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

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

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 filedes;
        char pathname[]="temp.txt";
        if((filedes=open(pathname,O_CREAT | O_RDWR | 0644)) == ERROR)
    // if((filedes=open(pathname,O_CREAT | O_RDWR ,0644)) == ERROR)
       {
                printf("File Open Error!\n");
                exit(1);
        }
        printf("fd=%d\n",filedes);

        close(filedes);

        return 0;
}
```

// 파일이 있다면 파일을 만들지마라.
// 파일을 읽고 쓸 수 있게 pathname 명으로 만들어라

EX2- file_io3.c

```
#include <fcntl.h>

int main(void)
{
	int filedes1,filedes2;

	filedes1 = open("data1.txt",O_WRONLY | O_CREAT | O_TRUNC, 0644);

	filedes2 = creat("data2.txt",0644);

	close(filedes1);
	close(filedes2);

	return 0;
}
```

//O_TRUNC : 무조건 한번 밀어버린다.(작업을 할때마다 새로 갱신함) 임시저장시 사용함.

Flag 에 인자로 넘겨주는 값

- O_CREAT : 필요한 경우 파일을 생성한다.
- O_TRUNC : 존재하던 데이터를 모두 삭제한다.
- O_APPEND: 존재하던 데이터를 보존하고 뒤에 이어서 저장한다.
- O_RDONLY : 읽기 전용 모드로 파일을 연다
- O_WRONLY : 쓰기 전용 모드로 파일을 연다.
- O_RDWR : 읽기, 쓰기 겸용 모드로 파일을 연다.

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);
        fdout = open("temp2.txt",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);
        close(fdout);
        retrun 0;
}
```

```
// 읽기전용, 읽을게 없으면 안열림.
// 실행할때마다 안에있는거 밀어버림.

// 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 집어넣
   어줘
// write(fd,buf,홀 크기) : buf 에 있는 값이 nread 크기만큼 fd 에 써짐. 자
   기가 쓸 byte 크기 리턴
// nread 만큼 썼으므로 nread 보다 작을 수 없음.

 // 마찬가지로 system call 임.
```

→ cp 를 만듬. 파일복사..

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

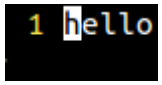
EX4- file_io5.c　파일의 용량 검사

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

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

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

data1.txt



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);
```

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[])

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

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.

//

```c
#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

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include "my_scanf.h"

int main(void)
{
        int nr;
        char buf[1024]={0};

        nr= my_scanf(buf,sizeof(buf));
        printf("nr=%d\n",nr);
        write(1,buf,nr);
        return 0;
}

```

```c
#include "my_scanf.h"

int my_scanf(char *buf, int size)
{
        int nr = read(0,buf,size);
        return nr;
}
```

write(1,~) : 모니터에 쓰기

```c
#ifndef __MY_SCANF_H__ // 선언된 것이 있으면 이 헤더파일을 들어가지마라.
#define __MY_SCANF_H__ // 선언된 것이  없으면 my_scanf_h 가 0 으로 됨. (그냥상수임!)

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

int my_scanf(char *, int);

#endif    // 두번 선언하는 것을 방지한다.


// 분할돼어있어 분간이쉬움
```

알고리즘 파트, 보드제어파트 , 순수한 소프트웨어파트, 영상처리 파트가 있을 때, 한공간에 묶어 놓으면 안된다.
분할되어있어 분간하기 쉬움. (<> : 시스템헤더, “ “: 사용자정의(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;
}
```

:vs :sp :e 파일명 컨트롤 +w w 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	다운받기.

1. 드라이빙하기

vi ~/.vimrc

```
"ctags 설정"
set tags=/root/compiler/gcc-4.5.0/tags
(bp 이후에 set tags=/home/lhs/kernel/linux-4.4/tags 로바꿔준다)
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 /root/compiler/gcc-4.5.0/cscope.out
(bp 이후에 cs add /home/lhs/kernel/linux-4.4/cscope.out 로바꿔준다)
set csto=0
set cst
func! Css()
    let css = expand("<cword>")
    new
    exe "cs find s ".css
    if getline(1) == ""
```

2. 명령어: 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
```

```
        exe "q!"
    endif
endfunc
nmap ,css :call Css()<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>
```

3. kernel 파일 → linux-4.4 파일 →

명령어: sudo apt-get install ctags cscope
명령어 : ctags -R

cd ~/ (홈으로)

명령어: chmod 755 ~/mkcscope.sh → mkcscope.sh 가 초록색이 되야함.
명령어: sudo ~/mkcscope.sh /usr/local/bin/
명령어 :ls /usr/local/bin/mkcscope.sh
명령어:mkcscope.sh → 빠져나오기 ctrl + D
명령어:vi -t task_struct ; 144 enter

/files_struct 찾고 ctrl+} 누른다 1 누르고 enter

struct file __rcu * fd_array[NR_OPEN_DEFAULT]; 위에 주석을 단다.

```
/* open()을 통해서 얻게 되는 FIle Descriptor 의 번호는
   결국 이 배열의 인덱스에 해당한다.
   커널은 별도의 정보를 제공하지 않고
   이 인덱스 정보만을 제공하므로
   시스템 내부에 치명적인 손상을 줄 수 있는
   포인터 주소등을 주지 않고도 유저가 파일을 제어할 수 있게 해줌
   그래서 read, write, close 등에는 숫자만 전달하게됨
   이 요청을 커널이 받으면 숫자값을 보고
   어떤 파일을 제어해야 하는지 빠르게 파악할 수 있음. */
```

152 include- 파일 구조체의 시작점.
→ q 누루고 152 를 친다.

file_operations 에서 ctrl + } → 10
빠져나올때는 ctrl + T

path → 93 → struct dentry

파일 묶어놓기

```
#include <fcntl.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);
	fsize=lseek(fd,0,SEEK_END);
	lseek(fd,old,SEEK_SET);
	return fsize;
}
```

res.tar 2000

a.txt 700

a.txt	700
a.txt 의 내용	
b.txt	100
b.txt 의 내용	
c.txt	10
c.txt 의 내용	

int main(void)
{
  int src, dst, ret;
  char buf[1024];
  F_info info;
  int i;
  dst = open(argv[argc -1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
  for(i=0;i<argc-2;i++)
  {
  src = open(argv[i+1],O_RDONLY);
  strcpy(info.fanme,argv[i+1];
  info.fsize = file_size(src);
  write(dst,&info,sizeof(info));
  while(ret=read(src,buf,sizeof(buf)))
  write(dst,buf,ret);
  close(src);
  }
  close(dst);
  return 0;
}
	file 이름 info.fanme 이라는 이름에 복사. → 구조체를 보자.

a.txt → hello
b.txt → linux system
c.txt → system call

./debug a.txt b.txt c.txt res.tar
xxd res.tar

파일 압축풀기

```
#include <fcntl.h>

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

#define min(x,y)       (((x)<(y))?(x):(y))

int main(int argc, char *argv[])
{
   int src, dst, len, ret;
   F_info info;
   char buf[1024];
   src = open(argv[1], O_RDONLY);       //
   while(read(src,&info,sizeof(info)))  // src
   {
      dst = open(info.fname,O_WRONLY|O_TRUNC|O_CREAT,0644);  //
a.txt 열어서 쓰기전용
      while(info.fsize >0)
      {
         len = min(sizeof(buf), info.fsize);  //  fsize 가 1024 를 넘으므로 최소
값을 구함.
         ret = read(src,buf,len);
         write(dst,buf,ret);
         info.fsize -= ret;
```

파일 삭제 후
./debug res,tar

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

복사해와서 읽기

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

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

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

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

실행 시킨 것 task_struct

open 을 하면 별도의 타입 디스크립트가 생긴다.

Quiz

임의의 난수를 10 개 발생시켜서 이 값을 배열에 저장하고, 배열에 저장된 값을 파일에 기록한다. (중복안됨.)
그리고 이 값을 읽어서 Queue 를 만든다.
이후에 여기 저장된 값 중 짝수만 선별하여 모두 더한 후에 더한 값을 파일에 저장하고, 저장한 파일을 읽어 저장된 값을 출력하도록한다.

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

int* randomm(int arr[])
{
   int i,k;
   for(i=0;i<10;i++)
   {
         arr[i]=rand()%10+1;
         for(k=0;k<i;k++)
         {
            while(arr[i]==arr[k])
            {
                  arr[i]=rand()%10+1;
                  k=0;
            }
         }
   }
   return arr;
}
int main(void)
{
   int arr[10];
```

```
    int* arr_m=randomm(arr);
    int i;
    int fd_array;
    int buf[1024]={0};

    for(i=0;i<10;i++)
          printf("%d\n",arr_m[i]);


    fd_array = open("fd_array.txt",O_WRONLY | O_CREAT | O_TRUNC,
0644);

    for(i=0;i<10;i++)
    {
          sprintf(buf,"%d\n",arr_m[i]);
        // printf("buf=%s\n",buf);

              write(fd_array,buf,strlen(buf));

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

오답노트 기터브 x 다음주월요일 , 메일 ,gmail.com
오늘학습내용정리]