Lecture 10: Exec, Exit, Wait

**12201898 박준형**

**2. Homework**

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

ex1.c:

#include <stdio.h>

void main(){

printf("I am ex1\n");

}

ex2.c:

#include <stdio.h>

#include <unistd.h>

void main(){

execve(“./ex1”,0 , 0); // change to ./ex1 with no additional argument

printf("I am ex2\n");

}

$ gcc -o ex1 ex1.c

$ gcc -o ex2 ex2.c

$ ex2

I am ex1

For Cygwin, the above code will not work. Change as below:

ex2.c:

#include <stdio.h>

#include <unistd.h>

void main(){

char \*argv[5];

argv[0]=”./ex1”;

argv[1]=0;

execve(argv[0], argv, 0); // change to ./ex1 with no additional argument

printf("I am ex2\n");

}

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

다음과 같이 ex1프로그램이 ex2프로그램에서 실행되었다. ex2를 실행하여 execve함수를 만나면 기존 메모리상에 올라가있던 ex2가 메모리에서 지워지고, 해당 위치에 ex1이 로드되어 올라간다. 따라서 ex2의 execve아래부분의 코드는 실행되지 못하고 ex1의 코드만 실행이 된다.

To check whether execve succeeds or not:

#include <stdio.h>

#include <unistd.h>

void main(){

char \*argv[5];

argv[0]=”./ex1”;

argv[1]=0;

int y=execve(argv[0], argv, 0); // change to ./ex1 with no additional argument

if (y < 0){ // we have an error in execve

perror(“err:”); // show the reason of error

exit(0);

}

printf("I am ex2\n");

}

결과>

exit()함수를 이용하기 위해 stdlib.h를 포함하기만 하면 문제없이 실행된다.

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

myexec.c:

#include <stdio.h>

#include <unistd.h>

void main(){

char \*k[10];

k[0]=”/bin/ls”;

k[1]=0; // make sure you put 0 at the end

execve(k[0], k, 0); // change to /bin/ls with no additional argument

}

The above program will exec to /bin/ls and print the listing of files in the current directoy.

결과>

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

ls 명령어를 execv로 실행하여 현재디렉토리내의 모든 파일을 보여주고 있다.

1-3) Run myexec below. Explain the result.

myexec.c:

#include <stdio.h>

#include <unistd.h>

void main(){

char \*k[10];

k[0]=”/bin/ls”;

k[1]=”-l”; // l is small L not number 1

k[2]=0; // make sure you put 0 at the end

execve(k[0], k, 0); // change to “/bin/ls –l”

}

The above program will exec to “/bin/ls –l” and print a long listing of files in the current directoy.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

인자로 “-l”을 넘기니 ls -l을 한 것과 같은 결과를 나타낼 수 있게되었다.

1-4) Run myexec below and explain the result.

myexec.c:

void main(){

char \*x[10];

x[0]="/bin/cat";

x[1]="f1";

x[2]=0; // argument list should end with 0

execve(x[0], x, 0); // change to /bin/cat with one argument f1

}

The above program will exec to "/bin/cat f1" which will print the contents of f1.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

execve함수로 cat명령어를 실행하였고, 인자로 f1을 전달해 f1파일을 열어 보여주도록했다.

2) Change myexec such that it execs to “/bin/ls -l -a”. Note "l" is small L, not number 1.

코드>

텍스트, 스크린샷, 폰트, 바이올렛색이(가) 표시된 사진

자동 생성된 설명

두번째 인자로 -a를 추가하였다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

-a 인자가 추가되어 숨겨져있는 디렉토리를 보여주게 되었다.

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

코드>

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

보시다시피 f1이 복제되어 f2가 생김을 알 수 있다.

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

코드>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

fork를 이용해 자식프로세스에서 execve가 실행되도록하고 해당 자식프로세스가 종료될때까지 기다리기위해 wait함수를 사용하였다. 그리고 자식프로세스가 종료되면, job done을 출력하도록 하였다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

다음과 같이 ls -l이 실행되고, job done이 출력됨을 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.

$ myexec

command> /bin/cat f1

….myexec execs to "/bin/cat f1"

$ myexec

command> /bin/cp f1 f2

….myexec execs to "/bin/cp f1 f2"

코드>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 보라색이(가) 표시된 사진

자동 생성된 설명

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

여러 개의 인자를 직접 입력받아 해당 명령어를 실행할 수 있다.

6) Same as 5), but myexec will repeat the process forever until you stop the program with “ctrl-c”. Also display the result of strtok as shown below.

$ myexec

command> /bin/cat f1

argc: 2 argv[0]: /bin/cat argv[1]: f1 argv[2]:0

….display the contents of f1

command> /bin/cp f1 f2

argc: 3 argv[0]: /bin/cp argv[1]: f1 argv[2]: f2 argv[3]: 0

….copy f1 to f2

command> /bin/cat f2

………….

코드>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

token화가 완료되면, 인수를 반복문을 통해 출력하도록 하였다. 그리고 계속해서 myexec가 실행되어 인수를 입력받고 해당 명령어를 실행하게하기위해 while문을 이용해 무한반복시켜주었다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

다음과 같이 계속해서 인수를 입력받고 해당 명령어를 실행시킬 수 있었다.

7) Same as 6), but change the prompt to the current location and ‘$’ as follows. You may need “getcwd”. Also add code to detect “exit” command and stop the program when the user types “exit”.

$ myexec

[/home/sp1/12345]$ /bin/cat f1

argc: 2 argv[0]: /bin/cat argv[1]: f1 argv[2]:0

….display the contents of f1

[/home/sp1/12345]$ /bin/cp f1 f2

argc: 3 argv[0]: /bin/cp argv[1]: f1 argv[2]: f2 argv[3]: 0

….copy f1 to f2

[/home/sp1/12345]$ /bin/cat f2

………….

[/home/sp1/12345]$ exit

bye

$

코드>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

6번문제와 여기서 달라진 것은 위의 빨간색박스로 친 영역이다.

getcwd를 이용해 현재 디렉토리의 위치를 출력해주는 것과,

exit를 입력받는 경우 종료하는 것이다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

다음과 같이 현재경로를 앞에 잘 표시하고 있고, exit를 입력하면 정상적으로 종료되는 것을 확인할 수 있다.

4. exit, wait

exit: A program calls exit() to exit.

- remove the body

- becomes a zombie until the parent calls wait()

wait: A program calls wait() and waits for the child to exit.

- if the child already called exit()

remove its process descriptor

else

wait until the child exits

5. shell

algorithm:

 for(;;){

          printf("$");

          scanf("%s", buf);

          x=fork();

          if (x==0) execve(buf,0,0);

          else wait();

        }

Actual code: mysh.c

#include <stdio.h>

#include <unistd.h>

#include <string.h>

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 one

      printf("$");

      scanf("%s", buf); // get command.

      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);

   }

}

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

반복해서 명령어를 입력받을 수 있다. 그러나 인수는 받지 못하게 되어있다.

6. process tree

(For Cygwin, use “ps –W” to see all processes including window processes. To see ppid in windows, download “process explorer” and run it.)

kernel (pid=0)

fork: init (pid=1)

exec: /sbin/init (pid=1)

fork & exec: iscsid

rsyslogd

..........

/usr/sbin/sshd (pid=1262)

fork : sshd : linuxer (pid=5198)

fork : sshd: linuxer@pts/0 (pid=5201)

fork & exec: -bash (pid=5202)

fork & exec: vi cli.c (pid=6420)

7. debugging a program with fork

In gdb, use "set follow-fork-mode child" or "set follow-fork-mode parent" to debug child or parent process. (Cygwin does not support “set follow-fork-mode”. Just debug up to “fork” in cygwin.)

**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.

1. 명령어를 실행하기 위해 해당 실행파일이 있는 디렉토리까지 모두 작성해야함.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

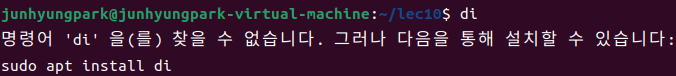
자동 생성된 설명

2. 인수를 입력할 수 없음

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

3. 새로운 명령어를 설치할 수 없음



텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

4. 디렉토리 위치를 바꿀 수 없음

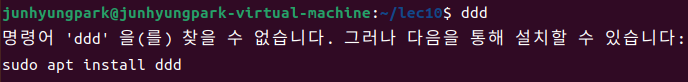
텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

5. 잘못된 명령어를 입력했을 때 나타나는 메시지가 다름.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명



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.

login shell -> pid: 6475, ppid: 6443



* login shell의 파일이름은 bash다.
* /proc 디렉토리를 확인하여 프로세스의 정보를 확인해보니 bash는 /usr/bin폴더내에 존재함을 알 수 있었다.
* 
* /bin 디렉토리 내에 존재한다.
* 텍스트, 스크린샷, 폰트이(가) 표시된 사진

  자동 생성된 설명

/usr/libexec/gnome-terminal-server -> pid: 6443, ppid: 1463



* bash파일의 부모 프로세스는 /usr/libexec/gnome-terminal-server 이다.
* 아마도 해당 부모 프로세스는 fork를 이용해 자식 프로세스를 만들고, 자식프로세스에서 exec를 통해 bash를 실행한 것으로 보인다.
* x=fork()
* if(x==0) {
* execve(“/usr/bin/bash”,0,0);
* }
* else {
* wait();
* }
* 다음과 같이 실행되는 것으로 보인다.

/lib/system/system –user -> pid: 1463, ppid: 1



/sbin/init auto noprompt splash -> pid:1, ppid: 0



계속해서 부모의 pid를 추적하였더니 최종적으로 /sbin/init 파일로 연결됨을 알 수 있었다.

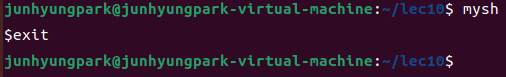
3) (Builtin Command) Improve mysh such that it exits when the user types "exit". You have to handle “exit” before “fork”. Explain why. This kind of commands that the shell has to handle before fork are called built-in commands.

코드>

텍스트, 폰트, 스크린샷, 라일락이(가) 표시된 사진

자동 생성된 설명

부모 프로세스에서 fork를 하기전에 exit가 입력되면, exit될 수 있도록 코드를 추가해주었다.



정상적으로 종료되는 것을 확인할 수 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.

코드>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

fgets로 공백을 포함해 문자열을 입력받은뒤 tokenize를 통해 공백을 기준으로 분리한뒤, argv를 프로그램이름, 인자1, 인자 2, … 순으로 배열을 만들어준다. 그리고 execve의 첫번째 인자에 argv[0]을 통해 실행시킬 프로그램 이름을 입력해주었다.

결과>

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명

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.

코드>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

디렉토리를 변경하기 위해 unistd.h에 있는 chdir함수와 getcwd함수를 이용하였다.

chdir함수를 이용해 cd를 구현할 수 있었다. 이때 cd 다음에 인자가 없는 경우, 에는 getenv함수를 이용해 HOME으로 설정되어있는 환경변수에서 유저디렉토리를 얻어내 해당 디렉토리로 이동하도록 하였고,



인자가 있는 경우에는 인자로 주어진 경로로 이동하도록 하였고, 2개이상 인자가 주어진 경우, 오류메세지를 주도록 했다.

getcwd함수를 이용해 pwd를 구현할 수 있었다. cwd변수에 getcwd함수를 이용해 현재 디렉토리의 값을 저장하고 이를 출력하였다. 2개이상인자가 주어진 경우, 오류메세지를 주도록 했다.

결과>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

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.

코드>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 친필이(가) 표시된 사진

자동 생성된 설명

결과>

& 없이 사용한 경우,

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명& 를 사용한 경우,

ex1을 실행할 때, 부모가 wait을 하지 않기 때문에 바로 다음 명령어를 작성하는 단계로 넘어가는 것을 확인할 수 있다. 이때 백그라운드에서 실제 동작하는지 여부를 파악하기 위해 ps-ef 명령어로를 통해 전체 프로세스를 출력했고, 여기서 ex1 프로세스가 백그라운드에서 동작중이라는 것을 확인할 수 있었다.

여기서 중요한 것은 &을 탐지한 이후에 &을 없앤 argv 배열을 자식에게 넘겨주어야한다는 것이었다.

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.

코드>

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

빌트인 커맨드 체크 바로 다음에 해당 코드를 집어넣어주었다. 해당 코드는 argv[0]의 명령어를 /bin/을 붙여주어 절대경로로 바꾸어주고, 이를 다시 argv[0]에 넣어주는 코드다.

결과>

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

다음과 같이 절대경로를 써주지 않고도 명령이 잘 실행됨을 알 수 있다.

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

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

코드>

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

getenv함수를 이용해 환경변수목록을 불러오고, 이를 :를 기준으로 나누어 출력하도록하였다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

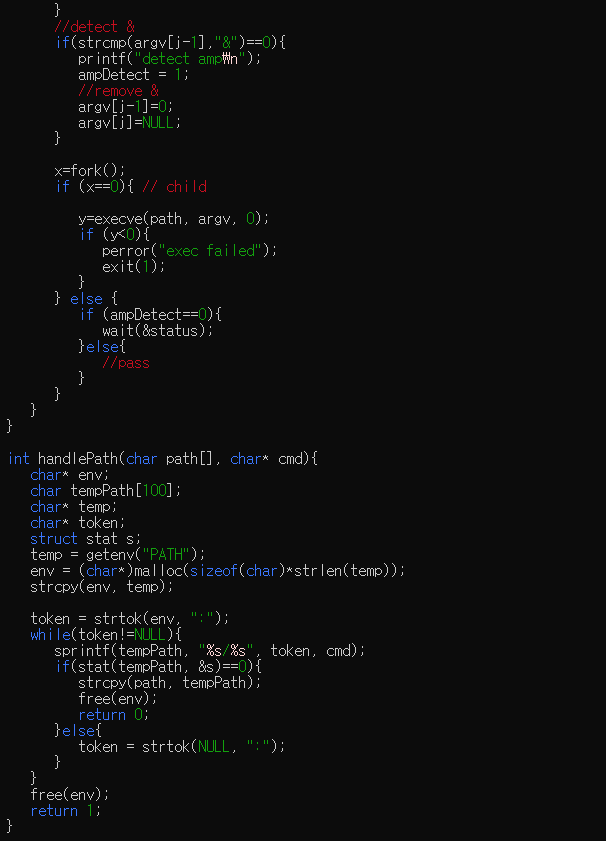
자동 생성된 설명

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.

코드>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명



path를 체크하는 함수인 handlePath를 만들었다. 우선 해당 함수는 getenv로 받은 환경변수 목록을 temp에 저장하고 해당크기만큼 env를 동적할당한다. 이 과정을 진행하는 이유는 getenv로 받은 문자열을 tokenize를 하게되면 strtok이 원본 문자열을 파괴하기 때문에 한번은 괜찮지만, 두번째 명령어를 입력받고 다시 해당 함수를 실행 시 환경변수를 찾을 수 없기때문이다. 따라서 env에 이를 복사해놓는 것이다.

이어서 tokenize한 것을 token과 입력된 명령어를 합쳐 절대경로를 생성한다. 그리고 이를 stat함수를 이용해 해당 위치에 해당파일이 존재하는지 확인한다. stat함수는 파일이 존재하면 0을 출력한다. 따라서 0이 반환되면, 인자로 받은 path에 해당 경로를 복사하고 함수를 종료한다. 못찾으면 이를 계속 반복한다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

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);

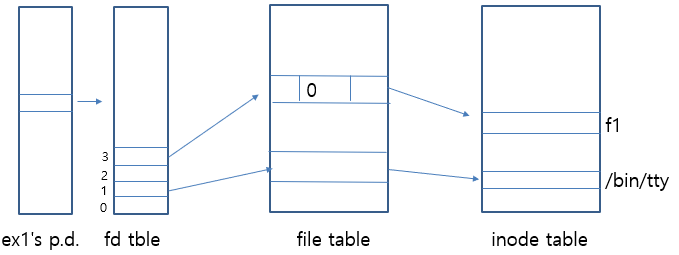
결과>

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

이유>

f1파일을 열면, 다음과 x에는 fd table의 다음 가능한 index인 3이 저장되고, inode table에서 f1을 찾아 file table에서 이를 가리키는 값을 할당하고, file안에서의 읽을 위치인 0이 file table에 저장된다.

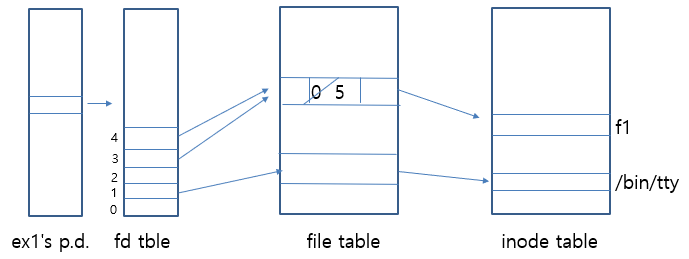


그리고 y=dup(x)를 하게되면, f1을 가리키는 file table 값을 가리키는 다른 하나의 fd table의 가능한 다음 index인 4가 생성되고 해당값이 y에 저장된다. 즉, x,y는 모두 같은 파일을 가리키게 된다.

도표, 라인, 기술 도면, 평행이(가) 표시된 사진

자동 생성된 설명

파일을 열고, 5바이트를 읽은 후 file pointer는 5를 가리키고 있게 된다. 가장 앞부터 5바이트를 읽었으므로, hello가 출력된다.



그리고 y를 이용해 read를 하면 y는 f1파일을 가리키는 상태이므로 f1을 읽게 된다. 따라서 해당 file table은 현재 file position이 5인상태이므로 5부터 다시 5바이트를 읽게 된다. 따라서 hello 다음부터인 (공백)my(공백)b 총 5바이트를 읽게된다. 그리고 나면 다음 같이 file table은 file position을 10으로 변경시킨다.

도표, 라인, 기술 도면, 평면도이(가) 표시된 사진

자동 생성된 설명

따라서 위와 같은 구조를 통해 file descriptor 를 복제할 수 있게된다.

이때 주의해야할점은 dup함수는 fd table에서 가능한 가장 작은 index값을 반환한다는 점이다.

fd table은 다음과 같이 기본적으로 3가지 file descriptor를 가진다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

따라서 close(1)을 통해 stdout을 삭제하면, y=dup(x)를 통해 dup함수를 이용해 복제한 뒤 반환되는 index값은 1이 된다. 따라서 printf와 같은 출력함수를 이용해 출력을 시도하면, 그 값이 y가 가리키는 file에 저장이 되게 됨을 주의하자.

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);

결과>

* 위 코드 실행결과  
    
  단순히 x에 저장된 fd table의 index가 저장되었다.
* f2파일에 저장된 값 확인  
  텍스트, 폰트, 스크린샷이(가) 표시된 사진

  자동 생성된 설명

이유>

위 코드에서 라인 별로 설명하도록하겠다.

**-close(1)**

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

fd table에서의 index 1은 표준출력을 담당하고 있는데 이를 해제하였으므로 다음 dup함수를 이용한 반환값은 1이된다. 이때 해당 값을 받은 file descriptor는 모든 출력함수들을 이용하면 해당 파일로 내용이 저장되게된다.

**-y=dup(x)**

현재 가능한 가장 작은 fd table에서의 index값은 1이므로 y에는 1이 들어간다. 그리고 y는 x가 가리키는 f2파일을 동일하게 가리키게된다.

도표, 라인, 기술 도면, 평행이(가) 표시된 사진

자동 생성된 설명

**-printf, write함수**

위 함수들 모두 fd table에서 1번 index로 값을 내보내기 때문에 현재 1번 index에 있는 y가 가리키는 f2파일에 원래는 터미널로 출력되야할 값들이 저장이 되게 된다.

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.

코드>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

우선 handleRedirection함수를 통해 argv 배열을 > 기호를 기준으로 왼쪽과 오른쪽 값으로 나누어 배열을 만든다. 이를 left, right배열이라 한다. 따라서 left 배열은 명령어가, right 배열은 file명이 들어간다. 이때 해당 함수는 >의 인덱스번호를 반환하므로 이를 d에 저장한다.

d의값이 argc(인자의 개수)와 같으면, >연산자가 없는 것이므로 원래하던데로 execve함수를 호출한다.

그러나 아닌 경우, >연산자가 존재하는 것이므로, for문을 반복하면서 right배열의 값을 하나씩 가져와 파일을 생성하고, close(1)을 통해 표준 출력을 fd table 1번 인덱스에서 제거하고, d=dup(file)을통해 file을 가리키는 file descriptor를 만든다. 그러면, d는 1번 인덱스를 가지게 되어 앞으로 실행되는 출력함수는 d가 가리키는 file에 출력내용이 저장된다.

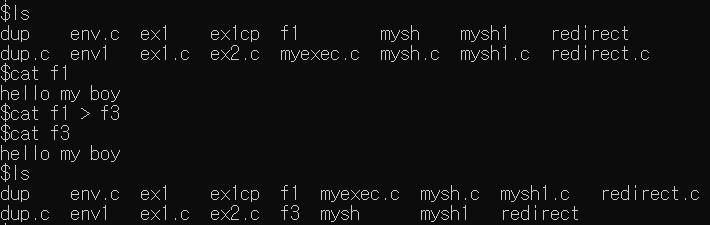
텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

handleRedirection함수는 >기호의 위치를 찾고, 이를 기준으로 왼쪽과 오른쪽을 따로 배열로 만드는 역할을 하는 함수이다.

p.s.) left, right 배열을 만드는 것을 동적할당으로 하고 싶었지만, 오류가 많이 나 고정배열로 선언뒤 사용하였습니다. 추가로 시도해보고, 동적할당으로도 구현해보도록 하겠습니다.

결과>



다음과 같이 redirection을 통해 f1의 출력 데이터가 f3파일에 저장된 것을 확인할 수 있다.