TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/3/20
수업일수	19 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

1. system call

- ~_t 로 끝나는 자료형

~_t 로 끝나는 자료형은 시스템의 종속적인 요소가 아니다.(int 같은 자료형이나 포인터들) 그래서 시스템이 32bit 에서 64bit 로 바뀌었을 때 코드를 수정하지 않고 헤더파일에 선언된 자료형들을 재정의 해주면 된다.
〈sys/types.h〉 헤더파일 사용

off_t = long 형 정수
size_t = unsigned int
ssize_t = signed int

-File Descriptor

open(), openat() 함수 : 리눅스, 유닉스 시스템에서 이미 존재하는 파일을 열 때 open()혹은 openat()함수를 사용. 리턴되는 파일 디스크립터를 인자로 이용하여 read(), write() 등을 수행 할 수 있다. 일반적으로 0이 아닌 정수 값을 갖는다.

리눅스, 유닉스 시스템에서 0은 표준입력(키보드), 1은 표준출력(모니터), 2는 표준에러이다.

create() 함수 : 새로운 파일을 생성한다. 하지만 write 모드로만 열려 다시 읽기 위해서는 O_RDONLY 또는 O_RDWR 의 플래그를 명시해서 다시 열어야한다.

close() 함수 : open()으로 연 파일을 close()로 닫는다.

2. linux kernel

-함수 포인터를 사용하는 이유

리눅스 지원하는 파일시스템이 1000개가 넘는데, open, read, write 등의 명령어들을 사용할 때 사용되는 인자의 수가 매우 많다.
따라서 이를 일일히 매번 기록하여 사용할 수 없기 때문에 함수를 가리키는 포인터를 사용한다면 편리하고 효율적으로
사용가능하기에 함수 포인터를 사용한다.

-커널 탐색을 위한 준비
‘ / ’ : 찾을 파일
Ex) /struct

(1)vi ~/.vimrc

```
"ctags 설정"
set tags =/home/xeno/kernel/linux-4.4/tags
if version >= 500
func! Sts()
        let st = expand("<cword>")
        exe "sts ".st
endfunc
nmap ,st :call Sts()<cr>
func! Tj()
        let st = expand("<cword>")
        exe "tj ".st
```

```
endfunc
nmap ,tj :call Tj()<cr>
endif
"cscope  설정"
set csprg =/usr/bin/cscope
set nocsverb
cs add /home/xeno/kernel/linux-4.4/cscope.out
set csto=0
set cst
func! Css()
        let css = expand("<cword>")
        new
        exe "cs find s ".css
        if getline(1) == ""
                exe "q!"
        endif
        endfunc
        nmap ,css :call Css()<cr>

func! Csc()
        let csc = expand("<cword>")
        new
        exe "cs find c ".csc
        if getline(1) == ""
                exe "q!"
        endif
```

```
endfunc
nmap ,csc :call Csc()<cr>
func! Csd()
                let csd = expand("<cword>")
                new
                exe "cs find d ".csd
                if getline(1) == ""
                        exe "q!"
                endif
        endfunc
        nmap ,csd :call Csd()<cr>

func! Csg()
        let csg = expand("<cword>")
        new
        exe "cs find g ".csg
        if getline(1) == ""
                exe "q!"
        endif
        endfunc
        nmap ,csg :call Csg()<cr>
```

(2)vi mkcscope.sh

```
#!/bin/sh
rm -rf cscope.files cscope.files
find. ₩(-name'*.c'-o-name '*.cpp'-o-name'*.cc'-o-name'*.h'-o-name'*.S'₩)-print 〉cscope.files
cscope -i cscope.files
```

chmod 755 ~/mkscsope.sh 명령어를 이용해 mkcscope.sh 를 활성화함
커널 탐색을 위한 준비가 모두 마쳐졌으면 vi mkcscope.sh 를 실행시킴.

- 커서위치의 정보찾기 : ctrl + }
- 뒤로가기 : ctrl + t
- 찾기 : /찾을내용

구조체나 함수가 시작되는 부분을 보고싶을 때
include 찾을부분{
을 찾으면 된다.

3.압축

-압축하기

```
#include <fdntl.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>

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

int file_size(int fd)
{
        int fsize,old;
        old=lseek(fd,0,SEEK_CUR);          //old 에는 현재위치를 저장
        fsize=lseek(fd,0,SEEK_END);        //fsize 엔 파일의 마지막 위치를 저장한다.
        lseek(fd,old,SEEK_SET);     //파일의 처음 위치에서 old 에 저장된 현재위치만큼 이동한다
        return fsize;
}
```

```
int main(int argc,char *argv[])
{
	int src,dst,ret;
	char buf[1024];
	F_info info;
	int i;
	dst=open(argv[argc-1],O_WRONLY|O_CREAT|O_TRUNC,0644);
	//가장 마지막인자이기 때문에 argv-1
	for(i=0;i<argc-2;i++)
	{
		src=open(argv[i+1],O_RDONLY);
		strcpy(info.fname,argv[i+1]);		//info.fname 에 argc[i+1]의 값을 복사한다.
		info.fsize=file_size(src);	//src 의 파일 마지막 위치를 info.fsize 에 너머겨 크기 값을 저장
		write(dst,&info,sizeof(info));
		while(ret=read(src,buf,sizeof(buf)))
			write(dst,buf,ret);
		close(dst);
	}
	close(dst);
	return 0;
}
```

~결과

```
xeno@xeno-NH:~/proj/0320$ gcc tar.c
xeno@xeno-NH:~/proj/0320$ ./a.out a.txt b.txt c.txt res.tar
```

```
xeno@xeno-NH:~/proj/0320$ xxd res.tar
00000000: 612e 7478 7400 0000 d868 e8a2 fe7f 0000    a.txt....h......
00000010: c70e e03d 0600 0000 6865 6c6c 6f0a 622e    ...=....hello.b.
00000020: 7478 7400 0000 d868 e8a2 fe7f 0000 c70e    txt....h........
00000030: e03d 0d00 0000 4c69 6e75 7820 7379 7374    .=....Linux syst
00000040: 656d 0a63 2e74 7874 0000 00d8 68e8 a2fe    em.c.txt....h...
00000050: 7f00 00c7 0ee0 3d0c 0000 0073 7973 7465    ......=....syste
00000060: 6d20 6361 6c6c 0a                          m call.
```

a.txt 는 hello 라는 내용을 입력하여 저장했고, b.txt 는 Linux system 이라는 내용을, c.txt 는 system call 이라는 내용을 입력하여 저장하였다.
위 소스파일을 실행시키면 res.tar 이라는 압축파일에 a.txt , b.txt , c.txt 파일이 압축되어 들어있다. xxd res.tar 을 치면 각각의 txt 파일에 들어있는 내용을 볼 수 있다.

-압출풀기

~소스

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

typedef struct
{
        char fname[20];
```

```
        int fsize;
}F_info;

#define min(x,y)        (((x)<(y))?(x):(y))
//x 나 y 에 복잡한 수식이 올 수 있어서 (x),(y)처럼 괄호를 쳐준다.

int main(int argc,char *argv[])
{
        int src,dst,len,ret;
        F_info info;
        char buf[1024];
        src = open(argv[1],O_RDONLY);
        //src 에 압축 풀 ret.tar 를 넣는다.
        while(read(src,&info,sizeof(info)))
        {
                dst=open(info.fname,O_WRONLY|O_TRUNC|O_CREAT,0644);
                while(info.fsize>0)
                {
                        len=min(sizeof(buf),info.fsize);
                        //fsize 가 1024byte 넘을수있으므로 최소값을 찾음
                        ret=read(src,buf,len);
                        write(dst,buf,ret);
                        //dst 에 write 함 src 에서 읽은 것들을
                        info.fsize-=ret;
                        //남은 데이터들을 처리하는 문장.
                        //ret : 읽은 바이트 수
```

```
                }
                close(dst);
        }
        close(src);
        return 0;
}
```

~결과

```
xeno@xeno-NH:~/proj/0320$ gcc tar-free.c
xeno@xeno-NH:~/proj/0320$ ./a.out res.tar
```

를 하면 res.tar 압축이 풀려 a.txt , b.txt, c.txt 가 생기게 된다.

3.
~소스

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

int main(void)
{
	char buff[1024];
	int fd[2];

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

	fd[1]=open("mytar.c",O_RDONLY);
	read(fd[1],buff,10);
	write(1,buff,10);

	return 0;
}
```

~결과

```
#include<f#include<f
```

같은 내용이 출력된다. Fd[0]과 fd[1]에 저장된 파일은 같은 것이고 이를 10byte 만큼만 출력했으니 fd[0]도 fd[1]도 같은 내용이 올 수 밖에 없다.

리눅스를 할때 핵심 구조체 : tast_struct

-문제 1
임의의 난수 10개를 발생시켜서 이 값을 배열에 저장하고 배열에 저장된 값을 파일에 기록하다.
그리고 이 값을 읽어서 queue 를 만든다.
이후에 여기 저장된 값 중 짝수만을 선별하여 모두 더한 후에 더한 값을 파일에 저장하고 저장한 파일을 읽어 저장된 값을 출력하도록 한다.
System call 기반으로 구현해야한다.

-문제2
카페이 있는 50번 문제 (성적 관리 프로그램)
기존의 입력받고 저장한 정보가 프로그램이 종료되면 날아갔었는데,
입력한 정보를 영구히 유지할 수 있는 방식으로 만들기.

조건
1. 파일을 읽어서 이를 정보와 성적 정보를 가져온다.
2. 초기 구동시 파일이 없을 수 있는데 이런 경우엔 읽어서 가져올 정보가 없다.
3. 학생 이름과 성적을 입력 할 수 있도록 한다.
4. 입력된 이름과 성적은 파일에 저장되어야 한다.
5. 통계 관리도 되어야 한다.(평균, 표준편차)
6. 프로그램을 종료하고 다시 키면 파일에서 앞서 만든 정보들을 읽어와서 내용을 출력해줘야 한다.
7. 언제든 원하면 내용을 출력할 수 있는 출력함수를 만든다. 특정버튼을 입력하면 출력이 되게 만듬
8. System call 기반으로 구현해야한다.

quiz1)

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

int extract_idx;

typedef struct __queue
{
	int data;
	struct __queue *link;
} queue;

bool is_dup(int *arr, int cur_idx)
{
	int i, tmp = arr[cur_idx];
	for(i = 0; i < cur_idx; i++)
		if(tmp == arr[i])
			return true;
	return false;
}//난수의 중복을 찾는 함수이다 중복이면 1, 중복이 아니면 0을 반환한다.

void init_rand_arr(int *arr, int size)
{
	int i;
	for(i = 0; i < size; i++)
	{
redo:
		arr[i] = rand() % 10 + 1;
		if(is_dup(arr, i))
		{
			printf("%d dup! redo rand()₩n",
arr[i]);
			goto redo;//난수가 중복일 경우에
redo 로 가게 된다. 그리고 다시 난수를 만듦.
		}
	}
}
void print_arr(int *arr, int size)
{
	int i;
	for(i = 0; i < size; i++)
		printf("arr[%d] = %d₩n", i, arr[i]);
}//난수가 저장된 배열을 출력한다.
queue *get_queue_node(void)
{
	queue *tmp;
	tmp = (queue *)malloc(sizeof(queue));
```

```
	tmp->link = NULL;
	return tmp;
}
void enqueue(queue **head, int data)
{
	if(*head == NULL)
	{
		*head = get_queue_node();
		(*head)->data = data;
		return;
	}
	enqueue(&(*head)->link, data);
}
void extract_even(queue *head, int *extract)
{
	queue *tmp = head;
	while(tmp)
	{
		if(!(tmp->data % 2))//짝수이면
			extract[extract_idx++] =
tmp->data; //짝수인 경우 값을 더한다.
		tmp = tmp->link;
	}
}
```

```
int main(void)
{
	int i, fd, len, sum = 0;
	char *convert[10] = {0};
	int arr[11] = {0};
	char tmp[32] = {0};
	int extract[11] = {0};
	int size = sizeof(arr) / sizeof(int) - 1;
	queue *head = NULL;
	srand(time(NULL));
	init_rand_arr(arr, size);//난수를 만듦
	print_arr(arr, size);//난수를 출력함
	for(i = 0; i < size; i++)
		enqueue(&head, arr[i]);//난수를 저장
	extract_even(head, extract);//짝수일 경우 더한다.
	printf("₩nExtract:₩n");
	print_arr(extract, extract_idx);
	fd = open("log.txt", O_CREAT | O_WRONLY |
O_TRUNC, 0644);//log.txt 라는 파일 생성
	for(i = 0; i < extract_idx; i++)
		sum += extract[i];
	sprintf(tmp, "%d", sum);//sprintf 쓰는 이유를
모르겠음
	write(fd, tmp, strlen(tmp));
	close(fd);
```

왼쪽의 #if 0, #endif 를 모르겠다.

```
#if 0
	for(i = 0; i < extract_idx; i++)
	{
		int len;
		char tmp[32] = {0};
		sprintf(tmp, "%d", extract[i]);
		len = strlen(tmp);
		convert[i] = (char *)malloc(len + 1);
		strcpy(convert[i], tmp);
		printf("tmp = %s₩n", tmp);
	}
#endif
	return 0;
}
```