open()
mkdir(), link()
fork()
dup()
write(), fsync()
opendir(), readdir, closedir()
unlink(), rmdir()

22. Files and Directories

▼ Abstractions for Storage

▼ File

byte의 linear array

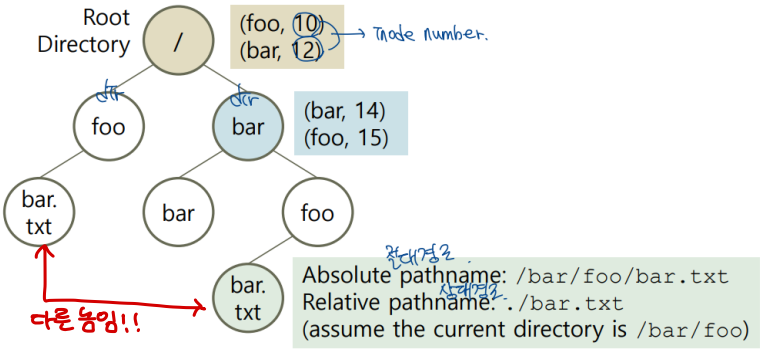
- 각 file은 low-level name을 가짐 ⇒ inode number
- OS : file에 대한 정보를 많이 알지 못 함

  (OS가 file에 대해 어떤 확장자를 가지는지..까지는 모름)

▼ Directory

(user-readable name, low-level name) pair들 list를 가지고 있음

- 각 directory 또한 low-level name을 가짐 ⇒ inode number



▼ File system interface

내부 operation이 어떻게 돌아가는지 알아보자

▼ Creating files/dir

- open file → *open()* with *O_CREAT* flag

```
int fd = open("foo", O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
```

  - O_CREAT : 같은 이름을 가진 file이 존재하지 않는다면 생성
  - O_WRONLY : Open for writing only

- O_TRUNC : file이 이미 존재한다면 0byte로 바꿔버린 후 존재하는 content 모두 삭제
- S_IRUSR|S_IWUSR : permission 설정 ⇒ owner에 의해 readable, writeable

- **file descriptor**

  open이 성공적으로 끝나면 return되는 integer 값 ⇒ 각 file의 고유한 번호

  - file descriptor을 이용하여 file read 혹은 wirte 수행(permission 가지고 있다면)
  - file의 pointer로 생각해도 됨
  - private per process
    - 각 process는 fd list를 유지, 관리하며 각 file은 system 전체의 open 된 file table에 있는 항목 중 하나임
    - example (xv6)

```
struct proc {
  ...
  struct file *ofile[NOFILE]; // Open files
  ...
};
```

- open file table
  - OS : system 내부에 현재 open되어 있는 file 정보를 table로도 관리
    - table 내부의 각 entry : current offset + 다른 relevant detail 가지고 있음
  - example(xv6)

```
struct {
    struct spinlock lock;
    struct file file[NFILE];
} ftable;
```

```
struct file {
    int ref;
    char readable, writable;
    struct inode *ip;
    uint off;
};
```

  - file : open file table의 특정 entry를 가짐
    - 다른 process가 동시에 같은 file read하더라도 open file table에 각자 자신만의 entry를 가지고 있음
    - reading 혹은 writing은 독립적임

- Making Directories
  - *mkdir()* ⇒ dir 만들기

    - empty dir 생성
    - default entries
        - . : 현재 자기 자신 dir
        - .. : 부모 dir

- Making file
    - *link() → hard links* ⇒ 다른 이름이 파일이지만 같은 inode 가리키도록 구현 가능

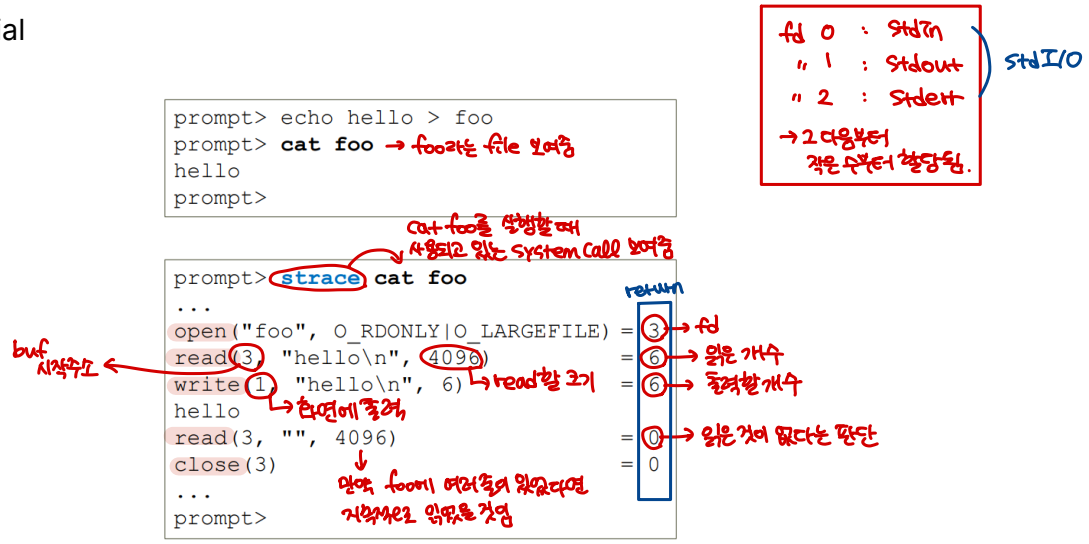
```
prompt> echo hello > file
prompt> cat file
hello
prompt> ln file file2
prompt> cat file2
hello
prompt> ls -i file file2
67158084 file
67158084 file2
prompt>
```

ln command ⇒ link 걸기

같은 inode 번호

▼ Accessing files/dir

- sequential



- random
    - Os : 항상 “current” offset 기억함
        - 해당 파일 내부에서 다음번 read 혹은 wirte가 어디서부터 read 혹은 write될 지 offset 기억함 (lseek 사용해서 바꿀수도 있음)
    - implicit update

    - read, write가 N byte 실행 → current offset += N
      ↳ 이만큼 offset 증가
  - explicit update

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

whence → 어디서부터 이 offset을 적용할 것인지!
- SEEK_SET: is set to offset bytes ⇒ offset
- SEEK_CUR: is set to its current location plus offset bytes ⇒ 현재 위치 + offset
- SEEK_END: is set to the size of the file plus offset bytes ⇒ 맨끝 + offset
```

- Shared file entries
  - *fork()*
    - child와 parent는 같은 file offset 공유 → update 적용됨.

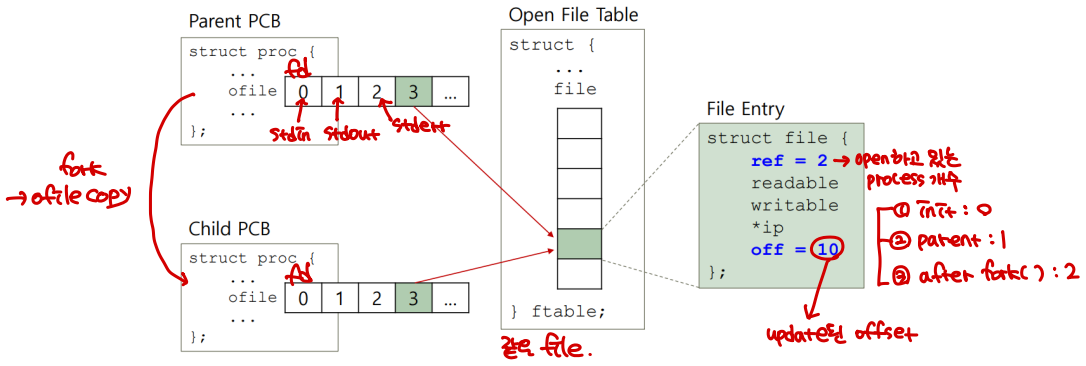
```
int main(int argc, char *argv[]) {
  int fd = open("file.txt", O_RDONLY);
  assert(fd >= 0);
  int rc = fork();
  if (rc == 0) {
    rc = lseek(fd, 10, SEEK_SET);
    printf("C: offset %d\n", rc);
  } else if (rc > 0) {
    (void) wait(NULL); ⇒ child가 무조건 먼저 실행됨!
    printf("P: offset %d\n", (int) lseek(fd, 0, SEEK_CUR));
  }
  return 0;
}
```

child
parent
↳ offset 출력

```
prompt> ./fork-seek
child: offset 10
parent: offset 10
prompt>
```

같이 update된 것 확인할 수 있음.

↳ 자료구조도 찾아보자



  - *dup()*
    - process가 이미 fd가 존재하는 open된 file을 나타내는 새로운 fd를 생성하는 것을 허락하는 operation

      ⇒ 사용되지 않고 있는 가장 낮은 번호의 fd를 사용하여 fd 복사본 만드는 것을 허용

```c
int fd=open("output.txt", O_APPEND|O_WRONLY);
close(1);
dup(fd); //duplicate fd to file descriptor 1
printf("My message\n");
```

- Writing immediately
  - *write()*
    - OS : data write를 buffer에다가 모두 저장해놓은 뒤 한꺼번에 disk write
      - disk scheduling 효율성 높임
      - 해당 write → storage device에 무조건 문제가 생길거임
  - fsync()
    - file system : disk에 모든 dirty data 작성
- Reading Dir
  - opendir(), readdir(), closedir()

```c
int main(int argc, char *argv[]) {
  DIR *dp = opendir(".");
  struct dirent *d;
  while ((d = readdir(dp)) != NULL) {
    printf("%lu %s\n", (unsigned long) d->d_ino, d->d_name);
  }
  closedir(dp);
  return 0;
}

struct dirent {
  char d_name[256]; // filename
  ino_t d_ino; // inode number
  off_t d_off; // offset to the next dirent
  unsigned short d_reclen; // length of this record
  unsigned char d_type; // type of file
};
```

▼ Removing files/dir

- Removing Files
  - unlink() → 아예 file delete

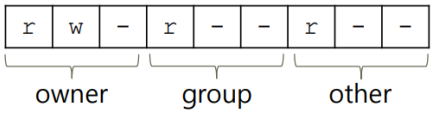
```
prompt> rm file
removed 'file'
```

```
prompt> cat file2
hello
```

- Deleting Dir
  - rmdir()
    - dir : 무조건 empty한 dir이어야 함. non-empty를 지우려고 하면 fail

▼ Mechanisms for Resource Sharing

- abstraction of a process
  - cpu, memory 모두 private하게 virtualization
- file system
  - disk virtualization → files and dir
  - files : 다른 user 혹은 process 사이에 공유됨 → not private
  - permission bits → 접근 가능한 대상 설정 가능



    - directory → execute bit(e)는 user가 directory 수정할 수 있게 해줌
  - mkfs
    - making a file system → file, dir 어떤 식으로 저장되게 할건지 정의
      - 빈 file system 새로 생성 후 root dir부터 시작하도록 disk 정의

```
prompt> mkfs -t ext4 /dev/sda1
```

    - uniform한 file-system tree 내에서 접근 가능하도록 만들어 줄 필요가 있음.
  - mount
    - target mount point로서 존재하는 dir 가져온 뒤 그 자리에 새로운 file system 복사 후 붙여넣기

```
prompt> mount -t ext4 /dev/sda1 /home/users
```