

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.

```
kyumin@DESKTOP-NUDFAPK ~  
$ ps  
  PID   PPID   PGID   WINPID   TTY      UID    STIME  COMMAND  
  950    949    950    15580    pty0     197609 09:17:20 /usr/bin/bash  
  949      1    949    31528    ?        197609 09:17:20 /usr/bin/mintty  
  988    950    988    24788    pty0     197609 09:30:42 /usr/bin/ps  
  
kyumin@DESKTOP-NUDFAPK ~  
$ mysh  
$/bin/ps  
I am child to execute /bin/ps  
  PID   PPID   PGID   WINPID   TTY      UID    STIME  COMMAND  
  950    949    950    15580    pty0     197609 09:17:20 /usr/bin/bash  
  992    950    992    30232    pty0     197609 09:32:41 /cygdrive/c/User  
s/kyumin/AppData/Roaming/SPB_Data/mysh  
  949      1    949    31528    ?        197609 09:17:20 /usr/bin/mintty  
  993    992    992    19612    pty0     197609 09:32:44 /usr/bin/ps  
$
```

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.

```
void main(){
    int x,y,status, i;
    char buf[50];
    char * argv[10];

    for(i=0;i<10;i++){ // use a finite loop instead of an infinite loop
        printf("$");
        scanf("%s", buf); // get command.
        if(strcmp(buf, "exit") == 0) exit(0);

        argv[0]=buf;
        argv[1]=0;

        x=fork();
        if (x==0){ // child
            printf("I am child to execute %s\n", buf);
            y=execve(buf, argv, 0);
            if (y<0){
                perror("exec failed");
                exit(1);
            }
        }
        else wait(&status);
    }
}
```

Fork를 하고 자식 프로세스가 exit을 하면 자식만 바다가 삭제되고, 부모 프로세스는 삭제되지 않는다. 그래서 부모의 코드에서 exit을 해줘야한다. 그래서 위의 코드처럼 fork를 하기 전 부모 프로세스에서 exit이 입력된다면 exit을하도록 코드를 작성하였다.

```
kyumin@DESKTOP-NUDFAPK ~
$ mysh
$/bin/pwd
I am child to execute /bin/pwd
/cygdrive/c/Users/kyumin/AppData/Roaming/SPB_Data
$exit
```

Exit을 입력한다면 정상적으로 프로그램이 종료된다.

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.

```
void main(){
    int x,y,status, i;
    char buf[50];
    char * argv[10];

    for(i=0;i<10;i++){ // use a finite loop instead of an infin
        printf("$");
        //scanf("%s", buf); // get command.
        fgets(buf, 50, stdin);
        buf[strlen(buf) - 1] = '\0';

        argv[0] = strtok(buf, " ");
        if(strcmp(argv[0], "exit") == 0) exit(0);

        for(int i = 1;i< strlen(buf); i++)
        {
            argv[i] = strtok(NULL, " ");
            if(argv[i] == NULL) break;
        }

        x=fork();
        if (x==0){ // child
            printf("I am child to execute %s\n", buf);
            y=execve(argv[0], argv, 0);
            if (y < 0){
                perror("exec failed");
                exit(1);
            }
        }
        else wait(&status);
    }
}
```

우선 여러 argument 를 받기 위해선 여러 단어를 입력 받을 수 있어야한다. Scanf 의 경우 단어 하나만 입력받기 때문에 띄어쓰기가 입력되면 앞 단어만 읽는다.

그래서 fgets 를 사용했고, 이 함수는 입력 시 Enter 를 줄바꿈으로 읽기 때문에 마지막 글자를 'W0'으로 바꿨다. 그리고 strtok 함수를 사용해서 여러 단어가 입력되면 토큰을 나누고, 그 토큰을 argv 배열에 따로 저장하였다. 그리고 execve 함수의 argument 로 argv 를 사용했다.

```

kyumin@DESKTOP-NUDFAPK ~
$ mysh
$/bin/ls -l
I am child to execute /bin/ls
total 1675
drwx-----+ 1 kyumin 2월 0  Mar 15  2021 cdssetup
drwxr-xr-x+ 1 kyumin 2월 0  Apr 14 15:45 d1
drwxr-xr-x+ 1 kyumin 2월 0  Mar  6 12:47 d2
-rwxr-xr-x+ 1 kyumin 2월 1  Apr 18 23:21 digitfile
-rw-r--r--+ 1 kyumin 2월 249 Apr 15 09:10 ex0.c
-rwxr-xr-x+ 1 kyumin 2월 66203 Apr 15 09:11 ex0.exe
-rw-r--r--+ 1 kyumin 2월 60  Apr 29 09:04 ex1.c
-rwxr-xr-x+ 1 kyumin 2월 66542 Apr 29 09:02 ex1.exe
-rw-r--r--+ 1 kyumin 2월 566  Apr 18 23:21 ex10.c
-rwxr-xr-x+ 1 kyumin 2월 68716 Apr 18 23:21 ex10.exe

```

수정된 프로그램으로 실행해보면 단순히 ls 명령어가 실행된게 아니라 -l 옵션으로 실행이 된 것을 볼 수 있다.

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.

```
void main(){
    int x,y,status, i;
    char buf[50];
    char * argv[10];

    for(;;)
    {
        printf("$");
        fgets(buf, 50, stdin);
        buf[strlen(buf) - 1] = '\0';

        argv[0] = strtok(buf, " ");

        //exit 입력된 경우
        if(strcmp(argv[0], "exit") == 0) exit(0);

        //cd 입력된 경우
        else if(strcmp(argv[0], "cd") == 0)
        {
            argv[1] = strtok(NULL, " ");
            chdir(argv[1]);
        }

        //pwd 입력된 경우
        else if(strcmp(argv[0], "pwd") == 0)
        {
            char pwd[256];
            getcwd(pwd, 256);
            printf("%s\n", pwd);
        }
    }
}
```

```

//내장 명령어가 아닌 경우
else
{
    for(int i = 1; i < strlen(buf); i++)
    {
        argv[i] = strtok(NULL, " ");
        if(argv[i] == NULL) break;
    }

    x=fork();
    if (x==0){ // child
        printf("I am child to execute %s\n", buf);
        y=execve(argv[0], argv, 0);
        if (y < 0){
            perror("exec failed");
            exit(1);
        }
    }
    else wait(&status);
}
}
}
~

```

Else if를 통해 cd가 입력된 경우와 pwd가 입력된 경우를 나눴다.

우선 cd가 입력된 경우 다음 argument를 chdir에 사용해서 현 위치를 다른 디렉토리로 옮길 수 있도록 했다.

Pwd가 입력된 경우 getcwd를 사용해서 현재 위치를 출력하도록 만들었다.

```

kyumin@DESKTOP-NUDFAPK ~
$ mysh
$pwd
/cygdrive/c/Users/kyumin/AppData/Roaming/SPB_Data
$cd d1
$pwd
/cygdrive/c/Users/kyumin/AppData/Roaming/SPB_Data/d1
$

```

정상적으로 pwd가 작동하고, cd가 작동하는 것을 볼 수 있다.

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 you detect it.

```
void main(){
    int x,y,status, i;
    char buf[50];
    char * argv[10];

    for(;;)
    {
        printf("$");
        fgets(buf, 50, stdin);
        buf[strlen(buf) - 1] = '\0';

        argv[0] = strtok(buf, " ");

        //exit 입력된 경우
        if(strcmp(argv[0], "exit") == 0) exit(0);

        //cd 입력된 경우
        else if(strcmp(argv[0], "cd") == 0)
        {
            argv[1] = strtok(NULL, " ");
            chdir(argv[1]);
        }

        //pwd 입력된 경우
        else if(strcmp(argv[0], "pwd") == 0)
        {
            char pwd[256];
            getcwd(pwd, 256);
            printf("%s\n", pwd);
        }
    }
}
```



```

//내장 명령어가 아닌 경우
else
{
    int flag = 0;
    for(int i = 1; i < strlen(buf); i++)
    {
        argv[i] = strtok(NULL, " ");
        if(argv[i] == NULL) break;

        //& 문자가 입력된 경우
        if(strcmp(argv[i], "&") == 0){
            flag = 1; //플래그 세우기
            argv[i] = 0; //execve에 들어가지 않게
        }
    }

    x=fork();
    if (x==0){ // child
        printf("I am child to execute %s\n", buf);
        y=execve(argv[0], argv, 0);
        if (y < 0){
            perror("exec failed");
            exit(1);
        }
    }
    else if(flag == 0) wait(&status);
}
}
}

```

strtok으로 단어를 나눌 때 ‘&’이 있다면 flag를 1로 올린다. 그리고 자식 프로세스에서 execve 인자에 ‘&’이 들어가지 않도록 0으로 바꿔준다. 그리고 코드는 실행된다.

부모 코드에서 flag가 0일 때만 wait을 실행한다.

즉 &이 없다면 for문이 다시 실행되므로 명령어 입력이 가능해진다.

그러나 이 코드에는 문제점이 있다. &을 입력 시 자식 프로세스는 종료되더라도 바디만 삭제되고 process descriptor가 살아있는 상태로 부모의 for문이 돌게된다. 그래서 자식의 process descriptor가 하나씩 매번 살아있는상태가 되는데 이를 해결하기위해선 wait이 아닌 waitpid 함수를 사용하면 된다. Waitpid는 자식의 프로세스를 종료하므로 이 문제를 해결할 수 있다.

```
kyumin@DESKTOP-NUDFAPK ~  
$ mysh  
$ex1  
I am child to execute ex1  
|
```

```
kyumin@DESKTOP-NUDFAPK ~  
$ mysh  
$ex1 &  
$I am child to execute ex1  
/bin/pwd  
I am child to execute /bin/pwd  
/cygdrive/c/Users/kyumin/AppData/Roaming/SPB_Data  
$
```

Ex1은 무한 루프가 도는 프로그램이다. Ex1은 현 위치의 실행 파일이므로 절대 주소를 입력할 필요가 없다.

Ex1 프로그램만 실행하면 무한 루프가 돌아서 다른 명령어 실행이 불가능하다. 그러나 ex1 &을 입력하면 다른 명령어 실행이 가능하고 위 사진에서 정상적으로 실행된 것을 볼 수 있다. 그러나 가끔 스케줄러가 execve가 아닌 부모 프로세스를 먼저 실행해서 &이 먼저 출력되고 execve의 결과가 나중에 나오기도 한다.

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.

```
void main(){
    int x,y,status, i;
    char buf[50];
    char * argv[10];

    for(;;)
    {
        printf("$");
        fgets(buf, 50, stdin);
        buf[strlen(buf) - 1] = '\0';

        argv[0] = strtok(buf, " ");

        //exit 입력된 경우
        if(strcmp(argv[0], "exit") == 0) exit(0);

        //cd 입력된 경우
        else if(strcmp(argv[0], "cd") == 0)
        {
            argv[1] = strtok(NULL, " ");
            chdir(argv[1]);
        }

        //pwd 입력된 경우
        else if(strcmp(argv[0], "pwd") == 0)
        {
            char pwd[256];
            getcwd(pwd, 256);
            printf("%s\n", pwd);
        }
    }
}
```

```

//내장 명령어가 아닌 경우
else
{
    int flag = 0;
    for(int i = 1;; i++)
    {
        argv[i] = strtok(NULL, " ");
        if(argv[i] == NULL) {
            argv[i] = 0;
            break;
        }

        //문자가 입력된 경우
        if(strcmp(argv[i], "&") == 0){
            flag = 1; //플래그 세우기
            argv[i] = 0; //execve에 0인 인자가 필요
        }
    }

    x=fork();
    if (x==0){ // child
        printf("I am child to execute %s\n", buf);

        //상대 경로 입력
        char temp[50];
        sprintf(temp, "/bin/%s", argv[0]);

        y=execve(temp, argv, 0);
        if (y < 0){
            perror("exec failed");
            exit(1);
        }
    }
    else if(flag == 0) wait(&status);
}
}
}

```

절대 경로가 아닌 상대 경로만 입력 시 작동하게 만들기 위해서 execve의 첫 번째 인자의 문자열을 수정하면 된다. If(x==0) 내부에서 char temp[50]을 선언해줬고, sprintf 함수를 사용해서 “/bin/” 문자열과 명령어가 저장된 argv[0]를 이어서 temp에 저장했다. 그리고 이 temp를 execve의 첫 번째 인자로 사용했다.

Sprintf 사용 시 주의할 점이 있다. 바로 크기를 보장해주지 못한다는 점이다. Argv[0]에 “ls”가 저장되어있고 argv[1]에 “-l”이 저장되어있을 때 sprintf를 이용하여 긴 문장이 저장된다면 argv[1]에 데이터를 침범한다. 그래서 argv[1]의 내용이 다른 내용으로 바뀐다. 그래서 이 문제를 해결하기 위해서 위와 같이 새로운 캐릭터 배열을 선언하여 그곳에 sprintf를 사용해줬다.

```

kyumin@DESKTOP-NUDFAPK ~
$ mysh
$ls -l
I am child to execute ls
total 1678
drwx-----+ 1 kyumin 없음      0 Mar 15  2021 cdssetup
drwxr-xr-x+ 1 kyumin 없음      0 Apr 14 15:45 d1
drwxr-xr-x+ 1 kyumin 없음      0 Mar  6 12:47 d2
-rwxr-xr-x+ 1 kyumin 없음      1 Apr 18 23:21 digitfile
-rw-r--r--+ 1 kyumin 없음    249 Apr 15 09:10 ex0.c
-rwxr-xr-x+ 1 kyumin 없음   66203 Apr 15 09:11 ex0.exe
-rw-r--r--+ 1 kyumin 없음     45 May  9 00:03 ex1.c
-rwxr-xr-x+ 1 kyumin 없음   66315 May  9 01:15 ex1.exe
-rw-r--r--+ 1 kyumin 없음     566 Apr 18 23:21 ex10.c
-rwxr-xr-x+ 1 kyumin 없음   68716 Apr 18 23:21 ex10.exe
-rw-r--r--+ 1 kyumin 없음     361 Apr 29 09:23 ex2.c
-rwxr-xr-x+ 1 kyumin 없음   67119 Apr 29 09:26 ex2.exe
-rwx-----+ 1 kyumin 없음   30268 Mar 31 18:54 swvader03.wav
-rw-r--r--+ 1 kyumin 없음  128652 Apr  1 09:05 x
$cp f1 f2
I am child to execute cp
$cat f2
I am child to execute cat
I have a dream
that one day this nation
will rise up and
live out the true
meaning of its creed

```

“ls -l”과 “cp f1 f2” 명령어 모두 정상적으로 실행된다.

F2파일을 제거한 후 위 명령어를 실행해봤고, f2의 내용을 출력해보면 정상적으로 출력된다.

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
.....
```

```
void main(){
    int x,y,status, i;
    char buf[50];
    char * argv[10];

    //문자열에 PATH 환경 변수의 주소를 저장
    char env[1024];
    strcpy(env, getenv("PATH"));

    char *PrintEnv[256]; //토큰을 저장하기 위해 선언
    PrintEnv[0] = strtok(env, ":");
    printf("%s\n", PrintEnv[0]);

    //각 배열에 각 주소를 하나씩 저장
    for(int i = 1; i++)
    {
        PrintEnv[i] = strtok(NULL, ":");
        if(PrintEnv[i] == NULL) break;
        printf("%s\n", PrintEnv[i]);
    }

    for(;;)
    {
        printf("$");
    }
}
```

main함수에 환경변수 PATH에 대해 출력하기 위해 코드를 추가하였다. 문자열 env를 선언하고, strcpy를 통해서 환경 변수의 모든 내용을 저장한다. 그리고 토큰을 저장하기 위한 포인터를 배열로 선언하고, for문을 돌려서 하나씩 추출 및 출력을 한다. 환경변수의 내용을 보면 주소가 “:”으로 구별된다. 그래서 `strtok(PrintEnv[i], “:”)`를 입력했다.

```
kyumin@DESKTOP-NUDFAPK ~
$ mysh
/usr/local/bin
/usr/bin
/cygdrive/c/WINDOWS/system32
/cygdrive/c/WINDOWS
/cygdrive/c/WINDOWS/System32/wbem
/cygdrive/c/WINDOWS/System32/WindowsPowerShell/v1.0
/cygdrive/c/WINDOWS/System32/OpenSSH
/cygdrive/c/Program Files/Git/cmd
/cygdrive/c/Program Files/dotnet
/cygdrive/c/Program Files/PUTTY
/cygdrive/c/Program Files/Bandizip
/cygdrive/c/Users/kyumin/AppData/Local/Programs/Python/Python37/Scripts
/cygdrive/c/Users/kyumin/AppData/Local/Programs/Python/Python37
/cygdrive/c/Users/kyumin/AppData/Local/Microsoft/WindowsApps
$
```

환경변수 PATH를 줄 단위로 출력되는 것을 볼 수 있다.

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.

```
void main(){
    int x,y,status, i;
    char buf[50];
    char * argv[10];

    for(;;)
    {
        printf("$");
        fgets(buf, 50, stdin);
        buf[strlen(buf) - 1] = '\0';

        //입력된 문장을 토큰으로 나누기
        argv[0] = strtok(buf, " ");
        int flag = 0;
        for(int i = 1;; i++)
        {
            argv[i] = strtok(NULL, " ");
            if(argv[i] == NULL) {
                argv[i] = 0;
                break;
            }

            //& 문자가 입력된 경우
            if(strcmp(argv[i], "&") == 0){
                flag = 1;          //플래그 세우기
                argv[i] = 0;       //execve에 0인 인자가 필요
            }
        }

        //exit 입력된 경우
        if(strcmp(argv[0], "exit") == 0) exit(0);

        //cd 입력된 경우
        else if(strcmp(argv[0], "cd") == 0) chdir(argv[1]);

        //pwd 입력된 경우
        else if(strcmp(argv[0], "pwd") == 0)
        {
            char pwd[256];
            getcwd(pwd, 256);
            printf("%s\n", pwd);
        }
    }
}
```

```

//내장 명령어가 아닌 경우
else
{
    x=fork();
    if (x==0){ // child
        printf("I am child to execute %s\n", buf);

        //환경 변수 PATH의 경로를 저장
        char env[1024];
        strcpy(env, getenv("PATH"));

        //토큰을 저장하기 위해 선언
        char *EnvTok[256];
        EnvTok[0] = strtok(env, ":");

        //경로 별로 하나씩 실행
        for(int i = 0;; i++)
        {
            char temp[50];
            strcpy(temp, EnvTok[i]);
            strcat(temp, "/");
            strcat(temp, argv[0]);

            y=execve(temp, argv, 0);

            //실행에 실패할 경우 다음 경로 실행
            if(y < 0){
                EnvTok[i + 1] = strtok(NULL, ":");

                //마지막까지 못 찾을 경우
                if(EnvTok[i + 1] == NULL) {
                    perror("exec failed");
                    exit(1);
                }
            }
        }

    }
    else if(flag == 0) wait(&status);
}
}

```

우선 fgets 함수로 사용자로부터 명령어를 입력받으면 strtok함수를 사용해서 단어별로 포인터 argv에 저장한다.

그리고 내장 명령어가 아닌 다른 명령어가 입력되었다면 두 번째 사진의 else가 실행된다. 여기서 프로세스 복제가 이뤄지고, 자식 프로세스에서는 명령어를 실행한다.

사용자로부터 명령어를 절대경로가 아닌 명령어를 입력받으면 작동하게 만들기 위해서 PATH라는 환경변수를 이용했다. 이 환경변수에는 여러 주소가 저장되어있고, 이 주소는 “:”으로 구별되니 strtok을 이용해서 주소를 포인터인 EnvTok에 각각 저장해줬다. 이 때 명령어를 실행하기 위해서 strcpy와 strcat 함수를 이용해서 주소 + 명령어(실행파일) 형태로 만들었고, 만들어진 주소를 execve에 사용한다. 그러나 실행이 되지 않는 경우가 있는데 PATH의 다음 주소를 이용해서 위 과정을 다시 시도한다. 그렇게해서 실행이 될 때까지 진행한다.


```

kyumin@DESKTOP-NUDFAPK ~
$ mysh
$ls -l
I am child to execute ls
total 1679
drwx-----+ 1 kyumin 없 음      0 Mar 15  2021 cdssetup
drwxr-xr-x+ 1 kyumin 없 음      0 Apr 14 15:45 d1
drwxr-xr-x+ 1 kyumin 없 음      0 Mar  6 12:47 d2
-rwxr-xr-x+ 1 kyumin 없 음      1 Apr 18 23:21 digitfile
-rw-r--r--+ 1 kyumin 없 음    249 Apr 15 09:10 ex0.c
-rwxr-xr-x+ 1 kyumin 없 음  66203 Apr 15 09:11 ex0.exe
-rw-r--r--+ 1 kyumin 없 음     45 May  9 00:03 ex1.c
-rwxr-xr-x+ 1 kyumin 없 음  66315 May  9 01:15 ex1.exe
-rw-r--r--+ 1 kyumin 없 음     566 Apr 18 23:21 ex10.c
-rwxr-xr-x+ 1 kyumin 없 음  68716 Apr 18 23:21 ex10.exe
-rw-r--r--+ 1 kyumin 없 음     361 Apr 29 09:23 ex2.c
-rwxr-xr-x+ 1 kyumin 없 음  67119 Apr 29 09:26 ex2.exe

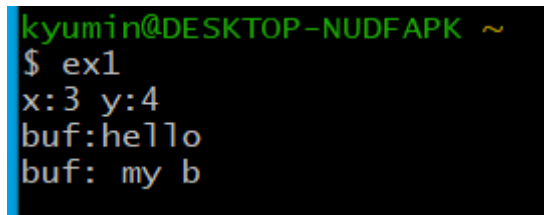
```

정상 작동하는 것을 볼 수 있다.

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);
```



```
kyumin@DESKTOP-NUDFAPK ~
$ ex1
x:3 y:4
buf:hello
buf: my b
```

```
int x = open("f1", ~);
```

x는 파일 디스크립터이다. 위 경우 3을 할당 받았다. 0, 1, 2는 기본적으로 할당되는 파일 디스크립터고, 각각은 표준 입력 / 표준 출력 / 표준 에러를 의미한다. X는 이 다음 번호인 3을 할당 받은 것이다. 3은 파일 테이블에서 f1을 가리킨다. 이 때 처음 오픈을 한 경우 file position은 0이다.

```
y=dup(x);
```

y도 파일 디스크립터이며, x와 동일한 파일 테이블을 가리킨다. 파일 디스크립터 3은 이미 사용 중이므로 다음 번호인 4를 할당받았다.

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

파일 디스크립터 x는 f1 파일을 5 만큼 읽는다. buf에는 "hello"라는 메시지가 저장된다. 이 때 파일 테이블에서 f1 파일의 file position은 5가 된다. 다시 read를 할 때 5부터 읽는다는 의미다.

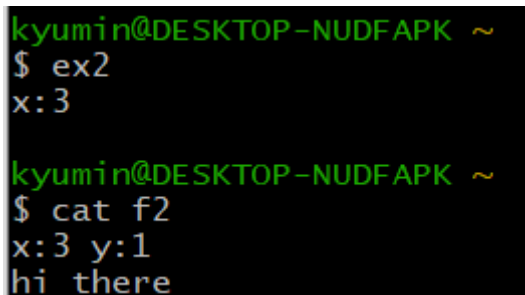
```
k=read(y, buf, 5);
```

y도 f1파일을 읽는다. 이 때 y는 x와 동일한 파일 테이블을 가리킨다. 이전에 f1 파일의 file position은 5였으므로 5부터 5글자를 읽는다. buf에는 “ my b”가 저장된다. 이 때 파일 테이블에서 f1 파일의 file position은 10이 될 것이다.

위 파일을 실행하면 위 사진과 같이 출력된다.

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);
```



```
kyumin@DESKTOP-NUDFAPK ~
$ ex2
x:3

kyumin@DESKTOP-NUDFAPK ~
$ cat f2
x:3 y:1
hi there
```

```
x=open("f2",~);
```

f2파일을 쓰기 모드로 열었다. 파일 디스크립터는 0, 1, 2는 표준 입출력으로 사용 중이므로 x는 3을 할당받았다.

```
close(1);
```

파일디스크립터 1은 표준 출력을 가리킨다. 해당 함수는 디스크립터 1번을 닫았으라는 의미고, 표준 출력을 사용하지 않는다.

```
y=dup(x);
```

y는 x의 파일 테이블을 가리킨다. 복제는 파일디스크립터 작은 번호부터 할당받으므로 y는 1을 할당 받는다.

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

printf의 기존 의미는 파일디스크립터 1번에 메시지를 작성하라는 의미다. 즉 표준 출력에 메시지를 작성하라는 의미이므로 화면에 메시지가 출력된다. 그러나 위 코드에서 y가 1을 할당받았으므로 printf는 1번 즉 f2 파일에 메시지를 작성하라는 의미가 된다.

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

1번 파일 디스크립터에 메시지를 작성하라는 의미다.

위 프로그램을 실행하면 f2파일은 사진과 같이 “x:3 y:1 hi there” 메시지가 저장된다.

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.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

void main(){
    int x, y, status;
    char buf[50];
    char * argv[10];

    for(;;)
    {
        printf("$");
        fgets(buf, 50, stdin);
        buf[strlen(buf) - 1] = '\0';

        //입력된 문장을 토큰으로 나누기
        argv[0] = strtok(buf, " ");
        int flag = 0;
        for(int i = 1;; i++)
        {
            argv[i] = strtok(NULL, " ");
            if(argv[i] == NULL) {
                argv[i] = 0;
                break;
            }

            //& 문자가 입력된 경우
            if(strcmp(argv[i], "&") == 0){
                flag = 1; //플래그 세우기
                argv[i] = 0; //execve에 0인 인자가 필요
            }
        }
    }
}
```

```

//exit 입력된 경우
if(strcmp(argv[0], "exit") == 0) exit(0);

//cd 입력된 경우
else if(strcmp(argv[0], "cd") == 0) chdir(argv[1]);

//pwd 입력된 경우
else if(strcmp(argv[0], "pwd") == 0)
{
    char pwd[256];
    getcwd(pwd, 256);
    printf("%s\n", pwd);
}

//내장 명령어가 아닌 경우
else
{
    x=fork();
    if (x==0){ // child
        printf("I am child to execute %s\n", buf);

        //cat f1 > f3 입력되는 경우
        if(argv[2] != NULL && strcmp(argv[2], ">") == 0)
        {
            close(1);
            int fd = open(argv[3], O_WRONLY|O_CREAT|O_TRUNC, 00777);
            argv[2] = '\0';
        }

        //환경 변수 PATH의 경로를 저장
        char env[1024];
        strcpy(env, getenv("PATH"));

        //토oken을 저장하기 위해 선언
        char *EnvTok[256];

```

```

//경로 별로 하나씩 실행
for(int i = 0;; i++)
{
    char temp[50];
    strcpy(temp, EnvTok[i]);
    strcat(temp, "/");
    strcat(temp, argv[0]);

    y = execve(temp, argv, 0);

    //실행에 실패할 경우 다음 경로 실행
    if(y < 0){
        EnvTok[i + 1] = strtok(NULL, ":");

        //마지막까지 못 찾을 경우
        if(EnvTok[i + 1] == NULL) {
            perror("exec failed");
            exit(1);
        }
    }
}
}
else if(flag == 0) wait(&status);
}
}
}

```

Cat f1 > f2 명령어는 f1의 내용을 f3에 복사하는 기능이다. 이 기능을 구현해 보았다. 위 사진에서 `if(argv[2] != NULL && strcmp(argv[2], ">") == 0)` 내부 내용을 보면 `close(1)`이 먼저 실행된다. 이는 표준 출력을 사용하지 않는다는 의미다. `Argv[3]`이 이름인 파일을 오픈했는데 이 때 쓰기 모드로 열었고, 파일 디스크립터는 1번이 비어있으므로 1번을 할당받는다. 그리고 ">"가 저장되어있는 `argv[2]`는 'w0'으로 수정해준다.

그 이유는 `execve` 실행 시 두 번째 argument로 `argv`가 들어가는데 이 때 "cat f1"만 입력되도록 만들기 위해서다. Cat은 원래 f1의 파일을 화면에 출력하는 명령어다. 그러나 표준 출력의 파일 디스크립터는 1번이지만 현재 1번은 f3파일에 할당되어있다. 그래서 cat f1 명령어가 실행되면 f3 파일에 f1의 내용이 써진다.

그래서 사용자가 "cat f1 > f3"를 입력하면 f1의 내용이 f3에 복사되는 것이다.

```

kyumin@DESKTOP-NUDFAPK ~
$ mysh
$cat f1 > f3
I am child to execute cat
$cat f1
I am child to execute cat
hello my boy
$cat f3
I am child to execute cat
hello my boy
$

```

Cat f1 > f3를 실행해보면 f3에는 f1의 내용과 동일한 내용이 입력된 것을 볼 수 있다.