TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-03-20 (19회차)

강사: Innova Lee(이상훈)
gcccompil3r@gmail.com
학생: 정유경
ucong@naver.com

linux kernel 의 소스코드 분석을 위한 환경 설정

Task_struct → files_struct

struct file *fd_array[]

open()을 통해서 얻게되는 File Descriptor의 번호는 결국 이 배열의 인덱스에 해당
커널은 별도의 정보를 제공하지 않고 이 인덱스 정보만을 제공
따라서 시스템 내부에 치명적 손상을 줄 수 있는 포인터 주소등을 주지 않고도
유저가 파일 제어 가능
그래서 read, write, close등에는 숫자만 전달
커널이 요청을 받으면 숫자값을 보고 어떤 파일을 제어해야 하는지 빠르게 파악가능

파일 시스템(file system)

- 파일과 그 안에 든 자료를 저장하고 찾기 쉽도록 유지, 관리하는 방법을 말한다.

- 윈도우는 FAT, NTFS의 2가지이고 리눅스는 1000여개의 파일시스템(EXT2, EXT3, EXT4...)을 모두 지원한다.

*. 커널 쪽에서는 파일 시스템이나 드라이버 구현의 상세한 내용을 알지 못해도,

구조체에 있는 멤버의 이름으로 호출을 할 수 있다....???

File_operations={함수포인터?!}

- 하나의 시스템에 여러 개의 파일 시스템을 사용하는 것이 가능해진다.

- 리눅스 커널에서 파일 시스템 관련부분을 보면 VFS(가상파일시스템)을 쓴다. 동일한 파일시스템 함수들을 정해놓고, 정작 그 구현은 파일 시스템에 따라 달라야 하기 때문이다.
- 즉, 동일한 인터페이스를 써야 하는데 그 구현은 각기 달라야 하는 모듈들을 구현해야 하기 때문에 함수포인터를 사용한다.

1. mytar.c

typedef struct{
char fname[20];
int fsize;
} F_info;

old = lseek(fd, 0, SEEK_CUR);// 파일 **현재위치** 0//파일포인터 복구시킬위치
Fsize = lseek(fd, 0, SEEK_END); // **파일 끝** // 전체 파일 사이즈
Lseek(fd, old, SEEK_SET);
// fd가 나타내는 파일의 현재위치를 **시작점**에서 old로 이동

dst = open(argv[argc-1], O_WRONLY|O_CREAT|O_TRUNC,0644);
argv[argc-1]: 가장 마지막에 들어온 인자

for(i=0;i<argc-2;i++)
argc-2: 가장 마지막인자 res.tar 전까지 즉, a.txt b,txt c.txt

src = open(argv[i+1], O_RDONLY);
argv[i+1]: 실행파일은 넣지 않는다.

strcpy(info.fname, argv[i+1]); // 파일이름 복사
info.fsize = file_size(src);

write(dst, &info, sizeof(info)); // res.tar에 info구조체의 내용을 기록
while(ret = read(src, buf, sizeof(buf))) // src의 내용 읽어서 버퍼에 저장
write(dst, buf, ret); // 버퍼에서 res.tar에 저장

a.txt 파일이름	700
a.txt 파일내용	

*. 실행: ./a.out a.txt b
*. xxd res.tar 로 확인

2. mytarfree.c

#define min(x,y) (((x) < (y)) ? (x) : (y))
// 수식 많이 들어갈 수 있으니 괄호!

Src = open(argv[1],O_RDONLY);
실행시: ./a.out res.tar

While(read(src, &info , sizeof(info))) // 읽을것 있으면 계속 반복
Src맨앞의 구조체를 info구조체로 읽어온다.
Dst=open(info.루믄, O_WRONLY|O_TRUNC|O_CREAT,0644);

While(info.fsize>0)
Len = min(sizeof(buf), info.fsize);
Info.fsize 가 버퍼사이즈보다 클수도 있다. (즉 1024보다 큼)
최소값을 찾아서 넘지 않는 값만 write

Ret = read(src,buf,len); // res.tar에서 len만큼 읽어서 buf에 저장
write(dst,buf,ret); // 읽은만큼 버퍼에서 dst에 쓴다

Info.fsize-=ret; // info.fsize에서 읽은만큼을 뺀다.
2000—1024라면 976이 되어 while한번 더 돈다.

3. mymove.c

같은 파일을 두번 열어주면 파일 포인터가 이동할까?

이동하지 않는다.

Open할 때 마다 별도의 fd가 생성되기 때문이다.

따라서 출력결과는 같다.

```
fd[0] = open("mytar.c", O_RDONLY);
read(fd[0], buf,10);
write(1, buf,9);

fd[1] = open("mytar.c", O_RDONLY);
read(fd[1],buf,9);
write(1,buf,9);
```

Task_struct → files_struct* → file*

*. task_struct: 프로세스에 관한 모든 정보를 보관하는 프로세스 서술자

Quiz 1

임의의 난수를 10 개 발생시켜서 이 값을 배열에 저장하고
배열에 저장된 값을 파일에 기록한다(중복은 안됨).
그리고 이 값을 읽어서 Queue 를 만든다.

이후에 여기 저장된 값중 짝수만 선별하여 모두 더한 후에
더한 값을 파일에 저장하고
저장한 파일을 읽어 저장된 값을 출력하도록 한다.

(반드시 System Call 기반으로 구현하도록 함 - 성능이 압도적임)

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdbool.h>
#include <fcntl.h>


int extract_index;

typedef struct Queue {
    int data;
    struct Queue* link;
} Queue;

// 노드생성
Queue* getnode()
{
    Queue *ptmp;
    ptmp = (Queue*)malloc(sizeof(Queue));
    ptmp->link = NULL;
    return ptmp;
}


// 노드추가
void Enqueue(Queue** ppHead, int data)
{

    if(*ppHead == NULL)
```

```c
// 노드추가
void Enqueue(Queue** ppHead, int data)
{

    if(*ppHead == NULL)
    {
        *ppHead = getnode();
        (*ppHead)->data = data;
        return;
    }
    Enqueue(&(*ppHead)->link, data);
}


void print_queue(Queue* pHead)
{
    if (pHead)
    {
        printf("%d\n", pHead->data);
        print_queue(pHead->link);
        return;
    }

}


Queue* dequeue(Queue *pHead, int data)
{

    Queue* ptmp = pHead;
```

```c


Queue* dequeue(Queue *pHead, int data)
{

    Queue* ptmp = pHead;
    if (ptmp->data == data)
    {
        ptmp = pHead->link;
        free(pHead);
        return ptmp;
    }
    else
        ptmp->link = dequeue(ptmp->link, data);
    return pHead;
}


//중복체크
bool is_dup(int arr[], int index)
{
    for (int i = 0; i < index; i++)
        if (arr[index] == arr[i])
            return true;
    return false;
}


//랜덤배열생성 int* arr , int size를 인자로 받는다
void random_arr(int arr[], int size)
{
```

```c
//랜덤배열생성 int* arr , int size를 인자로 받는다
void random_arr(int arr[], int size)
{
    srand((unsigned)time(NULL)); // main으로 옮길것
    for (int i = 0; i<size; i++)
    {
    redo:
        arr[i] = (rand() % 10) + 1; //1~10
        if (is_dup(arr, i)) // is_dupÀÌ trueÀÌžé
            goto redo;
    }
}

//배열출력
void print_arr(int arr[], int size)
{
    for(int i=0; i<size;i++)
    {
        printf("arr[%d]=%d\n",i,arr[i]);
    }
}

//queue에서 짝수를 뽑아서 배열에 저장
void extract_even(Queue *pHead, int *extract)
{
    Queue* tmp=pHead;
    while(tmp)
    {
        if(!(tmp->data % 2)) // Even
            extract[extract_index++] = tmp->data;
        tmp = tmp->link;

```

```c
int main()
{
    Queue *pHead = NULL;
    int i, fd;
    int arr[10] = { 0 }; //int arr[11] = {0}; 11????
    int size = sizeof(arr) / sizeof(int);
    // int size = sizeof(arr) / sizeof(int) - 1; -1???
    random_arr(arr, size); // 기본 랜덤배열 구하기
    print_arr(arr, size);
    for (i = 0; i < size; i++) // 배열을 queue에 넣는다.
        Enqueue(&pHead, arr[i]);

    int extract[11]={0}; // 11..??
    extract_even(pHead, extract); // 짝수 배열 구하기
    print_arr(extract, extract_index);

    // array.txt 파일 생성한다. 없으면 만들고, WR가능, 갱신, 0644권한
    fd = open("array.txt", O_CREAT | O_WRONLY | O_TRUNC, 0644);
    int sum = 0;
    for ( i = 0; i < extract_index; i++)
        sum += extract[i]; // 짝수배열요소 합 구하기
    char buf[32] = {0}; // 32...????
    sprintf(buf, "%d", sum); // %d 숫자를 문자열로 바꾼다
    //sprintf: 다양한 서식의 값을 문자열로 만들어 준다
    //sum을 "%d"로 서식을 지정하여 buf에 저장
    write(fd, buf, strlen(buf)); // array.txt 파일에 buf(문자열로 저장된 sum)를 write
    close(fd);
    return 0;
}
```

Quiz 2

카페에 있는 50 번 문제(성적 관리 프로그램)을 개조한다.

기존에는 입력 받고 저장한 정보가 프로그램이 종료되면 없어진다. 입력한 정보를 영구히 유지할 수 있는 방식으로 만들면 더 좋지 않을까 ?

* 조건

1. 파일을 읽어서 이름 정보와 성적 정보를 가져온다.

2. 초기 구동 시 파일이 없을 수 있는데 이런 경우엔 읽어서 가져올 정보가 없다.

3. 학생 이름과 성적을 입력할 수 있도록 한다.

4. 입력된 이름과 성적은 파일에 저장되어야 한다.

5. 당연히 통계 관리도 되어야한다(평균, 표준 편차)

6. 프로그램을 종료하고 다시 켜면 파일에서 앞서 만든 정보들을 읽어와서 내용을 출력해줘야 한다.

7. 언제든 원하면 내용을 출력할 수 있는 출력함수를 만든다. [특정 버튼을 입력하면 출력이 되게 만듦]

(역시 System Call 기반으로 구현하도록 함)