. main에서 환경변수가 몇 번 tokenizing 되었는지에 대한 정보가 필요하므로 그 정보를 넘겨줌.

1. Readcommand 함수

Buffer와, getchar()함수로 사용자로부터 command를 받아옴.

1. **shell.c** : execvp() 함수로 구현한 코드
2. main 함수



현재 작업 디렉토리를 받아와 보여줘 기존 shell 과 같이 보이도록 함.

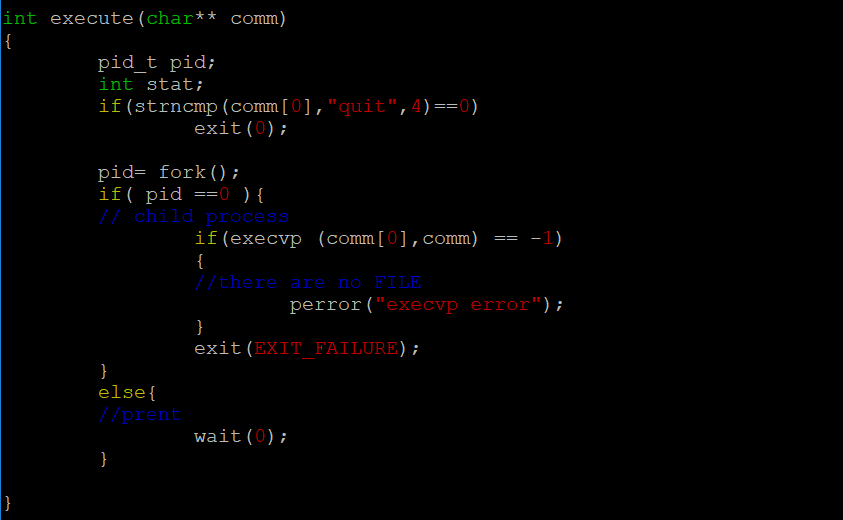
PATH를 tokenizing해서 확인할 필요가 없기 때문에 read\_command()함수를 이용하여 사용자로부터 명령어를 받아오고, 그 명령어만 tokenize\_command() 함수로 tokenizing 함.

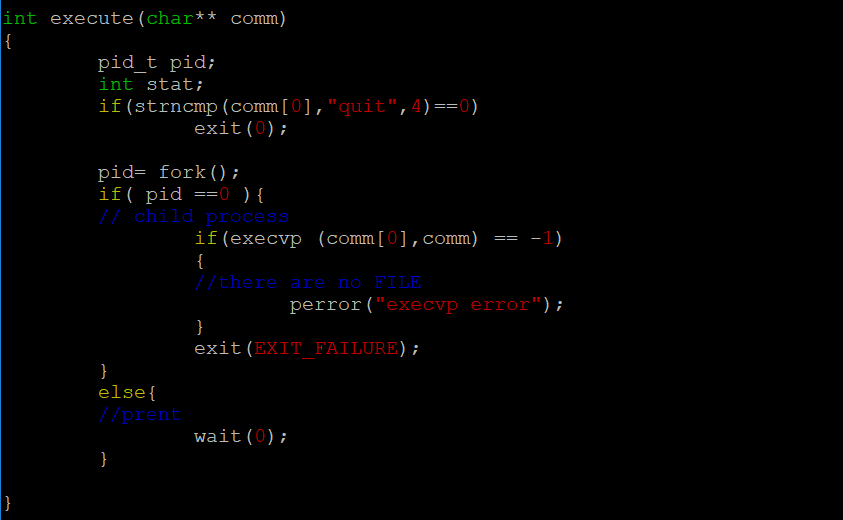
🡪 sh\_main.c에 있는 read\_command, tokenize 함수와 동일함.

execute() 함수를 통해 사용자가 입력한 명령어를 실행 함.

execute함수의 리턴 값을 flag 로 저장하여 while문을 control 함.

1. execute함수





“quit “ 는 파일이 없기 때문에 코드 내부적으로 정의해줘야 함.

strncmp함수를 통해 비교 해 줌. 만약 quit 일 경우는 exit 시켜 줌.

pid 가 0 인 경우: 자식 프로세스

execvp()함수에 파일명(ex) ls,echo 과, command 전체를 가리키는 각 각의

포인터를 넘겨 줌. 만약 파일이 없을 경우, -1을 리턴함. 이 때, error

Message를 넘김.

부모 프로세스

자식 프로세스가 모두 끝날 때까지 기다림.

1. **결과 및 느낀점**
2. 결과
3. sh\_main.c 코드

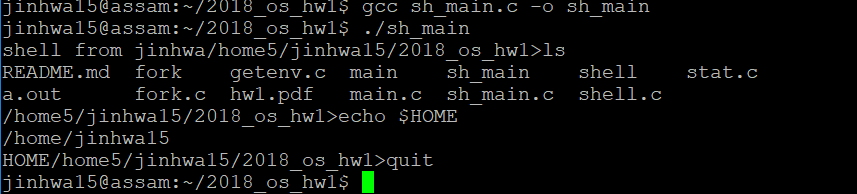
Build 환경

-Compilation: Linux Assam, with GCC

-To compile, please type: **gcc sh\_main.c -o sh\_main**

-To run, please type: **./sh\_main**

결과



1. shell.c 코드

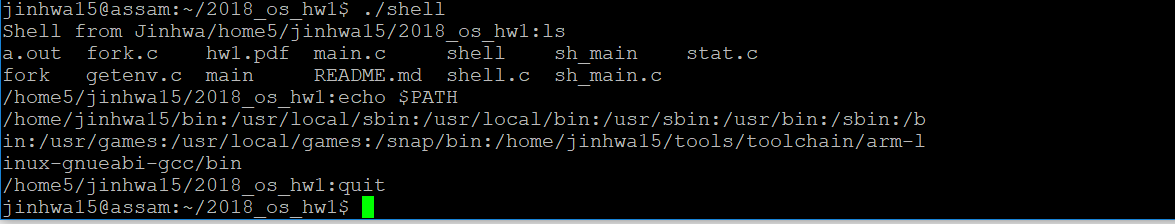
Build 환경

-Compilation: Linux Assam, with GCC

-To compile, please type: **gcc shell.c -o shell**

-To run, please type: **./shell**

결과



1. 느낀점

먼저, 코드를 작성하기 전에 shell 에 대해 정확한 개념을 알지 못해 구글링을 통해 shell의 개념과 원리를 정확하게 이해하려고 노력했으며, 여러가지 기존에 나와있는 코드들을 분석하면서 조금 더 디테일 하게 원리를 이해했습니다.

execvp()함수로 코드를 먼저 작성했는데, PATH에 대한 tokenizing과 다른 작업들이 필요 없어 간단하고 빠르게 코드를 작성할 수 있었으나, execve()로 코드를 작성할 때는 tokenizing 부분을 reference를 보지 않고 짜다 NULL pointer error, memory 할당 에러 등과 같은 여러 에러들을 만나게 되었고 거기에서 조금 시간이 많이 걸렸습니다. 그래도 이 부분을 해결하면서 포인터를 사용하는 방법에 대해 스스로 다시 조금 더 정리할 수 있었습니다. 또한 execve()의 경우에는 $PATH 같은 부분도 다 처리를 해줘야 해서 조금 까다로운 느낌이 들었으나, 여러 기능을 직접 구현할 수 있어 재미있게 코딩할 수 있었습니다.

다음에는 파일이 없는 CD 함수나 makefile도 공부하여 더 구현해 보고 싶습니다.