파이썬 – HW6

임베디드스쿨1기

Lv1과정

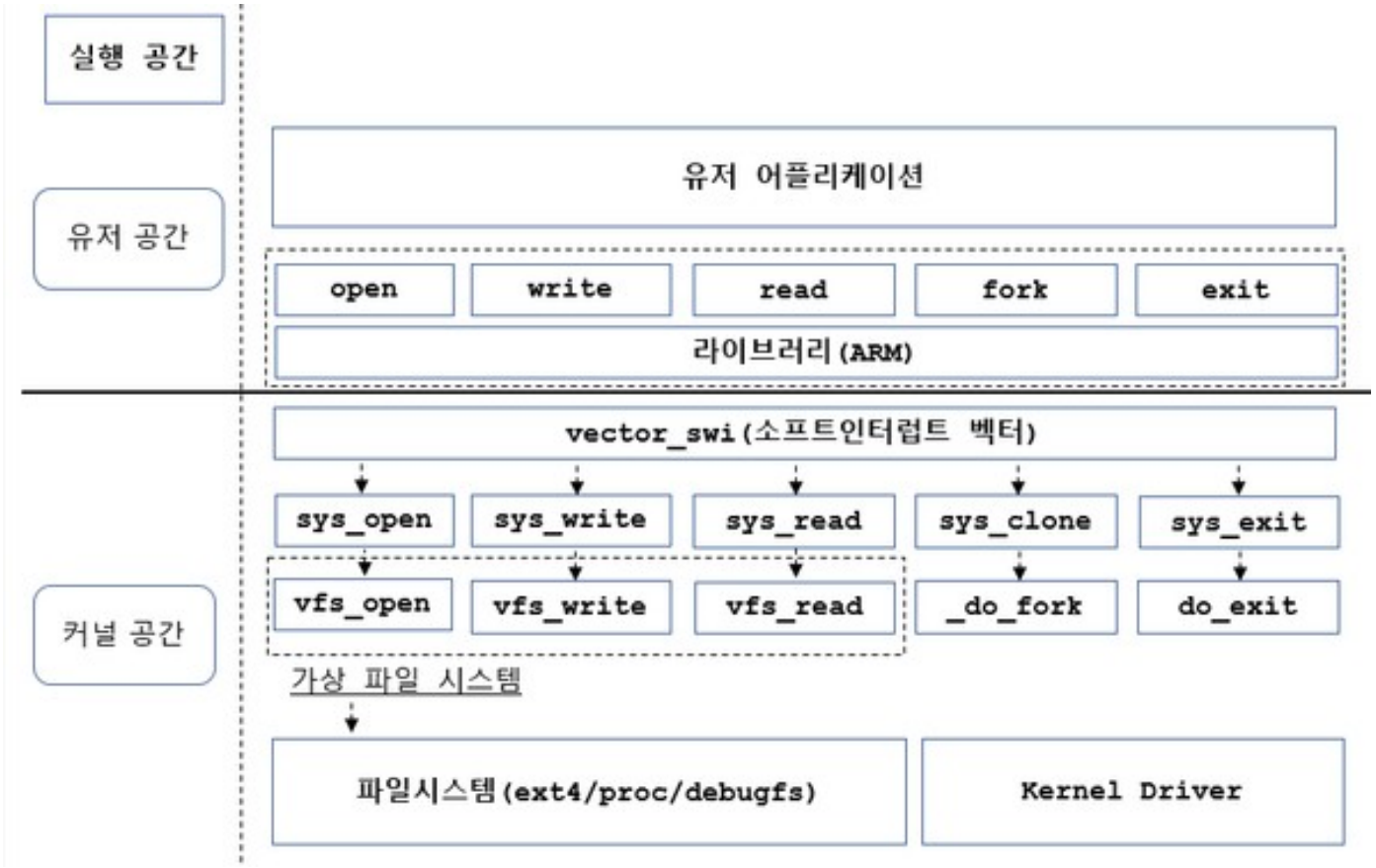
2020. 09. 15

박하늘

1. [Linux Kernel] 가상 파일 시스템(VFS)이란?

1) 가상 파일 시스템 특징

- Open(), Write(), Read() 함수를 유저 어플리케이션에서 호출하면 시스템 콜 실행으로 다양한 파일 시스템에 접근할 수 있는 소프트웨어 계층. (프로제스 구조체 → 파일 구조체)
- 파일시스템 인터페이스를 사용자 어플리케이션에 제공하고 모든 파일 시스템은 가상 파일시스템(VFS)을 통해 접근 가능.
-리눅스는 거의 모든 파일 시스템 지원(ext, ext2, ext3, ext4, fat, fat32, ntfs, hpfs, smb 등)
- VFS안에서 파일을 가장 빠르고 효율적으로 사용할 수 있도록 구현되어 있어서 캐시 기능과 이중 연결 구조를 활용함.



2. task_Struct

1) task_struct 내에 파일을 다루기 위한 멤버 변수 files 와 fs가 존재함

```
#endif
        /* Filesystem information: */
        struct fs_struct                *fs;

        /* Open file information: */
        struct files_struct             *files;
```

- struct fs_struct *fs ;
: 현재 작업 디렉토리, 루트 디렉토리, chroot 명령으로 변경된 루트 디렉토리 정보

- struct files_struct *files ;
: 프로세스 경로와 관련된 정보 관리하는 구조체, 파일을 관리하는데 사용

: 현재 프로세스가 사용중인 파일은 배열로 관리

2. file_struct

1) files_struct

- file_operations: 읽기/쓰기 함수와 연결
- vfsmount: 현재 마운트된 파일 시스템 정보 – flip 캐시 관리
(마운트 하려는 위치의 디렉토리 정보, 부모 파일시스템의 정보, 마운트 트리의 루트 디렉토리 정보 등)
- dentry: 파일이 위치한 디렉토리에 대한 정보의 구조체

```
/*
 * Open file table structure
 */
struct files_struct {
  /*
   * read mostly part
   */
        atomic_t count;
        bool resize_in_progress;
        wait_queue_head_t resize_wait;

        struct fdtable __rcu *fdt;
        struct fdtable fdtab;
  /*
   * written part on a separate cache line in SMP
   */
        spinlock_t file_lock ____cacheline_aligned_in_smp;
        unsigned int next_fd;
        unsigned long close_on_exec_init[1];
        unsigned long open_fds_init[1];
        unsigned long full_fds_bits_init[1];
        struct file __rcu * fd_array[NR_OPEN_DEFAULT];
};

struct file_operations;
struct vfsmount;
struct dentry;
```

2. file_operations

1) 가상 파일 시스템에서 파일 시스템 별로 파일을 열고 쓰고 읽는 함수 포인터 테이블을 지원.

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
        int (*iopoll)(struct kiocb *kiocb, bool spin);
        int (*iterate) (struct file *, struct dir_context *);
        int (*iterate_shared) (struct file *, struct dir_context *);
        __poll_t (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        unsigned long mmap_supported_flags;
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, loff_t, loff_t, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
        int (*check_flags)(int);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
        int (*setlease)(struct file *, long, struct file_lock **, void **);
        long (*fallocate)(struct file *file, int mode, loff_t offset,
                          loff_t len);
        void (*show_fdinfo)(struct seq_file *m, struct file *f);
#ifndef CONFIG_MMU
        unsigned (*mmap_capabilities)(struct file *);
#endif
        ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
                        loff_t, size_t, unsigned int);
        loff_t (*remap_file_range)(struct file *file_in, loff_t pos_in,
                                   struct file *file_out, loff_t pos_out,
                                   loff_t len, unsigned int remap_flags);
        int (*fadvise)(struct file *, loff_t, loff_t, int);
} __randomize_layout;
```

[llseek] loff_t (*llseek) (struct file *file, loff_t offset, int whence) :
파일 포인터를 offset 값으로 갱신

[read] ssize_t (*read) (struct file *file, char __user *buf, size_t count, loff_t *offset);
파일 오프셋(offset) 위치에서 count 바이트만큼 읽습니다. 이 동작을 수행하면서 파일 오프셋인 *offset은 업데이트됩니다.

[write] ssize_t (*write) (struct file *file, const char __user buf*, size_t count, loff_t offset*);
파일의 오프셋(*offset) 위치에 count 바이트만큼 buf에 있는 데이터를 써줍니다.

[poll] unsigned int (*poll) (struct file *, struct poll_table_struct *);
파일 동작을 점검하고 파일에 대한 동작이 발생하기 전까지 휴면 상태에 진입합니다.

[ioctl] long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
각각 멤버들은 포인터로 정의돼 있으며 함수 포인터로 파일 시스템 마다 지정된 서로 다른 함수들을 호출합니다.

Q&A 1. Open을 통해서 fd를 얻어온다.

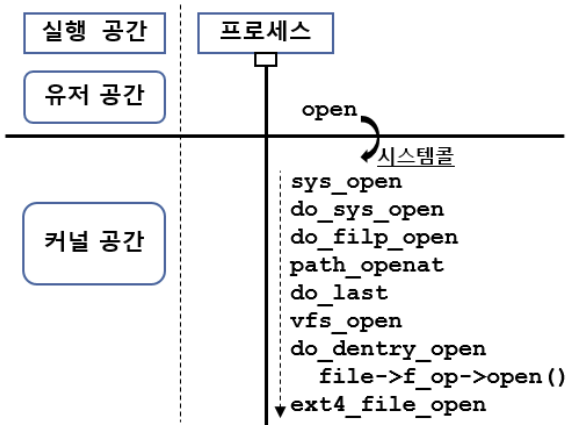
1) 실행 흐름

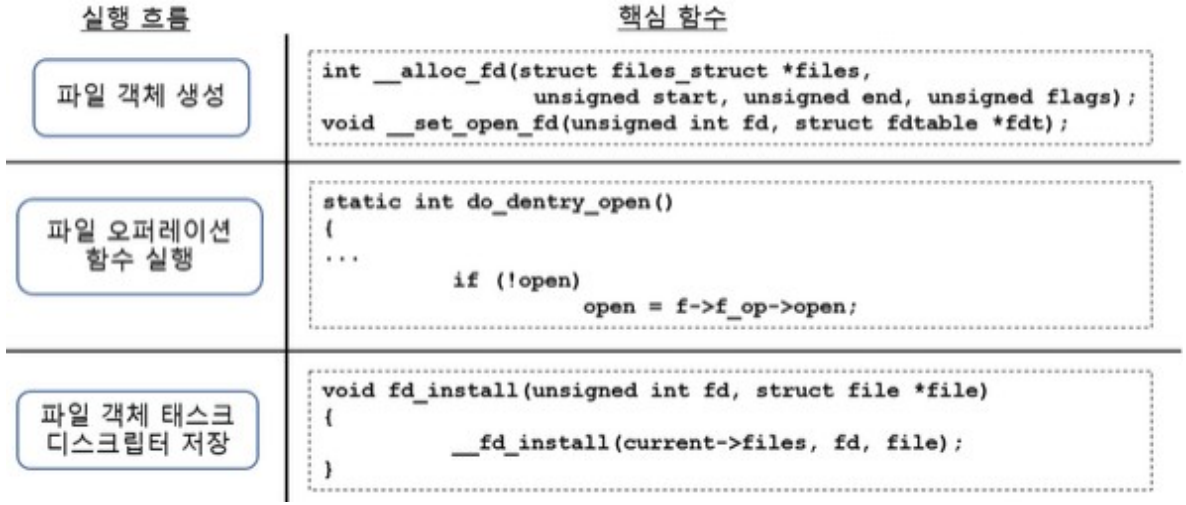
:유저 공간에서 open() 함수를 호출하면 시스템 콜을 발생시켜 실행 흐름이 커널 공간으로 바뀐다. 이후 open() 함수에 해당하는 시스템 콜 핸들러 함수인 sys_open() 함수가 실행 한 후 ext4 파일시스템에서 관리하는 파일 오픈 함수인 ext4_file_open() 함수를 호출

1단계: 파일 객체 생성
파일 오픈을 할 때 가장 먼저 파일 객체를 생성. 이 때 __alloc_fd() 함수를 호출.

2단계: 파일 오퍼레이션 실행
각 파일시스템에서 관리하는 파일 종류별로 등록된 open() 함수를 호출합니다. “/home/pi/sample_text.text” 이란 파일을 오픈할 경우 ext4_file_open() 함수를 실행합니다.

3 단계: 파일객체 태스크 디스크립터 등록
파일 오픈에 대한 함수 오퍼레이션을 마무리하면 프로세스에서 관리하는 파일 디스크립터 테이블에 파일 객체를 등록
파일을 오픈하면 파일 디스크립터 테이블에 파일 디스크립터를 저장합니다





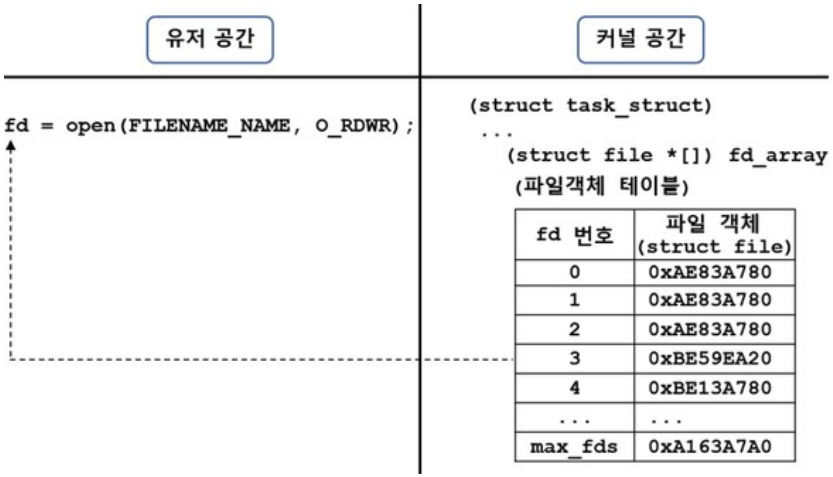
Q&A 2. 유닉스/리눅스에서 왜 모든것이 파일인가?

1) File Descriptor (fd) 로 모든 파일 관리 가능

- 파일 디스크립터로 아이노드, 슈퍼 블락 및 파일 오퍼레이션 등등 모든 파일을 관리할 수 있는 객체에 접근 가능.
- 유저 어플리케이션은 fd란 정수형 인자로 손쉽게 파일 관리가능. 파일 상세 내용은 모르는 대신 정수형 인자 하나로 파일을 핸들링하는 것. 프로세스가 이 정수형 파일 디스크립터를 관리한다.

파일을 오픈하고 난 다음 읽고 쓰고 닫을 때 프로세스가 파일 디스크립터를 어떻게 관리할까?
1 단계: 파일 객체 및 파일 디스크립터 생성 후 프로세스 파일 디스크립터 테이블에 등록
2 단계: 프로세스 파일 디스크립터 테이블에서 파일 디스크립터와 파일 객체 로딩
3 단계: 프로세스 파일 디스크립터 테이블에서 파일 디스크립터와 파일 객체 삭제

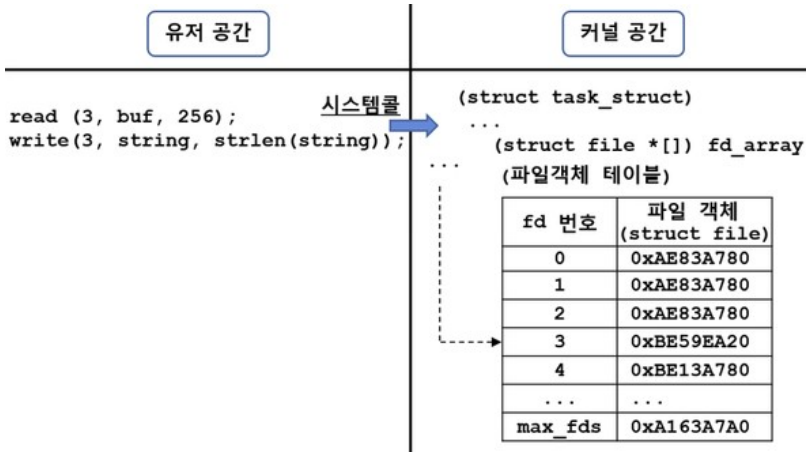
[1단계] - 파일 객체 파일 디스크립터 테이블 등록
- open() 함수를 호출해서 획득한 정수형 값인 파일 디스크립터는 커널 공간 파일 디스크립터 테이블 배열 인덱스를 의미
- 각 프로세스 마다 파일 객체를 관리하는 파일 디스크립터 테이블이 있음
- 유저 공간에서 파일에 대한 핸들링은 3으로 관리하는데, 커널 공간에서는 3번 배열에 해당하는 파일 객체로 파일을 관리



fd 번호	파일 객체 (struct file)
0	0xAE83A780
1	0xAE83A780
2	0xAE83A780
3	0xBE59EA20
4	0xBE13A780
...	...
max_fds	0xA163A7A0

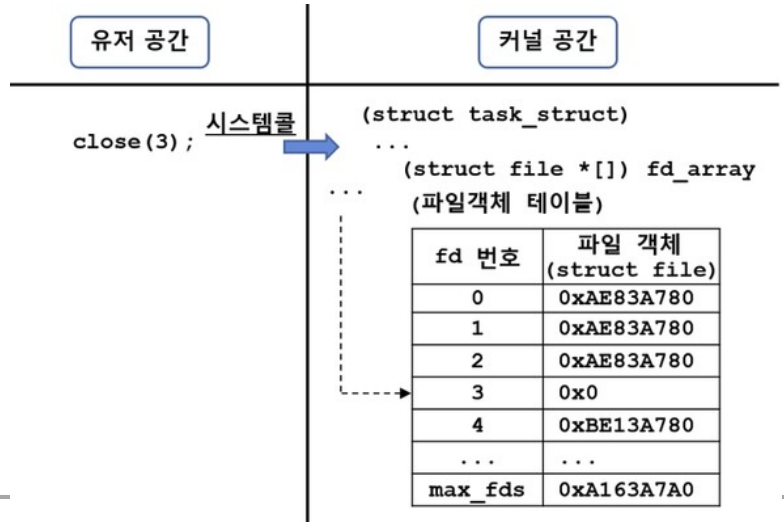
Q&A 2. 유닉스/리눅스에서 왜 모든것이 파일인가?

[2단계] - 파일 디스크립터로 파일 객체 로딩
- 유저 공간에서 3번 파일 디스크립터로 파일을 읽고 쓰고 파일 포인터를 설정
- 커널 공간에서는 파일 디스크립터 테이블에서 3번 배열에 해당하는 파일 객체(0xBE59EA20)로 파일을 읽고 쓰고 파일 포인터를 설정 동작을 관리



fd 번호	파일 객체 (struct file)
0	0xAE83A780
1	0xAE83A780
2	0xAE83A780
3	0xBE59EA20
4	0xBE13A780
...	...
max_fds	0xA163A7A0

[3단계] - 파일 디스크립터 해제
- 파일을 닫을 때 유저 공간에서 close() 함수를 호출하는데 이 때 정수형 파일 디스크립터를 인자로 전달
- 널 공간에서 파일 디스크립터 테이블에 접근해서 3번 파일 디스크립터 테이블에 있는 파일 객체를 0x0으로 초기화

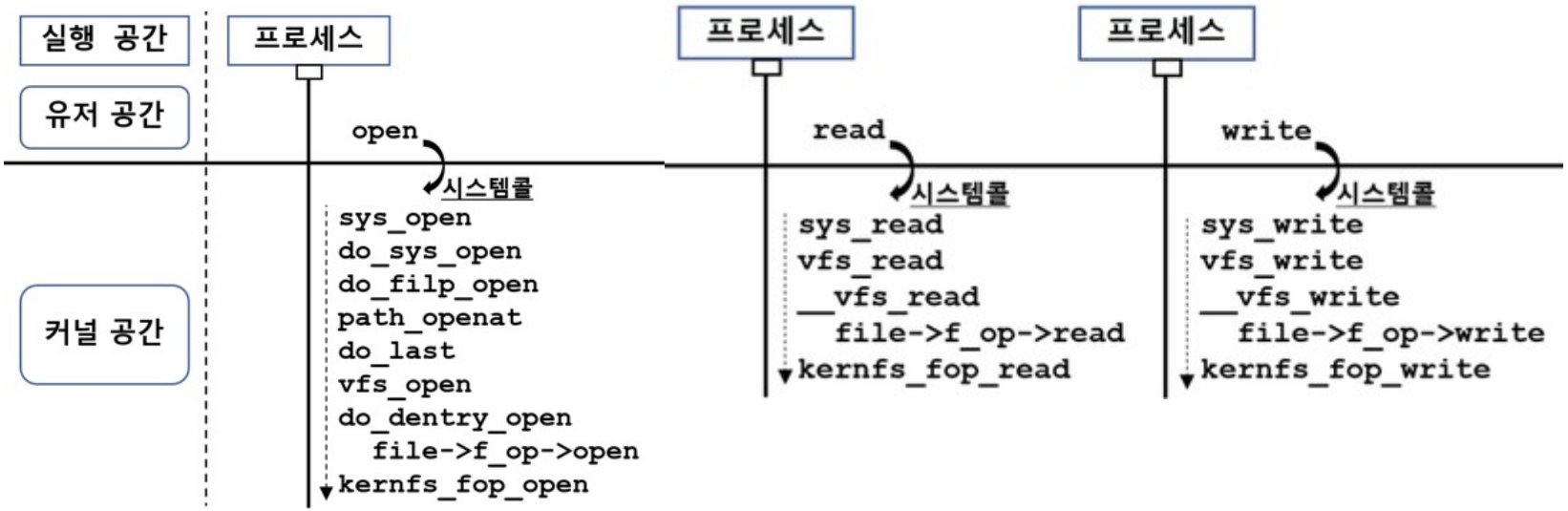


fd 번호	파일 객체 (struct file)
0	0xAE83A780
1	0xAE83A780
2	0xAE83A780
3	0x0
4	0xBE13A780
...	...
max_fds	0xA163A7A0

Q&A 3. open, read, write는 무엇을 제어하는가?

- 파일을 오픈해서 열고 쓰고 닫을 때 가상 파일시스템 계층에서 각 파일 시스템 별 함수 테이블로 분기 시킨다.
- 이런 동작이 가능한 이유는 가상 파일시스템에서 파일 객체란 struct file 자료 구조가 파일 시스템 별 파일 오픈 및 쓰고 읽기 동작에 대한 관리를 해줌

: struct file 구조체 멤버 중인 file_operations에 접근해서 open,read,write란 포인터가 가르키는 주소를 호출



Review. Class 관련 용어

1. 객체 (object) : 클래스로 만들어낸 피조물
   Ex) a = Cookie()
2. 생성자 : 객체를 생성할 때 자동 호출, 특정 이름이 정해져 있으며, 메소드와 유사한 구조로 이루어져 있음
   Ex) def __init__(self):
3. 클래스 (class) : 일종의 청사진
   Ex) class Cookie :
4. 필드 (field) : 클래스에 내장된 변수
5. 메소드 (method) : 클래스 내부에 선언된 함수
   Ex) def Make_Cookie() :
6. 속성 (attribute) : 필드 + 메소드 = 속성
7. 인스턴스 (instance) : 클래스를 통해 생성된 결과물로 '실체'라고도 한다.
   Ex) a = Cookie() 에서 a가 인스턴스에 해당한다.
8. 정적 메소드 (static method) : self 매개변수를 가지지 않는 메소드로 인스턴스 필드에 접근이 불가능하다.
   정적 메소드는 메소드 선언 앞에 @staticmethod라고 표기해야 한다.
9. 인스턴스 메소드 (instance method) : self로 인스턴스 필드에 접근하는 메소드
   Ex) def Make_Cookie(self) :
10. 클래스 메소드 (class method) : cls로 클래스 필드에 접근하는 메소드이며, 정적 메소드처럼 @classmethod라고 메소드 선언 앞에 표기해야 한다.
    Ex) def Make_Cookie(cls) :
11. 인스턴스 변수 : self가 붙은 변수
    Ex) self.variablename
12. 클래스 변수 : class 내부에 선언된 변수이며, 일반적인 변수와 동일한 형태이다.
    Ex) Cookies = 0

정적 메소드나 클래스 메소드는 필드에 접근하지 않을 때 사용하며, 특별히 클래스 변수에 접근해야 할 때는 클래스 메소드를 사용함.

13. 인스턴스를 이용한 접근
    Ex) instancename.methodname()
14. 클래스를 이용한 접근
    Ex) classname.methodname()

```
Class 클래스명:
    변수 = 값 (없어도 된다)
    Def 메소드명(self,매개변수)
        메소드 구문
```

Review. 객체 변수 초기화 방법 2가지

[방법 1] - 메소드를 통해

- "setValue" 메소드를 통해 객체 변수를 초기화

("classNum"는 클래스 변수 / "objNum"는 객체 변수)

```
class Computer:
    classNum = 0

    def setValue(self , num):
        self.objNum = num

cp1 = Computer()
cp2 = Computer()

cp1.setValue(10)
cp2.setValue(20)

print("cp1 : {0} cp2 : {1}" .format(cp1.objNum,cp2.objNum)) # cp1 : 10 cp2 :  20

Computer.classNum = 20

print("cp1 : {0} cp2 : {1}" .format(cp1.classNum,cp2.classNum)) #  cp1 : 20 cp2 :  20
```

[방법 2] - 생성자 사용

- 객체 변수를 초기화하기 위한 특별한 메소드
- 메소드 명에 "__init__"을 작성하고 똑같이 매개변수와 객체 변수를 통해 초기화

(좀 더 간편하게 구별 가능)

```
class Computer:
    def __init__(self,a,b):
        self.num1 = a
        self.num2 = b

cp1 = Computer(10,30)

print("num1 : {0} num2 : {1}" .format(cp1.num1,cp1.num2)) # num1 : 10 num2 : 30
```

HW. 클래스, 정적 매서드

Q. ??

```
##클래스 메서드(classmethod)와 정적 메서드(staticmethod) 이해하기

class CntManager:
    cnt = 0
    def __init(self):
        CntManager.cnt += 1
    def staticPrintCnt():
        print("Instance cnt", CntManager.cnt)
    sPrintCnt = staticmethod(staticPrintCnt)

    def classPrintCnt(cls):
        print("Instance cnt: ",cls.cnt)
    cPrintCnt = classmethod(classPrintCnt)

a, b, c = CntManager(), CntManager(), CntManager()

CntManager.sPrintCnt()
b.sPrintCnt()

CntManager.sPrintCnt()
b.cPrintCnt()
```

```
Instance cnt 0
Instance cnt 0
Instance cnt 0
Instance cnt:  0
```

감사합니다.