lect6 HW

시스템프로그래밍 1분반 12180626 성시열

1) Make a file, "f1", and fill it with more than 20 bytes.

```
-bash-4.2$ vi f1
```

```
I have a dream
that one day this nation
will rise up and
live out the ture
meaning of its creed
that all men are created equal.
~
~
```

f1파일을 vi 편집기로 open한 뒤, 'i'키를 통해 insert 모드로 변경하였다. 이후 위와 같은 문자열을 입력하고 esc키와 :wq 입력하여 문자열을 저장하고 f1 파일을 빠져나온다.

2) Try the code in 6-0), 6-1), 6-2), 6-3).

6-0)

ex2.c를 생성하여 코드를 작성해본다.

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

int main(){
        int x, y;
        char buf[50];

        x = open("f1",O-RDONLY,00777);
        y = read(x, buf, 20);
        buf[y] = 0;
        prrintf("%s\n", buf);

        return 0;
}
~
~
```

```
-bash-4.2$ vi ex2.c
-bash-4.2$ g++ -o ex2 ex2.c
-bash-4.2$ ./ex2
I have a dream
that
```

파일 open, read 등의 함수를 사용하기 위해 #include를 추가한다.
코드를 보면 f1파일을 읽기형태로 open하고 그 파일의 파일번호를 x에 return 한다.
이후 x의 파일번호를 통해 f1 파일로부터 20bytes를 읽어 buf에 저장을 한다.
이후 실제 읽은 byte가 y값에 return 된다. string의 마지막에는 0이 들어가야 하기 때문에 buf[y] = 0을 넣어 string의 마무리를 나타낸다.

컴파일 후 실행해보면
['I', ' ', 'h', 'a', 'v', 'e', ' ', 'a', ' ', 'd', 'r', 'e', 'a', 'm', ' ', 't', 'h', 'a', 't', ' '] 총 20byte 까지 출력됨을 알 수 있다.

6-1)

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

int main(){
        int x, y;
        char buf[50];

        x = open("f1",O_RDONLY,00777);
        y = read(x, buf, 20);
        write(1, buf, y);

        return 0;
}
~
~
```

```
-bash-4.2$ vi ex2.c
-bash-4.2$ g++ -o ex2 ex2.c
-bash-4.2$ ./ex2
I have a dream
that -bash-4.2$ vi ex2.c
```

6-0)에서 변형된 부분을 설명하면 f1 파일로부터 최대 20bytes까지 읽은 값을 buf에 저장하고 y에는 실제 read한 byte 수가 return 된다.(20) 이후 write를 해석하면 buf에 저장된 값을 1번 파일에 최대 y만큼 write하는 것이다. 여기서 1번 파일은 "standard output file"로 화면에 출력하는 것을 의미한다.

컴파일 후 실행해보면 우리가 보는 화면에 20bytes 까지 출력되어 있는 것을 확인할 수 있다.

6-2)

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

int main(){
        int x1, x2, y;
        char buf[50];

        x1 = open("f1", O_RDONLY,00777);
        x2 = open("f2", O_RDWR | O_CREAT | O_TRUNC, 00777);
        y = read(x1, buf, 20);
        write(x2, buf, y);

        return 0;
}
~
~
```

```
-bash-4.2$ vi ex2.c
-bash-4.2$ g++ -o ex2 ex2.c
-bash-4.2$ ./ex2
```

코드를 보면 f1파일을 읽기모드로 open하고 x1에 파일번호를 return한다. f2파일을 읽기, 쓰기가 가능하며, f2파일이 존재하지 않으면 직접 생성하고, 불러온 파일에 기존 데이터가 존재한다면 모두 지워주는 모드로 open하고 x2에 파일번호를 return한다.
x1에 저장된 파일번호를 통해 f1 파일로부터 최대 20bytes 만큼 읽어와 buf에 저장한다.
y에는 실제 read한 byte수가 저장된다(20).
x2 파일번호를 통해 f2파일에 buf 값을 y(20)만큼 write한다.
컴파일 후 실행하면 아무런 값이 출력되지 않음을 알 수 있다. 이는 write에서 값을 x2 파일번호에 write 했기 때문이다.

```
-bash-4.2$ ls
d1  d3   ex1.c  ex2.c  ex5    f1  f3  f5  f7  y.c
d2  ex1  ex2    ex4.c  ex5.c  f2  f4  f6  x
-bash-4.2$ cat f2
I have a dream
that -bash-4.2$
```

cat f2를 통해 f2에 저장된 값을 보면 20byte만큼 저장됨을 알 수 있다.

6-3)

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

int main(){
        int x1, x2, y;
        char buf[50];

        x1 = open("f1", O_RDONLY,00777);
        x2 = open("f2", O_RDWR | O_CREAT | O_TRUNC, 00777);
        for(;;){
                y = read(x1, buf, 20);
                if (y==0) break;
                write(x2, buf, y);
        }
        return 0;
}
~
~
```

```
-bash-4.2$ vi ex2.c
-bash-4.2$ g++ -o ex2 ex2.c
-bash-4.2$ ./ex2
-bash-4.2$ ls
d1  d3   ex1.c  ex2.c  ex5    f1  f3  f5  f7  y.c
d2  ex1  ex2    ex4.c  ex5.c  f2  f4  f6  x
-bash-4.2$ cat f2
I have a dream
that one day this nation
will rise up and
live out the ture
meaning of its creed
that all men are created equal.
```

6-2)에서 read와 write를 무한 루프에 넣어주고 y가 0이 될 때 빠져나오는 코드로 변경하였다. 이를 통해 f1을 통해 buf에 저장된 긴 문자열을 20byte 읽고 f2파일에 write하는 과정을 무한 반복한다. 이후 read함수에서 실제 read하는 byte가 0이 될 때, 모든 문장을 모두 읽었을 때 y는 0이 되어 반복문을 탈출한다.

컴파일 후 실행한 뒤 cat f2를 통해 f2 파일을 확인해보았다. f1에 저장된 모든 문자열이 f2에 저장됨을 확인할 수 있었다.

3) Find the byte size of f2 with "ls –l f2". Use xxd to find out the actual data stored in f2.

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

int main(){
        int x1, x2, y;
        char buf[50];
        int byte_size = 0;

        x1 = open("f1", O_RDONLY,00777);
        x2 = open("f2", O_RDWR | O_CREAT | O_TRUNC, 00777);
        for(;;){
                y = read(x1, buf, 20);
                byte_size = byte_size + y
                if (y==0) break;
                write(x2, buf, y);
        }
        printf("f2's size = %d", byte_size)
        return 0;
}
```

```
-bash-4.2$ vi ex2.c
-bash-4.2$ g++ -o ex2 ex2.c
-bash-4.2$ ./ex2
f2's size = 128-bash-4.2$
```

y의 값을 축적하여 더한 결과 f2가 128bytes 인 것을 알 수 있었다.

```
-bash-4.2$ ls -l
total 132
d--------- 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
-rwxrwxr-x 1 12180626 12180626  8503 Jun 22 08:35 ex1
-rw-rw-r-- 1 12180626 12180626   194 Jun 22 08:34 ex1.c
-rwxrwxr-x 1 12180626 12180626  8887 Jun 23 10:07 ex2
-rw-rw-r-- 1 12180626 12180626   433 Jun 23 10:07 ex2.c
-rw-rw-r-- 1 12180626 12180626    52 Jun 21 10:30 ex4.c
-rwxrwxr-x 1 12180626 12180626 13214 Jun 22 17:32 ex5
-rw-rw-r-- 1 12180626 12180626  1286 Jun 22 17:45 ex5.c
-rw-rw-r-- 1 12180626 12180626   128 Jun 23 09:20 f1
-rwxrwxr-x 1 12180626 12180626   128 Jun 23 10:07 f2
-rw-rw-r-- 1 12180626 12180626     6 Jun 20 11:18 f3
-rw-rw-r-- 1 12180626 12180626     6 Jun 20 11:17 f4
-rw-rw-r-- 2 12180626 12180626     6 Jun 20 18:10 f5
-rw-rw-r-- 2 12180626 12180626     6 Jun 20 18:10 f6
-rw-rw-r-- 1 12180626 12180626     0 Jun 20 18:32 f7
-rw-rw-r-- 1 12180626 12180626 35633 Jun 21 10:28 x
-rw-rw-r-- 1 12180626 12180626    52 Jun 21 10:30 y.c
```

ls –l을 통해서도 f2의 크기를 확인해본 결과 128임을 알 수 있다.

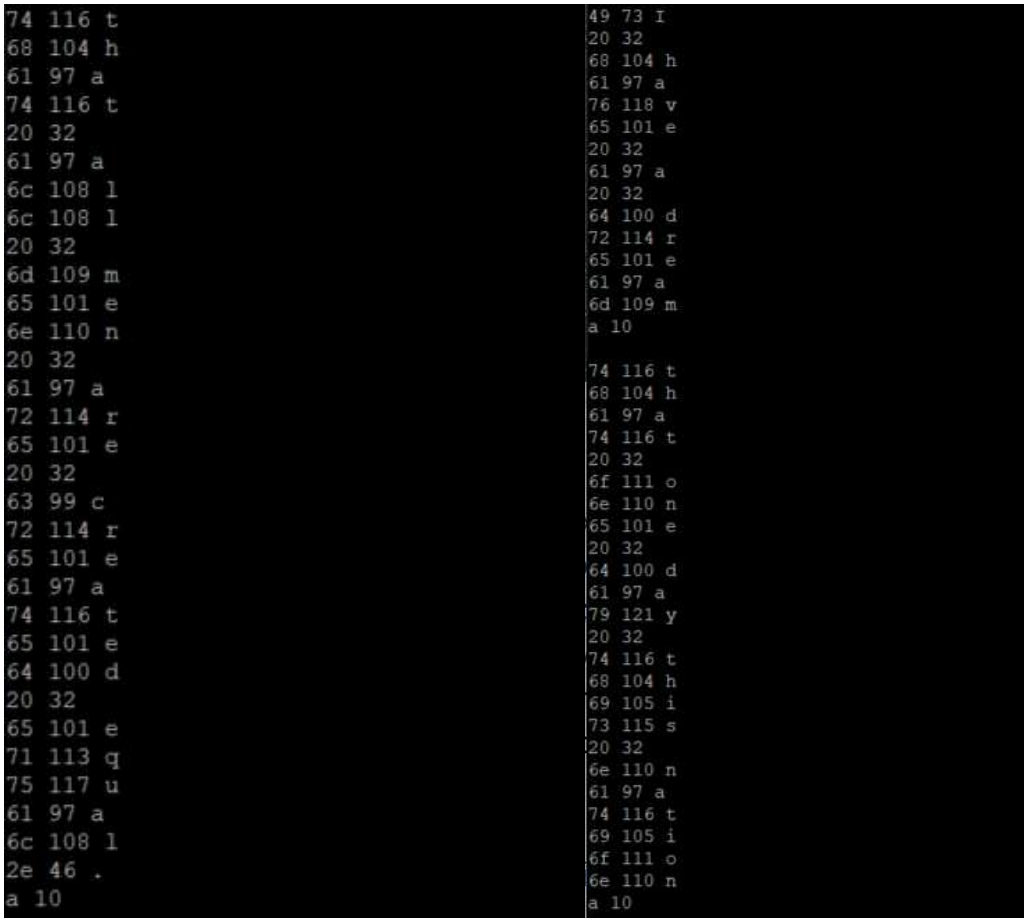
```
-bash-4.2$ xxd f2
0000000: 4920 6861 7665 2061 2064 7265 616d 0a74  I have a dream.t
0000010: 6861 7420 6f6e 6520 6461 7920 7468 6973  hat one day this
0000020: 206e 6174 696f 6e0a 7769 6c6c 2072 6973   nation.will ris
0000030: 6520 7570 2061 6e64 0a6c 6976 6520 6f75  e up and.live ou
0000040: 7420 7468 6520 7475 7265 0a6d 6561 6e69  t the ture.meani
0000050: 6e67 206f 6620 6974 7320 6372 6565 640a  ng of its creed.
0000060: 7468 6174 2061 6c6c 206d 656e 2061 7265  that all men are
0000070: 2063 7265 6174 6564 2065 7175 616c 2e0a   created equal..
```

xxd f2를 통해 f2에 저장된 값을 16진수로 표현한 결과를 확인하였다.

4) Write a program "hw4.c" that opens f2 and shows each byte of it in hexadecimal number, decimal number, and character. Use printf("%x %d %c₩n", ...) to display a number in various format.

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

int main(){
        int x, y;
        char buf[20];

        x = open("f2", O_RDONLY, 00777);
        for(;;){
                y = read(x, buf, 1);
                if(y==0) break;
                printf("%x %d %c\n",buf[0], buf[0], buf[0]);
        }
}

~
~
```

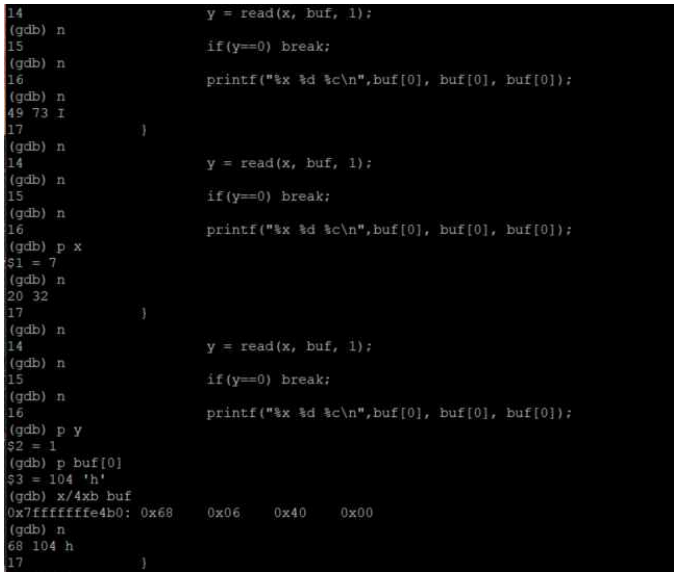
```
-bash-4.2$ vi ex2.c
-bash-4.2$ vi hw4.c
-bash-4.2$ g++ -o hw4 hw4.c
-bash-4.2$ ./hw4
```

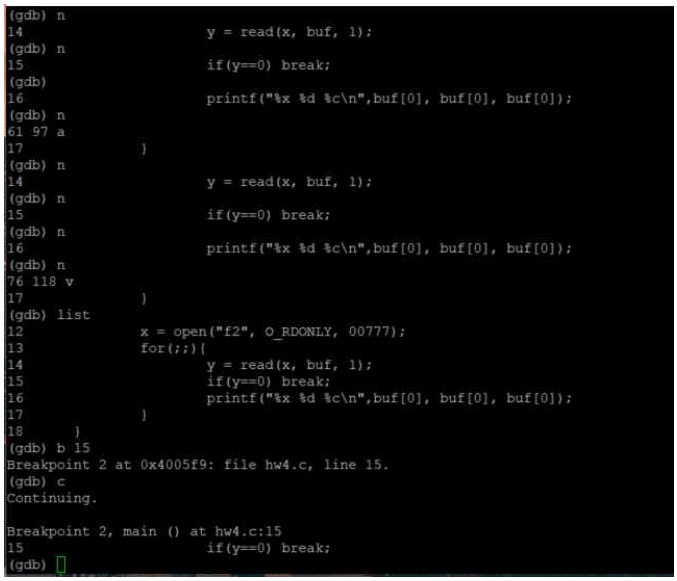


코드를 보면 f2 파일을 읽기모드로 open하고 파일 번호를 x에 return 해준다. 무한 루프에 read 함수를 보면 파일번호 x를 통해 f2 파일로부터 1byte만큼 buf에 저장해주고, 실제 read한 byte수만큼 y에 return 해준다. (모든 문자열을 읽기 전까지 y에 1이 return 됨. 모든 문자열을 읽게되면 y에 0이 return 됨.) 1 byte씩 읽는데, 그 글자의 16진수 형태, 정수(10진수) 형태, 문자 형태를 출력해준다. 이러한 과정을 반복문에 넣어줌으로써 모든 문자열의 16진수 10진수 문자를 출력하고, 모든 byte를 다 읽게 되면 조건문을 통해 반복문을 탈출한다. 컴파일 후 실행해보면 저장된 모든 문자의 16진수와 10진수가 출력됨을 알 수 있다.

5) Compile hw4.c with –g option and run gdb to execute each instruction one by one. Use “p” or “x” to check the value of a variable.

```
(gdb) b main
Breakpoint 1 at 0x4005c4: file hw4.c, line 12.
(gdb) r
Starting program: /home/sp5/12180626/hw4

Breakpoint 1, main () at hw4.c:12
12              x = open("f2", O_RDONLY, 00777);
Missing separate debuginfos, use: debuginfo-install glibc-2.16-34.fc18.x86_64
(gdb) list
7
8       int main(){
9               int x, y;
10              char buf[20];
11
12              x = open("f2", O_RDONLY, 00777);
13              for(;;){
14                      y = read(x, buf, 1);
15                      if(y==0) break;
16                      printf("%x %d %c\n",buf[0], buf[0], buf[0]);
(gdb) n
```





gdb는 변수의 변화를 추척할 수 있는 기능 중 하나이다. b main을 통해 breakpoint를 main 내에서 설정하고 r을 통해 시작한다.

n을 통해 한 줄씩 한 줄씩 실행을 시키고 결과값을 확인한다. list는 코드 전체를 보여주는 명령어이고, p x를 통해 명령어 실행 시점의 x값을 출력한다. 7이 출력된 것으로 보아 f1의 파일 번호가 7번임을 알 수 있다.

p buf[0]을 통해 명령어 실행 시점의 buf[0]값을 출력하면 'h'임을 알 수 있다.

x/4xb buf를 통해 byf에 저장된 값을 4byte만큼 16진수로 알려준다. 0x68이 'h'임을 알 수 있다.

b 15를 통해 breakpoint를 line 15로 설정하고, c를 통해 breakpoint까지 반복한다.

q를 통해 gdb를 탈출한다.

6) Write a program that creates a file and writes “hello there”in it. Use open() and write(). Confirm the result with "cat".

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

int main()
{
        int x;
        x = open("f3", O_RDWR | O_CREAT | O_TRUNC, 00777);
        write(x, "hello there", 11);

        return 0;
}
~
~
```

```
-bash-4.2$ vi ex6.c
-bash-4.2$ g++ -o ex6 ex6.c
-bash-4.2$ ./ex6
-bash-4.2$ cat f3
hello there-bash-4.2$
```

f3를 읽고 쓸 수 있고, f3가 없다면 생성하고, f3에 기존 내용이 있다면 지우는 모드로 open 하고 f3의 파일번호가 x에 저장된다.
이후 파일번호 x를 통해 f3에 “hello there”이라는 문자열을 최대 11byte까지 저장한다.
이처럼 문자열이 저장된 파일뿐만 아니라“hello there”이라는 문자열을 직접 넣어줄 수 있다.
cat f3를 통해 hello there이 저장됨을 확인할 수 있다.

7) Write a program that makes a copy for file "hw4.c" into another file "cphw4.c". Use open(), read(),and write(). Confirm that they are same with "cat" and "ls –l".
문제에서 제시된 알고리즘은 ex2.c에 해당하는 알고리즘 이므로
cp ex2.c hw4.c를 통해 hw4.c에 ex2.c내용을 붙여넣는다.

```
-bash-4.2$ cp hw4.c cphw4.c
```

이후 cp hw4.c cphw4.c를 통해 hw4.c의 내용을 cphw4.c에 붙여넣는다.

```
-bash-4.2$ vi cphw4.c
```

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

int main(){
        int x1, x2, y;
        char buf[20];

        x1 = open("f1", O_RDONLY,00777);
        x2 = open("f2", O_RDWR | O_CREAT | O_TRUNC, 00777);
        for(;;){
                y = read(x1, buf, 20);
                if (y==0) break;
                write(x2, buf, y);
        }

        return 0;
}
~
~
"cphw4.c" 21L, 347C                                   12,2-9        All
```

vi cphw4.c를 통해 복사가 잘 되었는지 확인하였다.

```
-bash-4.2$ ls -l hw4.c
-rw-rw-r-- 1 12180626 12180626 439 Jun 23 11:03 hw4.c
-bash-4.2$ ls -l cphw4.c
-rw-rw-r-- 1 12180626 12180626 439 Jun 23 11:03 cphw4.c
```

ls –l을 통해서도 파일크기 및 파일 이름, 권한 등 모두 일치하는 것을 확인하였다.

8) Write a program that makes a copy for file "hw4" (the executable file for "hw4.c") into another file cphw4. Confirm that they are same with "xxd" and "ls -l".
Execute cphw4 to see if it runs ok.

```
-bash-4.2$ g++ -o cphw4 cphw4.c
-bash-4.2$ ./cphw4
```

```
-bash-4.2$ xxd f2
0000000: 4920 6861 7665 2061 2064 7265 616d 0a74  I have a dream.t
0000010: 6861 7420 6f6e 6520 6461 7920 7468 6973  hat one day this
0000020: 206e 6174 696f 6e0a 7769 6c6c 2072 6973   nation.will ris
0000030: 6520 7570 2061 6e64 0a6c 6976 6520 6f75  e up and.live ou
0000040: 7420 7468 6520 7475 7265 0a6d 6561 6e69  t the ture.meani
0000050: 6e67 206f 6620 6974 7320 6372 6565 640a  ng of its creed.
0000060: 7468 6174 2061 6c6c 206d 656e 2061 7265  that all men are
0000070: 2063 7265 6174 6564 2065 7175 616c 2e0a   created equal..
```

cphw4.c를 컴파일 한 뒤 실행하면 f2에 값이 write된다.
xxd f2를 통해 결과를 확인하면 값이 잘 저장되어있음을 알 수 있다.

```
-bash-4.2$ ls -l hw4.c
-rw-rw-r-- 1 12180626 12180626 439 Jun 23 11:03 hw4.c
-bash-4.2$ ls -l cphw4.c
-rw-rw-r-- 1 12180626 12180626 439 Jun 23 11:03 cphw4.c
```

ls –l을 통해서도 파일크기 및 파일 이름, 권한 등 모두 일치하는 것을 확인하였다.

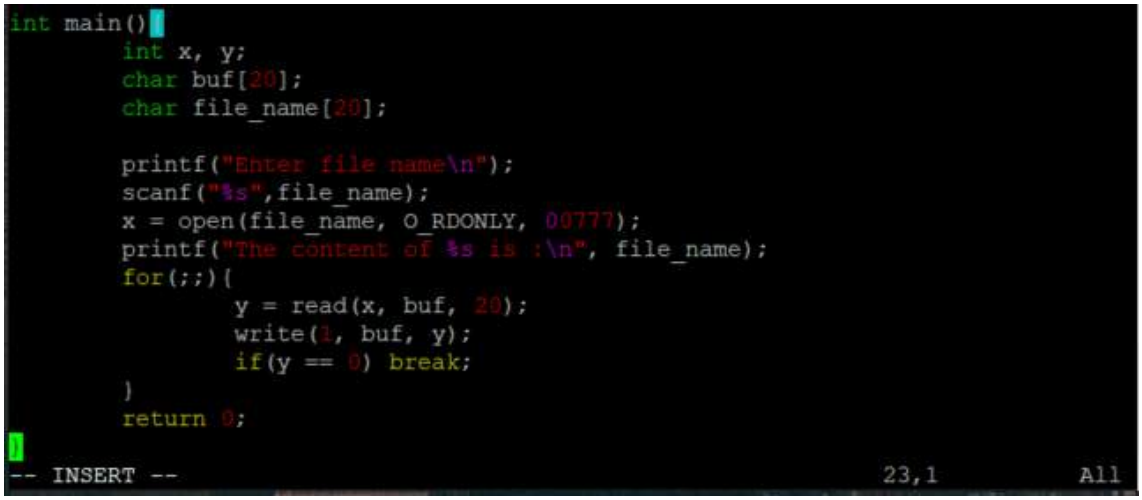
9) Repeat 7). But get the name of the files from the user. Confirm that the result of copy with "cat" and "ls –l".
(#include 목록은 생략하였습니다.)

```
int main(){
        int x1, x2, y;
        char buf[20];
        char src[256];
        char dest[256];

        printf("Enter src file name\n");
        scanf("%s", src);
        printf("Enter dest file name\n");
        scanf("%s", dest);

        x1 = open(src, O_RDONLY, 00777);
        x2 = open(dest, O_RDWR | O_CREAT | O_TRUNC, 00777);

        for(;;){
                y = read(x1, buf, 20);
                if(y==0) break;
                write(x2, buf, y);
        }
        printf("%s is copied into %s successfully", src, dest);
        return 0;
}
                                                        21,0-1        Bot
```

```
-bash-4.2$ vi ex9.c
-bash-4.2$ g++ -o ex9 ex9.c
-bash-4.2$ ./ex9
Enter src file name
hw4.c
Enter dest file name
newhw4.c
hw4.c is copied into newhw4.c successfully-bash-4.2$
```

7)번에서 사용한 코드와 유사하나 이미 생성된 파일의 이름을 지정하는 것이 아닌 사용자로부터 파일이름을 받아서 파일 내용을 read하고 write하는 코드이다.
컴파일 후 실행결과 hw4.c와 newhw4.c를 입력하게 되면 hw4.c의 문자열을 read하고 buf에 저장한 뒤 buf의 값을 newhw4.c에 write한다.

10) Write "mycat" that displays the contents of a user-input file in the terminal in characters. Give a text file and a non-text file to mycat and explain the difference.



```
int main()
        int x, y;
        char buf[20];
        char file_name[20];

        printf("Enter file name\n");
        scanf("%s",file_name);
        x = open(file_name, O_RDONLY, 00777);
        printf("The content of %s is :\n", file_name);
        for(;;){
                y = read(x, buf, 20);
                write(1, buf, y);
                if(y == 0) break;
        }
        return 0;
}
-- INSERT --                                                    23,1          All
```

```
-bash-4.2$ vi mycat.c
-bash-4.2$ g++ -o mycat mycat.c
-bash-4.2$ ./mycat
Enter file name
f1
The content of f1 is :
I have a dream
that one day this nation
will rise up and
live out the ture
meaning of its creed
that all men are created equal.
```

mycat이라는 이름의 파일을 생성하고, 사용자가 직접 파일의 이름을 넣어준다. 이전 파일이름을 두 개 받아서 했던 코드는 코드를 복사하여 붙여넣은 뒤 저장하는 반면에, 이 코드는 파일 하나를 받아서 그 파일의 문자열을 어딘가에 또 저장하지 않고, 1번 파일을 활용해 화면에 출력해주기만 한다는 차이가 있다.

(출력 결과가 길어 일부 생략하였습니다.)

mycat을 실행시켜 hw4의 내용을 확인한 결과, hw4에는 hw4.c를 컴파일하여 변환된 기계어가 들어가있으므로 값이 깨져서 출력되는 것을 확인할 수 있다.

11) Write "myxxd" that displays the contents of a user-input file in the terminal in hexadecimal numbers. Give a text file and a non-text file to myxxd and explain the difference. You need to use printf("%x ", buf[i]) to display a byte in a hexadecimal number. Also declare the buffer as an array of unsigned char. Compare the result from the result of xxd.

```
-bash-4.2$ cp mycat.c myxxd.c
```

10)번의 코드와 유사하므로 cp mycat.c myxxd.c를 활용해 코드를 복사한 후 수정한다.

```
int main(){
        int x, y;
        unsigned char buf[20];
        char file_name[20];

        printf("Enter file name\n");
        scanf("%s",file_name);
        x = open(file_name, O_RDONLY, 00777);
        printf("The content of %s is :\n", file_name);
        for(;;){
                y = read(x, buf, 1);
                printf("%x ", buf[0]);
                if(y == 0) break;
        }
        printf("\n");
        return 0;
"myxxd.c" 24L, 443C                                    10,2-9        Top
```

```
-bash-4.2$ vi myxxd.c
-bash-4.2$ g++ -o myxxd myxxd.c
-bash-4.2$ ./myxxd
Enter file name
f1
The content of f1 is :
49 20 68 61 76 65 20 61 20 64 72 65 61 6d a 74 68 61 74 20 6f 6e 65 20 64 61 79
20 74 68 69 73 20 6e 61 74 69 6f 6e a 77 69 6c 6c 20 72 69 73 65 20 75 70 20 61
6e 64 a 6c 69 76 65 20 6f 75 74 20 74 68 65 20 74 75 72 65 a 6d 65 61 6e 69 6e 6
7 20 6f 66 20 69 74 73 20 63 72 65 65 64 a 74 68 61 74 20 61 6c 6c 20 6d 65 6e 2
0 61 72 65 20 63 72 65 61 74 65 64 20 65 71 75 61 6c 2e a a
```

printf("%x ", buf[i])를 사용하기 위해 read를 1byte씩 진행하고, 읽어낼 때마다 printf를 통해 출력한다. 만약 모든 값을 다 읽어서 read의 return 값이 0이 되면 반복문을 탈출한다.
컴파일 후 실행한 뒤 파일 이름에 f1을 입력하면 f1의 내용이 16진수로 출력됨을 알 수 있다.

```
Enter file name
hw4
The content of hw4 is :
7f 45 4c 46 2 1 1 0 0 0 0 0 0 0 0 0 2 0 3e 0 1 0 0 0 ffffffd0 4 40 0 0 0 0 0 40 0
 0 0 0 0 0 0 28 14 0 0 0 0 0 0 0 0 0 0 40 0 38 0 9 0 40 0 23 0 20 0 6 0 0 0 5 0 0
 0 40 0 0 0 0 0 0 0 40 0 40 0 0 0 0 0 40 0 40 0 0 0 0 0 fffffff8 1 0 0 0 0 0 0 ff
fffff8 1 0 0 0 0 0 0 8 0 0 0 0 0 0 0 3 0 0 0 4 0 0 0 38 2 0 0 0 0 0 0 38 2 40 0 0
 0 0 0 38 2 40 0 0 0 0 0 1c 0 0 0 0 0 0 0 1c 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0
0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 40 0 0 0 0 0 0 0 40 0 0 0 0 0 fffffffc 7 0 0 0 0 0
0 fffffffc 7 0 0 0 0 0 0 0 0 20 0 0 0 0 0 1 0 0 0 6 0 0 0 10 e 0 0 0 0 0 0 10 e 6
0 0 0 0 0 0 10 e 60 0 0 0 0 0 34 2 0 0 0 0 0 0 38 2 0 0 0 0 0 0 0 0 20 0 0 0 0 0
2 0 0 0 6 0 0 0 28 e 0 0 0 0 0 0 28 e 60 0 0 0 0 0 28 e 60 0 0 0 0 0 ffffffd0 1 0
 0 0 0 0 0 ffffffd0 1 0 0 0 0 0 0 8 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 54 2 0 0 0 0 0
0 54 2 40 0 0 0 0 0 54 2 40 0 0 0 0 0 44 0 0 0 0 0 0 0 44 0 0 0 0 0 0 0 4 0 0 0 0
 0 0 0 50 ffffffe5 74 64 4 0 0 0 fffffff0 6 0 0 0 0 0 0 fffffff0 6 40 0 0 0 0 0 f
ffffff0 6 40 0 0 0 0 0 34 0 0 0 0 0 0 0 34 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 51 fffff
fe5 74 64 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 52 ffffffe5 74 64 4 0 0 0 10 e 0 0 0 0 0 0 10 e
60 0 0 0 0 0 10 e 60 0 0 0 0 0 fffffff0 1 0 0 0 0 0 0 fffffff0 1 0 0 0 0 0 0 1 0
0 0 0 0 0 0 2f 6c 69 62 36 34 2f 6c 64 2d 6c 69 6e 75 78 2d 78 38 36 2d 36 34 2e
73 6f 2e 32 0 4 0 0 0 10 0 0 0 1 0 0 0 47 4e 55 0 0 0 0 0 2 0 0 0 6 0 0 0 20 0 0
0 4 0 0 0 14 0 0 0 3 0 0 0 47 4e 55 0 ffffff8f ffffffd8 67 ffffffec 1b ffffffad f
fffffe9 25 fffffffd 24 1 ffffff9f 1e ffffff97 fffffff5 ffffffac 38 6 21 79 1 0 0
0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

myxxd를 실행시켜 hw4의 내용을 확인한 결과, hw4에는 hw4.c를 컴파일하여 변환된 기계어가 들어가있으므로 값이 이상하게 출력되는 것을 확인할 수 있다.

12) Run following code and display f8 with cat and xxd respectively. Explain the results.

```
int main() {
        x = open("f8", O_CREAT | O_RDWR | O_TRUNC, 00777);
        write(x, "ab", 2);
        int y=10;
        write(x, &y, 4);
        write(x, "cd", 2);
        y=235;
        write(x, &y, 4);

        return 0;
}
~
~
```

f8 파일을 열어준 후 write를 사용하여 값을 넣어주었다. 이때 " "사이의 값은 문자열로 인식하여 정상적으로 출력되지만 y=10과 y=235가 출력되지 않는 것을 볼 수 있다.
이로 보아 10과 235에 해당하는 아스키코드의 동작을 한다는 것을 알 수 있었다.
10은 줄 바꿈을 의미하고 235는 Ù 문자를 의미한다. 출력창에선 이미지가 깨져 보이는 것을 볼 수 있다.
xxd를 하여 확인하니 ad와 cd 즉, 문자열만 저장된 것을 확인할 수 있다.

13) Write a program that divides a given file into three small files of roughly equal size. Use fstat() to find out the size of a file.

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

int main()
{
        struct stat st;
        char file_name[20];
        int x, y, z;
        int x1, x2, x3;
        int file_info;
        char file_name1[20];
        char file_name2[20];
        char file_name3[20];
        char buf[20];

        printf("Enter file name to split\n");
        scanf("%s", file_name);

        x = open(file_name, O_RDONLY, 0777);
        y = fstat(x, &st);

        int file_size = st.st_size;
        int div_size = file_size / 3;

        strcpy(file_name1, file_name);
        strcpy(file_name2, file_name);
        strcpy(file_name3, file_name);

        file_name1[strlen(file_name)]='1';
        file_name2[strlen(file_name)]='2';
        file_name3[strlen(file_name)]='3';

        x1 = open(file_name1, O_RDWR | O_CREAT | O_TRUNC, 0777);
        x2 = open(file_name2, O_RDWR | O_CREAT | O_TRUNC, 0777);
        x3 = open(file_name3, O_RDWR | O_CREAT | O_TRUNC, 0777);

        int i;
        for(i = 0; i < div_size; i++){
                z = read(x, buf, 1);
                write(x1, buf, z);
        }

        for(i = 0; i < div_size; i++){
                z = read(x, buf, 1);
                write(x2, buf, z);
        }

        for(i = 0; i < div_size; i++){
                z = read(x, buf, 1);
                if(z == 0) break;
                write(x3, buf, z);
        }

        printf("%s is split into %s, %s, and %s\n", file_name, file_name1, file_name2, file_name3);
        return 0;
}
```

```
-bash-4.2$ vi ex13.c
-bash-4.2$ g++ -o ex13 ex13.c
-bash-4.2$ ./ex13
Enter file name to split
f9
f9 is split into f91, f92, and f93
```

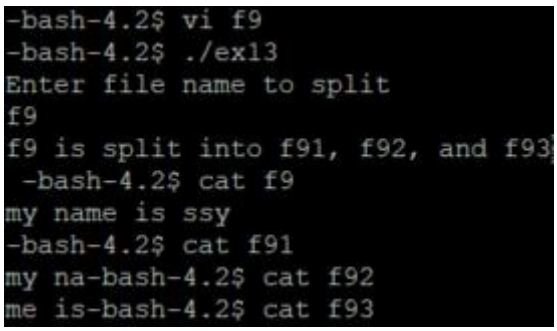
fstat는 파일에 관한 정보를 얻는 함수이다. 먼저 사용자에게 파일 이름을 입력 받는다. 파일 이름을 읽기 전용으로 open하고, 해당 파일 번호를 x에 return한다.

fstat(x, &st)의 return 값을 y에 저장한다. 입력받은 파일 이름을 file_name 1,2,3에 복사하고, 파일 이름 길이번 째 인덱스에 각각 1,2,3 문자를 저장한다.

이후 각각의 파일을 읽고쓰기가 가능하고, 없다면 새로운 파일을 생성하며 내용이 이미 있다면 지우는 모드로 x1, x2, x3를 open한다.

이후 read와 write를 통해 x1, x2, x3에 x값을 넣어준다.

먼저 f9에 my name is ssy라는 14글자 문장을 저장한다. (맨 끝 0포함 15글자)



컴파일 후 실행하면 f9가 f91, f92, f93으로 나누었고,
f91 - my na, f92 – me is가 저장됨을 확인하였다.

14) What is wrong with following program?

```
char temp0[20], *temp1[10], *temp2[10];
printf("enter src file name₩n");
gets(temp0);
temp1[0]=temp0;
printf("enter dest file name₩n");
gets(temp0);
temp2[0]=temp0;
printf("src file:%s dest file:%s₩n", temp1[0], temp2[0]);
```

주어진 코드를 분석해보면 src file과 dest file을 gets를 통해 받고 printf()를 통해 출력할 때, 각각 입력한 값이 출력되어야 하는데 src file과 dest file 모두 dest file에 입력한 값이 출력된다.
temp1과 temp2는 포인터 변수이므로 주소 값을 저장할 수 있다. 그러므로 temp1[0]=temp0; 과 temp2[0]=temp0;을 봤을 때 temp0에 특정 주소값이 저장된다. 즉 temp1과 temp2가 같은 주소 값을 가지게 되고 이는 같은 결과 창을 출력하는 오류를 일으킨다는 것을 알 수 있었다.

15) What is wrong with following program. Find the problem with GDB and fix it.

```
int x, x1, y;
x=open("f1", O_RDONLY, 00777);
x1=open("f2", O_WRONLY|O_CREAT|O_TRUNC,00777);
char buf[512];
int cnt=0;
for(;;){
    y=read(x,buf,1);
    if (y==0) break;
    cnt++;
}
write(x1, buf, cnt);
```

f2의 값을 확인하면 쓰레기값이 들어간 것을 알 수 있다.
gdb를 통해 오류를 찾기 위해 컴파일 시 –g 옵션을 추가한다.

```
(gdb) list
1       #include <fcntl.h>
2       #include <sys/stat.h>
3       #include <sys/types.h>
4       #include <unistd.h>
5       #include <string.h>
6       #include <stdio.h>
7
8       int main(){
9               int x, x1, y;
10              x = open("f1", O_RDONLY, 00777);
(gdb) b main
Breakpoint 1 at 0x400787: file ex3.c, line 10.
(gdb) r
Starting program: /home/sp5/12180626/ex3

Breakpoint 1, main () at ex3.c:10
10              x = open("f1", O_RDONLY, 00777);
Missing separate debuginfos, use: debuginfo-install glibc-2.16-34.fc18.x86_64 libgcc-4.7.2-8.fc18.x86_64 libstdc++-4
.7.2-8.fc18.x86_64
(gdb) n
11              x1 = open("f2", O_WRONLY | O_CREAT | O_TRUNC, 00777);
(gdb) n
14              int cnt = 0;
(gdb) n
16                      y = read(x, buf, 1);
(gdb) n
17                      if(y==0) break;
(gdb) n
18                      cnt++;
(gdb) n
15              for(;;){
(gdb) n
16                      y = read(x, buf, 1);
(gdb) n
17                      if(y==0) break;
(gdb) n
18                      cnt++;
(gdb) n
15              for(;;){
(gdb) p buf[0]
$1 = 32 ' '
(gdb) n
16                      y = read(x, buf, 1);
(gdb) n
17                      if(y==0) break;
(gdb) n
18                      cnt++;
(gdb) n
15              for(;;){
(gdb) p buf[1]
$2 = 0 '\000'
```

list를 통해 코드를 확인한 뒤 main함수에서 break point를 걸어준 후 r로 프로그램을 시작하여 주었다. 이후 n으로 한 줄씩 코드를 실행시켰다. 무한루프로 들어가기까지 오류가 없었다. 무한루프에서 p로 buf[0]의 값을 찍어보니 처음 반복 때는 I가 잘 들어간 것을 볼 수 있었다.

하지만 두 번째 반복 때 buf[0]에 ' ' 뛰어쓰기가 들어가 있었고 buf[1]엔 0, 즉 아무것도 저장되지 않은 것을 확인했다. 혹시나 싶은 마음에 세 번째 반복한 후 buf[0]를 찍어보았지만 역시나 다음 문자인 h가 저장되어 있었고 buf[1]에는 아무것도 저장되어 있지 않았다. 이로써 buf[0]의 주소에만 값을 저장하고 있어 출력 시에 나머지 인덱스에서는 쓰레기 값을 출력하는 것을 확인할 수 있었다.

```
int main(){
        int x, x1, y;
        x = open("f1", O_RDONLY, 00777);
        x1 = open("f2", O_WRONLY | O_CREAT | O_TRUNC, 00777);

        char buf[512];
        int cnt = 0;
        for(;;){
                y = read(x, buf+cnt, 1);
                if(y==0) break;
                cnt++;
        }
        write(x1, buf, cnt);
}
```

이를 수정하기 위해 무한루프에서 read 부분 두 번째 argument를 buf+cnt로 변경한다.

```
-bash-4.2$ vi ex3.c
-bash-4.2$ g++ -o ex3 ex3.c
-bash-4.2$ ./ex3
-bash-4.2$ cat f2
I have a dream
that one day this nation
will rise up and
live out the ture
meaning of its creed
that all men are created equal.
```

다시 실행했을 때 f2에 복사가 잘 되었음을 확인하였다.