TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/3/19
수업일수	18 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

1. Linux system programming

-리눅스란

유닉스를 기반으로 개발한 공개용 오퍼레이팅 시스템(OS)이다. 우리가 무언가를 하고자 할 때 리소스(resource)를 제공해줌
리소스는 컴퓨터 시스템자체와 운영체제에 포함되는 하드웨어 기구나 기능의 총칭이다. 또한 처리에 요하는 시간이나 오퍼레이터, 입출력장치, 주기억 장치, 제어프로그램, 처리프로그램 등을 가리키는 경우도 있다. 리소스에는 하드웨어,소프트웨어 둘 다 포함한다.

SW 3대장
1. OS(Operating System)
2. compiler
3. DB(Data Base)

-Free SW, Open SW
free SW　　//자유, 책임 x
open SW　　//책임 o

-system call mechanism
: system call 은 user 가 kernel 에게 요청하는 작업을 의미

2. File descriptor

1-1. open()

~소스

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

#define ERROR    -1

int main(void)
{
        int filedes;
        char pathname[]="temp.txt";
        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;
}
```

~결과

temp.txt 만들어짐

1-2

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

#define ERROR     -1

int main(void)
{
            int filedes;
            char pathname[]="temp.txt";
            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;
}
```

~결과

```
File Open Error!
```

- **file descriptor**
  파일 식별자라고 생각하면 된다. 숫자값을 리턴한다.

- **open**
  open 이라는 명령어를 이용하여 파일을 생성할 수 있다.
  open(파일이름, 파일특성)
  파일 특성에는 O_CREAT, O_EXCL, O_RDWR, O_RDONLY, O_WRONLY, O_TRUNC 등이 있다.
  O_CREAT 는 앞에서 지정한 파일 이름으로 파일을 생성할 때 사용한다.
  O_EXCL 는 앞에서 지정한 파일 이름으로 이미 파일이 존재할 때 에러를 되돌려주며 파일을 생성하는데 실패한다.
  O_RDWR 는 파일을 읽고 쓸 수 있도록 지정한다.
  O_RDONLY 는 읽기전용의 파일로 지정한다. 읽을 내용이 없으면 열리지 않는다.
  O_WRONLY 는 쓰기 전용 파일로 지정한다.
  O_TRUNC 는 이전에 저장되어 파일의 내용을 지운다.

1-1과 1-2의 차이는 빨간 색으로 표시된 하나의 문자 때문에 결과 값이 달라지게 된다.

```
1-1   if((filedes=open(pathname,O_CREAT|O_RDWR,0644))==ERROR)
1-2   if((filedes=open(pathname,O_CREAT|O_RDWR|0644))==ERROR)
```

Open 명령어는 숫자 값을 반환하게 되는데 그 값이 ERROR(-1)을 만족하면 “File Open Error!”라는 문자를 출력하고 빠져나오게 되며, 반환 값이 -1이 아닌 경우에는 filedes 에 저장된 open 의 반환 값이 출력되게 된다.

2.creat()
~소스

```
#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;
}
```

~결과

```
xeno@xeno-NH:~/proj/0319$ gcc 2.c
xeno@xeno-NH:~/proj/0319$ ./a.out
xeno@xeno-NH:~/proj/0319$ ls
1.c   2.c   a.out   data1.txt   data2.txt   temp.txt
```

Open()과 creat()둘 다 새로운 파일을 생성할 수 있다.
처음 실행했을 때에는 data1과 data2의 txt 파일이 생성되고 data1에 값을 입력한 뒤 다시 실행하면 입력했던 값이 지워진다.
입력했던 값이 지워지는 이유는 O_TRUNC 라는 oflag 때문이다.

3. fd

~소스

```
#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)
       {        //nread 는 내가 몇 바이트를 읽었나를 받게 됨

                if(write(fdout,buf,nread)<nread)
                {
                        close(fdin);
                        close(fdout);
                }
        }
        close(fdin);
        close(fdout);

        return 0;
```

}
~결과
temp1.txt 라는 파일을 만들어 내용을 기입하고 소스파일을 실행시키면 temp2.txt 가 생성되고 temp1에 있던 내용이 입력된다.

cp 명령어를 만듦?

//read(fd,buf,읽을 크기)
//write(fd,buf,쓸 크기)

- system call
close
open
read
write
하드웨어 연산이 들어가면 전부 system call 이다.

4.
~소스

```
#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);
                        //(off_t)0 시작을 0부터하겠다
//newpos 에 파일의 사이즈를 저장
//lseek :파일사이즈
printf("file size : %d₩n",newpos);
}
```

~결과

```
file size : 7
```

data1.txt 에 123456을 넣고 실행시키면 7이 출력됨

5. cp 명령어 만들기

~소스

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

int main(int argc,char**argv)
{
	int i,fdin,fdout,nread;
	char buf[1024]={0};

	if(argc!=3)
	{
		printf("인자 입력 3개");	// 인자 입력이 3개가 되지 않으면 오류처리 해버린다.
		exit(-1);
	}
	for(i=0;i<argc;i++)
	{
		printf("입력한 인자 = %s₩n",argv[i]);	//argv 배열에 입력된 인자를 출력한다.
	}

	fdin = open(argv[1],O_RDONLY);
	fdout = open(argv[2],O_WRONLY|O_CREAT|O_TRUNC);
```

```
        while((nread = read(fdin,buf,1024))>0)
        {
                if(write(fdout,buf,nread)<nread)
                //오류가있으면  작업중단
                {
                        close(fdin);
                        close(fdout);

                        exit(-1);
                }
        }
        close(fdin);
        close(fdout);

        return 0;
}
```

~결과

```
xeno@xeno-NH:~/proj/0319$ ./a.out test.txt test2.txt
입력한  인자  = ./a.out
입력한  인자  = test.txt
입력한  인자  = test2.txt
```

test.txt 파일이 test2.txt 파일로 그대로 복사됨 test.txt 파일은 그대로있음.

6.

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

int main(void)
{
	FILE *fp= fopen("mycat.c","r");
	char buf[1024]="₩0";
	int ret;
	while(ret=fread(buf,1,sizeof(buf),fp))
	//fp 파일을 1바이트씩 1024만큼 읽어서 buf 에 넣어라
	{
		usleep(1000000);
		//us:마이크로 세컨드
		// 1000000us 씩 출력하라
		fwrite(buf,1,ret,stdout);

	}
	fclose(fp);
	return 0;
}
```

~결과

```
xeno@xeno-NH:~/proj/0319$ ./a.out 6.c
#include<stdio.h>
#include<unistd.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;
}
```

./a.out 6.c 를 입력하면 6.c 에 저장되었던 소스파일이 그대로 출력된다.

궁금점
FILE
fwrite
fclose

7.

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

int main(int argc,char **argv)
{
        int fd,ret;
        char buf[1024];
        if(argc !=2)
        //argc :
        //argv : 문자열
        {
                printf("usage : mycat filename₩n");
                exit(-1);
        }
        fd=open(argv[1],O_RDONLY);
        while(ret=read(fd,buf,sizeof(buf)))
        {
                write(1,buf,ret);
                //1번-표준출력 (모니터에 출력)
        }
        close(fd);
        return 0;
}
```

```
xeno@xeno-NH:~/proj/0319$ gcc 7.c
xeno@xeno-NH:~/proj/0319$ ./a.out 7.c
#include<stdio.h>
#include<fcntl.h>
#include<stdlib.h>
#include<unistd.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;
}
```

6번예제와 마찬가지로 7.c 의 소스파일이 그대로 모니터에 출력된다. 6번과 7번의 차이점은 7번은 system call 을 이용하여 처리속도가 매우 빠르지만 6번은 7번에 비해 처리속도가 느리다.

8.

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

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

        int fd,ret,nread,file1,nr;
        char buf[1024]={0};

        nr=read(0,buf,sizeof(buf));
        printf("%d",nr);
        write(1,buf,read(0,buf,nr));

        return 0;
}
```

~결과

내가 키보드로 입력한 문자를 그대로 출력 가능하다. scanf 에서는 ‘ ‘가 되지 않았었는데 8번 예제와 같이 system call 을 이용하면 ‘ ‘를 입력할 수 있다.

9. 파일나누기

-9.c

```
#include<stdio.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;
}
```

-my_scanf.c

```
#include "my_scanf.h"

int my_scanf(char *buf,int size)
{
```

```
int nr=read(0,buf,size);
return nr;
}
```

-my_scanf.h

```
#ifndef __MY_SCANF_H__
#define __MY_SCANF_H__

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

int my_scanf(char *,int);
#endif
```

~결과

```
xeno@xeno-NH:~/proj/0319$ gcc 9.c my_scanf.c
xeno@xeno-NH:~/proj/0319$ ./a.out
adfagag a adf a
nr=16
adfagag a adf a
```

위의 8번 함수를 여러 파일로 쪼개보았다.

-헤더파일 만들기
<> → 시스템 헤더, 라이브러리 헤더
“” → 사용자 정의 헤더

#ifndef __MY_SCANF_H__
#endif
위의 2개는 하나의 세트처럼 사용해야 하고 헤더파일이 여러 번 중복되어 사용 될 경우 오류를 내보낸다.
#define __MY_SCANF_H__
헤더파일을 만들어서 사용할 때에는 위의 3 문장이 기본적으로 깔려 있어야한다.
이렇게 파일을 쪼개어 소스코드를 작성하는 이유는 소스코드를 많이 작성하다 보면 너무 소스코드의 양이 많아 컴파일 자체가 안될 때가 있고 사용자가 보기 편하게 하기 위한 목적도 있다.

10. wc 명령어 : 갯수를 세는...

~소스

```
#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)   //인자가 2개가 들어오지 않았을 때 밑의 문장들을 출력시킨다.
        {
                printf("You need 1 more parameter₩n");
                printf("usage : mywc filename₩n");
                exit(-1);
        }

        if((fd = open (argv[1],O_RDONLY)<0))
        {
```

```
		perror("opem()");	//어떤 오류가 있는지 나타내는 명령어이다.
		exit(-1);
	}
	while(read(fd,&ch,1))		//ch 변수를 ch[1]의 배열처럼 사용한다. Fd 파일을 1바이트씩 ch 변수를 읽는다.
	{
		cnt++;		//while 문이 한번 돌 때 마다 카운팅되어 띄어쓰기 라인 바꾸기,문자, 숫자 모두를 카운팅한다.
		if(ch=='₩n')
			line++;		// ₩n 을 사용할 경우, 즉 줄바꾸기를 사용할 경우 line 변수값을 1 증가시킨다.
		if(ch != '₩n' && ch!='₩t' && ch != ' ')	//문자,숫자만이 해당되는 경우
		{
			if(flag==0) //단어를 카운팅하기 위한 조건식
			{
				word++;
				flag=1;
			}
		}
		else
		{
			flag=0;		//띄어쓰기나 tap, 줄바꾸기 등이 실행되면 flag 를 0으로 만들어 다시 if 문으로 들어갈 수
있도록 조건식을 바꾼다.
		}
	}
	close(fd);
	printf("%d %d %d %s₩n",line,word,cnt,argv[1]);
	return 0;
}
```