TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-03-22 (21 회차)

강사: Innova Lee(이상훈)

gcccompil3r@gmail.com

학생: 정유경

ucong@naver.com

1. ls 기본기능 구현하기

```
#include <sys/types.h>
#include <dirent.h> // directory entry : opendir(), readdir(), closedir()
#include <stdio.h>

int main(void)
{
	DIR* dp;
	// DIR* 파일포인터와 유사하다고 일단은 알아두자 typedef처럼 헤더파일 안에 있을것??
	int i = 0;
	struct dirent* p; // 디렉토리 내의 내용물, 파일 리스트가 들어있다??
	dp = opendir("."); // 현재 디렉터리를 열어서 dp 파일포인터를 얻는다
	while (p = readdir(dp)) // readdir(dp) 해서 p로 디렉토리 리스트가 넘어온다
	{ // 리스트가 있을때
		if (p->d_name[0] == '.') // 맨 앞글자가 '.'이면 건너뛴다 (숨김파일)
			continue; // 반복문의 나머지 부분 건너뛰고 다시 while문 돈다
		printf("%-16s", p→d_name); // 맨 앞글자가 '.'아니면 d_name출력한다
		if ((i + 1) % 5 == 0)		//5번 단위로 개행한다.
			printf("₩n");
		i++;
	}
	printf("₩n");
	closedir(dp);
	return 0;
}
```

*. %-[자릿수]s
printf("%-5s", 'ABC'); : 총 5 자리로 표현되는 점은 위와 같으나, ABC 공백 공백 이 출력
*. %[자릿수]s
printf("%5s", 'ABC'); : 총 5 자리로 표현되고, 공백 공백 ABC 가 출력

2. getopt( ) 사용하기 – 옵션 한 글자씩 받아오기

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

int main(int argc, char** argv)
```

```
{

        int cmd;        //argc(전달인자갯수), argv(문자열 주소배열) -> 현재 옵션 몇개 있는지 확인
        while ((cmd = getopt(argc, argv, "ab")) > 0) //getopt( ) C라이브러리, 시스템 콜
        {
        //"ab" 처리할 수 있는 옵션을 나열한다. 이중 getopt가 입력받은 옵션 찾아서 cmd에 준다
                switch (cmd)
                {
                case 'a':
                        printf("a option₩n");
                        break;
                case 'b':
                        printf("b option₩n");
                        break;
                default:
                        printf("unknown option₩n");
                }
        }
        return 0;
}
```

3. main 에서 옵션을 받도록 해보자

```
//ls -a: 숨김파일까지 모두 다 보여준다 = 현재 lsmodule3.c 의 a.out -a 랑 같다
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/types.h>

int main(int argc, char** argv)
{
        DIR* dp;
        int   i = 0, cmd;
        struct dirent* p;
        cmd = getopt(argc, argv, "a");
        dp = opendir(".");
        while (p = readdir(dp)) // DIR* dp가 가리키는 디렉토리 내의 리스트를 struct dirent*p로 받는다
        {
```

```
                if (cmd != 'a') // a아닐때
                {
                        if (p→d_name[0] == '.') //'.'이면 건너뛴다 (숨김파일)
                                continue; // 파일의 나머지 부분을 실행하지 않고 다시 루프를 돈다
                }

                printf("%-16s", p→d_name);
                if ((i + 1) % 5 == 0)
                        printf("₩n");
                i++;
        }

        printf("₩n");
        closedir(dp);
        return 0;
}
```

4. flag 활용하여 옵션을 주는 방법

(비트를 이용한 옵션분할-공간효율성과 속도측면에서 좋다)
비트연산은 cpu 클럭을 최대로 활용 가능하다
int 형 변수 는 32 비트이므로 변수를 32 개 선언하는 것보다 int 형 변수 하나로 옵션을 처리하는 방법이 낫다

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

#define  A        (1 << 0) //1
#define  B        (1 << 1) //2
#define  C        (1 << 2) //4
#define  D        (1 << 3) //8
#define  E        (1 << 4) //16
#define  F        (1 << 5) //32
#define  G        (1 << 6) //64

void check_flag(int flag)
{
```

```
	int i, tmp = flag;

	for (i = 0; i < 7; i++)
	{
		switch (tmp & (1 << i)) // flag의 7개 비트에 옵션이 들어가 있는지를 검사하여 출력한다
		{
		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) // flag의 7개 비트에 옵션을 준다
		{
```

```
                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;
}
```

5. flag 활용한 'ls-a'구현

```
#include <stdio.h>
#include <sys/types.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 ) // flag 를 구한다
        {
                switch(cmd)   // cmd 로 들어오는 옵션에 따라 각 flag 위치에 1 을 설정해준다
                {
                        case 'a':
                                flag|=1;
                                break;
                        case 'l':
                                flag|=2;
                                break;
                        case 'R':
                                flag|=4;
                                break;
                        case 'i':
                                flag|=8;
                                break;
                }
        }

        dp=opendir("."); // 현재 자기자신을 열고 DIR* dp (파일포인터)반환한다
        while(p=readdir(dp)) //   DIR* dp 가 가리키는 곳을 읽어서 struct dirent *p(파일리스트를 가리키는 구조체 포인터)를 반환한다
        {
                if(!(flag&1)) // !(첫번째 옵션 a 있으면=1)=0 즉, a 가 없으면!
                {
```

```
                        if(p->d_name[0] == '.')
                                continue;
                }
                printf("%-16s", p→d_name); //숨김파일 아닐경우 이름을 출력한다
                if((i+1)%5==0) // 5 개마다 개행한다
                        printf("₩n");
                i++;
        }
        printf("₩n");
        closedir(dp);
        return 0;
}

/*./a.out 과 ./a.out -a 를 비교해보자*/
```

ls -l : 파일 세부사항 보기
ls -R : 디렉토리 내부 순회하면서 안의 내용을 전부 다 보여준다
ls -i : inode 번호 (자세한 내용은 커널들어가서 배우자)
ls -li : 옵션 2 개 -l,-i

6-1. 'ls-l' 을 구현하기 위한 사전작업 1 – 파일종류확인

*. 파일의 종류
d : 디렉토리, -: 일반파일, p: 파이프, l: link 바로가기, s : 소켓, c : 캐릭터디바이스, b : 블록디바이스

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


int main(int argc, char** argv)
{
        // 구조체 정보 보는 법: struct stat을 알고 싶으면 -> man_s2 stat
        struct stat buf;
        char ch;            // argv의 파일을 받아서 그 파일의 상태stat을 보고 buf에 저장한다
        stat(argv[1], &buf);
        if (S_ISDIR(buf.st_mode)) // 디렉토리인지 확인, (&연산 하여 있으면 디렉토리로 인식)
                ch = 'd';
```

```
        if (S_ISREG(buf.st_mode))   // regular 일반파일
                ch = '-';
        if (S_ISFIFO(buf.st_mode)) // 파이프
                ch = 'p';
        if (S_ISLNK(buf.st_mode)) // 바로가기 파일
                ch = 'l';
        if (S_ISSOCK(buf.st_mode)) // 소켓(네트워크관련 장비)
                ch = 's';
        if (S_ISCHR(buf.st_mode)) // 캐릭터 디바이스
                ch = 'c';
        if (S_ISBLK(buf.st_mode)) //블록디바이스
                ch = 'b';
        printf("%c₩n", ch); // 파일의 정보를 출력한다. ls-l의 맨 앞에 들어가는 부분
        return 0;
}
```

6-2. 'ls-l' 을 구현하기 위한 사전작업 2 – 파일권한설정

***. chmod [파일명][숫자]**

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

int main(int argc, char** argv)
{

        struct stat buf;
        char ch;
        char perm[11] = "----------";
        // [파일의종류| [rwx(root및 사용자권한)] [rwx(그룹권한-회사에서 사용)] [rwx(제3자권한)]
        // (r: read(4) / w : write(2) / x : execute(1))
        // 644 (rw | r | r)      755(rwx | rx | rx)
                char rwx[4] = "rwx"; // 3개 즉 8가지이므로, 8진수로 표현하면 좋겠다(0644)
        int i;
        stat(argv[1], &buf);

        if (S_ISDIR(buf.st_mode)) // 디렉토리인지 확인, (&연산 하여 있으면 디렉토리로 인식)
                perm[0] = 'd';
```

```
        // buf.st_mode 상위 4ibt는 파일종류, 하위 9bit는 권한설정 rwxrwxrwx
        if (S_ISREG(buf.st_mode))
                perm[0] = '-';
        if (S_ISFIFO(buf.st_mode))
                perm[0] = 'P';
        if (S_ISLNK(buf.st_mode))
                perm[0] = 'r';
        if (S_ISSOCK(buf.st_mode))
                perm[0] = 's';
        if (S_ISCHR(buf.st_mode))
                perm[0] = 'c';
        if (S_ISBLK(buf.st_mode))
                perm[0] = 'b';

        for (i = 0; i<9; i++)
                if ((buf.st_mode >> (8 - i)) & 1)
                        perm[i + 1] = rwx[i % 3]; //perm[1~9]까지 권한설정한다
        // (8 - i) & 1 = (8 - 0)이면 즉, 1000 & 1이 참이면
                printf("%s₩n", perm);
        return 0;
}
```

6-3. 'ls-l'을 구현하기 위한 사전작업 3 – usr_id, gr_id

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.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))
                perm[0] = 'd';
        if (S_ISREG(buf.st_mode))
                perm[0] = '-';
        if (S_ISFIFO(buf.st_mode))
                perm[0] = 'P';
        if (S_ISLNK(buf.st_mode))
                perm[0] = 'r';
        if (S_ISSOCK(buf.st_mode))
                perm[0] = 's';
        if (S_ISCHR(buf.st_mode))
                perm[0] = 'c';
        if (S_ISBLK(buf.st_mode))
                perm[0] = 'b';

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

        printf("%lu", buf.st_nlink); // 하드링크의 개수
        pw = getpwuid(buf.st_uid); // user_id를 pw구조체에 저장하고 출력
        printf("%s", pw->pw_name);
        gr = getgrgid(buf.st_gid); // group_id를 gr구조체에 저장하고 출력
        printf("%s", gr->gr_name);

        return 0;
}
```

6-4. 'ls-l'을 구현하기 위한 사전작업 4
– set user id, set gr id, sticky bit & 시간출력하기

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.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))
                perm[0] = 'd';
        if (S_ISREG(buf.st_mode))
                perm[0] = '-';
        if (S_ISFIFO(buf.st_mode))
                perm[0] = 'P';
        if (S_ISSOCK(buf.st_mode))
                perm[0] = 's';
        if (S_ISCHR(buf.st_mode))
                perm[0] = 'c';
        if (S_ISBLK(buf.st_mode))
                perm[0] = 'b';
```

```
        for (i = 0; i<9; i++)
                if ((buf.st_mode >> (8 - i)) & 1) //우측쉬프트해서 8,7,6,5,4,3,2,1,0 쉬프트 했을때 비트가 1이면
                        perm[i + 1] = rwx[i % 3]; //perm[1,2,3,4,5,6,7,8,9]에 각각 rwx설정, rwx[0,1,2]={r,w,x}
        for (i = 0; i<3; i++)
                if ((buf.st_mode >> (11 - i)) & 1) // 우측 쉬프트해서 0,1,2 쉬프트
                        if (perm[(i + 1) * 3] == '-') // perm[3,6,9]가 -이면
                                perm[(i + 1) * 3] = sst[i] ^ 0x20; // 대문자로 전환한다
                        else
                                perm[(i + 1) * 3] = sst[i]; // '-'없으면 소문자를 설정한다
                                // x → s,s,t가 덮어쓰게 된다
                                // 스티키비트 → swap또는 공유폴더
        printf("%s", perm);
        printf("%lu", buf.st_nlink); // 하드링크된 파일 갯수
        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;
}
```

*. struct stat 구조체에 대하여 알아보자!! 내부는 어떻게 생겼을까

리눅스는 다중 사용자 OS 이기 때문에 각 파일의 접근 가능 정도를 아는 것이 중요하다.

리눅스는 파일에 대한 정보를 얻어올 수 있는 stat 구조체를 제공한다.

```
All of these system calls return a stat structure, which contains the following fields:

    struct stat {
        dev_t     st_dev;         /* ID of device containing file */
        ino_t     st_ino;         /* inode number */
        mode_t    st_mode;        /* protection */
        nlink_t   st_nlink;       /* number of hard links */
        uid_t     st_uid;         /* user ID of owner */
        gid_t     st_gid;         /* group ID of owner */
        dev_t     st_rdev;        /* device ID (if special file) */
        off_t     st_size;        /* total size, in bytes */
        blksize_t st_blksize;     /* blocksize for filesystem I/O */
        blkcnt_t  st_blocks;      /* number of 512B blocks allocated */

        /* Since Linux 2.6, the kernel supports nanosecond
           precision for the following timestamp fields.
           For the details before Linux 2.6, see NOTES. */

        struct timespec st_atim;  /* time of last access */
        struct timespec st_mtim;  /* time of last modification */
        struct timespec st_ctim;  /* time of last status change */

    #define st_atime st_atim.tv_sec      /* Backward compatibility */
    #define st_mtime st_mtim.tv_sec
    #define st_ctime st_ctim.tv_sec
    };
```

struct stat {

dev_t st_dev; /* 장치파일의 위치 및 여부 device id */

ino_t st_ino; /* 파일의 inode 번호 */

mode_t st_mode; /* file permission 정보 */

nlink_t st_nlink; /* 하드링크의 갯수 */

uid_t st_uid; /* user id */

gid_t st_gid; /* group id */

dev_t st_rdev; /* 장치파일(inode)를 기술 */

off_t st_size; /* 해당 파일의 총 바이트크기*/

blksize_t st_blksize; /* 효율적인 I/O 파일 시스템 위한 블럭 사이즈*/

blkcnt_t st_blocks; /* 할당된 블럭 사이즈 */

};

*. struct stat.st_mode

set_uid	set_gid	Sticky_bit	rwx	rwx	rwx

set_uid 에 권한을 주면 파일명이 빨갛게 보인다.

*.set_uid : 유저에게 root 권한 일시부여한다
set_gid : 그룹에게 root 권한 일시부여한다
(root 권한 → sudo)

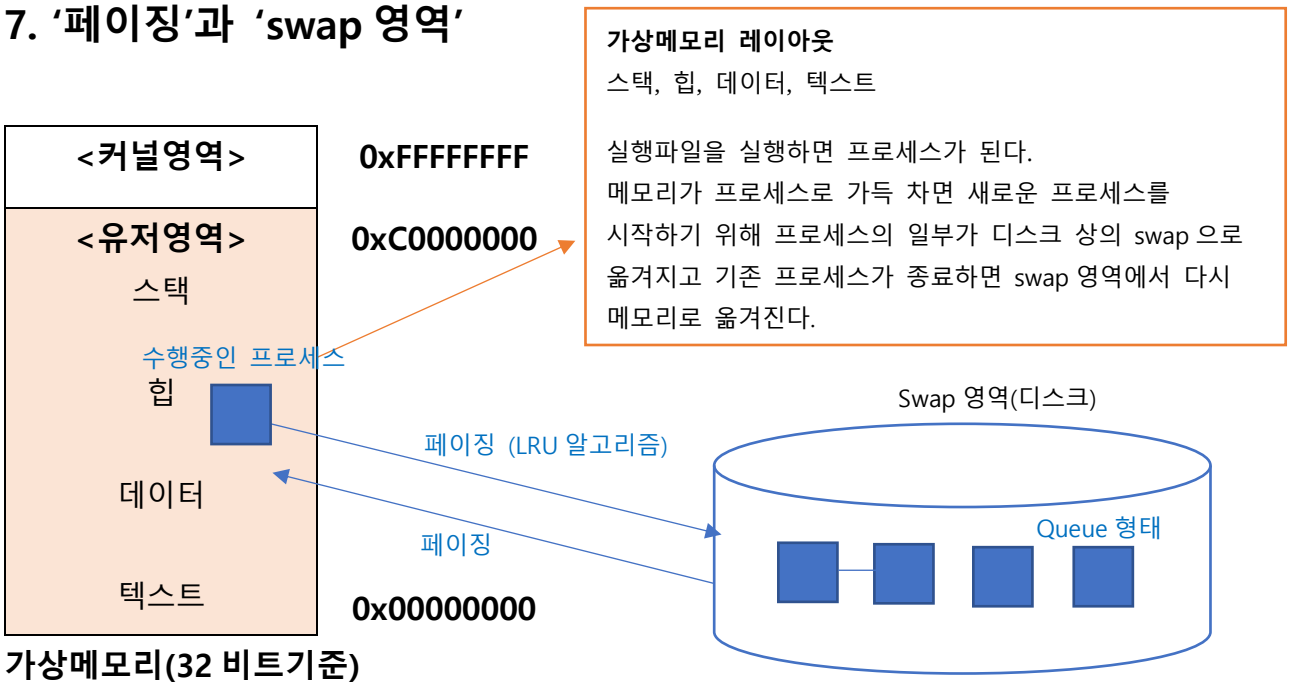
*.파일이 빨간색 블록처리가 되어있다면-→ set_uid 가 설정되어 있다는 뜻
'ls-l/usr/bin/sudo'보면 알수 있다

*.'ls-l/'하면 tmp 가 초록색이다.
Drwx rwx rw**t** 폴더에 붙어있는 t 는 공유폴더가 된다는 뜻이므로
tmp 내부에 넣어놓으면 공유하게 된다.

*. 스티키비트
ex. **d**rwxrwxrw**t**
1. 디렉토리에 붙을 경우 : 공유폴더가 된다 → 초록색
2. 파일에 붙을 경우 : swap 영역을 사용한다. → 빨간색

Cf. 주의할 부분
*. stat(pathname , &)은 첫번째 인자로 pathname 을 받으며 stat 구조체에 그 정보를 저장한다

7. '페이징'과 'swap 영역'



가상메모리(32 비트기준)

계속해서 메모리에 할당하면 컴퓨터의 속도는 느려진다.
메모리에는 적당히 할당하고 현재 필요한 정도만 페이징 하는 것을 "Demand on Paging"이라고 한다.

프로그램이 100G 일 경우 즉, 물리메모리가 감당할 수 없는 용량일 때 swap 영역을 사용한다.
페이지 자체를 swap 영역(디스크)에 넣는다 이때, LRU 알고리즘이 사용된다.
*. LRU 알고리즘: 자주 사용하지 않으면서 다시 재활용 할 페이지 선정하여 swap 영역에 넣는 알고리즘
*. swap 영역은, Sticky bit 옵션을 사용하여 SW 캐시를 서포트 한다.

8. 커널이 물리메모리를 찾는 메커니즘

가짜주소가 실제 물리메모리 주소로 변환된다
Ex. 0xbf884c8c
10/10/12 비트로 나눈다
1011 1111 10/00 1000 0100 /1100 1000 1100
766 / 132 / 3212
1. pgd 의 766 번째 인덱스를 확인한다.
2. 766 번째 인덱스의 배열포인터를 참조하여 다음 배열의 132 번째 인덱스로 이동한다.
3. 132 번째 인덱스에 들어있는 실제 물리메모리의 주소를 참조한다.
*. Pgd 위치: task_struct > mm_struct > pgd