

# 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

## [1] 디렉토리 포인터 DIR \*

파일 → FILE 이듯, 디렉토리 → DIR 이다. 디렉토리의 많은 정보를 저장할 수 있는 구조체 필드로 구성된다. 포인터선언을 해준다.

#include <dirent.h> : struct dirent : 디렉토리 파일들의 정보 구조체

## [2] 디렉토리를 열어주는 opendir(), 닫아주는 closedir() 내부함수

파일과 마찬가지로 디렉토리를 사용하기 위해서는 open 과 close 되어야 한다.

DIR\* opendir (const char\* name); → name (경로) 를 열어서 입력을 받는다. 성공하면 DIR \*을 반환, 실패시 NULL 반환

int closedir(DIR\* dp);

```
#include <sys/types.h>
```

```
#include <dirent.h>
```

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

```
int main(void)
```

```
{
```

```
    DIR *dp;
```

```
    int i = 0;
```

```
    struct dirent *p;
```

```
    dp = opendir(".");
```

// 열고자하는 디렉토리 경로를 입력으로 받고 성공하면 DIR 구조체에 대한 포인터를 반환 한다. 아님 NULL

// “.” 현재 디렉토리를 입력 받는다.

```
    while(p=readdir(dp))
```

// ( struct dirent \*) 하나하나씩 반환 된다. 즉 디렉토리에 있는 리스트가 넘어온다.

```
    {
```

```
        if(p->d_name[0] == '.')
```

```
            continue;
```

```
        printf("%-16s ",p->d_name);
```

```
        if((i+1)%5==0)
```

```
        printf("\n");
    i++;
}
printf("\n");
closedir(dp);
return 0;
}
```

실행: touch a.txt b.txt c.txt  
ls -al

# getopt 함수

03/22 - ls\_al.c

옵션을 분석할 수 있게 제공하는 시스템 호출이 getopt 함수이다. getopt 함수에 첫 번째와 두 번째 인자는 main 함수의 argc 와 argv 를 그대로 전달하고 세 번째 option 에 제공하고자 하는 옵션을 전달한다. 만약 'a', 'b' 옵션을 전달하고자 한다면 "ab"이라고 전달한다. 옵션 뒤에 인자를 사용해야 한다면 ':'을 추가한다. 만약 'a'에는 옵션 뒤에 인자를 사용하고, 'l'에는 옵션 뒤에 인자를 사용하지 않는다면 "a:l"을 전달한다.(? 이말은 뭐냐면, ./a.out -a file1.c 이런식으로 뒤에 인자가 온다는 말) 마지막으로 getopt 함수는 옵션의 아스키코드 값을 반환한다. 옵션 뒤에 인자를 사용하고자 한다면 이미 선언한 optarg 를 이용한다. 그리고 더 이상 옵션을 발견하지 못하면 -1 를 반환.

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

int main(int argc, char **argv){
    int cmd;
    while((cmd = getopt(argc, argv, "ab"))>0){ // argc 인자의수, argv 옵션요소, -로 시작하는 문자는 옵션 문자가 됨.
        switch(cmd){ // while 문 한번에 문자가 순서대로 하나씩 반환되기 때문에, 끝날때까지 돈다~
            case 'a':
                printf("a option\n");
                break; // getopt 함수는
            case 'b': // 즉 이 함수를 통해 명령어: ls -al 이라는 것도 ls -a -l 로도 쓸 수 있고, ls -l -a 로도 사용 될 수 있다.
                printf("b option\n"); // 이러한 옵션을 쉽게 처리할 수 있도록 도와주는 함수인 것!
                break;

            default:
                printf("unknown option\n");
                break;}}
    return 0;}
```

실행

```
./a.out -a
./a.out --b a
./a.out --b
./a.out - abc
```

```
lsh@lsh-NH:~/my_proj/github/Homework/sanghoonlee/lec/lsh/linux_system/3_22$ ./a.out --b a
./a.out: invalid option -- '-'
unknown option
b option
lsh@lsh-NH:~/my_proj/github/Homework/sanghoonlee/lec/lsh/linux_system/3_22$ ./a.out --b
./a.out: invalid option -- '-'
unknown option
b option
lsh@lsh-NH:~/my_proj/github/Homework/sanghoonlee/lec/lsh/linux_system/3_22$ ./a.out -abc
a option
b option
./a.out: invalid option -- 'c'
unknown option
```

# dirent

03/22 - ls\_dirent.c

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

int main(int argc, char **argv)
{
    DIR *dp;
    int i = 0, cmd;
    struct dirent *p;
    cmd = getopt(argc, argv, "a"); // -a 로 a 가 걸린다면 cmd 에 a 가 들어감!
    dp = opendir("."); // 현재 디렉토리를 받아온다. → 현재 리스트를 받아온다. 구조체에..
    while(p = readdir(dp)) // 이것도 dp 가 while 문이 돌때마다 하나씩 리스트가 반환되서 읽힌다!
    {
        if(cmd != 'a') // ./a.out -a (이런식으로) -a 가 아니라 -ba, -bc, -bb 이런식으로 옵션인자를 넣었다면
        {
            if(p->d_name[0] != '.')
                continue;
        }
        printf("%-16s", p->d_name); // → d_name[0] == '.'
        if((i+1)%5 == 0) // 5 개마다 줄바꿈
            printf("\n");
        i++;
    }
    printf("\n");
    closedir(dp);
    return 0;
}
```

./a.out -a : 숨긴 파일이든 아니든 모두 보여줌 a option

# 비트연산의 이점

03/22 - get\_opt.c

```
#include <stdio.h>
#include <unistd.h>
/*
#define A    (1 << 0)
#define B    (1 << 1)
#define C    (1 << 2)
#define D    (1 << 3)
#define E    (1 << 4)
#define F    (1 << 5)
#define G    (1 << 6)
*/
void check_flag(int flag)
{
    int i, tmp = flag;

    for(i = 0; i < 7; i++)
    {
        switch(tmp & (1 << i))
        {
            case 1:
                printf("A\n");
                break;
            case 2:
                printf("B\n");
                break;
            case 4:
                printf("C\n");
                break;
            case 8:
                printf("D\n");
```

```

        break;
    case 16:
        printf("E\n");
        break;
    case 32:
        printf("F\n");
        break;
    case 64:
        printf("G\n");
        break;
    }
}

int main(int argc, char **argv)
{
    int cmd;
    int flag;

    /* 7 개의 옵션: a, b, c, d, e, f, g */
    while((cmd = getopt(argc, argv, "abcdefg")) > 0)
    {
        switch(cmd)
        {
            case 'a':
                flag |= 1 << 0;
                printf("a option\n");
                break;
            case 'b':
                flag |= 1 << 1;
                printf("b option\n");
                break;
            case 'c':
                flag |= 1 << 2;
                printf("c option\n");
                break;
            case 'd':

```



```
        flag |= 1 << 3;
        printf("d option\n");
        break;
    case 'e':
        flag |= 1 << 4;
        printf("e option\n");
        break;
    case 'f':
        flag |= 1 << 5;
        printf("f option\n");
        break;
    case 'g':
        flag |= 1 << 6;
        printf("g option\n");
        break;
    default:
        printf("unknown option\n");
    }
}
check_flag(flag);

return 0;
}
```

Flag 를 받아서 temp 비트연산을 함으로 즉시 해석되어 빠른 속도를 보여준다.

# ls 옵션 분할

03/22 - get\_opts2.c

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

int main(int argc, char **argv)
{
    DIR *dp;
    int i = 0, cmd;
    struct dirent *p;
    int flag = 0;

    while((cmd=getopt(argc,argv,"alRi"))>0)
    {
        switch(cmd)
        {
            case 'a':
                flag |= 1;
                break;
            case 'b':
                flag |= 2;
                break;
            case 'R':
                flag |= 4;
                break;
            case 'i':
                flag |= 8;
                break;
        }
    }
}
```

```
dp=opendir(".");
while(p= readdir(dp))
{
    if(!(flag & 1))
    {
        if(p->d_name[0]!='.')
            continue;
        printf("%-16s ",p->d_name);
    }
    if((i+1)%5==0)
        printf("\n");
    i++;
}
printf("\n");
closedir(dp);
return 0;
}
```

dp 를 통해 현재 디렉토리를 읽어온다. while 문 (p= dp 를 하나씩 읽어온다. ) if 문 a 만 뺀다. --> continue; 건너뛰기(true)

# 파일 상태 확인

03/22 - ls\_module5.c

## 어떤 상태의 파일인지 확인, 파일타입 확인

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

int main(int argc, char **argv)
{
    struct stat buf;
    char ch;
    stat(argv[1], &buf);

    if(S_ISDIR(buf.st_mode))    // d, p, i, s, c, b --> 파일타입확인.
        ch='d';               // directory 이세요?
    if(S_ISREG(buf.st_mode))   // reg(일반) 파일이세요?
        ch='-';
    if(S_ISFIFO(buf.st_mode))  // 파이프 파일이세요?
        ch='P';
    if(S_ISLNK(buf.st_mode))   // 바로가기 파일이세요?
        ch='l';
    if(S_ISSOCK(buf.st_mode))  // 소켓파일이세요?
        ch='s';
    if(S_ISCHR(buf.st_mode))   // 캐릭터 디바이스세요?
        ch='c';
    if(S_ISBLK(buf.st_mode))   // block device 세요?
        ch='b';
    printf("%c\n", ch);
    return 0;
}
```

# 파일권한설정

03/22 - ls\_module6.c

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

int main(int argc, char **argv)
{
    struct stat buf;
    char ch;
    char perm[11] = "....."; // 맨앞 4 비트가 파일 종류가된다. 4 9 13
    char rwx[4] = "rwx";      //파일권한설정할때 rwx 를 사용한다.
    int i;

    stat(argv[1],&buf);

    if(S_ISDIR(buf.st_mode)) // directory
        ch='d';
    if(S_ISREG(buf.st_mode)) // reg(일반)
        ch='-';
    if(S_ISFIFO(buf.st_mode)) // 파이프 파일
        ch='P';
    if(S_ISLNK(buf.st_mode)) // 바로가기 파일
        ch='l';
    if(S_ISSOCK(buf.st_mode)) // 소켓 파일
        ch='s';
    if(S_ISCHR(buf.st_mode)) // 캐릭터 디바이스
        ch='c';
    if(S_ISBLK(buf.st_mode)) // block 디바이스
        ch='b';

    for(i=0;i<9;i++)
```

```

{
    if(buf.st_mode >> (8-i) & 1)
        perm[i+1] = rwx[i%3];
    }

    printf("%s\n",perm);
    return 0;
}

```

drwxr-xr-x 2 root root 4096 Apr 22 16:59 conory

파일 Type **퍼미션**정보 **링크수** **소유자** **소유그룹** **용량** **생성날짜** **파일이름**

파일 Type	"d" : 디렉토리 , "l" : 링크파일 (바로가기), "-" : 일반파일 , "P" : fifo 파일, "s" : 소켓 파일, "c" : character device
<b>퍼미션정보</b>	해당 파일에 어떠한 퍼미션이 부여되어있는 지 표시
<b>링크수</b>	해당 파일이 링크된 수. 링크는 윈도우의 "바로가기"와 같다. "ln [대상파일] [링크파일]" 명령으로 링크파일을 만듦.
<b>소유자</b>	해당 파일의 소유자이름. (누구것인지)
<b>소유그룹</b>	해당 파일을 소유한 그룹이름. 특별한 변경이 없을 경우 소유자가 속한 그룹이 소유그룹으로 지정된다.

#### \* 퍼미션의 사용자지정

- 소유자 : 소유자에 대한 퍼미션지정
- 그룹 : 소유그룹에 대한 퍼미션지정
- 공개 : 모든사용자들에 대한 퍼미션지정

[1 비트 파일의 종류, 3 비트, 3비트, 3비트]  
[- :파일 ---: 사용자권한 ---:그룹권한 ---: 제 3 자]

- 읽기(r): 파일의 읽기권한 : 4

- 쓰기(w): 파일의 쓰기권한 : 2

- 실행(x): 파일의 실행권한 : 1

chmod [변경될 퍼미션값] [변경할 파일]

**chmod 755 a.txt** → 명령어 실행하면 **a.txt** 파일이 **755**에 해당하는 퍼미션으로 변경된다.

**chmod -R 777 a** → 명령어 실행하면 **a** 디렉토리의 하위에 위치한 모든 파일이 퍼미션 **777**로 변경된다.

# 파일 갯수 확인

03/22 - ls\_module7.c

**선생님말씀 : 디스크 분리되어 현재 디스크의 값이 나왔는데, 이해해라**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include <pwd.h>
#include <grp.h>

int main(int argc, char **argv)
{
    struct stat buf;
    struct passwd *pw;
    struct group *gr;
    char ch;
    char perm[11] = "-----";
    char rwx[4] = "rwx";
    int i;

    stat(argv[1], &buf);

    if(S_ISDIR(buf.st_mode)) // directory
        perm[0] = 'd';
    if(S_ISREG(buf.st_mode)) // regular 파일
        perm[0] = '-';
    if(S_ISFIFO(buf.st_mode)) // 파이프 파일
        perm[0] = 'P';
    if(S_ISLNK(buf.st_mode)) // 바로가기 파일
        perm[0] = 'l';
    if(S_ISSOCK(buf.st_mode)) // 소켓파일?
```



```
    perm[0]='s';
if(S_ISCHR(buf.st_mode))    // 캐릭터디바이스
    perm[0]='c';
if(S_ISBLK(buf.st_mode))    // block
    perm[0]='b';

for(i=0;i<9;i++)
{
    if(buf.st_mode >> (8-i) & 1)
        perm[i+1] = rwx[i%3];
}

printf("%s\n",perm);
printf("%lu ",buf.st_nlink);
pw=getpwuid(buf.st_uid);
printf("%s",pw->pw_name);
gr=getgrgid(buf.st_gid);
printf("%s",gr->gr_name);
return 0;
}
```

## Sticky bit 란?

03/22 - ls\_module8.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
```

```
int main(int argc, char **argv)
{
    struct stat buf;
    struct dirent *p;
    struct passwd *pw;
    struct group *gr;
    struct tm *tm;
    char ch;
    char perm[11] = "-----";
    char rwx[4] = "rwx";
```

```

char sst[4] = "sst";
int i;
stat(argv[1],&buf);
if(S_ISDIR(buf.st_mode)) // directory
    perm[0]='d';
if(S_ISREG(buf.st_mode)) // regular 일반파일
    perm[0]='-';
if(S_ISFIFO(buf.st_mode)) // 파이프
    perm[0]='P';
if(S_ISLNK(buf.st_mode)) // 바로가기 파일
    perm[0]='l';
if(S_ISSOCK(buf.st_mode)) // 소켓파일
    perm[0]='s';
if(S_ISCHR(buf.st_mode)) // 캐릭터디바이스
    perm[0]='c';
if(S_ISBLK(buf.st_mode)) // block 디바이스
    perm[0]='b';

for(i=0;i<9;i++)
{
    if(buf.st_mode >> (8-i) & 1)
        perm[i+1] = rwx[i%3];
}

for(i=0;i<3;i++)
{
    if((buf.st_mode >> (11-i)) & 1)
        if(perm[(i+1)*3] == '-')
            perm[(i+1)*3] = sst[i] ^ 0x20;
        else
            perm[(i+1)*3] = sst[i];
}
printf("%s",perm);
printf("%lu",buf.st_nlink);
pw = getpwuid(buf.st_uid);
printf("%s",pw->pw_name);
gr = getgrgid(buf.st_gid);

```

```

printf("%s",gr->gr_name);
printf("%lu",buf.st_size);
tm=localtime(&(buf.st_mtime));
    printf("%d-%02d-%02d %02d:%02d",
        tm->tm_year + 1900, tm->tm_mon + 1, tm-> tm_mday, tm-> tm_hour, tm-> tm_min);
printf("\n");
return 0;
}

```

```

chmod 4644 a.txt
ls -l a.txt
ls
chomd 4755 a.out
ls -l a.out

```

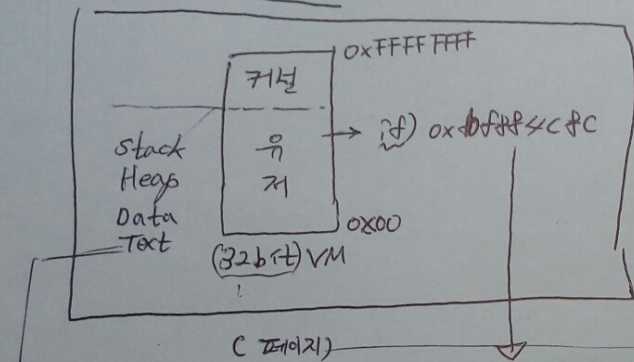
```

chmod 2755 a.out

```

**drwxrwxrwt** [디렉토리에 스테키 비트\(t\)가 붙으면 공유 파일이된다.](https://photo.cloud.naver.com/v2/#)

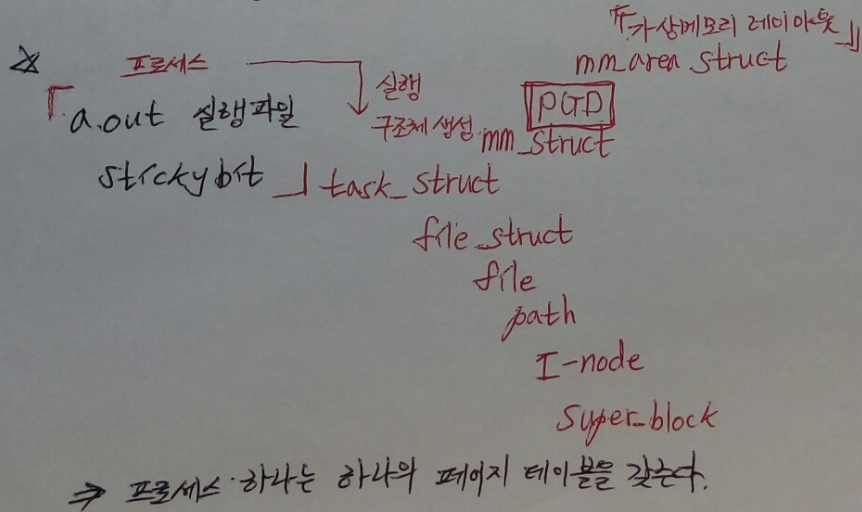
파일에 t 가 붙어있으면, dram 에 올라갈 때 제어 -> 스왑 (하드디스크 -> 물리메모리)  
 가상메모리 백업해봤자 필요있나? 물리메모리 백업해야함. 그러니까 결국 가상메모리를 물리메모리로 변환 하는 과정을 해야함. 그과정을 페이징 이라한다.



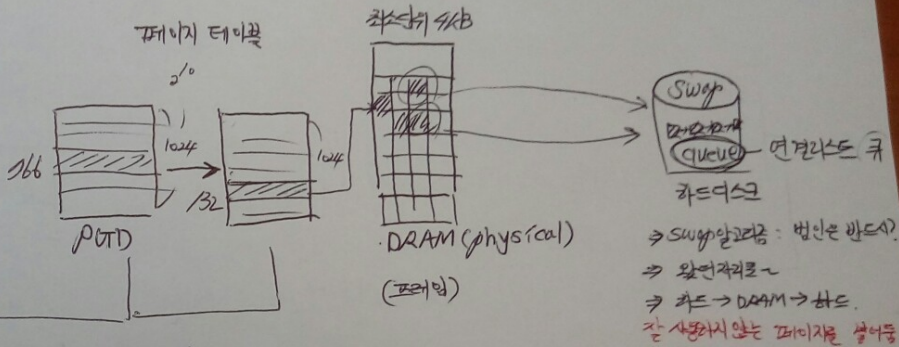
CPU가 기억어를 명령 시키는 구간은 먼저 페이지징한다

b	f	f	f	f	c	f	c
1011	1111	1000	1000	1000	1100	1000	1100
← 10bit →		← 10bit →		← 10bit →			
10bit		10bit		10bit			
10bit		10bit		10bit			

10bit = 1024



⇒ 프로세스 하나는 하나의 페이지 테이블을 갖는다.



- ⇒ 페이지가 하나의 프레임은 할당 받아 물리 메모리에 저장
- ⇒ 물리 메모리에 저장 값을 할당하면 주소가 그려짐 (memory, cache, reg)
- ⇒ 프로세스 하나가 페이지 테이블 하나를 만들어 물리 메모리에 값이 할당되어 사용한다. 이때 할당받지 못하거나 잘 사용하지 않는 페이지는 swap 한다.
- ? 만약 할당 영역 128bit는 같고 나머지가 다르다면 어떻게 할당할까?
- 그 또한 알고리즘이 있음.