

# Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)

[gcccompil3r@gmail.com](mailto:gcccompil3r@gmail.com)

학생 – hoseong Lee(이호성)

[hslee00001@naver.com](mailto:hslee00001@naver.com)



파일 I/O 제어,

프로세스 제어,

멀티 태스킹과 컨텍스트 스위칭,

signal 활용법,

IPC 기법

**핵심철학 : 모든것은 파일이다.**

**System call :** 유일한 소프트웨어 인터럽트, User 가 Kernel 에게 요청하는 작업을 의미한다.  
open 은 숫자를 retrun 한다.

## EX1- file\_io2.c

```
#include <stdlib.h>
#include <fcntl.h>
#include <stdio.h>
```

```
#define ERROR -1
```

```
int main(void)
{
    int filedес;
    char pathname[]="temp.txt";
    if((filedes=open(pathname,O_CREAT | O_RDWR | 0644)) == ERROR)
        // 파일이 있다면 파일을 만들지마라.
        // if((filedes=open(pathname,O_CREAT | O_RDWR ,0644)) == ERROR)
        // 파일을 읽고 쓸 수 있게 pathname 명으로 만들어라
    {
        printf("File Open Error!\n");
        exit(1);
    }
    printf("fd=%d\n",filedes);

    close(filedes);

    return 0;
}
```

## EX2- file\_io3.c

```
#include <fcntl.h>

int main(void)
{
    int filed1, filed2;

    filed1 = open("data1.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    //O_TRUNC : 무조건 한번 밀어버린다.(작업을 할때마다 새로 갱신함) 임시저장시 사용함.
    filed2 = creat("data2.txt", 0644);

    close(filed1);
    close(filed2);

    return 0;
}
```

## EX3- file\_io4.c 숫자값이 리턴된다.

```
#include <unistd.h>
#include <fcntl.h>

int main(void)
{
    int fdin, fdout;
    ssize_t nread;
    char buf[1024];

    fdin = open("temp1.txt", O_RDONLY);
    // 읽기전용, 읽을게 없으면 안열림.
```

```

fdout = open("temp2.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
// 실행할때마다 안에있는거 밀어버림.

while((nread = read(fdin, buf, 1024)) > 0)
// read(fd, buf, 읽을크기) : fd 를 읽어와서 buf 에 읽을 크기만큼 집어넣음, 자기가 읽을 byte 크기 리턴.
// → 읽을게 없으면 -값 리턴됨
// system call, read 로 fdin(인덱스)에 있는 값을 읽어와서 buf 에 집어넣어줌
{
    if(write(fdout, buf, nread) < nread)
    // write(fd, buf, 쓸 크기) : buf 에 있는 값이 nread 크기만큼 fd 에 써짐. 자기가 쓸 byte 크기 리턴
    // nread 만큼 썼으므로 nread 보다 작을 수 없음.

    {
        close(fdin);
        close(fdout);
    }
}

close(fdin); // 마찬가지로 system call 임.
close(fdout);

return 0;
}

```

→ cp 를 만들. 파일복사..

```

files_struct → files → f_pos, : 위치를 저장 , 이중 포인터로 관리됨(즉 포인터배열이다.) // files *fd[] 배열의 인덱스
                file_operations
                inode → path → super_block

```

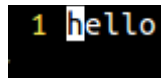
## EX4- file\_io5.c 파일의 용량 검사

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

int main(void)
{
    int filedес;
    off_t newpos;

    filedес = open("data1.txt",O_RDONLY);
    newpos = lseek(filedes, (off_t)0, SEEK_END); // 파일용량 검사
    printf("file size: %d\n",newpos);
}
```

data1.txt

A screenshot of a terminal window showing the content of a file named data1.txt. The text '1 hello' is displayed in a monospaced font, with the '1' in yellow and 'hello' in white on a black background.

결과 : 

## EX5- 파일 CP 코드 짜보기 → argc, argv

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    int fdin,fdout;
    ssize_t nread;
    char buf[1024];

    if(argc!=3)
    {
        printf("인자 입력 3 개 하라고!\n");
        exit(-1);
    }
    for(i=0;i<argc;i++)
        printf("당신이 입력한 인자는 = %s\n",argv[i]);

    fdin = open(argv[1],O_RDONLY);
    fdout = open(argv[2],O_WRONLY | O_CREAT | O_TRUNC , 0644);

    while((nread = read(fdin,buf, 1024))>0)
    {
        if(write(fdout,buf,nread) < nread)
        {
            close(fdin);
            close(fdout);
        }
    }

    close(fdin);
```

a

<pre>close(fdout); return 0; }</pre>	
--------------------------------------	--

argc : 프로그램을 실행할 때, 지정해 준 “명령행 옵션”의 “개수”가 저장되는 곳.

argv : 프로그램을 실행할 때, 지정해 준 “명령행 옵션의 문자열들”이 실제로 저장되는 배열  
→ (`**argv == *argv[]`)

결과 :

```
koitt@koitt-Z20NH-AS51B5U:~/my_proj/Homework/sanghoonlee/lec/lhs/linux_system$ ./debug file_cp.c file_cp2.c
당신이 입력한 인자는 = ./debug
당신이 입력한 인자는 = file_cp.c
당신이 입력한 인자는 = file_cp2.c
koitt@koitt-Z20NH-AS51B5U:~/my_proj/Homework/sanghoonlee/lec/lhs/linux_system$ ls
data1.txt  debug      file_cp.c  file_io3.c  file_io5.c  temp2.txt  XDG_VTNR=7
data2.txt  file_cp2.c file_io2.c  file_io4.c  temp1.txt   temp.txt
```



## System call 을 사용하지 않는 것과 사용하는 것의 속도차이??

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

```
int main(void)
```

```
{
```

```
    FILE *fp = fopen("mycat.c","r");
```

```
    char buf[1024] = "\0";
```

```
    int ret;
```

```
    while(ret=fread(buf,1,sizeof(buf),fp))
```

```
    {
```

```
        usleep(1000000);
```

```
        fwrite(buf,1,ret,stdout);
```

```
    }
```

```
    fclose(fp); return 0;
```

```
}
```

```
// O_RDONLY 와 같은뜻. 시스템콜 아님.
```

```
//1byte 씩 1024 바이트를 읽어라.
```

```
//
```

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
```

```
int main(int argc, char **argv)
{
    int fd, ret;
    char buf[1024];
    if(argc != 2)
    {
        printf("Usage: mycat filename\n");
        exit(-1);
    }
    fd = open(argv[1], O_RDONLY);
    while(ret = read(fd, buf, sizeof(buf)))
    {
        write(1, buf, ret);
    }
    close(fd);
    return 0;
}
```

시스템 콜을 사용하면 더빠른 속도로 파일을 열고 닫고 쓸 수 있다!!

## Read 시 키보드 지정해서 파일안에 쓰기

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
```

```
int main(void)
{
    int ret;
    char buf[1024]={0};
    int fd;
    fd = open("c.txt",O_CREAT|O_RDWR|0644);
    ret = read(0,buf,sizeof(buf));
    write(fd,buf,ret);

    close(fd);
    return 0;
}
```

Read( 0, ~,~ ) 키보드는 0 , 모니터는 1

## Read 시 키보드 지정해서 파일안에 쓰기

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include "my_scanf.h"
5
6 int main(void)
7 {
8     int nr;
9     char buf[1024]={0};
10
11     nr= my_scanf(buf,sizeof(buf));
12     printf("nr=%d\n",nr);
13     write(1,buf,nr);
14     return 0;
15 }
16
```

```
1 #include "my_scanf.h"
2
3 int my_scanf(char *buf, int size)
4 {
5     int nr = read(0,buf,size);
6     return nr;
7 }
```

write(1,~) : 모니터에 쓰기

```
1 #ifndef __MY_SCANF_H__ // 선언된 것이 있으면 이 헤더파일을 들어가지마라.
2 #define __MY_SCANF_H__ // 선언된 것이 없으면 my_scanf_h 가 0 으로 땀. (그냥상수임!)
3
4 #include <fcntl.h>
5 #include <unistd.h>
6
7 int my_scanf(char *, int);
8
9 #endif // 두 번 선언하는 것을 방지한다.
10
11
12 // 분할되어있어 분간이쉬움
```

알고리즘 파트, 보드제어파트 , 순수한 소프트웨어파트, 영상처리 파트가 있을 때, 한공간에 묶어 놓으면 안된다.  
분할되어있어 분간하기 쉬움.

:

< > 시스템 헤더

“ “ 사용자 정의(커스텀) 헤더

## FILE 을 읽어서 라인,단어 갯수 확인.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

int main(int argc, char **argv)
{
    int fd= open(argv[1],O_RDONLY);
    int line =0;
    int word = 0;
    int flag=0;
    int cnt=0;
    char ch;
    if(argc!=2)
    {
        printf("You need1 mor parameter\n");
        printf("Usage:mywc filename \n");
        exit(-1);
    }

    if((fd=open(argv[1],O_RDONLY))<0) // 0 작으면 에러
    {
        perror("open()");
        exit(-1);
    }
    while(read(fd,&ch,1))
```

숫자값 리턴, 0: 표준입력 1:표준출력 2:표준에러 3~5

```
{
    cnt++;
    if(ch == '\n')
        line++;
    if(ch!='\n' && ch!='\t' && ch != ' ')
    {
        if(flag==0)
        {
            word++;
            flag=1;
        }
    }
    else
    {
        flag=0;
    }
}
close(fd);
printf("%d %d %d %s\n",line,word,cnt,argv[1]);
return 0;
}
```

```
wget https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.4.tar.gz
tar zxvf linux-4.4.tar.gz
```

/kernel/linux-4.4/include/linux/sched.h → :set hlsearch /구조체

```
:vs
:sp
:e 파일명
```

컨트롤 +w w

오답노트 기터브 x 다음주월요일 , 메일 ,gmail.com  
오늘학습내용정리