File I/O

FILE

- file
  - unix에서의 파일은 단지 바이트들의 나열이다
  - operating system은 파일에 어떤 포맷도 부과하지 않는다
  - 파일의 내용은 바이트 단위로 주소를 줄 수 있다
- file descriptor
  - file descriptor는 0이나 양수이다
  - file은 open이나 creat로 file descriptor가 새로 할당되고, read, write할 때 이를 이용한다
  - 그러므로 user가 생성하는 첫번 째 파일의 file descriptor는 3이다.
- file type
  - regular file - 0 이상의 data block을 가진 일반적인 파일
  - directory - 파일과 파일 이름을 mapping 시켜주는 파일
  - special file - physical device를 file system에 mapping
  - link - 파일을 다른 이름으로 연결
  - symbolic link - 다른 파일을 가리키는 파일
  - named pipe(FIFO) - process간의 통신을 위한 파일

File Descriptor

- file descriptor
  - all open files are referred to by file descriptors
  - how to obtain file descriptor
    - return value of open( ), creat( )
  - when we want to read or write a file, we identify the file with the file descriptor
  - file descriptor is the index of user file descriptor table
  - STDIN_FILENO(0), STDOUT_FILENO(1), STDERR_FILENO(2) (<unistd.h>)

File Descriptor

- file descriptor (cont'd)
  - standard input, output, error
    - STDIN_FILENO (== 0)
    - STDOUT_FILENO (== 1)
    - STDERR_FILENO (== 2)
  - defined in <unistd.h>
  - opened by the shell
    - not by the kernel

open( )

- 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);

open( )

- description
  - attempts to open a file and return a file descriptor
  - flags (defined in <fcntl.h>)
    - O_RDONLY, O_WRONLY or O_RDWR
      - access mode
    - O_CREAT
      - If the file does not exist it will be created
    - O_EXCL
      - When used with O_CREAT, if the file already exists it is an error and the open will fail

open( )

- flags (cont'd)
  - O_NOCTTY
    - If pathname refers to a terminal device it will not become the process's controlling terminal even if the process does not have one
  - O_TRUNC
    - If the file already exists it will be truncated
  - O_APPEND
    - Initially, and before each write, the file pointer is positioned at the end of the file

open( )

- flags (cont'd)
  - O_NONBLOCK (O_NDELAY)
    - Neither the open nor any subsequent operations on the file descriptor which is returned will cause the calling process to wait
    - regular file: mandatory lock
    - FIFO: open, read, write
  - O_SYNC
    - Any writes on the resulting file descriptor will block the calling process until the data has been physically written to the underlying hardware

open( )

- flags (cont'd)
  - O_NOFOLLOW
    - If pathname is a symbolic link, then the open fails
    - define _GNU_SOURCE before including <fcntl.h>
  - O_DIRECTORY
    - If pathname is not a directory, cause the open to fail
    - define _GNU_SOURCE before including <fcntl.h>
    - Directory can be opened with open(), but can't read by read() system call. Should use readdir() system call.

open( )

- flags (cont’d)
  - O_LARGE
    - On 32-bit systems that support the Large Files System, allow files whose sizes cannot be represented in 31 bits to be opened.
    - define _GNU_SOURCE before including <fcntl.h>
  - cf) open64, lseek64

open( )

- mode
  - specifies the permissions to use if a new file is created
  - modified by the process's umask
    - the permissions of the created file are (mode & ~umask)
  - should always be specified when O_CREAT is in the flags, and is ignored otherwise
- return value
  - return the new file descriptor, or -1 if an error occurred

open( )

- example

```
int fd;
fd = open(“/etc/passwd”, O_RDONLY);
fd = open(“/etc/passwd”, O_RDWR);

fd = open(“ap”, O_RDWR | O_APPEND);
fd = open(“ap”, O_RDWR | O_CREAT | O_EXCL, 0644);
```

open( )

- example (synchronization)

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

#define LOCKFILE "lockfile"
#define DELAY 10000000

void delay(void)
{
 int i;
 for (i = 0; i < DELAY; i++);
}
```

open( )

```
int main(void)
{
     int fd, i;

     while ((fd = open(LOCKFILE, O_WRONLY | O_CREAT |
O_EXCL, 0644)) < 0) {
            if (errno != EEXIST) {
                    perror("open");
                    exit(1);
            }
     }
     for (i = 'a'; i <= 'z'; i++) {
            putchar(i);
            fflush(stdout);
            delay();
     }
close(fd);
     unlink(LOCKFILE);

     return 0;
}
```

creat( )

- synopsis
  - #include <sys/types.h>
  - #include <sys/stat.h>
  - #include <fcntl.h>
  - int creat(const char *pathname, mode_t mode);
- description
  - create a new file
  - equivalent to open with flags equal to O_CREAT|O_WRONLY|O_TRUNC

close( )

- synopsis
  - #include <unistd.h>
  - int close(int fd);

- description
  - closes a file descriptor
  - if the descriptor was the last reference to a file which has been removed using “unlink” the file is deleted

close( )

- when a process terminates, all open files are automatically closed by the kernel
- return value
  - zero on success, or -1 if an error occurred

close( )

- example

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

int main(void)
{
    int fd, n;
    char buf[10];
    fd = open("testfile", O_RDWR | O_CREAT | O_TRUNC,
0644);
    if (fd < 0) {
           perror("open");
           exit(1);
    }
```

close( )

```
    system("ls testfile");

    unlink("testfile");
    system("ls testfile");

    write(fd, "hello\n", 6);
    lseek(fd, 0, SEEK_SET);
    n = read(fd, buf, 10);
    write(STDOUT_FILENO, buf, n);

    close(fd);
}
```

lseek( )

- synopsis
  - #include <sys/types.h>
  - #include <unistd.h>
  - off_t lseek(int fildes, off_t offset, int whence);

- description
  - repositions the offset of the file descriptor fildes to the argument offset

lseek( )

- whence
  - SEEK_SET
    - The offset is set to offset bytes.
  - SEEK_CUR
    - The offset is set to its current location plus offset bytes.
  - SEEK_END
    - The offset is set to the size of the file plus offset bytes.

lseek( )

- hole
  - allows the file offset to be set beyond the end of the existing end-of-file of the file
  - If data is later written at this point, subsequent reads of the data in the gap return bytes of zeros
- return value
  - success : the resulting offset location as measured in bytes from the beginning of the file
  - error : -1

lseek( )

- example
  - off_t curpos;
  - curpos = lseek(fd, 0, SEEK_CUR);
  - lseek(fd, 0, SEEK_SET);
  - lseek(fd, 0, SEEK_END);
  - lseek(fd, -10, SEEK_CUR);
  - lseek(fd, 100, SEEK_END);

lseek( )

- example

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

int main(void)
{
 if (lseek(STDIN_FILENO, 0, SEEK_CUR) == -1)
        printf("cannot seek\n");
 else
        printf("seek OK\n");
 exit(0);
}
```

lseek( )

- result

```
$ gcc lseek.c
$ cat < /etc/motd | a.out
cannot seek
$ a.out < /etc/motd
seek OK
```

lseek( )

- example

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

int main(void)
{
        int fd;

        fd = creat("holefile", 0644);
        write(fd, "hello", 5);

        lseek(fd, 10, SEEK_CUR);
        write(fd, "world", 5);

        lseek(fd, 8192, SEEK_SET);
        write(fd, "bye", 3);

        close(fd);
        return 0;
}
```

lseek( )

- result

```
$ ls -l holefile
-rw-r--r--   1 kim   stud      8195 Jul 18 21:37 holefile
$ od -c holefile
0000000   h   e   l   l   o  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0   w
0000020   o   r   l   d  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000040  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0020000   b   y   e
0020003
$ du holefile
8   holefile
```

read( )

- synopsis
  - #include <unistd.h>
  - ssize_t read(int fd, void *buf, size_t count);

- description
  - attempts to read up to count bytes from file descriptor fd into the buffer starting at buf

read( )

- If count is zero, read() returns zero and has no other results
- return value
  - On success, the number of bytes read
  - zero indicates end of file
  - On error, -1 is returned

write( )

- synopsis
  - #include <unistd.h>
  - ssize_t write(int fd, const void *buf, size_t count);
- description
  - writes up to count bytes to the file referenced by the file descriptor fd from the buffer starting at buf

write( )

- return value
  - the number of bytes written
  - zero indicates nothing was written
  - On error, -1

read() & write()

```
#include <unistd.h>
#define BUFFSIZE 8192
int main(void)
{
        int n;
        char buf[BUFFSIZE];

        while ((n=read(STDIN_FILENO,buf,BUFFSIZE))>0)
                if (write(STDOUT_FILENO,buf,n)!=n)
                   printf("write error\n");

        if (n<0)
                printf("read error\n");
        exit(0);
}
```

Atomic Operations

- appending to a file

```
lseek(fd, 0, SEEK_END);
write(fd, buf, 256);
```

- creating a file

```
if ((fd = open(pathname, O_WRONLY)) < 0) {
    if (errno == ENOENT)
      fd = creat(pathname, 0644);
    else {
      perror(“open”);
      exit(1);
    }
}
```

dup( ) and dup2( )

- synopsis
  - #include <unistd.h>
  - int dup(int oldfd);
  - int dup2(int oldfd, int newfd);

- description
  - create a copy of the file descriptor oldfd and returns a new file descriptor
  - dup2 makes newfd be the copy of oldfd, closing newfd first if necessary.

dup( ) and dup2( )

- close_on_exec flag is also copied.
- The old and new descriptors may be used interchangeably
  - if the file position is modified by using lseek on one of the descriptors the position is also changed for the other
- two descriptors do not share the close-on-exec flag
- dup uses the lowest-numbered unused descriptor for the new descriptor

dup( ) and dup2( )

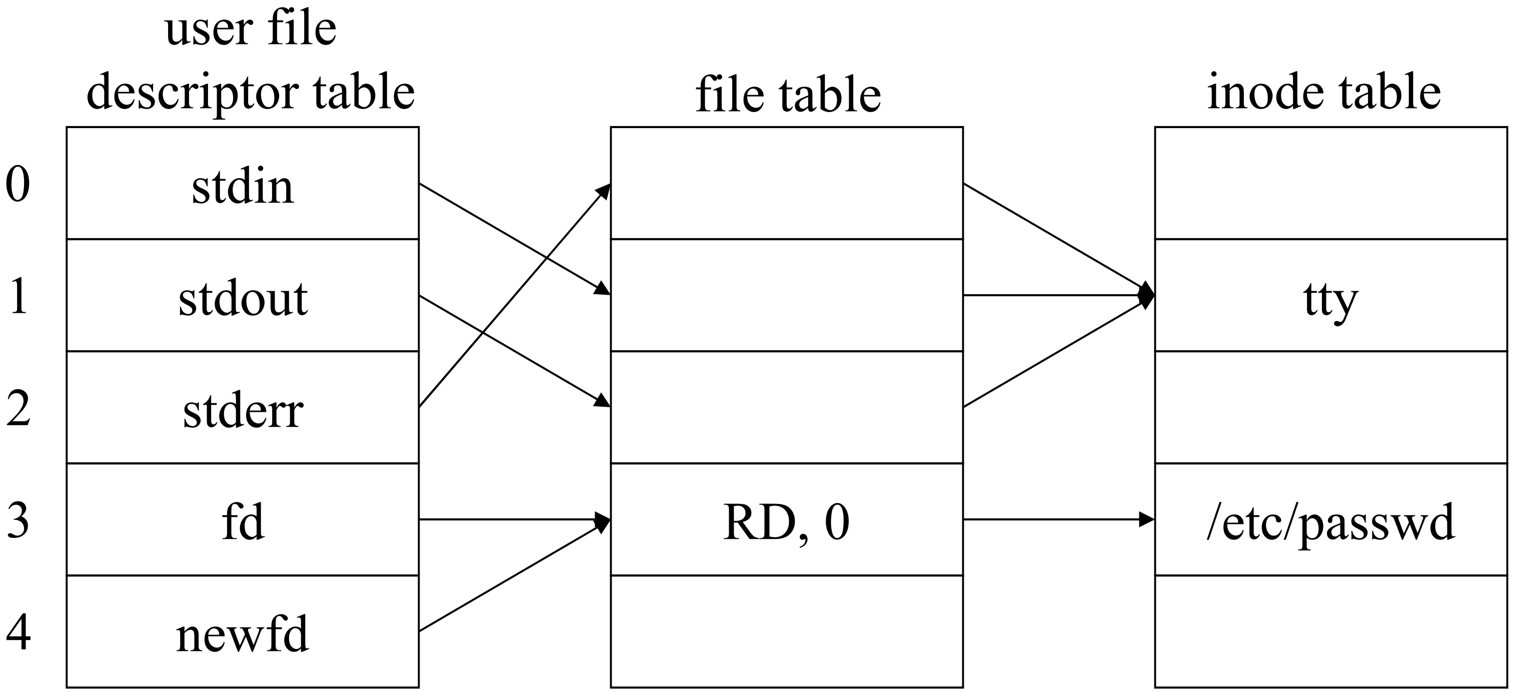
- return value
  - the new descriptor, or -1 if an error occurred

- example

```
1) int fd = dup(STDOUT_FILENO);
2) fd = open(“input_file”, O_RDONLY);
   dup2(fd, STDIN_FILENO);
3) fd = open(“input_file”, O_RDONLY);
   close(STDIN_FILENO);
   dup(fd);
```

dup( ) and dup2( )

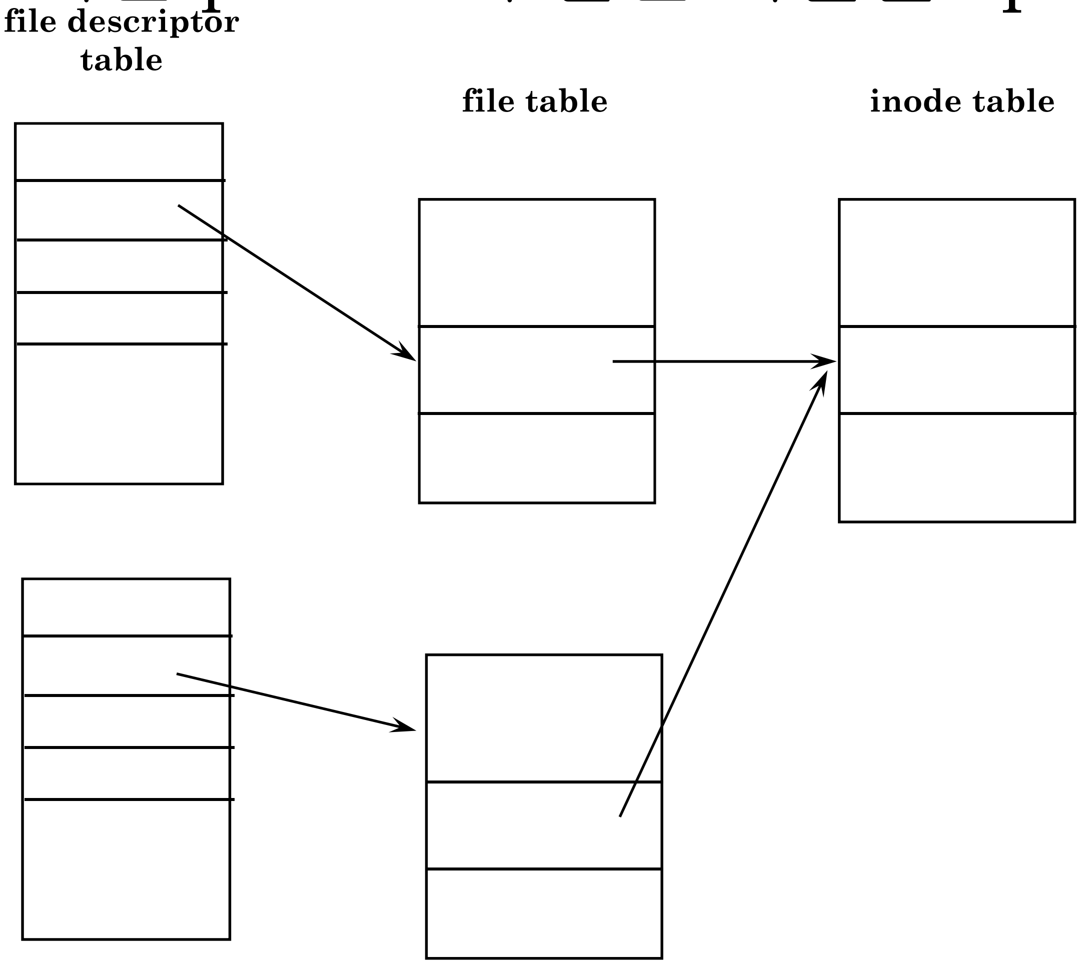
- fd = open("/etc/passwd", O_RDONLY);
- newfd = dup(fd);



	user file descriptor table	file table	inode table
0	stdin		
1	stdout		tty
2	stderr		
3	fd	RD, 0	/etc/passwd
4	newfd		

dup( ) and dup2( )

- 2개의 다른 process가 같은 파일을 open



dup( ) and dup2( )

- example

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

int main(void)
{
        int fd;
        fd = creat("dup_result", 0644);
        dup2(fd, STDOUT_FILENO);
        close(fd);
        printf("hello world\n");
        return 0;
}
```

```
$ cat dup_result
hello world
```

fcntl( )

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

  - int fcntl(int fd, int cmd);
  - int fcntl(int fd, int cmd, long arg);

fcntl( )

Description

- 이미 열려진 파일에 대해서 현재의 상태를 읽어 오거나 새로운 상태를 설정한다.
- manipulate file descriptor
- performs one of various miscellaneous operations on fd
- The operation in question is determined by cmd

fcntl( )

- cmd
  - F_DUPFD
    - Makes arg be a copy of fd
  - F_GETFD
    - Read the close-on-exec flag
    - If the low-order bit is 0, the file will remain open across exec
    - otherwise it will be closed.
  - F_SETFD
    - Set the close-on-exec flag to the value specified by arg
    - only the least significant bit is used
  - F_GETFL
    - Read the descriptor's flags
    - all flags (as set by open( )) are returned

fcntl( )

- example

```
int flag = fcntl(fd, F_GETFL);
if ((flag & O_ACCMODE) == O_RDONLY)
        printf(“opened with RDONLY”);
else if ((flag & O_ACCMODE) == O_WRONLY)
        printf(“opened with WRONLY”);
else
        printf(“opened with RDWR”);
if (flag & O_APPEND)
        printf(“ ,APPEND\n”);
else
        printf(“\n”);
```

fcntl( )

- F_SETFL
  - Set the descriptor's flags to the value specified by arg
  - Only O_APPEND, O_NONBLOCK and O_ASYNC may be set
  - the other flags are unaffected
- example
  - int flag = fcntl(fd, F_GETFL);
  - fcntl(fd, F_SETFL, flag | O_APPEND);

fcntl( )

- return value
  - F_DUPFD
    - The new descriptor
  - F_GETFD
    - the close-on-exec flag
  - F_GETFL
    - value of descriptor's flags
  - F_SETFD, F_SETFL
    - On success, 0
    - On error, -1

/dev/fd

- /dev/fd
  - opening the file /dev/fd/n is equivalent to duplicating descriptor n
  - example
  - fd = open(“/dev/fd/0”, O_RDONLY);
  - fd = dup(0);   /* STDIN_FILENO */

Example - copy

```
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

main(int argc, char *argv[]);
{
    int fd1, fd2, n;
    char buf[1024];
    if (argc < 3) {
        fprintf(stderr, “Usage; %s src dest\n”, argv[0]);
        exit(1);
    }
```

Example - copy

```
    if ((fd1 = open(argv[1], O_RDONLY)) < 0) {
            perror(“Error:”);
            exit(1);
    }
    if ((fd2 = creat(argv[2], 0644)) < 0) {
            perror(“Error:”);
            exit(1);
    }
    while ((n = read(fd1, buf, 1024)) > 0)
            write(fd2, buf, n);
    close(fd1);
    close(fd2);
}
```

Example – large file

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
main()
{
    int fd;
    loff_t p = 1LL << 32;
    if ((fd = open64("large", O_CREAT|O_RDWR|O_TRUNC, 0644)) < 0)
     {
        perror("open");
        exit(1);
    }
```

Example – large file

```
    if (lseek64(fd, p, SEEK_SET) < 0) {
        perror("llseek");
        exit(0);
    }
    write(fd, "hello", 5);
    close(fd);
}
```