TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
18일차 (2018. 03. 19)

목차

학습 내용 복습

- 운영체제?
- 모놀리식 커널과 마이크로 커널
- System call
- 리눅스 명령어
- 관련 예제

-운영 체제란? Operating system

컴퓨터의 하드웨어를 제어하고 응용 소프트웨어를 위한 기반 환경을 제공하여, 사용자가 컴퓨터를 사용할 수 있도록 중재 역할을 해 주는 프로그램을 말한다.

- 리눅스 : 유닉스를 기반으로 개발한 공개용 오퍼레이팅 시스템이다. 리눅스는 소스 코드를 완전 무료로 공개하여 전세계적으로 약 5백만 명이 넘는 프로그램 개발자 그룹을 형성하게 되었으며, 이들에 의해 단일 운영체제의 독점이 아닌 다수를 위한 공개라는 원칙하에 지속적인 업그레이드가 이루어지고 있다.

Software는 크게 3가지로 나뉜다.

1. OS
2. Compiler
3. DB

Free SW : 자유, 책임 x

Open SW : 책임 o 즉, 오픈 소프트웨어는 라이센서를 생각해야 한다.

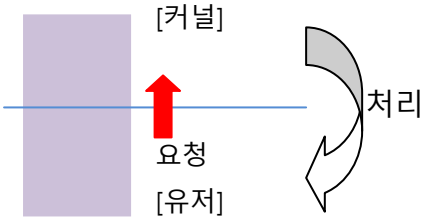
-모놀리식 커널과 마이크로 커널

마이크로 커널 : 조립형식으로 가능하다는 개념으로 붙이고 떼는 것이 된다.

모놀리식 커널 : 마이크로 커널 개념을 탑재하고 있으며 필요한 운영체제가 몇 가지 있다.

'디바이스 드라이버'- 레고블럭 같은 개념이 있다.

-system call이란?



시스템콜은 유저가 어떤 상황을 요청하면 커널에서 처리하고 다시 유저에게 돌려준다.

* system call : 유저가 커널에게 요청하는 작업을 의미한다.

뒤에 소스 코드에 나오는 close open read write 등 하드웨어 연산이 들어가면 전부 system call 이다.

- 리눅스 명령어

- mv [이름 변경할 파일명] [변경 이름] : 파일 이름 변경
- cp [복사할 파일명] [복사 될 파일명] : 파일 복사
- rm-rf [파일명] : 파일 삭제 ('rm –rf /' 는 사용하지 말 것!)
- ls : 현재 디렉토리에 있는 list
- cd [디렉토리명] : 디렉토리 이동
- mkdir [파일명] : 디렉토리 생성
- gcc [파일명] : 프로그램 디버깅
- mkfifo [옵션] [파일명] : FIFO 생성
- touch [옵션] [파일명] : 파일 변경시간을 시스템 현재 시간으로 변경
- cat [파일명] : 파일 밖에서 내용 보기
- tar [옵션] [파일명1/결과] [파일명2/대상] : 다수의 파일이나 디렉토리를 하나의 파일로 묶기

핵심 철학 : 모든 것은 파일이다.

Warning 나올 시에 ' #include <unistd.h> ' 를 넣어준다.

open () 이나 creat() 는 시스템 호출로 파일을 열어 파일을 생성한다. 또한 open(), creat()를 쓴 경우, close()

(=공간을 해제하라)를 꼭 해주어야 한다.
Main 함수에 인자를 받을 때,
int main(int argc, char **argv)
argc : 받을 파일 , 숫자 , 문자 등의 개수
argv : 받은 것들 마다의 주소(이중 포인터)

-예제 1

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

#define ERROR    -1

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

  if((filedes = open(pathname, O_CREAT | O_RDWR | O_EXCL, 0644))==ERROR)
  {
    printf("File Open Error!\n");
    exit(1);
  }
//    printf("fd=%d\n", filedes);
   close(filedes);

return 0;
}
~
```

> temp.txt 생성 / 터미널 창만 켜 있을 시에는 보통 3이 출력 된다.
Open 명령어는 숫자 값을 반환하게 되는데 그 값이 ERROR(-1)을 만족하면 "File Open Error!"라는 문자를 출력하고 빠져 나오게 되며, 반환 값이 -1이 아닌 경우에는 filedes 에 저장된 open 의 반환 값이 출력되게 된다.
0 = 표준입력(키보드) 1 = 표준 출력(출력) 2 = 표준 에러 3, 4, ... = 그 외 파일들
첫 실행을 제외하고선 if문을 통과하지 못하는데 O_EXCL의 특성 때문이다.

* 여기서 open 이라는 명령어는 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 생성하지 않는다.
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.

0644는 파일 접근 권한이다.

&(and) 연산이 아닌 | (or) 연산을 하는 이유는 '모든 옵션을 다 넣겠다.'는 의미이다. 쉽게 and는 교집합, or은 합집합이라고 생각하면 된다.

-예제 2

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

> data1.txt / data2.txt 를 생성한다.

open()과 creat()가 파일을 생성하는 함수이기 때문이다. 또한, data1에 값을 입력한 뒤 다시 실행하면 입력했던 값이 지워진다. 입력했던 값이 지워지는 이유는 O_TRUNC 때문이다.

-예제 3

```c
#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);

  return 0;
}
```

> temp1.txt을 생성해서 내용이 기입되어 있는 경우, temp2.txt가 생성되어 동일한 내용이 적힌다.

file descriptor(fd) : 파일 식별자라고 생각하면 된다. 숫자 값을 리턴한다.

[0 = 표준입력(키보드) 1 = 표준 출력(출력) 2 = 표준 에러 3, 4, ... = 그 외 파일들]

여기서 nread 는 내가 몇 바이트를 읽었나를 받게 된다. 0보다 작으면 읽을 것이 없다.

read(fd,buf,1024) → fd: 파일 식별자, buf :읽을 내용이 들어감, 1024: 총 읽은 크기

write(fd,buf,nread) → fd: 파일 식별자, buf : 쓸 내용이 들어감, nread: 총 쓸 크기

즉, read(fd,buf,읽을 크기) > 몇 byte 읽었는지가 리턴 된다. / write(fd,buf,쓸 크기) > 몇 byte를 썼는지가 리턴된다.

-예제 4

```c
#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);

  printf("file size :%ld\n", newpos);
```

< 실수로 return 0; 이 그림에서 잘림.

> data1.txt에 hello를 입력하면 file size : 6이 출력된다. > 한글자 +1로 출력 된다.

lseek()에서 filedes 식별자를 통해서 파일의 처음부터 끝까지 회전해서 파일의 사이즈를 newpos에 저장한다.

파일을 많이 만들었을 경우, 현재 올라온 파일의 용량이 알고 싶을 때, lseek()를 쓴다.

여기서 (off_t)0은 '시작을 0부터 하겠다.'는 뜻이다. 즉, **lseek(fd, 시작위치, 길이)**로 몇 byte로 읽었는지 리턴된다.

*cp 명령어 만들기

sue100012@sue100012-Z20NH-AS51B5U: ~/project/3_19

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


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


   if(argc != 3)
   {
     printf("인자 입력 3개 하라고!\n");
      exit(-1);
   }

   for(i=0; i < argc; i++)
   {
     printf("당신이 입력한 인자는 = %s\n", argv[i]);
   }

  in = open(argv[1], O_RDONLY);
  out = open(argv[2],  O_CREAT | O_WRONLY |O_TRUNC);

 while((nread = read(in, buf, 1024))>0)
  {
    if(write(out, buf, nread) < nread)
     {
      close(in);
      close(out);
     printf("비정상 종료\n");
       exit(-1);
     }
  }

   close(in);
   close(out);
   printf("정상 종료\n");

   return 0;
}
~
~
```

< 인자가 3개 입력 되지 않으면 오류 처리가 된다.

< argv 배열에 입력된 인자를 출력한다.

< 오류가 있으면 작업을 중단시킨다.

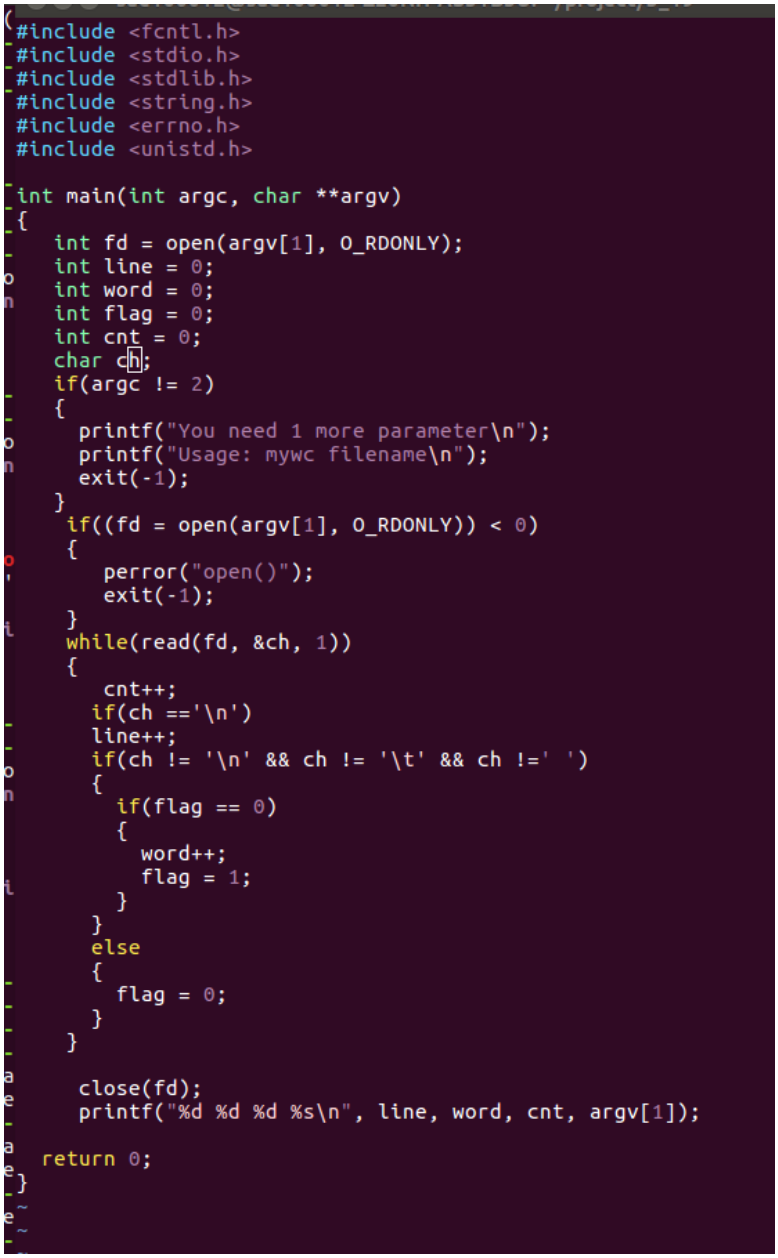
> ./a.out text.txt text2.txt 로 실행하면 test.txt 파일이 test2.txt 파일로 복사된다. test.txt 파일은 변경되지 않는다.

* scanf에서 받지 못했던 띄어쓰기 입력 받기

약간 변형된 형태로 사용자 정의의 헤더 파일을 만들어준 경우다.
헤더파일
<> : 시스템 헤더, 라이브러리 헤더
" " : 사용자 정의 헤더

#ifndef __MY_SCANF_H__ #endif 위의 2개는 하나의 세트처럼 사용해야 한다. #endif 가 없을 경우 에러가 발생한다. 헤더파일이 여러 번 중복되어 사용 될 경우 오류를 내보내기 때문이다. 중복 선언을 방지해준다. 어떠한 경우라도 함수는 이름이 같으면 안 된다.#define __MY_SCANF_H__ 으로 사용할 함수를 정의해주면 된다. define은 ifndef로 정의가 안 된 것을 정의해준다. 그 후, 프로토타입을 기입해준다. 헤더파일을 만들어서 사용할 때에는 위의 3 문장이 기본적으로 깔려 있어야 한다.

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.
:sp 화면을 행단위로 나눠준다. :e 파일명 파일을 만들거나 있는 것들을 vi 모드로 킬 수 있다.
:vs 화면을 수직 단위로 나눠준다. ctrl+w 방향키 화면을 바꾼다. 바꾼 곳에서 쓰려면 'w' 를 누르면 된다.



```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.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 need 1 more parameter\n");
      printf("Usage: mywc filename\n");
      exit(-1);
    }
     if((fd = open(argv[1], O_RDONLY)) < 0)
     {
        perror("open()");
        exit(-1);
     }
     while(read(fd, &ch, 1))
     {
        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;
}
```

> perror ("open()") = print error 라고 이해하면 쉽다. 전달 인자만 없고 error를 프린터한다.

&ch : 배열로 들어가서 주소값을 받는 것이라고 생각하면 된다.
좀 더 쉽게 이해하자면 char ch;가 결국 한 문자니까 char ch[1]; 와 비슷하다고 생각하여 이해하였습니다.
if문에서 다시 flag가 0이면 word(단어 수)가 증가하고 flag를 1로 만든다.
띄어쓰기가 나오면 else로 가서 flag를 0으로 초기화한다. 이는 flag로 단어를 세기 위함이다.