

삼성 청년 SW 아카데미

시스템 프로그래밍

<알림>

본 강의는 삼성 청년 SW아카데미의 콘텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

Day1-1. 시스템 프로그래밍 개요

시스템 프로그래밍은
컴퓨팅 시스템에서의 SW개발을 의미하며,
System Call 을 사용하여 Application 개발을 하는 수업이다.

시스템 / 네트워크 / 커널 프로그래밍으로 분류가 되어 있지만,
큰 틀에서는 모두 시스템 프로그래밍이다.

- 시스템 프로그래밍 수업에서 배운 내용을 네트워크/커널 프로그래밍에서 적용할 수 있다.
- 임베디드 SW 개발자라면, 반드시 알아야 하는 가장 기본적인 내용들로 구성되어 있다.

꼭! 연습하고, 복습하여 본인의 것으로 만들어야 한다!

1일차 : system call 소개, open/close/read/write 연습 등

2일차 : Thread 프로그래밍 등

3일차 : Process, System Architecture 등

Only Linux 만 사용

- CLI 환경 : mobaXterm
- vi 로 개발

시스템 프로그래밍은 사실 매우 방대하고 내용이 깊은 내용이다.

우린 그 기초를 배우고 학습하며, 면접을 대비한 CS 관련 역량도

많이 키우게 된다!

Day1-2. System

챕터의 포인트

- 시스템의 이해
- POSIX의 이해

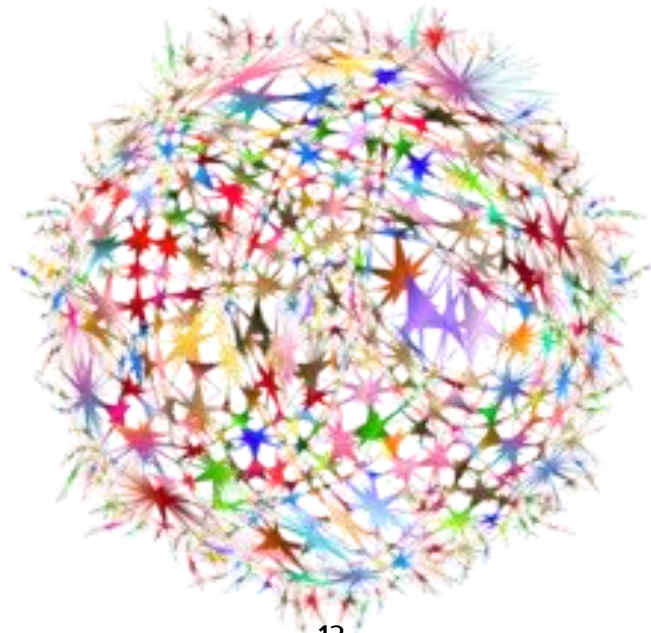
시스템의 이해

목표

- 시스템이란 무엇이고, 컴퓨팅 시스템과 임베디드 시스템의 차이를 이해한다.
- 시스템 프로그래밍이란 무엇을 의미하는 것인지 알아본다.

구성 요소들이 상호 작용하는 집합체를 시스템이라 한다.

- 어떤 구성 요소들이 모여서 서로 통신하면 시스템이다.
- 통신을 한다?! 데이터를 주고 받는다. 신호를 주고 받는다.
- 사람의 몸도 시스템이다.



CPU, 기억장치, 입출력장치 등이 상호작용을 하는 집합체



컴퓨팅 시스템 중, 전용 기능을 수행하도록 만들어진 시스템

- PC와 달리 특정 목적을 가짐

Firmware 는 임베디드 시스템이다.

OS 는 컴퓨팅 시스템 (범용) 이다.



특정한 목적이 있음 (ex. CCTV)

컴퓨팅 시스템 구성 1. HW

- HW

- CPU
- 메모리
- 페리퍼럴 (Peripheral, 주변장치)
 - 저장장치
 - Graphic
 - 입출력장치
 - LAN Card
 - USB Interface 등등



CPU, 메모리 외

모든 입력 출력을 하는 장치를 페리퍼럴 이라고 부른다.

SSD는 대표적인 입력 장치이다.

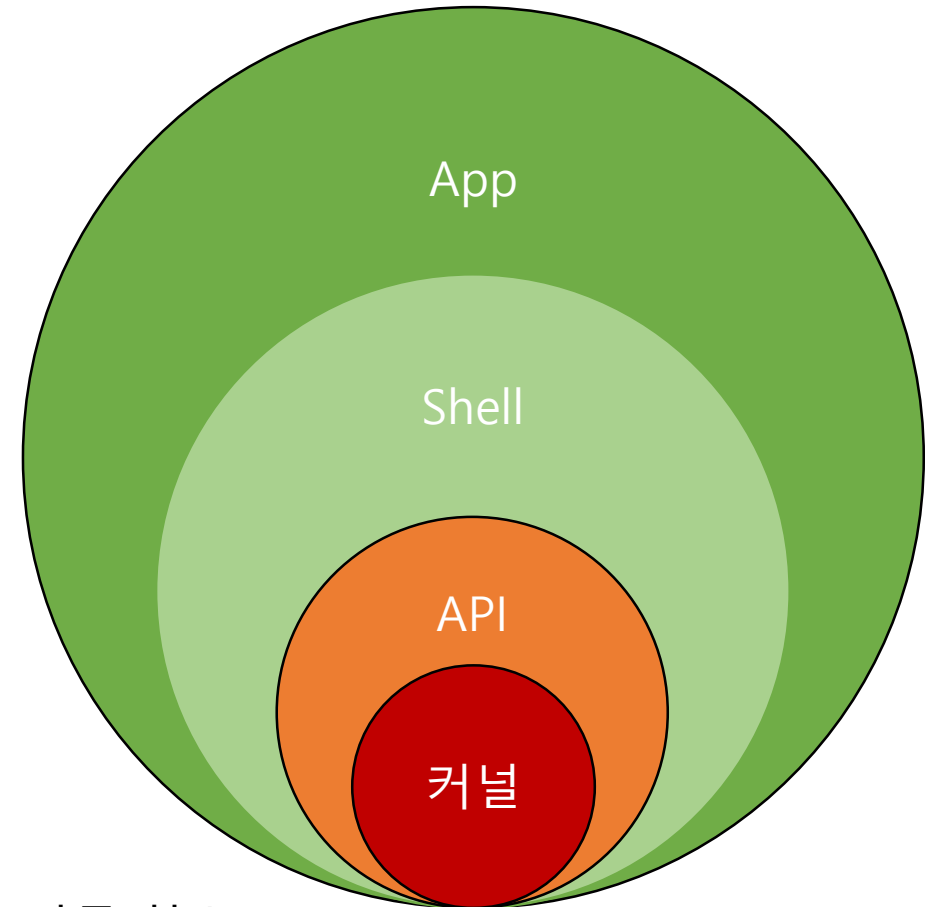
- SSD는 Nand Flash memory이며 메모리 어드레싱 방식이 아닌, 섹터 어드레싱 방식이라, 보조기억장치로 분류된다!
- USB 스틱이 바로 Flash memory!



<https://news.samsungsemiconductor.com/kr/category/%ea%b8%b0%ec%88%a0/%ec%9a%a9%ec%96%b4%ec%82%ac%ec%a0%84/page/2/>

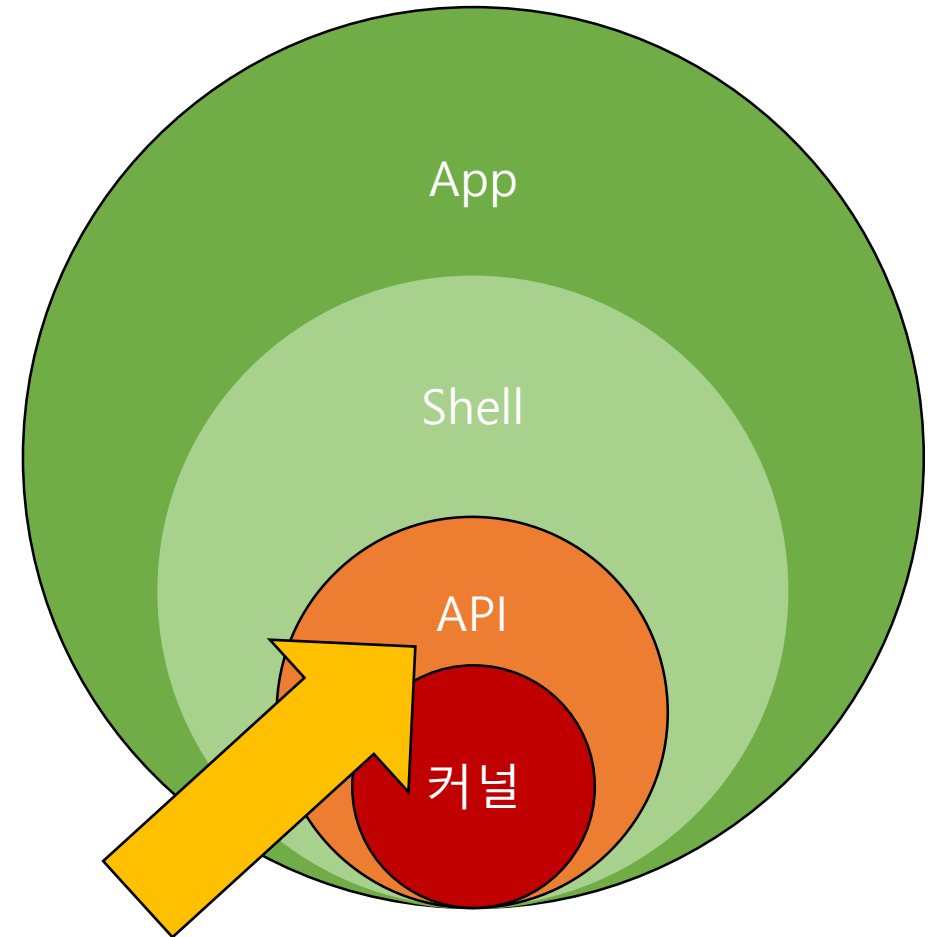
컴퓨팅 시스템 구성 2. SW

- **Application Level**
 - App
 - Shell
- **Middleware Level**
 - API , Library
- **Low Level**
 - ex) 리눅스 커널



API : Application Programming Interface, App Level이 쓰라고 만든 함수
Library : API 모음집

리눅스에서 App이
커널의 기능을 쓸 수 있도록 만든
API를 뭐라고 부를까요?



결론 : 시스템 프로그래밍 수업에서
시스템 프로그래밍이란 무슨 뜻인가!?

- 1) 시스템이라는 것은
컴퓨팅 시스템에서 S/W 개발을 의미하며,
- 2) System Call 을 사용한 Application 을 개발하는 수업이
시스템 프로그래밍 이다.

POSIX의 이해

목표

- 시스템 개발을 편리하게 해준 POSIX에 대해 학습한다.
- POSIX와 System call의 차이를 이해한다.

App 개발자들은 OS가 제공하는 API 를 알아야 한다.

- 임베디드 제품마다 사용되는 OS가 다를 수 있다.
- 다양한 OS와 다양한 OS 개발자들이 있었다.
 - ??? : linux 할 줄 아니까 iOS도 해줘?!



Application 개발자들을 위해
OS마다 제공되는 API들을
하나로 통일하였다.

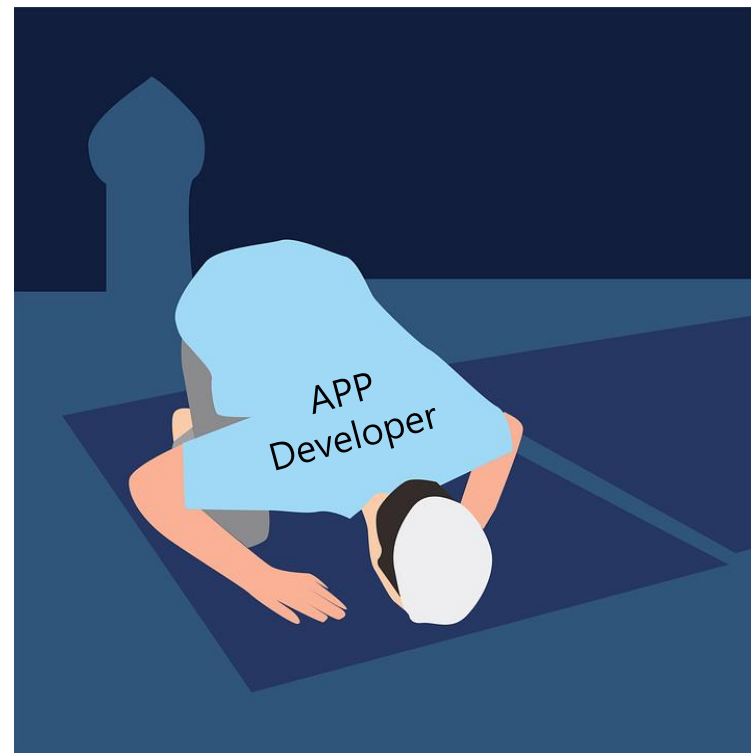
- 통일된 API 의 이름은?



포직스란 ?

- OS 들이 지원하는 API 들의 표준 규격
- IEEE에서 제정

POSIX API 만 배워두면
여러 임베디드 OS에서도
편리하게 App 개발이 가능하다.



Quiz 1.

POSIX 함수 형태는 똑같지만, 내부 구현은 OS마다 다를 수 있을까?

→ OS마다 구현 하는 방법이 다르다! , POSIX는 Interface 표준일 뿐

Quiz 2.

Linux 에서 POSIX API로 개발한 C언어 소스코드가 있다.

이것을 VxWorks같은 다른 운영체제 에서 빌드하면, 잘 동작할까?

- 어떤 OS 개발이든, POSIX 표준으로 개발한다면, 다른 OS에서도 동작하게 된다!

Quiz 3.

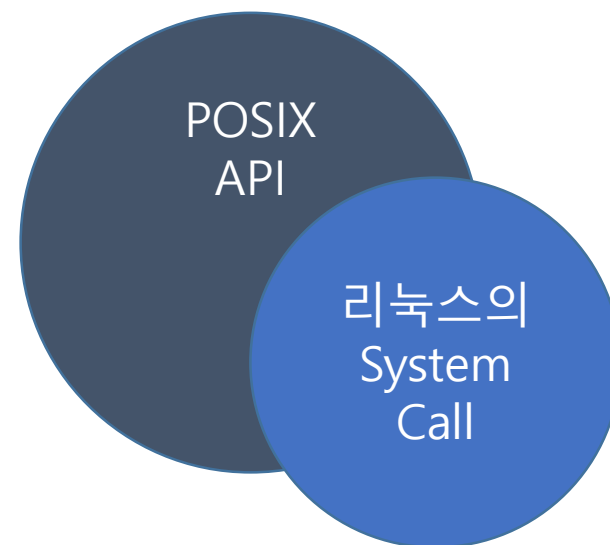
Windows App / Android App 개발할 때도, POSIX를 쓸 수 있을까?

- Windows 는 win8 부터 POSIX 지원이 안된다.
- 안드로이드는 일부만 지원한다.

POSIX : OS가 App에게 제공하는 **API들의 표준**

System Call : **리눅스**가 App 개발자들을 위해 제공하는 API

System Call에는
POSIX API 호환도 있고
아닌 것도 있다.



APP

- 전용 OS에 맞는,
맞춤 제작 필요 했음 (매번 새로 개발)

OS

- 맞춤 제작 필요 (매번 새로 개발)

HW

- 맞춤 제작 필요 (매번 새로 개발)
- ??? : 오늘 야근이야

자체 OS로 개발된
국산 게임기 GP32



APP

- System Call API (리눅스 인 경우에만)
- POSIX API 사용 (리눅스 / RTOS)

OS

- 자체 제작함
 - Firmware 개발
- 기존 만들어진 OS를 사용함
 - RTOS
 - Linux
 - Android

HW

- 맞춤 제작 필요 (매번 새로 개발 필요)



Unix 계열 OS 사용

OS를 쓰는 임베디드 시스템의
Application 개발자들은
POSIX API 를 쓰면서 App을 만들어낸다.

Day1-3. System Call

챕터의 포인트

- Low Level API
- open(), read(), write(), close()
- 파일 offset 과 lseek()

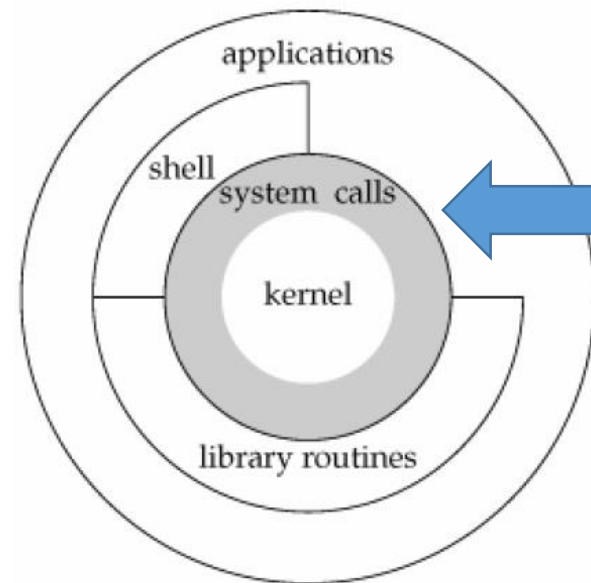
Low Level API

목표

- Low Level API 와 High Level API 에 대해 학습한다.
- System Call 인 open, read, write, close, lseek 에 대해 학습한다.

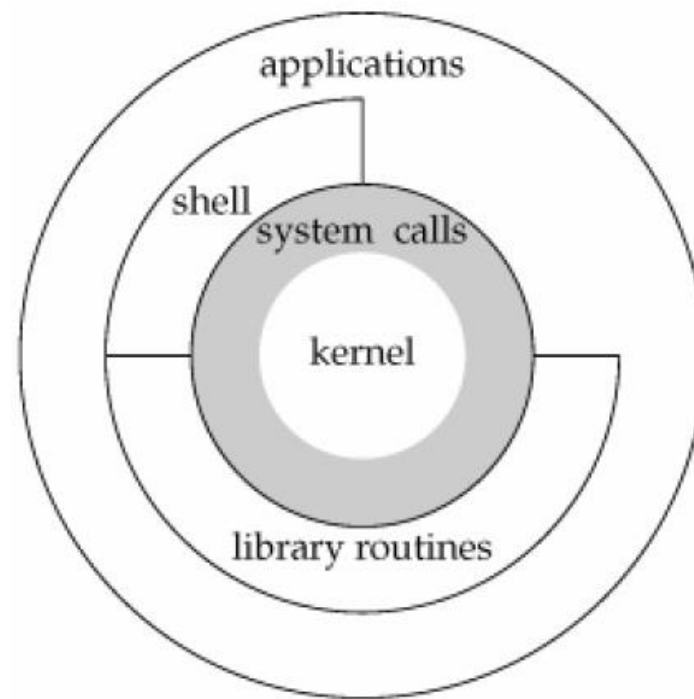
리눅스 App이
리눅스 커널에게 어떤 부탁을 하기 위해 만들어진 API를
System Call 이라고 한다.

- System Call 을 줄여서 **Syscall (시스콜)** 이라고 한다.



printf()/scanf() 는 System Call 로 만들어졌을까?

- printf() / scanf() 는 커널의 도움이 필요할까?



printf(), scanf() 는 System Call 로 만들어졌다.

- **printf()**

- 디스플레이 장치에 글자가 출력 되어야 한다.
- 커널에게, 그래픽 카드를 이용해 글자를 출력해줘 라는 부탁을 Syscall 로 해야 한다

- **scanf()**

- 키보드라는 입력장치의 동작을 읽어야 한다.
- 마찬가지로, 커널에게 부탁해야 한다.

파일에 값을 쓰고, 읽기 위해 사용하는
fopen 은 Syscall로 구현되어 있을까?

`fopen()` 은

Syscall 중 하나인 “`open()`” 을 사용한다.

- 커널은 모든 장치들을 관리한다.
- `open()` syscall로 커널에게 부탁을 한다.
 - Disk 저장장치에 저장된 값을 읽어주세요.

`fopen()` / `printf()` / `scanf()` 등등

- Syscall을 포함한 **Wrapper**(감싼) 함수
- **High Level API** 라고 부른다.

`open()` / `read()` / `write()` / `close()` 등등

- 커널의 도움을 받아 동작하는 Syscall 중 하나.
- **Low Level API** 라고 부른다.

파일처리 하는 System Call 을 주로 다룰 예정이다.

- `open()` : `fopen`이 사용하는 syscall
- `read()` : `fscanf`가 사용하는 syscall
- `write()` : `fprintf`가 사용하는 syscall
- `close()` : `fclose`가 사용하는 syscall

시스템 프로그래밍 시간에 배운 system call 을 이용해,
네트워크 프로그래밍, 커널 프로그래밍 때 사용한다.

open, read, write, close

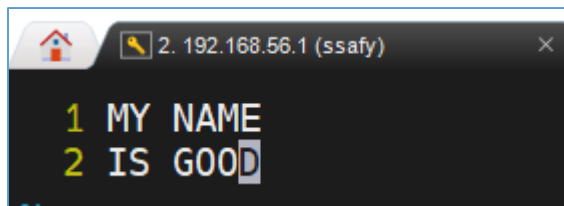
~/test/ 디렉토리 생성

- test.txt
- sample.c

sample.c

```
8  #include <stdio.h>
9
10 int main(){
11     int fd = open("./test.txt", O_RDONLY, 0);
12     if( fd<0 ){
13         printf("ERROR\n");
14         exit(1);
15     }
16
17     char buf[1000] = {0};
18     read(fd, buf, 1000-1);
19
20     printf("%s\n", buf);
21     close(fd);
22     return 0;
23 }
```

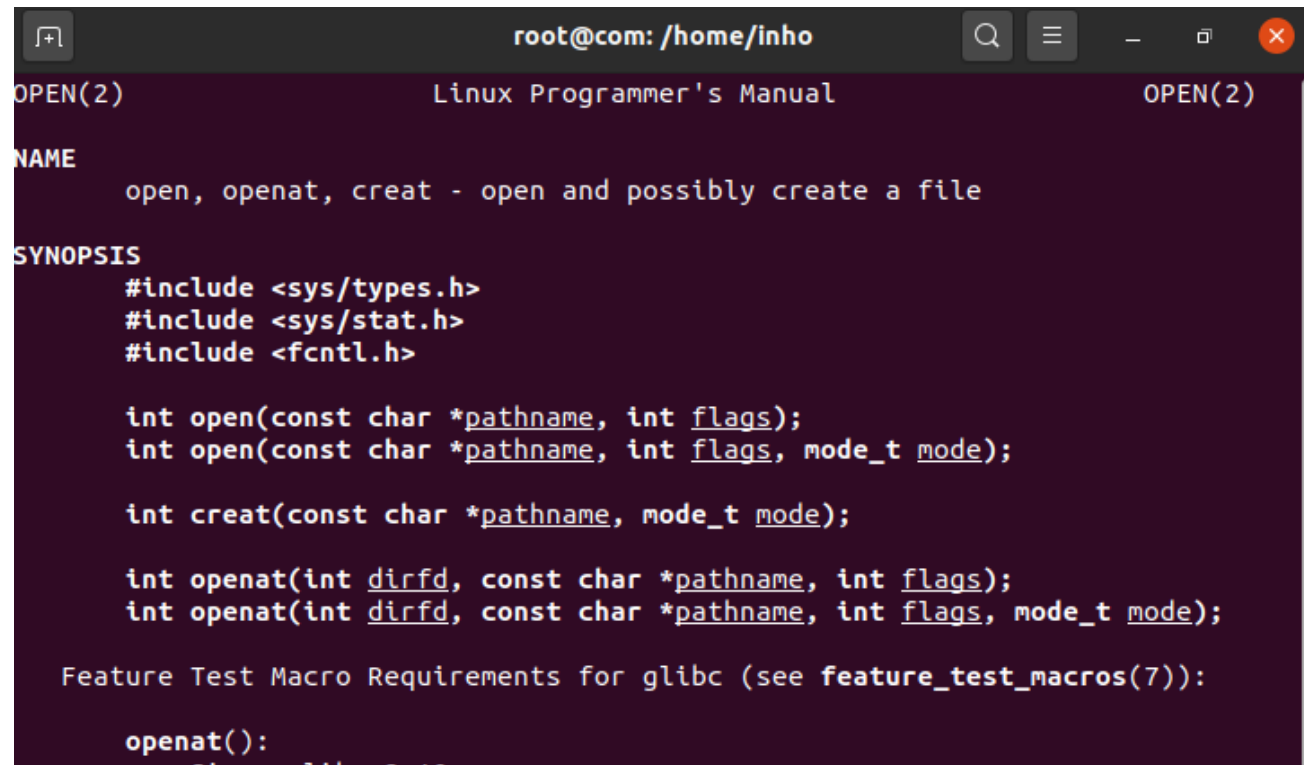
test.txt



<https://gist.github.com/hoconoco/36368c8e2b4b18a4e9baace8099e4508>

System Call API Interface 생각이 안 날 때는 2 + shift + k

- vim 에 API 이름을 적는다. (ex: open , command mode로 esc 키)
- [page 숫자] + shift + k
- 괄호에 있는 숫자는 종류를 뜻한다.
 - 1 : Linux Shell Command
 - 2 : System Call
 - 3 : Linux Library
- 필요 Header File과 함수 Interface를 확인할 수 있다.



```
root@com: /home/inho
OPEN(2) Linux Programmer's Manual OPEN(2)

NAME
    open, openat, creat - open and possibly create a file

SYNOPSIS
    #include <sys/types.h>
    #include <sys/stat.h>
    #include <fcntl.h>

    int open(const char *pathname, int flags);
    int open(const char *pathname, int flags, mode_t mode);

    int creat(const char *pathname, mode_t mode);

    int openat(int dirfd, const char *pathname, int flags);
    int openat(int dirfd, const char *pathname, int flags, mode_t mode);

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

    openat():
        Since glibc 2.10:
```

코드를 작성하고 빌드한다.

- gcc 를 이용해 빌드한다.
- gcc sample.c -o ./gogo

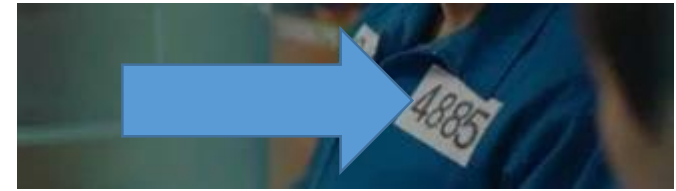
test.txt 의 정보를 읽어서 출력한다.

```
ssafy@ssafy-VirtualBox:~/test$ gcc sample.c -o ./gogo
ssafy@ssafy-VirtualBox:~/test$ ls
gogo sample.c test.txt
ssafy@ssafy-VirtualBox:~/test$ ./gogo
MY NAME
IS GOOD
```

```
8  #include <stdio.h>
9  #include <sys/types.h>
10 #include <sys/stat.h>
11 #include <fcntl.h>
12 #include <unistd.h>
13 #include <stdlib.h>
14 int main(){
15     int fd = open("./test.txt", O_RDONLY, 0);
16     if( fd<0 ){
17         printf("ERROR\n");
18         exit(1);
19     }
```

File Descriptor (Number)

- 한 프로그램이
파일에 접근하려고 할 때, 부여되는 정수를 뜻함



```
int fd = open("./test.txt", O_RDONLY, 0);
```

한 감옥에서 죄수들의 번호를 부여하는 것처럼

한 프로그램에서 사용되는 파일에게 **번호를 부여하는** 것이다.

```
int open(const char* path, int flag, mode_t mode)
```

flag

- 필수 옵션 (셋 중 하나 선택)
 - O_RDONLY
 - O_WRONLY
 - O_RDWR
- 추가옵션 (or 연산 사용)
 - O_CREAT : 없으면 새로 생성 (표기 주의)
 - O_APPEND : 덧붙이기
 - O_TRUNC : 파일 내용 제거 후 사용

int open(path, flag, mode)

mode

- 파일 생성할 때 파일 권한을 줄 수 있다. 0xxx 값을 넣으면 된다. (8진수)
 - 0777 : rwxrwxrwx
 - 0644 : rw-r--r--
- Open할 때 파일 권한은 없어도 된다.
- 기존에 작성된 파일에 대해서는 무시된다.
- other에 대한 권한은 보안상의 정책으로 무시될 수 있다고 한다.

open()은 error가 자주 일어난다.

- open() 의 return 값인 fd 는 정상 동작일 경우, 양수이다.
- 음수 값일 때는 error를 뜻한다.
- error 처리에 대한 로그 메시지를 꼭 남기자.
- exit(0) : 정상 종료
exit(1) : 오류로 인한 종료

```
12  int main(){
13      int fd = open("./test.txt", O_RDONLY, 0);
14      if( fd<0 ){
15          printf("[%s :: %d] FILE OPEN ERROR\n", __FILE__, __LINE__);
16          exit(1);
17      }
```

close(fd)

파일디스크립터만 적어주면 된다.

`ssize_t read(int fd, void buf[.count], size_t count)`

- `<unistd.h>` 필요
- return 값 : 동작이 성공하면 읽은 바이트 수를 반환, 오류 시 -1
- 파일을 읽어서 buf에 크기만큼 저장한다.

~/test2

- test.txt
- sample.c
- gcc sample.c -o ./gogo

 test.txt

```
1 123456789012345678901234567890
```

```
9  int main()
10 {
11     int fd = open("./test.txt", O_RDONLY, 0);
12     if (fd < 0) {
13         printf("[%s :: %d] FILE OPEN ERROR\n", __FILE__, __LINE__);
14         exit(1);
15     }
16
17     char buf[10] = {0};
18     ssize_t i = read(fd, buf, 10);
19
20     printf("%s %lu\n", buf, i);
21
22     close(fd);
23     return 0;
24 }
```

<https://gist.github.com/hoconoco/40af450f4b9a4aff6ede874d35e93894>

read() 의 return 값은 읽기 성공한 사이즈이다.

- ssize_t 타입 (signed size_t)
 - OS의 bit 상관 없이 개발을 할 수 있다.
- %lu : %ul 은 표준이 아니다.

```
ssafy@ssafy-VirtualBox:~/test2$ ./gogo  
1234567890 10
```

~/test3 디렉토리 생성

- sample.c
- gcc sample3.c -o ./gogo

```
16  int main()
17  {
18      int fd = open("./test.txt", O_WRONLY | O_CREAT, 0664);
19      if (fd < 0) {
20          printf("[%s :: %d] FILE OPEN ERROR\n", __FILE__, __LINE__);
21          exit(1);
22      }
23
24      char* buf = "SSAFY 10th\n";
25      write(fd, buf, strlen(buf));
26
27      close(fd);
28
29      return 0;
30  }
```

<https://gist.github.com/hoconoco/e8e1ba23285e00b963a1457db51e8fae>

실행 파일을 실행하면, test.txt 가 생성된다.

- write() 를 이용해 파일에 문자열 쓰는 샘플 코드

```
ssafy@ssafy-VirtualBox:~/test2$ ./gogo
ssafy@ssafy-VirtualBox:~/test2$ ls
gogo  sample2.c  test.txt
ssafy@ssafy-VirtualBox:~/test2$ cat test.txt
SSAFY 10th
```


`ssize_t write(int fd, const void buf[.count], size_t count)`

- `<unistd.h>` 필요
- `buf`의 내용을 파일에 `count` 만큼 쓰기
- 파일에 내용을 추가할 지, 새로 쓸지를 `open()` 에서 정한다.

~/test4 디렉토리 생성

- test.txt
- sample.c
- gcc sample.c -o ./gogo

```
9  int main(){
10      int fd = open("./test.txt", O_RDWR | O_TRUNC);
11      if (fd < 0) {
12          printf("[%s :: %d] FILE OPEN ERROR\n", __FILE__, __LINE__);
13          exit(1);
14      }
15
16      char buf[10] = "[NEW]";
17      write(fd, buf, strlen(buf));
18
19      close(fd);
20
21      return 0;
22 }
```

<https://gist.github.com/hoconoco/fd59de89f6d1f41e5224d7b1d4afd8fa>

open() 에서 적용한 flag 옵션으로 인해, [NEW] 만 출력된다.

- O_TRUNC

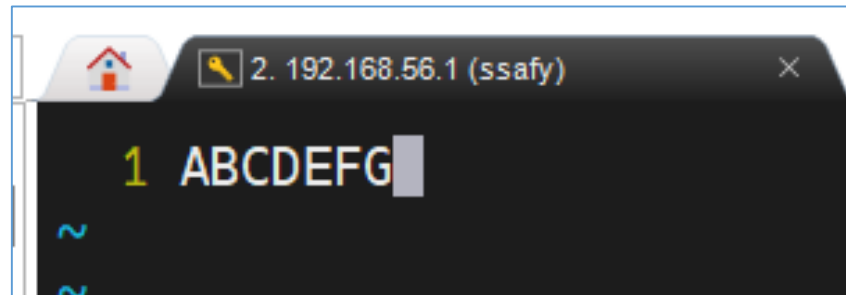
- 파일의 내용을 모두 제거하고 새로 기록을 한다.

```
ssafy@ssafy-VirtualBox:~/test4$ cat test.txt  
[NEW]ssafy@ssafy-VirtualBox:~/test4$ █
```

test.txt 를 다시 수정한다.

Confidential

다시 test.txt 파일을 다음과 같이 만든다.

A screenshot of a terminal window. The title bar shows a home icon, a key icon, and the text "2. 192.168.56.1 (ssafy)". The terminal content shows a line number "1" in yellow, followed by the text "ABCDEFG" in white. A white cursor is positioned at the end of the text. Below this line, the number "2" is visible in blue, indicating the next line in the file.

```
1 ABCDEFG
2
```

코드를 작성한 뒤 빌드한다.

- O_TRUNC 옵션이 없으니, 덮어쓰기가 된다.

```
int fd = open("./test.txt", O_RDWR);
if (fd < 0) {
    printf("[%s :: %d] FILE OPEN ERROR\n", __FILE__, __LINE__);
    exit(1);
}
```

```
ssafy@ssafy-VirtualBox:~/test4$ gcc sample.c -o ./gogo
ssafy@ssafy-VirtualBox:~/test4$ ./gogo
ssafy@ssafy-VirtualBox:~/test4$ cat test.txt
[NEW]FG
```

파일 offset 과 lseek()

파일 오프셋

- 저장장치에서 파일을 어디까지 읽었는지에 대한 Offset 값을 내부적으로 가지고 있다.
- 파일 `read()` / `write()` 할 때, 위치를 조작할 수 있다.

`lseek()` 이용하면, 파일 offset을 조작할 수 있다.

~/test5 디렉토리 생성

- test.txt
- sample.c
- gcc sample.c -o ./gogo

```
19      char buf[10] = {0};
20      int n = lseek(fd, -7, SEEK_END);
21      printf("%d\n", n);
22      read(fd, buf, 6);
23      printf("%s\n", buf);
24
25      memset(buf, 0, 10);
26      n = lseek(fd, 0, SEEK_SET);
27      printf("%d\n", n);
28      read(fd, buf, 10);
29      printf("%s\n", buf);
30
31      memset(buf, 0, 10);
32      n = lseek(fd, 5, SEEK_CUR);
33      printf("%d\n", n);
34      read(fd, buf, 6);
35      printf("%s\n", buf);
```


lseek() 을 이용해서 파일 오프셋 값을 조작하여,
원하는 부분만 read 한다.

- 파일 마지막에는 \0 (널문자) 가 있다.

```
ssafy@ssafy-VirtualBox:~/test5$ gcc sample.c -o ./gogo
ssafy@ssafy-VirtualBox:~/test5$ ./gogo
15
FINISH
0
0123456789
15
FINISH
```

`off_t lseek(int fd, off_t offset, int whence);`

- 기준점 에서 offset 만큼 떨어져 있는 곳으로 파일 offset으로 옮기는 시스템콜
- return 값 : 현재 위치 값
- off_t : 정수 (long), 음수가능
- offset : whence를 기준으로 한 offset 값 (+, - 가 능)
- whence : 기준
 - SEEK_CUR : 현재위치 에서 부터
 - SEEK_SET : 시작점 에서 부터
 - SEEK_END : 끝 지점에서 부터

~/test6 디렉토리 생성

- test.txt
- sample.c
- gcc sample.c -o ./gogo
- ASCII CODE 의 0x7F 를 이용해 글자를 지운다.

```
123 7B 173 &#123; {  
124 7C 174 &#124; |  
125 7D 175 &#125; }  
126 7E 176 &#126; ~  
127 7F 177 &#127; DEL
```

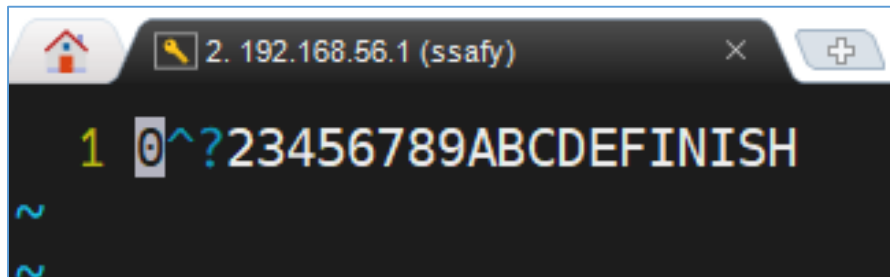
```
20      lseek(fd, 1, SEEK_SET);  
21      char buf[10] = {0x7F};  
22      write(fd, buf, strlen(buf));  
23  
24      close(fd);
```

실행 파일을 실행 한 뒤 test.txt 를 확인한다.

- cat 으로 읽으면, 0과 1이 사라진 것으로 보인다. (1 대신해서 DEL 이 작성, DEL 앞에 있는 0이 출력 안됨)

```
ssafy@ssafy-VirtualBox:~/test6$ ls
gogo sample.c test.txt
ssafy@ssafy-VirtualBox:~/test6$ cat test.txt
23456789ABCDEFINISH
```

- test.txt 를 직접 열면, 0이 삭제되지 않은 것이 보인다.



```
1 0^?23456789ABCDEFINISH
~
```

표준 입출력 (High Level API)

- fopen, fseek, fclose 등 존재
- 사용하기 쉽고, 기능이 많다.
- 개발속도가 빠르다.
- Low Level 명령어를 감싼 (Wrapping한 함수)

Low Level 명령어

- 쓰기 불편하다. open, write, lseek 등 존재
- 임베디드에서 **Device File** 을 다룰때 필요하기 때문에 써야한다.

Day1-4. 도전

임베디드 면접에서 물어볼 수 있는 용어를 조사해서 Word 파일로 정리 하자.

1. UFS
2. 인포테인먼트
3. ROS
4. 코봇
5. ECU

임베디드 면접에서 물어볼 수 있는 용어를 조사해서 Word 파일로 정리 하자.

1. Kernel
2. Interface
3. API
4. System Call
5. OS 와 Firmware
6. Memory
7. NVM (Non-volatile-memory)
8. API
9. System Call
10. POSIX
11. RTOS

man 페이지만을 참고하여 직접 개발해보자.

1. num.txt 파일에 수 하나 저장 ex) 1
2. 파일을 읽는다.
3. 숫자로 변환한다.
4. + 10 값을 출력한다.

먼저 cal.txt 파일 생성

- 파일 내용 : “1” 한 글자 적어 둬
- 644 권한

다음 순서대로 동작하는 프로그램 작성 (open / read / write)

1. 파일 읽기
2. 읽은 값을 출력하기
3. 읽은 값 x 2 값을 새로 쓰기
4. 파일 저장

한번 App 을 수행할 때 마다 x 2 값이 출력되어야 함

10 바이트씩 read하여 모든 내용 출력하는 프로그램 작성

- 반복문을 사용하여 구현한다.
- 읽어야하는 Text 파일의 Byte 를 모른다고 가정한다.
 - 만약 37바이트라면 정확히 4회 반복으로 동작되어야한다.

open할때 TIP

- buf 사이즈를 1로 잡아두고, 읽기 전마다 memset 하기
 - 항상 뒤에 널문자가 존재하는 것이기에 printf가 편리하다.

```
nojin@nojin-VirtualBox:~/test$ gcc ./hi.c && ./a.out
=====
0123456789
=====
10
=====
ABCDEABCDE
=====
10
=====
0123456789
=====
10
=====
FINISH
=====
7
```

파일 text.txt 파일에 A ~ Z 까지 저장하기

ABCDEF**GHI**JKLMN**OP**QRSTUVWXYZ

1. 문자열 첫 다섯글자만 읽고 출력

- 출력결과 : ABCDE

2. 문자열 맨 뒤 다섯글자만 읽고 출력

- 출력결과 : VWXYZ

3. 처음부터 순차적으로 세 글자씩 읽는다.

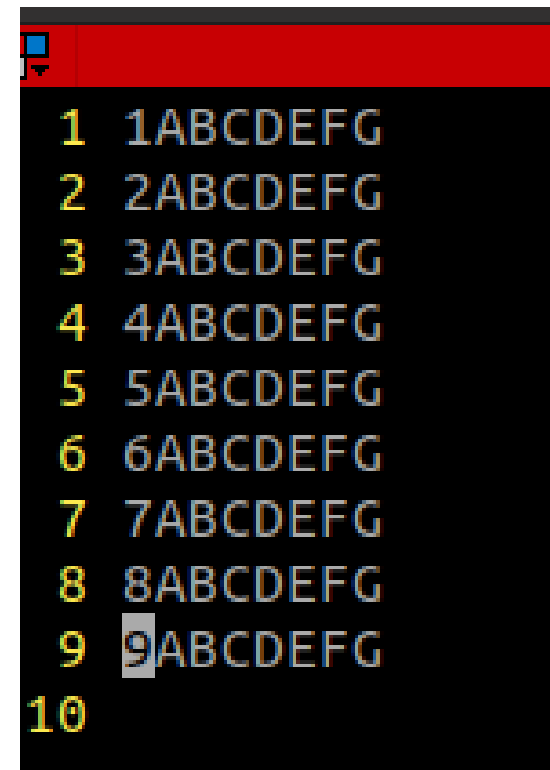
- 세 글자씩 탐색 중, 만약 **GHI** 라는 문자열을 발견하면
offset 을 다섯칸 이동 후, 세글자 읽고 출력
- 출력결과 : OPQ

한 줄 삭제하는 Application 제작하기

1. test.txt 를 오른쪽과 같이 제작한다.
2. 지울 줄 번호를 입력 받는다.
3. 특정 라인 줄만 삭제하여 test.txt에 저장한다.

힌트

1. 모든 내용을 읽는다.
2. Wn 을 기준으로 줄 단위로 문자열을 보관한다.
3. 입력 받은 Line을 제외하고 파일에 저장한다.



```
1 1ABCDEFGH
2 2ABCDEFGH
3 3ABCDEFGH
4 4ABCDEFGH
5 5ABCDEFGH
6 6ABCDEFGH
7 7ABCDEFGH
8 8ABCDEFGH
9 9ABCDEFGH
10
```

내일 방송에서 만나요!

삼성 청년 SW 아카데미