lect8 HW

시스템프로그래밍 1분반 12180626 성시열

0) Set up bash startup file: copy .bashrc and .bash_profile into your login directory

$ pwd

/home/sp11/12181234 -- make sure you are in the login directory

$ (use linuxer2 instead of linuxer1 in 152 server)

$ (use linuxer2 instead of linuxer1 in 152 server)

```
-bash-4.2$ pwd
/home/sp5/12180626
-bash-4.2$ cp ../../linuxer1/.bashrc .
-bash-4.2$ cp ../../linuxer1/.bash_profile .
```

로그인 디렉토리에서 cp ../../linuxer1/.bashrc .와 cp ../../linuxer1/.bash_profile .를 통해 bash start up file을 세팅한다.

이후 putty 창을 닫고 새로운 putty 창을 열면

[12180626@linuxer1 ~]$ 형태 이후 명령어를 실행시킬 수 있으며 컴파일 된 파일을 실행시킬 때 "./" 생략할 수 있다.

```
[12180626@linuxer1 ~]$ ex1
97 a 97 a
hello 0x40064d hello 0x40064d
```

./ex1 대신 ex1으로 ex1 파일을 실행시킬 수 있음을 확인하였다.

1) Try following program which doesn't receive command line arguments.

```
#include <stdio.h>

void main(){
        printf("hi\n");
}
~
~
```

```
[12180626@linuxer1 ~]$ vi ex0.c
[12180626@linuxer1 ~]$ gcc -o ex0 ex0.c
[12180626@linuxer1 ~]$ ex0
hi
```

ex0.c 파일에 c 프로그램을 작성하기 위해 vi 편집기를 통해 접속 후 코드를 작성하고 파일을 컴파일하였다.

ex0를 통해 파일을 실행시키면 c 프로그램이 잘 작동됨을 확인할 수 있다.

2) Try following program that receives one command line argument.

```
void main(int argc, char * argv[]){
        printf("hi\n");
        printf("%d\n", argc);
        printf("%s\n", argv[0]);
}
~
~
```

```
[12180626@linuxer1 ~]$ vi ex1.c
[12180626@linuxer1 ~]$ gcc -o ex1 ex1.c
[12180626@linuxer1 ~]$ ex1
hi
1
ex1
```

Command Line Argument를 사용한 코드를 ex1.c에 작성한다. main 함수의 parameter 자리에 int argc, char *argv[]를 추가하여 Command Line Argument의 개수와 해당 문자열이 포함된 char형 포인터 배열을 사용할 수 있다.

printf("hi₩n");
printf("%d₩n", argc);
printf("%s₩n", argv[0]);

를 통해 Argument의 개수와 0번째 Argument가 무엇인지 알 수 있다.
실제 컴파일 후 실행하면 1개의 아규먼트와 0번째 ex1이라는 아규먼트가 존재한다는 것을 알 수 있다. 여기서 0번째 아규먼트는 항상 실행되는 프로그램의 이름이다.

3) Try following program that receives two command line arguments.

```
void main(int argc, char *argv[]){
        printf("hi\n");
        printf("%d\n", argc);
        printf("%s\n", argv[0]);
        printf("%s\n", argv[1]);
}
~
~
```

```
[12180626@linuxer1 ~]$ cp ex1.c ex2.c
[12180626@linuxer1 ~]$ vi ex2.c
[12180626@linuxer1 ~]$ gcc -o ex2 ex2.c
[12180626@linuxer1 ~]$ ex2
hi
1
ex2
Segmentation fault (core dumped)
```

ex2.c 파일을 작성하기 앞서 ex1.c 파일과 흡사한 파일이므로 cp ex1.c ex2.c를 통해 ex1.c 파일을 복사하여 ex2.c 파일에 붙여넣는다.
이후 vi 편집기를 통해 코드를 수정한다.
ex2를 통해 실행을 하면 argv[0]까지는 0번째 파라미터인 실행 파일 이름이 잘 출력된다.
하지만 argv[1]에 해당하는 첫 번째 아규먼트를 입력해주지 않았으므로 Segmentation fault (core dumped) 에러가 발생한다. 이는 printf("%s₩n", argv[1]); 코드가 있는데 argv[1]에 아무런 값이 할당되지 않아 발생한 오류이다.

```
[12180626@linuxer1 ~]$ ex2 hello
hi
2
ex2
hello
```

ex2 hello를 통해 아규먼트를 추가하였다. argv[1]에 hello라는 문자열의 주소가 저장되고, printf("%s₩n", argv[1]);를 통해 "hello"라는 문자열을 출력한다.

```
[12180626@linuxer1 ~]$ ex2 hello uzbek tuit
hi
4
ex2
hello
```

ex2 hello uzbek tuit 는 총 4개의 아규먼트가 사용되었지만 코드에서 printf는 argv[0], argv[1]만 하였으므로 뒤에 uzbek과 tuit은 무시된다.

4) A program that receives three command line arguments.

```
void main(int argc, char *argv[])
        printf("hi\n");
        printf("%d\n", argc);
        printf("%s\n", argv[0]);
        printf("%s\n", argv[1]);
        printf("%s\n", argv[2]);
}
~
~
```

```
[12180626@linuxer1 ~]$ cp ex2.c ex3.c
[12180626@linuxer1 ~]$ vi ex3.c
[12180626@linuxer1 ~]$ gcc -o ex3 ex3.c
[12180626@linuxer1 ~]$ ex3 hello there
hi
3
ex3
hello
there
```

3)번과 같이 cp를 통해 ex2.c파일을 ex3.c에 붙여넣고 argv[2]를 출력하는 코드를 추가한다. 컴파일 후 ex3 hello there을 통해 아규먼트 개수와 3가지 아규먼트를 모두 출력하였다.

5) A program that receives any number of arguments.

```
void main(int argc, char *argv[])
        int i;
        printf("argc is %d\n", argc);
        for(i=0; i<argc; i++){
                printf("argv[%d] is %s\n", i, argv[i]);
        }
}
~
~
```

```
[12180626@linuxer1 ~]$ vi ex4.c
[12180626@linuxer1 ~]$ gcc -o ex4 ex4.c
[12180626@linuxer1 ~]$ ex4 x1 x2 x3 x4
argc is 5
argv[0] is ex4
argv[1] is x1
argv[2] is x2
argv[3] is x3
argv[4] is x4
```

vi ex4.c를 통해 ex4.c 코드를 작성한다. 코드를 보면 아규먼트 몇 개를 작성하든 for 반복문을 통해 작성된 아규먼트 만큼 출력을 해주는 코드를 작성하였다.
컴파일 후 ex4 x1 x2 x3 x4를 통해 실행하면 argv의 인덱스별로 어떤 문자열이 저장되어 있는지 알 수 있다.

6) Try following and explain the difference from echo command.

```
void main(int argc, char *argv[])
        int i;

        for(i=1; i<argc; i++){
                printf("%s ", argv[i]);
        }
        printf("\n");
}
~
~
```

```
[12180626@linuxer1 ~]$ cp ex4.c myecho.c
[12180626@linuxer1 ~]$ vi myecho.c
[12180626@linuxer1 ~]$ gcc -o myecho myecho.c
[12180626@linuxer1 ~]$ myecho hello
hello
[12180626@linuxer1 ~]$ echo hello
hello
[12180626@linuxer1 ~]$ myecho hello hi bye
hello hi bye
[12180626@linuxer1 ~]$ echo hello hi bye
hello hi bye
[12180626@linuxer1 ~]$ echo hi > f1
[12180626@linuxer1 ~]$ cat f1
hi
[12180626@linuxer1 ~]$ myecho hi > f2
[12180626@linuxer1 ~]$ cat f2
hi
```

myecho.c를 작성하기 전 ex4.c 파일과 유사하여 cp를 통해 복사 붙여넣기를 하였다. 이후 vi 편집기를 통해 코드를 작성하고 컴파일하였다. 코드는 argv[0], 즉 실행하는 파일이 들어간 아규먼트를 제외한 모든 아규먼트를 순차적으로 출력하는 코드이다. 이는 리눅스에서 기본으로 제공하는 echo 명령어와 같은 역할을 한다. 비교하여 아규먼트가 두 개일 때, 네 개일 때 실행시켜본 결과 같은 결과값이 출력되었고, echo 명령어 실행 파일을 f1에 저장하고, myecho hi를 실행한 값을 f2에 저장하여 cat f1, cat f2를 통해 비교했을 때 같은 결과값이 출력되는 것을 확인하였다.

7) Try following and explain the difference from cat command.

```
void main(int argc, char *argv[]){
        int x, y;
        char buf[20];

        x = open(argv[1], O_RDONLY, 00777);
        if (x==-1){
                perror("error in open");
                exit(1);
        }
        for(;;){
                y = read(x, buf, 20);
                if (y==0) break;
                write(1, buf, y);
        }
}
~
```

코드를 보면 argv[1] 즉, 코드를 실행시킬 때 실행 프로그램 바로 뒤에 나오는 파일을 open 하고 그 파일의 번호를 x에 return 한다.
if(x==-1){ ~ } 조건문은 파일이 없거나 하는 오류 상황에 대한 처리를 작성하였다.
무한 루프를 통해 argv[1]에 저장된 파일을 20byte씩 읽어와 buf에 저장하고 실제 읽은 byte 만큼 y에 저장한다. 이후 1번 파일(화면에 출력)에 buf 값을 y(실제 읽은 byte)만큼 write 한다. 무한 루프에서 더 이상 읽을 값이 없어 y==0일 때 반복문을 탈출한다.

```
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.
~
~
```

컴파일 후 값을 실행시키기 전 vi f1 을 통해 해당 문자열을 저장한다.

```
[12180626@linuxer1 ~]$ cat f1
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.
[12180626@linuxer1 ~]$ mycat f1
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.
```

cat f1 과 mycat f1 을 비교해본 결과 같은 기능을 하는 프로그램임을 알 수 있다.

```
[12180626@linuxer1 ~]$ cat f23
cat: f23: No such file or directory
[12180626@linuxer1 ~]$ mycat f23
error in open: No such file or directory
```

존재하지 않는 파일을 cat 과 mycat 을 통해 확인해보았을 때도 둘 다 오류가 발생하는 것을 알 수 있다.

8) Try following: mycat2.c. Use functions for your program.

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

void display_content(int x); //function prototype

void main(int argc, char *argv[]){
        int x;

        x = open(argv[1], O_RDONLY, 00777);
        if (x==-1){
                perror("error in open");
                exit(1);
        }
        display_content(x);
}

void display_content(int x){
        char buf[20];
        int y;
        for(;;){
                y = read(x, buf, 20);
                if (y==0) break;
                write(1, buf, y);
        }

}
```

```
[12180626@linuxer1 ~]$ cp mycat.c mycat2.c
[12180626@linuxer1 ~]$ vi mycat2.c
[12180626@linuxer1 ~]$ gcc -o mycat2 mycat2.c
[12180626@linuxer1 ~]$ mycat2 f1
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.
```

mycat.c 에서 read 하고 write 하는 부분을 display_content 라는 함수를 만들어서 코드를 작성하였다.

mycat2 f1 을 통해 mycat 과 같은 기능을 수행하는 코드임을 확인하였다.

9) Try following: mycat3.c.

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

void show_file(char *fname);
void display_content(int x); //function prototype

void main(int argc, char *argv[]){
        show_file(argv[1]);
}

void show_file(char *fname){
        int x;

        x = open(fname, O_RDONLY, 00777);
        if (x==-1){
                perror("error in open");
                exit(1);
        }
        display_content(x);
}

void display_content(int x){
        char buf[20];
        int y;
        for(;;){
                y = read(x, buf, 20);
                if (y==0) break;
                write(1, buf, y);
        }

}
```

```
[12180626@linuxer1 ~]$ vi mycat3.c
[12180626@linuxer1 ~]$ gcc -o mycat3 mycat3.c
[12180626@linuxer1 ~]$ mycat3 f1
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.
```

8)에서 작성한 mycat2 에서 main 에 있는 값을 한 번 더 함수로 묶어준 코드이다. mycat3.c 코드를 작성하고 컴파일하면 위와 같은 결과값이 출력됨을 알 수 있다.

10) You can debug programs with command line arguments as follows. Debug mycat3.c with gdb. To pass command line arguments to gdb, do "set args arg1 arg2

```
[12180626@linuxer1 ~]$ gdb mycat3
GNU gdb (GDB) Fedora (7.5.1-38.fc18)
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/sp5/12180626/mycat3...done.
(gdb) b main
Breakpoint 1 at 0x40065b: file mycat3.c, line 13.
(gdb) set args f1 f2 f3
(gdb) r
Starting program: /home/sp5/12180626/mycat3 f1 f2 f3

Breakpoint 1, main (argc=4, argv=0x7fffffffe548) at mycat3.c:13
13              show_file(argv[1]);
Missing separate debuginfos, use: debuginfo-install glibc-2.16-34.fc18.x86_64
(gdb) s
show_file (fname=0x7fffffffe7d8 "f1") at mycat3.c:19
19              x = open(fname, O_RDONLY, 00777);
(gdb) n
20              if (x==-1){
(gdb) n
24              display_content(x);
(gdb) n
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.
```

mycat3.c 의 main에서 argv[1]만 사용하는 것을 보면 set args f1 f2 f3를 했을 때, f1파일만 사용되는 것을 알 수 있다.

gdb를 통해 확인하였다.

gdb) b main : Breakpoint를 main 함수 내에 설정하였다.

gdb) set args f1 f2 f3 : argv 배열에 각각 "args", "f1", "f2", "f3"을 setting 한다.

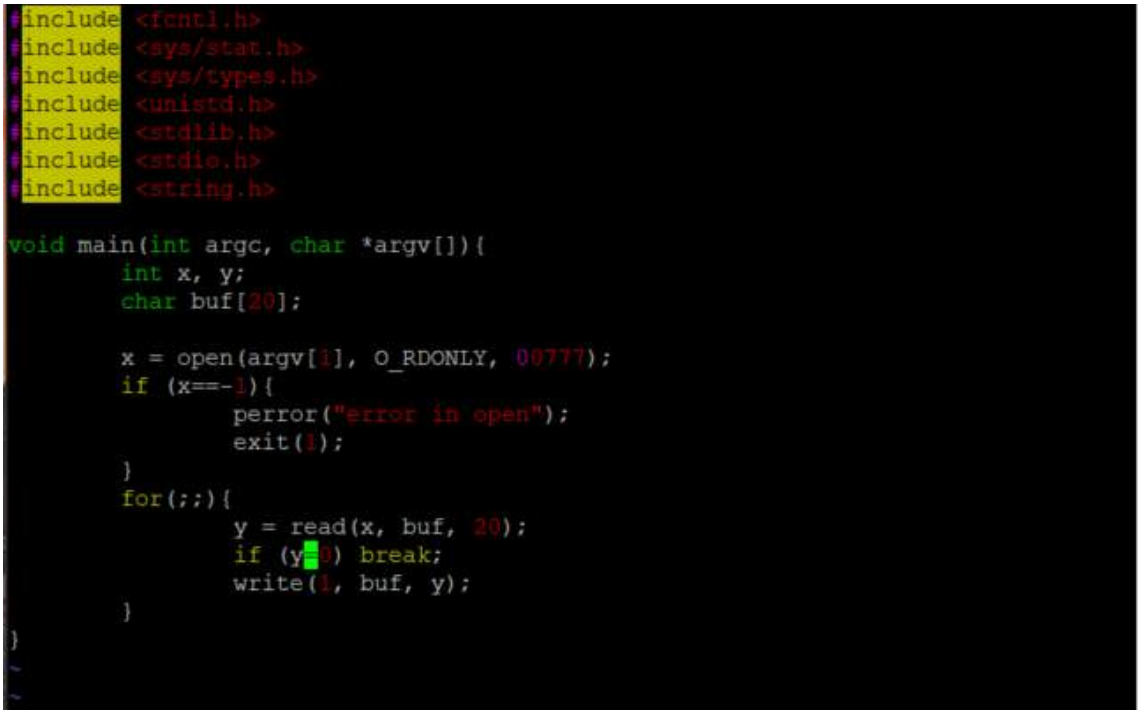
gdb) r : 파일을 실행시킨다.

gdb) s : (step) 현재 행 수행 후 정지한다. 함수 호출 시 함수 안으로 들어간다.

gdb) n : (next) 현재 행 수행 후 정지하나, 함수 호출시 함수 수행 다음 행으로 간다.

s 명령어를 통해 main 함수 첫 번째 행을 실행하였으나 main 함수에 한 줄의 함수밖에 없으므로 함수 내부로 들어가 실행하고 정지한다. 이후 n 명령어를 통해 show_file 함수의 실행문을 한 줄 씩 실행해본 결과 display_content(x) 까지 실행을 하였고, 그 결과 f1에 저장된 문자열이 출력됨을 확인하였다.

10-1) Following program falls into infinite loop when run: ex1 f1. Debug it with gdb.



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

void main(int argc, char *argv[]){
        int x, y;
        char buf[20];

        x = open(argv[1], O_RDONLY, 00777);
        if (x==-1){
                perror("error in open");
                exit(1);
        }
        for(;;){
                y = read(x, buf, 20);
                if (y==0) break;
                write(1, buf, y);
        }
}
~
~
```

vi 편집기를 통해 ex1.c 의 코드를 작성하고 컴파일한다. 무한 루프 내의 if 문 조건을 y=0 으로 하여 break 문에 도달하지 못하게 한다.

```
[12180626@linuxer1 ~]$ vi ex1.c
[12180626@linuxer1 ~]$ gcc -g -o ex1 ex1.c
[12180626@linuxer1 ~]$ ex1 f1
^C
[12180626@linuxer1 ~]$
```

ex1 f1을 통해 실행시킨 결과 무한루프에 빠지게 되어 아무런 출력값도 나오지 않고 파일이 계속 실행되었다. ctrl + c를 통해 무한 루프에서 강제로 탈출하였다.

```
[12180626@linuxer1 ~]$ gdb ex1
GNU gdb (GDB) Fedora (7.5.1-38.fc18)
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/sp5/12180626/ex1...done.
(gdb) b main
Breakpoint 1 at 0x40065b: file ex1.c, line 13.
(gdb) set args f1
(gdb) r
Starting program: /home/sp5/12180626/ex1 f1

Breakpoint 1, main (argc=2, argv=0x7fffffffe568) at ex1.c:13
13              x = open(argv[1], O_RDONLY, 00777);
Missing separate debuginfos, use: debuginfo-install glibc-2.16-34.fc18.x86_64
(gdb) n
14              if (x==-1){
(gdb) n
19                      y = read(x, buf, 20);
(gdb) n
20                      if (y=0) break;
(gdb) n
21                      write(1, buf, y);
(gdb) n
22              }
(gdb) n
19                      y = read(x, buf, 20);
(gdb) n
20                      if (y=0) break;
(gdb) n
21                      write(1, buf, y);
(gdb) n
22              }
```

이후 gdb ex1을 통해 디버깅한 결과 무한루프 내부의 if문에 도달하지 못하고 계속 반복하여 무한루프가 발생함을 알 수 있다.

```
[12180626@linuxer1 ~]$ vi ex1.c
[12180626@linuxer1 ~]$ gcc -g -o ex1 ex1.c
[12180626@linuxer1 ~]$ ex1 f1
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.
```

이를 해결하기 위해 ex1.c 파일의 무한루프 내부 if문을 수정한다. (y = 0 → y == 0)
이후 ex1 f1을 통해 파일을 실행하면 정상적으로 작동함을 확인할 수 있다.

11) Modify mycat.c such that it can handle two input files.

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

void main(int argc, char *argv[]){
        int x, y;
        char buf[20];

        int i;
        for(i=1; i<3; i++){
                x = open(argv[i], O_RDONLY, 00777);
                if (x==-1){
                        perror("error in open");
                        exit(1);
                }
                for(;;){
                        y = read(x, buf, 20);
                        if (y==0) break;
                        write(1, buf, y);
                }
                printf("\n\n");
        }
}
```

```
[12180626@linuxer1 ~]$ vi mycat.c
[12180626@linuxer1 ~]$ gcc -o mycat mycat.c
[12180626@linuxer1 ~]$ mycat f1
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.


error in open: Bad address
[12180626@linuxer1 ~]$ mycat f1 f2
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.


hi
```

기존 작성한 mycat 코드를 그대로 하되, 3개의 아규먼트까지 출력되도록 하는 코드를 작성한다. 아규먼트가 3개일 때는 코드가 잘 작동되지만 2개일 때는 한 번의 빈 반복문이 형성되어 에러가 발생함을 알 수 있다.

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

void main(int argc, char *argv[]){
        int x, y;
        char buf[20];

        int i;
        for(i=1; i<3; i++){
                x = open(argv[i], O_RDONLY, 00777);
                if (x==-1){
                        perror("error in open");
                        exit(1);
                }
                for(;;){
                        y = read(x, buf, 20);
                        if (y==0) break;
                        write(1, buf, y);
                }

                if(argc != 3) break;

                printf("\n\n");
        }
}
```

이러한 에러를 방지하기 위해 반복문 하나를 추가하였다. 파라미터 개수가 3개가 아닐 때는 반복문을 실행하고 바로 반복문을 빠져나올 수 있도록 했다. 이러한 방식은 매우 복잡하면서도 편협된 상황에만 적용된다.

```
[12180626@linuxer1 ~]$ vi mycat.c
[12180626@linuxer1 ~]$ gcc -o mycat mycat.c
[12180626@linuxer1 ~]$ mycat f1
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.
[12180626@linuxer1 ~]$ mycat f1 f2
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.


hi
```

실행하면 아규먼트가 2개인 상황과 3개인 상황에서는 원하는대로 출력됨을 알 수 있다.

12) Modify mycat such that it can handle any number of files.

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

void main(int argc, char *argv[])
        int x, y;
        char buf[20];

        int i;
        for(i=1; i<argc; i++){
                x = open(argv[i], O_RDONLY, 00777);
                if (x==-1){
                        perror("error in open");
                        exit(1);
                }
                for(;;){
                        y = read(x, buf, 20);
                        if (y==0) break;
                        write(1, buf, y);
                }
                printf("\n\n");
        }
}
~
```

11)번의 비효율적인 코드를 수정하여 아규먼트가 몇 개이든 모든 아규먼트 파일의 내용을 출력하는 코드를 출력하는 반복문을 완성하였다.
i 를 1 부터 아규먼트 개수 –1 만큼 반복하고
argv[i]의 파일을 open 하고 read 하여 모든 내용을 출력한다.
그렇게 되면 아규먼트의 개수가 4 개일 때 i 는 1 부터 3 까지 반복하고
argv [1], [2], [3]의 파일을 open 하고 read 하는 것을 알 수 있다.

```
[12180626@linuxer1 ~]$ vi mycat.c
[12180626@linuxer1 ~]$ gcc -o mycat mycat.c
[12180626@linuxer1 ~]$ mycat f1 f2 f3
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.


hi


hello there

[12180626@linuxer1 ~]$ mycat f1 f2 f3 f4
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.


hi


hello there

hello
```

실제로 컴파일하여 실행시켜보면 아규먼트가 몇 개이든 모든 파일의 내용이 잘 출력됨을 알 수 있다.

13) Implement mycp that works similarly to "cp".

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

void main(int argc, char *argv[]){
        int x1, x2;
        char buf[20];
        int y;

        x1 = open(argv[1], O_RDONLY, 00777);
        x2 = open(argv[2], O_RDWR, 00777);

        if (x1==-1 || x2 == -1){
                perror("error in open");
                exit(1);
        }
        for(;;){
                y = read(x1, buf, 20);
                if (y==0) break;
                write(x2, buf, y);
        }
}
```

```
[12180626@linuxer1 ~]$ vi mycp.c
[12180626@linuxer1 ~]$ gcc -o mycp mycp.c
[12180626@linuxer1 ~]$ mycp f1 f2
[12180626@linuxer1 ~]$ cat f1
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.
[12180626@linuxer1 ~]$ cat f2
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.
```

argv[1]의 파일의 내용을 복사하여 argv[2]파일에 붙여넣는 코드를 작성하였다. 이전의 mycat 은 1 번 파일(화면에 출력)에 write 하였다면 이번에는 argv[2] 파일에 write 해야하기 때문에 x2 라는 변수를 추가하여 argv[2] 파일 번호를 저장하고 해당 파일에 값을 저장하여 값을 복사할 수 있었다.

14) Implement myxxd that works similarly to "xxd". Run "myxxd mycat.c". Compare the result with "xxd mycat.c".

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

void main(int argc, char *argv[]){
        int x, y;
        char buf[20];

        x = open(argv[1], O_RDONLY, 00777);
        if (x==-1){
                perror("error in open");
                exit(1);
        }
        for(;;){
                y = read(x, buf, 1);
                if (y==0) break;
                printf("%x ", buf[0]);
        }
}
~
~
```

```
[12180626@linuxer1 ~]$ cp mycat.c myxxd.c
[12180626@linuxer1 ~]$ vi myxxd.c
[12180626@linuxer1 ~]$ vi f1
[12180626@linuxer1 ~]$ vi fabc
[12180626@linuxer1 ~]$ cat fabc
abc
[12180626@linuxer1 ~]$ xxd fabc
0000000: 6162 630a                                abc.
```

```
[12180626@linuxer1 ~]$ myxxd fabc
61 62 63 a [12180626@linuxer1 ~]$ vi myxxd.c
```

fabc라는 파일에 abc라는 문자열을 저장하였다.
read를 통해 x(argv[1] 파일의 번호)의 문자열을 1byte씩 buf에 저장하고
buf[0]에 저장된 값을 printf("%x ",buf[0])을 통해 16진수로 출력한다.
이를 반복하여 더 이상 출력할 값이 없을 때 y는 0이 되고, 반복문을 탈출한다.
xxd fabc를 보면 6162 630a 가 출력됨을 확인할 수 있는데 아스키코드로 볼 때 a는 61, b는 62, c는 63이고 문자열이므로 끝에 0a가 붙음을 알 수 있다.
myxxd도 컴파일 후 값을 출력하면 61 62 63 a가 출력됨을 확인할 수 있다.

xxd mycat.c와 myxxd mycat.c를 통해 좀 더 명확한 비교를 해보자

```
[12180626@linuxer1 ~]$ xxd mycat.c
0000000: 2369 6e63 6c75 6465 203c 6663 6e74 6c2e  #include <fcntl.
0000010: 683e 0a23 696e 636c 7564 6520 3c73 7973  h>.#include <sys
0000020: 2f73 7461 742e 683e 0a23 696e 636c 7564  /stat.h>.#includ
0000030: 6520 3c73 7973 2f74 7970 6573 2e68 3e0a  e <sys/types.h>.
0000040: 2369 6e63 6c75 6465 203c 756e 6973 7464  #include <unistd
0000050: 2e68 3e0a 2369 6e63 6c75 6465 203c 7374  .h>.#include <st
0000060: 646c 6962 2e68 3e0a 2369 6e63 6c75 6465  dlib.h>.#include
0000070: 203c 7374 6469 6f2e 683e 0a23 696e 636c   <stdio.h>.#incl
0000080: 7564 6520 3c73 7472 696e 672e 683e 0a0a  ude <string.h>..
0000090: 766f 6964 206d 6169 6e28 696e 7420 6172  void main(int ar
00000a0: 6763 2c20 6368 6172 202a 6172 6776 5b5d  gc, char *argv[]
00000b0: 297b 0a09 696e 7420 782c 2079 3b0a 0963  ){..int x, y;..c
00000c0: 6861 7220 6275 665b 3230 5d3b 0a09 0a09  har buf[20];....
00000d0: 696e 7420 693b 0a09 666f 7228 693d 313b  int i;..for(i=1;
00000e0: 2069 3c61 7267 633b 2069 2b2b 297b 0a09   i<argc; i++){..
00000f0: 0978 203d 206f 7065 6e28 6172 6776 5b69  .x = open(argv[i
0000100: 5d2c 204f 5f52 444f 4e4c 592c 2030 3037  ], O_RDONLY, 007
0000110: 3737 293b 0a09 0969 6620 2878 3d3d 2d31  77);...if (x==-1
0000120: 297b 0a09 0909 7065 7272 6f72 2822 6572  ){....perror("er
0000130: 726f 7220 696e 206f 7065 6e22 293b 0a09  ror in open");..
0000140: 0909 6578 6974 2831 293b 0a09 097d 0a09  ..exit(1);...}..
0000150: 0966 6f72 283b 3b29 7b0a 0909 0979 203d  .for(;;){....y =
0000160: 2072 6561 6428 782c 2062 7566 2c20 3230   read(x, buf, 20
0000170: 293b 0a09 0909 6966 2028 793d 3d30 2920  );....if (y==0)
0000180: 6272 6561 6b3b 0a09 0909 7772 6974 6528  break;....write(
0000190: 312c 2062 7566 2c20 7929 3b0a 0909 7d0a  1, buf, y);...}.
00001a0: 0909 7072 696e 7466 2822 5c6e 5c6e 2229  ..printf("\n\n")
00001b0: 3b0a 097d 0a7d 0a                        ;..}.}.
```

xxd mycat은 줄바꿈마다 0a가 들어가고 숫자 4개(2byte)마다 공백이 있다. 또한, 실제 사용된 문자열이 무엇인지 우측에 표현된다.

```
[12180626@linuxer1 ~]$ myxxd mycat.c
23 69 6e 63 6c 75 64 65 20 3c 66 63 6e 74 6c 2e 68 3e a 23 69 6e 63 6c 75 64 65
20 3c 73 79 73 2f 73 74 61 74 2e 68 3e a 23 69 6e 63 6c 75 64 65 20 3c 73 79 73
2f 74 79 70 65 73 2e 68 3e a 23 69 6e 63 6c 75 64 65 20 3c 75 6e 69 73 74 64 2e
68 3e a 23 69 6e 63 6c 75 64 65 20 3c 73 74 64 6c 69 62 2e 68 3e a 23 69 6e 63 6
c 75 64 65 20 3c 73 74 64 69 6f 2e 68 3e a 23 69 6e 63 6c 75 64 65 20 3c 73 74 7
2 69 6e 67 2e 68 3e a a 76 6f 69 64 20 6d 61 69 6e 28 69 6e 74 20 61 72 67 63 2c
 20 63 68 61 72 20 2a 61 72 67 76 5b 5d 29 7b a 9 69 6e 74 20 78 2c 20 79 3b a 9
 63 68 61 72 20 62 75 66 5b 32 30 5d 3b a 9 a 9 69 6e 74 20 69 3b a 9 66 6f 72 2
8 69 3d 31 3b 20 69 3c 61 72 67 63 3b 20 69 2b 2b 29 7b a 9 9 78 20 3d 20 6f 70
65 6e 28 61 72 67 76 5b 69 5d 2c 20 4f 5f 52 44 4f 4e 4c 59 2c 20 30 30 37 37 37
 29 3b a 9 9 69 66 20 28 78 3d 3d 2d 31 29 7b a 9 9 9 70 65 72 72 6f 72 28 22 65
 72 72 6f 72 20 69 6e 20 6f 70 65 6e 22 29 3b a 9 9 9 65 78 69 74 28 31 29 3b a
9 9 7d a 9 9 66 6f 72 28 3b 3b 29 7b a 9 9 9 79 20 3d 20 72 65 61 64 28 78 2c 20
 62 75 66 2c 20 32 30 29 3b a 9 9 9 69 66 20 28 79 3d 3d 30 29 20 62 72 65 61 6b
 3b a 9 9 9 77 72 69 74 65 28 31 2c 20 62 75 66 2c 20 79 29 3b a 9 9 7d a 9 9 70
 72 69 6e 74 66 28 22 5c 6e 5c 6e 22 29 3b a 9 7d a 7d a [12180626@linuxer1 ~]$
```

myxxd는 줄마꿈마다 a가 들어가고 숫자 2개(1byte)마다 공백이 존재한다. 두자리 숫자에서도 앞이 0이면 0이 생략된다. 실제 문자열이 어떻게 표현되어있는지 확인할 수 없다.

15) Modify mycat to handle various options. The second argument is either a file or an option. If it is a file, just display the contents. If it is an option (starting with '-'), perform the following corresponding actions.

옵션을 직접 만들어보는 문제이다.

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

void cp(char *second, char *third);
void hx(char *second);
void pw(char *second);
void dir(char *second);

void main(int argc, char *argv[]){
        if(strcmp(argv[1],"-c")==0) cp(argv[2], argv[3]);
        else if(strcmp(argv[1],"-x")==0) hx(argv[2]);
        else if(strcmp(argv[1],"-p")==0) pw(argv[2]);
        else if(strcmp(argv[1],"-d")==0) dir(argv[2]);
}
```

(mycat.c 함수가 매우 길어 함수별로 첨부하였습니다.)

```
[12180626@linuxer1 ~]$ vi mycat.c
[12180626@linuxer1 ~]$ gcc -o mycat mycat.c
```

코드를 모두 작성한 뒤 컴파일하여 각 함수별로 실행시켜보았다.

```c
void cp(char *second, char *third){
        int x1, x2;
        char buf[20];
        int y;

        x1 = open(second, O_RDONLY, 00777);
        x2 = open(third, O_RDWR | O_CREAT | O_TRUNC, 00777);

        if (x1==-1 || x2 == -1){
                perror("error in open");
                exit(1);
        }

        for(;;){
                y = read(x1, buf, 20);
                if (y==0) break;
                write(x2, buf, y);
        }
}
```

```
[12180626@linuxer1 ~]$ mycat -o f1 flout
[12180626@linuxer1 ~]$ cat f1
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.
[12180626@linuxer1 ~]$ cat flout
I have a dream
that one day
this nation will rise up.
live out the true meaning of its creed.
```

앞에서 사용했던 복사 붙여넣기 코드를 활용하여 cp 함수를 만들었다.
실행시킨 뒤 cat f1, cat f2를 통해 값이 복사됨을 확인하였다.

```c
void hx(char *second){
        int x, y;
        char buf[20];

        x = open(second, O_RDONLY, 00777);

        if (x==-1){
                perror("error in open");
                exit(1);
        }

        for(;;){
                y = read(x, buf, 1);
                if (y==0) break;
                printf("%x ", buf[0]);
        }
}
```

```
[12180626@linuxer1 ~]$ mycat -x f1
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
a 74 68 69 73 20 6e 61 74 69 6f 6e 20 77 69 6c 6c 20 72 69 73 65 20 75 70 2e a 6
c 69 76 65 20 6f 75 74 20 74 68 65 20 74 72 75 65 20 6d 65 61 6e 69 6e 67 20 6f
66 20 69 74 73 20 63 72 65 65 64 2e a [12180626@linuxer1 ~]$
```

앞에서 사용했던 16진수 출력 코드를 활용하여 hx 함수를 만들었다.
실행시키면 f1에 저장된 문자열이 16진수로 출력됨을 확인하였다.

```c
void pw(char *second){
        FILE *F;
        F = fopen(second, "r");
        char buf[100];
        char *tokens[7];

        if(F == NULL) printf("file can't read\n");

        for(;;){
                if(fgets(buf, sizeof(buf), F)==NULL) break;
                fgets(buf, sizeof(buf), F);
                tokens[0] = strtok(buf, ":");

                int i;
                for(i=1; i<7; i++) tokens[i] = strtok(NULL, ":");
                printf("id : %s  ", tokens[0]);
                printf("password : %s  ", tokens[1]);
                printf("uid : %s  ", tokens[2]);
                printf("gid : %s  ", tokens[3]);
                printf("desc : %s  ", tokens[4]);
                printf("home dir : %s  ", tokens[5]);
                printf("shell : %s  \n", tokens[6]);
        }
}
```

```
[12180626@linuxer1 ~]$ mycat -p /etc/passwd
id : bin  password : x  uid : 1  gid : 1  desc : bin  home dir : /bin  shell : /sbin/nologin

id : adm  password : x  uid : 3  gid : 4  desc : adm  home dir : /var/adm  shell : /sbin/nolo
gin

id : sync  password : x  uid : 5  gid : 0  desc : sync  home dir : /sbin  shell : /bin/sync

id : halt  password : x  uid : 7  gid : 0  desc : halt  home dir : /sbin  shell : /sbin/halt

id : uucp  password : x  uid : 10  gid : 14  desc : uucp  home dir : /var/spool/uucp  shell :
 /sbin/nologin
```

```
id : 12180626  password : x  uid : 1232  gid : 1232  desc : Student Account  home dir : /home
/sp5/12180626  shell : /bin/bash

id : 12181735  password : x  uid : 1234  gid : 1234  desc : Student Account  home dir : /home
/sp31/12181735  shell : /bin/bash

id : 12181740  password : x  uid : 1236  gid : 1236  desc : Student Account  home dir : /home
/sp31/12181740  shell : /bin/bash

id : 12181746  password : x  uid : 1238  gid : 1238  desc : Student Account  home dir : /home
/sp31/12181746  shell : /bin/bash
```

pw 함수는 아규먼트로 받은 파일을 fopen을 통해 읽기모드로 open하고 F에 파일을 저장한다. 받은 파일이 NULL값일 때는 경고문을 출력하는 조건문도 작성하였다.
fgets를 통해 F에 저장된 파일에서 buf 사이즈만큼 buf에 저장하고, 이 값이 NULL 값일 때 무한 루프를 탈출한다.
strtok를 통해 buf에 저장된 값을 공백 단위로 끊어서 tokens[]에 저장한다.
이후 형식에 맞게 출력한다.
실행 결과 다양한 파일들의 정보를 파악할 수 있었다.

```
void dir(char *second){
        DIR* dir = opendir(second);
        struct dirent *x;

        while((x = readdir(dir)) != NULL) printf("%s\n",x->d_name);

        closedir(dir);
}
```

```
[12180626@linuxer1 ~]$ mycat -d d1
f2
..
f1
.
[12180626@linuxer1 ~]$ ls d1
f1  f2
```

dir 함수는 opendir를 활용하여 디렉토리를 open하고 x->d_name을 통해 받아온 디렉토리 하위에 있는 디렉토리의 이름을 출력해준다.

opendir를 통해 open한 directory는 closedir를 통해 닫아준다.

mycat –d d1을 통해 실행시킨 뒤 ls d1을 통해 값을 확인하면 d1 디렉토리 내부에 f2, f1 디렉토리가 존재함을 확인할 수 있다.