12201922
이규민

8. Homework

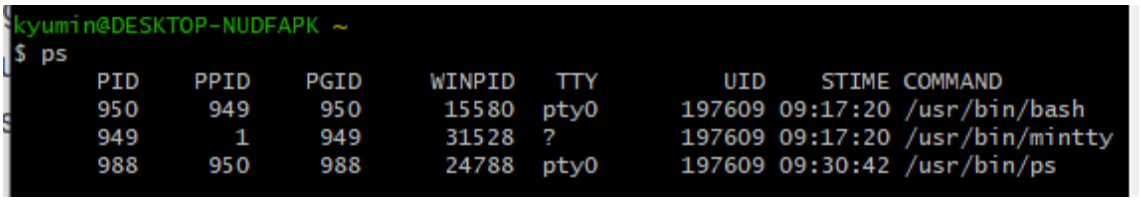
1) Compile and run mysh.c in section 5. What is the difference between mysh and the system shell(the login shell that runs when you log in)? Show at least 5 differences.

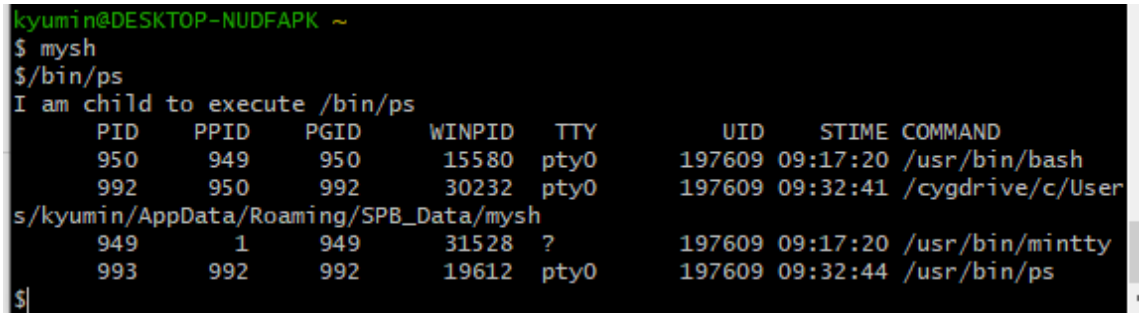
```
kyumin@DESKTOP-NUDFAPK ~
$ mysh
$pwd
I am child to execute pwd
exec failed: No such file or directory
$
```

```
kyumin@DESKTOP-NUDFAPK ~
$ mysh
$/bin/ls
I am child to execute /bin/ls
cdssetup   ex2.c     ex7.c    f3          mycat3.c    myxxd.c
d1         ex2.exe   ex7.exe  f8          mycat3.exe  myxxd.exe
d2         ex3.c     ex8.c    hw4.c       mycp.c      newhw4
digitfile  ex3.exe   ex8.exe  hw4.exe     mycp.exe    newhw4.c
ex0.c      ex4.c     ex9.c    midexam.c   myecho.c    sw2.wav
ex0.exe    ex4.exe   ex9.exe  midexam.exe myecho.exe  swvader03.wav
ex1.c      ex5.c     exam.c   mycat.c     myexec.c    x
ex1.exe    ex5.exe   exdir    mycat.exe   myexec.exe
ex10.c     ex6.c     f1       mycat2.c    mysh.c
ex10.exe   ex6.exe   f2       mycat2.exe  mysh.exe
$
```

1. 명령어만 입력 시 오류가 발생한다.
2. /bin/ls 형태로 입력해야 작동한다.
3. 10번까지만 명령어를 실행할 수 있다.
4. 명령어 실행 시 자식이 실행했다는 메시지가 뜬다.
5. Argument가 두 개 이상 들어갈 수 없다.

2) What is the process name of your login shell? What is the executable file name of your login shell and how can you find it? Who is the parent of your login shell? Explain how the parent of your login shell can create your login shell by showing its C code(roughly). Find all ancestor processes of your login shell.





로그인 쉘의 프로세스 이름은 “pty0”이고, 파일 명은 bash이며, /usr/bin/bash에 있다.
로그인 쉘의 부모의 pid는 949이고, 이름은 보이지 않고 “?”로 나타난다.
Mysh를 실행하고 ps 명령어로 프로세스를 확인해봤다. Mysh 프로세스의 pid는 992다. Mysh의 부모 pid는 950이다.

3) (Builtin Command) Improve mysh such that it exits when the user types "exit". You have to handle “exit” before “fork”. Explain why. This kind of commands that the shell has to handle before fork are called built-in commands.

Fork를 하고 자식 프로세스가 종료되면 바디는 지워지지만, process descriptor는 남아있는 상태가 된다. 그래서 부모가 wait을 해줘야하는 번거로움이 있다.

4) Improve mysh further such that it can handle a command with arguments, such as "/bin/ls -l". Use gets() or fgets() to read the command.
4-1) Improve it further so that it can handle "cd" comand. Also improve it so that it can handle "pwd" command. Note “cd” and “pwd” are other examples of built-in command.
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. Make sure you delete "&" at the end of the command once your detect it.

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.

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
.....

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.

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.