lect9 HW

시스템프로그래밍 1분반 12180626 성시열

1-1) Try below and explain why the output is "I am ex1" when ex2 runs.

```c
#include <stdio.h>

void main()
{
        printf("I am ex1\n");
}
```

```c
#include <stdio.h>

void main()
{
        execve("./ex1",0);
        printf("I am ex2\n");
}
```

```
[12180626@linuxer1 ~]$ vi ex1.c
[12180626@linuxer1 ~]$ cp ex1.c ex2.c
[12180626@linuxer1 ~]$ vi ex2.c
[12180626@linuxer1 ~]$ gcc -o ex1 ex1.c
[12180626@linuxer1 ~]$ gcc -o ex2 ex2.c
[12180626@linuxer1 ~]$ ex2
I am ex1
```

ex2 코드를 보면 execve를 통해 ex2 프로그램을 ex1프로그램으로 변환한다.
두 파일 모두 컴파일을 하고 ex2를 실행하면 영혼은 그대로이나 body가 ex1으로 바뀌었으므로 "I am ex1"이 출력된다.

1-2) Run myexec below. Explain the result.

```c
#include <stdio.h>

void main()
{
        char *k[10];
        k[0] = "/bin/ls";
        k[1] = 0;
        execve(k[0], k, 0);
}
```

```
[12180626@linuxer1 ~]$ vi myexec.c
[12180626@linuxer1 ~]$ gcc -o myexec myexec.c
[12180626@linuxer1 ~]$ myexec
cphw4     ex1.c    ex3      ex5     f1      f9       mycat2    myxxd
cphw4.c   ex11     ex3.c    ex5.c   f1.c    f91      mycat2.c  myxxd.c
d1        ex11.c   ex33     ex6     f1out   f92      mycat3    newhw4.c
d2        ex13     ex33.c   ex6.c   f2      f93?     mycat3.c  outfiles
d3        ex13.    ex4      ex7     f3      fabc     mycp      sw2.wav
echo      ex13.c   ex4.c    ex7.c   f4      hw33     mycp.c    swvader03.wav
ex.8      ex2      ex44     ex8     f5      hw4      myecho    x
ex0       ex2.c    ex44.c   ex8.c   f6      hw4.c    myecho.c  y.c
ex0.c     ex22     ex444    ex9     f7      mycat    myexec
ex1       ex22.c   ex444.c  ex9.c   f8      mycat.c  myexec.c
```

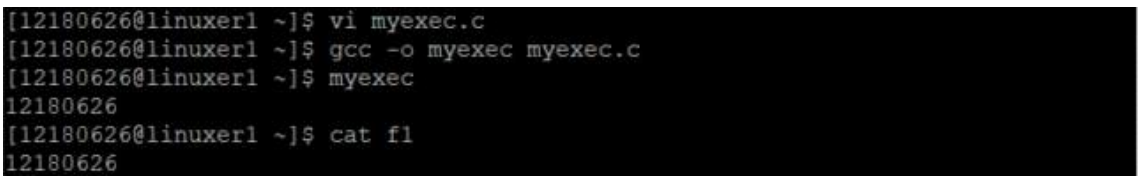
myexec 프로그램의 body를 /bin/ls로 바꾸어준다. execve의 아규먼트를 보면 첫 번째는 변환할 파일이고 두 번째는 그 파일에서 사용되는 아규먼트들을 모아둔 리스트를 넣어준다. 아규먼트를 모두 넣어줬다면 마지막 인덱스에 0을 넣어 리스트가 마무리됨을 알려준다.
컴파일 후 실행하면 ls 명령어와 같은 결과가 나온 것을 알 수 있다.

1-3) Run myexec below and explain the result.

```
#include <stdio.h>

void main()
        char *x[10];
        x[0] = "/bin/cat";
        x[1] = "f1";
        x[2] = 0;
        execve(x[0], x, 0);
}
~
~
```

myexec 프로그램의 body를 /bin/cat으로 바꾸어준다. execve의 아규먼트를 보면 첫 번째는 변환할 파일이고 두 번째는 그 파일에서 사용되는 아규먼트들을 모아둔 리스트를 넣어준다. x[0]에는 cat 명령어 위치를 넣어주고, x[1]에는 cat 명령어를 사용할 때 필요한 아규먼트를 넣어준다.



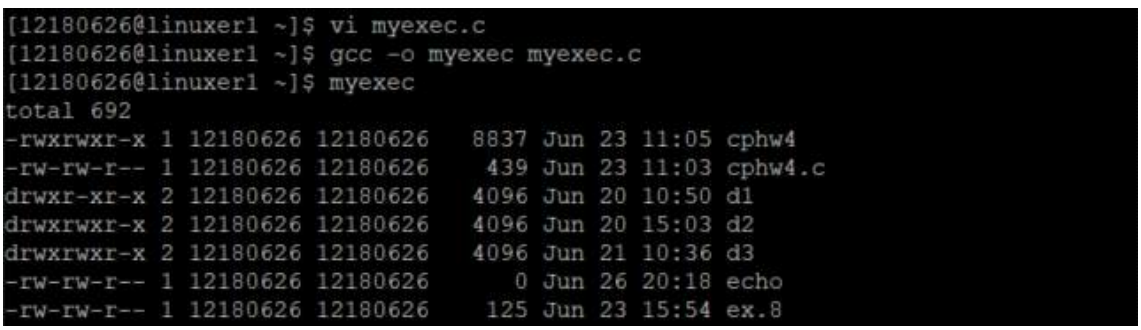
```
[12180626@linuxer1 ~]$ vi myexec.c
[12180626@linuxer1 ~]$ gcc -o myexec myexec.c
[12180626@linuxer1 ~]$ myexec
12180626
[12180626@linuxer1 ~]$ cat f1
12180626
```

컴파일 후 실행하면 cat f1과 같은 결과가 나온 것을 알 수 있다.
The above program will exec to "/bin/cat f1" which will print the contents of f1.

2) Change myexec such that it execs to "/bin/ls -l".

```
#include <stdio.h>

void main()
        char *x[10];
        x[0] = "/bin/ls";
        x[1] = "-l";
        x[2] = 0;
        execve(x[0], x, 0);
}
~
~
```



```
[12180626@linuxer1 ~]$ vi myexec.c
[12180626@linuxer1 ~]$ gcc -o myexec myexec.c
[12180626@linuxer1 ~]$ myexec
total 692
-rwxrwxr-x 1 12180626 12180626   8837 Jun 23 11:05 cphw4
-rw-rw-r-- 1 12180626 12180626    439 Jun 23 11:03 cphw4.c
drwxr-xr-x 2 12180626 12180626   4096 Jun 20 10:50 d1
drwxrwxr-x 2 12180626 12180626   4096 Jun 20 15:03 d2
drwxrwxr-x 2 12180626 12180626   4096 Jun 21 10:36 d3
-rw-rw-r-- 1 12180626 12180626      0 Jun 26 20:18 echo
-rw-rw-r-- 1 12180626 12180626    125 Jun 23 15:54 ex.8
```

body를 변경하고, 필요한 아규먼트를 포함하여 리스트를 전달하는데, 이 때 옵션까지 포함한 리스트를 만들어 값을 execve의 아규먼트에 넣어준다.
출력 결과 ls –l과 같은 결과값이 출력됨을 알 수 있다.

3) Change myexec such that it execs to "/bin/cp f1 f2".

```
#include <stdio.h>

void main()
        char *x[10];
        x[0] = "/bin/cp";
        x[1] = "f1";
        x[2] = "f2";
        x[3] = 0;
        execve(x[0], x, 0);
}
~
~
```

```
[12180626@linuxer1 ~]$ vi myexec.c
[12180626@linuxer1 ~]$ gcc -o myexec myexec.c
[12180626@linuxer1 ~]$ myexec
[12180626@linuxer1 ~]$ cat f1
12180626
[12180626@linuxer1 ~]$ cat f2
12180626
```

cp 함수는 아규먼트가 두 개 필요하므로 x[1]과 x[2]에 값을 넣어주어 x[1] 파일을 x[2]에 복사하는 함수로 myexec를 바꿔준다. 컴파일 후 실행하고, cat f1 cat f2를 통해 값이 복사됨을 확인하였다.

4) Change myexec such that it runs "/bin/ls -l" and prints "job done" after "/bin/ls -l" is finished.

```
#include <stdio.h>

void main()
        char *x[10];

        int y = fork();

        if(y==0){
                x[0] = "/bin/ls";
                x[1] = "-l";
                x[2] = 0;
                execve(x[0], x, 0);
        }

        else{
                wait();
                printf("job done\n");
        }
}
~
~
```

2)처럼 ls –l을 실행하고 "job done"을 출력한다. 이를 위해 fork()와 wait()을 사용한다. fork()를 통해 값을 복제하면 부모와 자식으로 나뉘는데, 부모의 return 값은 0아 아니고(자식의 pid) 자식의 return 값은 0이다. 이를 활용하여 조건문을 구성한다. 자식이 실행될 때, myexec 프로그램이 "ls –l"로 변환되게 한다. 부모가 실행될 때, "job done"을 출력한다.
단, scheduler의 복잡한 계산에 의해 자식 프로세스가 먼저 실행될지 부모 프로세스가 실행될지 결정된다. wait()은 자식 프로세스가 끝날 때까지 기다려주는 함수이므로 부모가 먼저 실행되려할 때 wait()을 통해 자식 프로세스가 먼저 실행되고 이후 부모 프로세스가 실행되게 한다. 자식 프로세스를 통해 프로그램 body를 바꿔주고, 자식 프로세스가 종료되면 이후의 부모 프로세스의 printf()문을 실행한다.

```
[12180626@linuxer1 ~]$ vi myexec.c
[12180626@linuxer1 ~]$ gcc -o myexec myexec.c
[12180626@linuxer1 ~]$ myexec
total 692
-rwxrwxr-x 1 12180626 12180626   8837 Jun 23 11:05 cphw4
-rw-rw-r-- 1 12180626 12180626    439 Jun 23 11:03 cphw4.c
drwxr-xr-x 2 12180626 12180626   4096 Jun 20 10:50 d1
drwxrwxr-x 2 12180626 12180626   4096 Jun 20 15:03 d2
-rwxrwxr-x 1 12180626 12180626      0 Jun 28 08:53 outfiles
-rw-rw-r-- 1 12180626 12180626  30268 Jun 24 14:57 sw2.wav
-rw-rw-r-- 1 12180626 12180626  30268 Jun 24 09:16 swvader03.wav
-rw-rw-r-- 1 12180626 12180626 126760 Jun 24 14:44 x
-rw-rw-r-- 1 12180626 12180626     52 Jun 21 10:30 y.c
job done
```

컴파일 후 실행하면 ls –l이 실행되고 job done이 출력된 것을 알 수 있다.

5) Change myexec such that it reads a command and execs to the given command. Your code should be able to exec any command. Read the command line with gets() or fgets() and use strtok() to extract all arguments and perform exec to run the command. Remember you have to remove the last character('₩n') if you use fgets.

```
#include <stdio.h>
#include <string.h>

void main(){
        int j;
        char buf[256];
        char *token[256];
        printf("command > ");
        gets(buf);
        int i = 0;
        token[i] = strtok(buf, " ");
        for(i = 1; ; i++){
                token[i] = strtok(NULL, " ");
                if (token[i] == NULL) break;
        }
        token[i] = 0;
        execve(token[0], token, 0);
}
```

명령어를 fgets()로 입력받아서 실행하는 코드이다.

fgets로 명령어를 입력받은 뒤 strtok를 통해 공백단위로 값을 나누고
해당 값을 리스트에 하나씩 넣은 뒤, execve의 argument로 보내준다.

```
[12180626@linuxer1 ~]$ vi myexec.c
[12180626@linuxer1 ~]$ gcc -o myexec myexec.c
myexec.c: In function 'main':
myexec.c:9:2: warning: 'gets' is deprecated (declared at /usr/include/stdio.h:64
0) [-Wdeprecated-declarations]
[12180626@linuxer1 ~]$ myexec
command > /bin/cat f1
12180626
[12180626@linuxer1 ~]$ myexec
command > /bin/cp f1 f2
[12180626@linuxer1 ~]$ cat f1
12180626
[12180626@linuxer1 ~]$ cat f2
12180626
```

해당 코드를 컴파일하고 실행한 뒤 명령어를 입력해본 결과 잘 실행됨을 확인하였다.

6) Same as 5), but myexec will repeat the process 5 times running different linux commands each time.

```c
#include <stdio.h>
#include <string.h>

void main(){
        int y;
        int j;
        for(j=0; j<5; j++){
                y = fork();
                if(y==0){
                        char buf[256];
                        char *token[256];
                        printf("command > ");
                        gets(buf);
                        int i = 0;
                        token[i] = strtok(buf, " ");
                        for(i = 1; ; i++){
                                token[i] = strtok(NULL, " ");
                                if (token[i] == NULL) break;
                        }
                        token[i] = 0;
                        execve(token[0], token, 0);
                }
                else
                        wait();
        }
}
```

5)번 코드의 실행문을 반복문을 통해 5번 실행할 수 있게 하였다. 단순히 5번 코드를 for문 안에 넣어줬을 때는 한번 실행하면 프로그램이 종료되는 것을 확인하였다.
반복을 위해 y=fork();를 활용하여 값을 복제하고, 부모, 자식 프로세스를 실행시킨다. wait()을 통해 자식 프로세스가 먼저 실행되게 한다. 자식 프로세스가 실행될 때 y에 리턴되는 값이 0이므로 5)번 코드에서 했던 과정을 진행한다.

```
[12180626@linuxer1 ~]$ myexec
command > /bin/cat f1
12180626
command > /bin/cp f1 f2
command > /bin/cat f2
12180626
command > /bin/ls
cphw4     ex1.c    ex3      ex5     f1      f9       mycat2    myxxd
cphw4.c   ex11     ex3.c    ex5.c   f1.c    f91      mycat2.c  myxxd.c
d1        ex11.c   ex33     ex6     f1out   f92      mycat3    newhw4.c
d2        ex13     ex33.c   ex6.c   f2      f93?     mycat3.c  outfiles
d3        ex13.    ex4      ex7     f3      fabc     mycp      sw2.wav
echo      ex13.c   ex4.c    ex7.c   f4      hw33     mycp.c    swvader03.wav
ex.8      ex2      ex44     ex8     f5      hw4      myecho    x
ex0       ex2.c    ex44.c   ex8.c   f6      hw4.c    myecho.c  y.c
ex0.c     ex22     ex444    ex9     f7      mycat    myexec
ex1       ex22.c   ex444.c  ex9.c   f8      mycat.c  myexec.c
command > /bin/ls -l
total 692
-rwxrwxr-x 1 12180626 12180626   8837 Jun 23 11:05 cphw4
-rw-rw-r-- 1 12180626 12180626    439 Jun 23 11:03 cphw4.c
drwxr-xr-x 2 12180626 12180626   4096 Jun 20 10:50 d1
```

컴파일하여 실행한 결과 5번의 명령어 실행이 잘 이루어졌음을 확인할 수 있다.

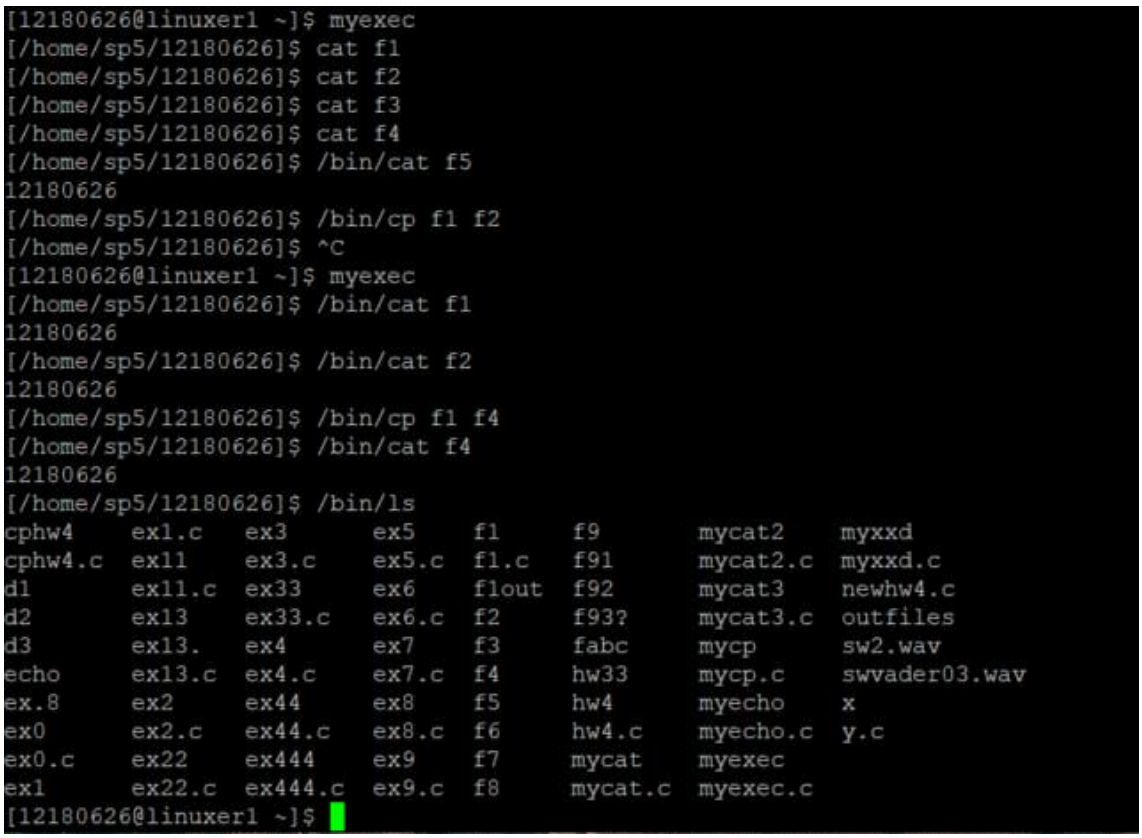
7) Same as 6), but change the prompt to the current location and '$' as follows. You may need "getcwd".

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>

void main(){
        int y;
        int j;
        char dir_name[256];
        getcwd(dir_name, 256);

        for(j=0; j<5; j++){
                y = fork();
                if(y == 0){
                        char buf[256];
                        char *token[256];
                        printf("[%s]$ ", dir_name);
                        gets(buf);
                        int i = 0;
                        token[i] = strtok(buf, " ");
                        for(i = 1; ; i++){
                                token[i] = strtok(NULL, " ");
                                if (token[i] == NULL) break;
                        }
                        token[i] = 0;
                        execve(token[0], token, 0);
                }
                else
                        wait();
        }
}
```

getcwd(vuf, size)는 현재 작업중인 directort 이름을 size만큼 buf에 복사하는 함수이다. 이를 활용하여 printf("comman >") 대신 터미널에서 명령어를 사용하는 것처럼 [현재위치]$를 printf()를 한다.



컴파일 후 실행하면 [현재위치]$를 통해 터미널에서 명령어를 실행시키는 것과 비슷한 것을 확인할 수 있다. 가장 아랫 줄을 통해 정확하게는 다른 표현인 것을 알 수 있다.