lec

exit, wait 설명 + **Shell**

scanf = 한문장만

gets = line받을 수 있음

4p = 자식이 process 하는동안 부모가 기다림

끝나면 parent도 wait을 빠져나와서 for문실행하고 또 wait,

shell을 만들어보기\*

bash=shell prog



child proc dbg방법

환경변수 - system의 환경 변수

 $ - 환경변수

PATH의 환경변수

오늘 숙제는 어제 여러분이 만든 myexec을 발전시켜서 myshell을 만드는 겁니다. myshell을 시스템이 제공하는 shell 프로그램(bash)과 유사한 기능을 갖도록 만드는 것이 목표입니다.

쉘은 사용자의 명령어를 처리하는 프로그램을 일반적으로 부르는 명칭입니다. 리눅스에서 사용자가 명령어를 치면 그 명령어를 대신 실행시켜주는 프로그램이 쉘입니다. 윈도우에서는 그래픽 쉘이라고 부르는데 그래픽하게 아이콘들을 띄우고 사용자가 특정 아이콘을 더블클릭하면 그 아이콘에 해당하는 프로그램을 실행시켜줍니다.

원리는 모두 마찬가지로 화면에 프롬프트(리눅스 경우 $ 사인, 윈도우 경우 아이콘들)를 띄우고 사용자가 명령어를 치거나 아이콘을 더블클릭하면 자신을 복제해서 자식을 만든 후 자식이 해당 프로그램으로 exec하게 하고 부모는 기다렸다가 자식이 종료하면 다시 다음 명령어를 기다리는 것입니다.

wait()은 자식이 종료할때까지 대기하라는 의미입니다. wait(&status)는 자식이 exit()으로 종료할 때 exit(1) 등과 같이 어떤 숫자를 전달할 수 있는데 이 숫자를 status에 받으라는 의미입니다.부모는 wait(&status) 후에 status 값을 찍어봄으로써 자식이 마지막으로 어떤 숫자를 전달하고 죽었는지 알 수 있습니다. 이는 자식이 eixt하는 지점이 여러곳일 때 각 지점마다 다른 값을 전달하게 함으로써 자식의 종료지점을 부모가 알 수 있도록 하기 위해서입니다.

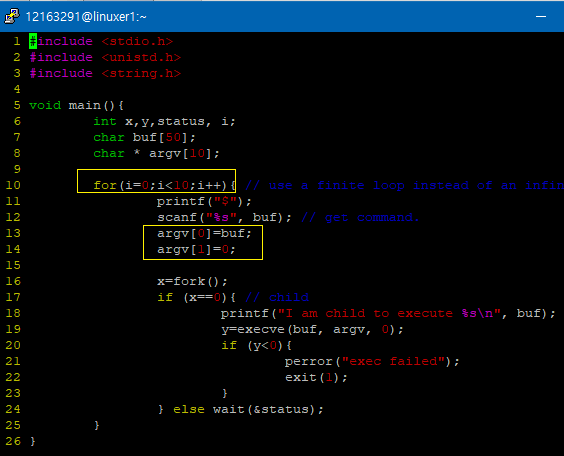
hw

(\* to avoid system down, change infinite loops to finte loops:

for(;;){…} 🡺 for(i=0;i<5;i++){…}

\*)

1) Try the shell example. What is the difference between your shell and the system shell?

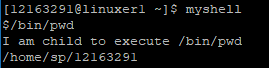


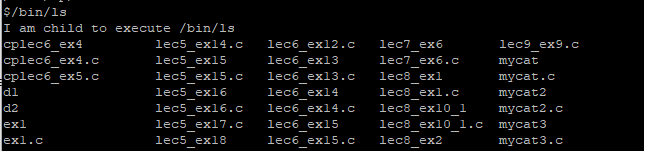
10번 고정으로 반복됩니다.

arg로는 명령어 밖에 받지 못합니다. (옵션 등 커맨드 뒤의 값들 인지 불가)

왜냐하면, argv[0]으로 하나의 buf값만 받기 때문입니다.

만약 더 많은 값들을 받고싶다면, tokenize를 하는 과정이 필요할 것 입니다.



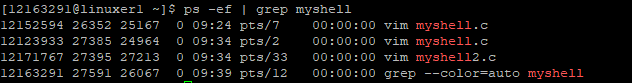




2) Find all ancestor processes of your shell.

ps -ef 와 PPID 정보를 이용하여 ancestor process를 확인할 수 있습니다.

ps -ef | grep 프로세스명  = 특정 프로세스의 pid, ppid 확인 가능

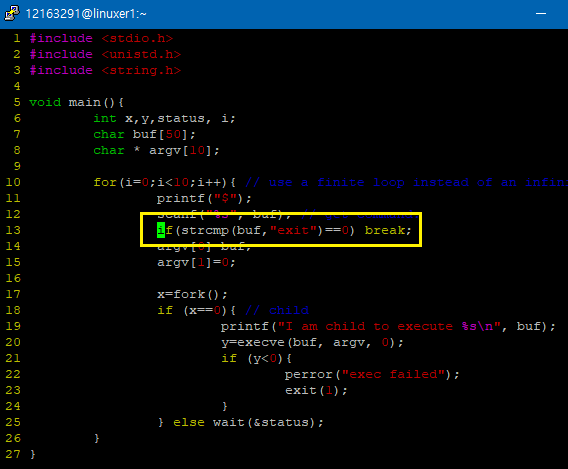


27591, 26067

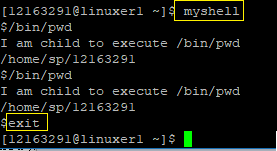




3) (Builtin Command) Change the shell such that it exits when the user types "exit".

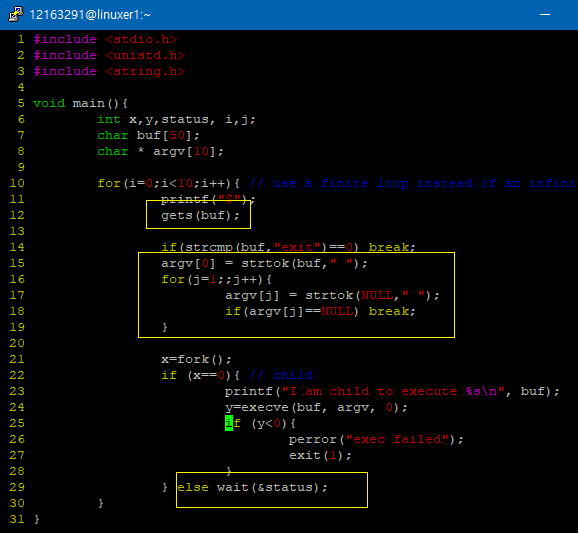


strcmp로, 들어온 입력값과 exit을 비교해줍니다.



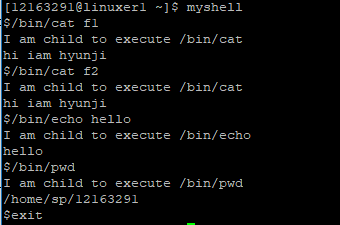
잘 실행되다가, exit이 입력되니 shell이 종료되는 것을 확인할 수 있었습니다.

4) Change the shell such that it can handle a command with arguments. Use gets() or fgets() to read the command.



scanf -> gets 방식으로 바꿔, 명령어를 line단위로 받을 수 있도록 합니다.

그리고, tokenize 를 사용하여, 다양한 argv를 받을 수 있도록 하였습니다.



결과를 보면, child proc가 먼저 실행된 후, parent proc가 실행되는 것을 확인할 수 있습니다.

tokenize를 통해 명령을 받으니, 명령어 뿐만아닌, 두번쨰 arg들도 인식되는 것을 확인할 수 있었습니다.

10번을 모두 실행하지 않기 위해서, (for 10번 loop)

exit을 통해 종료해주었습니다.

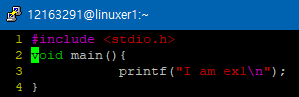
5) (Handling &) **Change the shell such that it can handle '&' at the end of the command**.

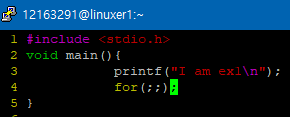
$ ex1

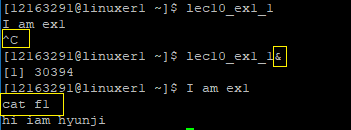
In above, the shell waits until ex1 (the child) is finished. You should make ex1 to have an infinite loop to see the effect.

$ ex1 &

In above, the shell does not wait and immediately prints the next prompt and waits for the next user command.

 에

 무한루프 포함

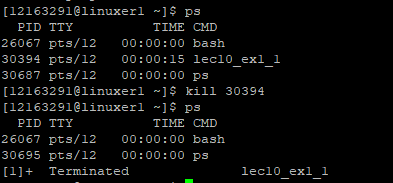


무한루프가 포함되었기에, 프로그램을 실행할 때, 맨뒤에 &가 없다면, 무한루프에서 벗어나지못해

Ctrl C 로 강제종료해야만 그 process를 kill할 수 있습니다.

위의 경우와 달리, 뒤에 &를 첨가해주면, 바로 다음 커맨드를 입력할 수 있게됩니다.

하지만 이는 실제로 process가 마쳐저, 다음 process를 진행할 수 있게 된 것이아닙니다.



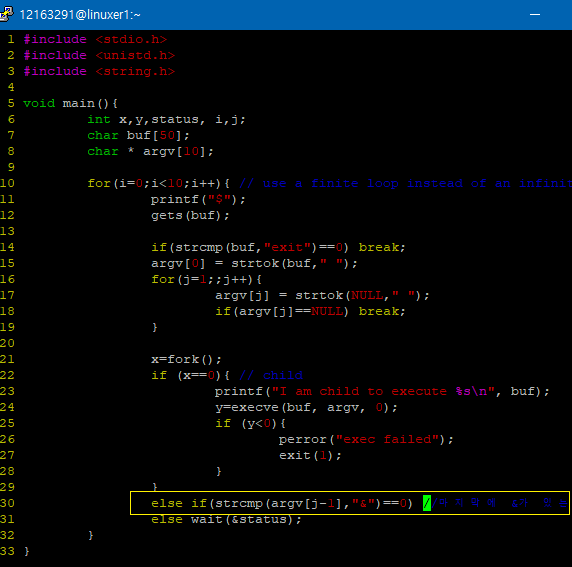
ps로 확인해보면 lec10\_ex1\_1이 아직 실행되고 있는 것을 확인할 수 있습니다.

이를 kill 명령어로 kill해주어야 process가 종료 되게 됩니다.

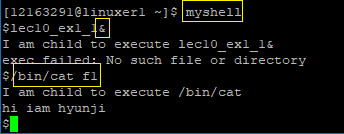
이를 myshell에서 구현하기위한 코드는다음과 같습니다

child process에서 &를 캐치할 수 있어야 합니다.

&가 있는 경우를 구분하기 위해서 - strcmp를 사용해야 합니다. 아마도



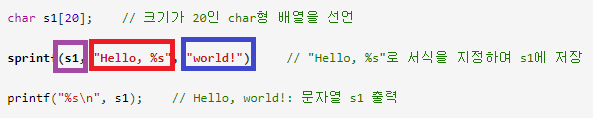
마지막 부분에 &가 있는지 확인합니다. 이에 wait을 넣어주지 않습니다 = child를 기다리지 않고, parent가 먼저 끝나버립니다. 바로 다음 커맨드 창으로 넘어갈 수 있게 됩니다.



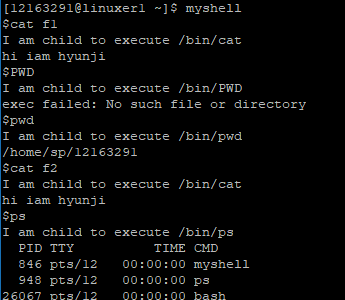
맨 뒤에 &를 붙여주니, 진행될 수 없는 상황임에도 child proc를 기다리지 않고 부모 proc가 끝나버리므로 바로 다음 커맨드를 내릴 수 있는 상황으로 넘어가는 것을 확인할 수 있었습니다.

6) (Handling relative path) Make your shell handle **relative** paths assuming the executable file always exists in /bin directory. When the user enters only the command name (e.g. "ls -l", "cp f1 f2", etc), build a full path such as "/bin/ls", "/bin/cp", etc. and perform exec. Use sprintf() to build the full path.

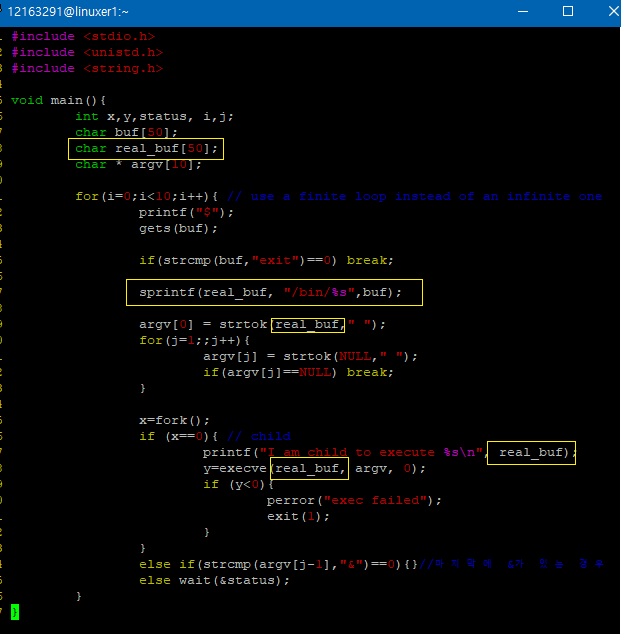
sprintf() =  [printf](https://modoocode.com/35) 에서 화면에 출력하는 대신에, 화면에 출력할 문자열을 인자로 지정한 문자열에 쓴다는 것이다.



빨강 부분+ 파랑부분 되어, 보라부분에 저장



/bin/는 이미 커맨드에 입력되어있으므로 그 뒤의 부분만 입력해도 결과가 출력됩니다.



sprintf() 함수를 사용하기 위해, char배열 변수를 하나더 정의합니다.

sprintf함수를 이용하여, buf로 입력받을 값의 앞에 /bin/를 자동으로 붙여줄 수 있도록 합니다.

/bin/ + buf로 입력받은 명령어 = real\_buf입니다.

따라서, 이부분 이후로, 원래 buf변수로 이루어 졌던 코드들을 real\_buf로 변결해줍니다.

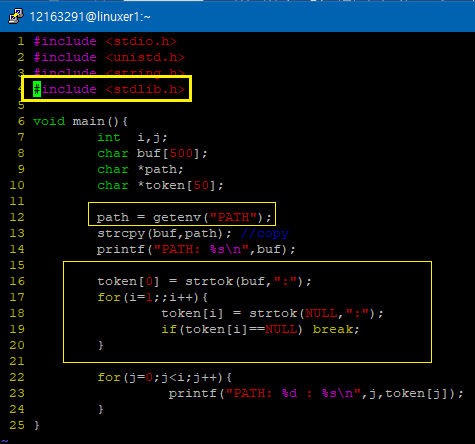
6-1) Use getenv("PATH") to retrieve PATH environment variable and use strtok() to extract each system path. Display each system path line by line.

/usr/lib64/ccache

/usr/local/bin

/usr/bin

.............

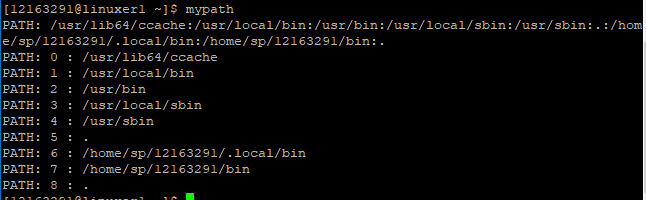


getenv(“PATH”)를 char형 \* path에 저장하고, char[]로 선언된 buf에 복사하였습니다.

buf에 입력된 내용을 출력하여 env - path를 확인하였고, 그 후에 tokenize 함수를 사용하여 ‘:’를 구분자로 사용하여, 잘라주었습니다.

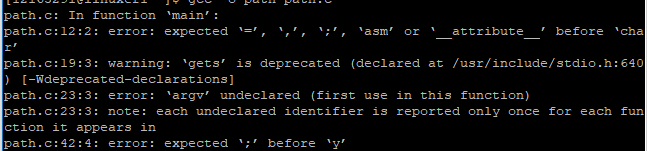
그리고 tokenize 된 총 path를 for를 사용하여 한 줄씩 출력하였습니다.

\*헤더파일 중요



결과는 이왁 같습니다.

7) (Handling relative path) Change the shell such that it can handle relative path for the command in general. The shell will search the PATH environment variable to compute the full path of the command when it is given as a relative path name. Use getenv(“PATH”) to obtain the pointer to the value of the PATH environment variable. Note you need to copy the string of the PATH variable into another char array before you start extracting each path component with strtok() since strtok() destroys the original string.





전 문제와 슷하게getenv 함수를 사용하여 path를 불러옵니다..

기존 string을 파괴하는 것을 막기 위해 그 path를 strcpy 함수로 다른 char[]변수에 복사하였습니다. 그리고 이 복사된 string token 함수를 통해 ‘:’을 기준으로 잘라 tokenpath에 저장하였습니다.

child process에서 sprintf함수를 사용하여 각각의 path의 뒤에 명령어를 이어 붙이고 comamdn에 저장하였습니다.

그 후에 exeve를 실행하였을 때, 반환 되는 값이 0보다 작으면 그 path에는 입력된 명령어가 없는 것으로 간주되며 실패를 알리는 문장이 출력됩니다.

for 10번반복 이므로 다음 path에 대해 같은 작업을 수행합니다.

//여기부턴 할 수 있으면 하세요

8) dup(x) duplicates fd[x] in the first empty entry in the fd table. Run following program and explain the output. Assume f1 has

hello my boy

x=open("f1", O\_RDONLY, 00777);

int y;

y=dup(x);

printf("x:%d y:%d\n", x, y);

char buf[50];

int k=read(x, buf, 5);

buf[k]=0;

printf("buf:%s\n", buf);

k=read(y, buf, 5);

buf[k]=0;

printf("buf:%s\n", buf);

9) (Standard output redirection) Explain the output of the following code.

x=open("f2", O\_WRONLY|O\_CREAT|O\_TRUNC,00777);

printf("x:%d\n", x);

int y;

close(1);

y=dup(x);

printf("x:%d y:%d\n", x, y);

write(1, "hi there", 8);

10) (Standard output redirection) Change the shell such that it can handle standard output redirection.

$ cat f1 > f3

will redirect the output of "cat f1" to file f3.