


2. 저수준 파일 입출력

바이트 단위로 입·출력 수행  일반 파일 뿐만 아니라 특수 파일도 읽고 쓸 수 있음


파일 기술자 (File Descriptor)

저수준 파일 입출력에서 열린 파일을 참조하는데 사용하는 지시자 역할을 함

open 메소드로 파일을 열었을 때 번호가 부여됨, 그 중 0 ~ 2번은 미리 할당돼 있음(변경할 수도 있음)

파일 기술자	파일
0	표준 입력 (Standard Input) - 키보드
1	표준 출력 (Standard Output) - 모니터
2	표준 오류 (Standard Error) - 모니터

새로운 파일을 열 경우 다음 번호(3번)부터 할당됨

+ 유닉스는 모든 리소스(디렉토리, 링크파일, USB 등)를 파일로 간주함  why? 이렇게 할 경우 모든 리소스를 파일처럼 관리할 수 있기 때문에 시스템을 단순화 시킴

+ Solaris에서 한 프로세스가 열 수 있는 최대 파일 개수는 256개임 (안 쓰는 파일은 close 메소드로 닫아서 파일 기술자를 반환해야 함)

파일 열기 (open)

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

```
int open(const char *path, int oflag [, mode_t mode]); // [ ] 안에 있는 거
는 옵션임 (플래그가 O_CREAT일 때 사용)
```

open 메소드는 파일 열기에 성공하면 파일 기술자, **정수**(0 이상의)를 리턴한다. 파일 열기에 실패시 -1을 리턴하므로 조건문을 사용하여 파일이 제대로 열렸는지 확인할 수 있다. (errno 변수에 오류 코드를 저장하므로 왜 오류가 났는지도 확인 가능하다)

■ 주요 oflag

종류	기능
O_RDONLY	파일을 읽기 전용으로 연다
O_WRONLY	파일을 쓰기 전용으로 연다
O_RDWR	파일을 읽기, 쓰기용으로 연다
O_CREAT	파일이 없는 경우 생성 (파일 접근 권한을 지정해줘야 함 [mode_t mode])
O_EXCL	O_CREAT 플래그와 함께 사용, 이미 있는 경우 생성하지 않고 오류 메시지를 출력한다
O_TRUNC	파일을 생성할 때, 이미 있는 파일이고 쓰기 옵션(O_WRONLY)으로 열었으면 내용을 모두 지우고 파일 길이를 0으로 변경한다

더 있으니 필요한 경우 찾아보기

플래그는 OR 연산자(|)로 연결해 여러 개를 지정할 수 있다.

ex) O_WRONLY | O_TRUNC : 파일을 쓰기 전용으로 열고, 이미 있는 경우 내용을 모두 지우고 길이를 0으로 변경함

■ mode

플래그	모드	설명
S_IRWXU	0700	소유자 읽기/쓰기/실행 권한
S_IRWXG	0070	그룹 읽기/쓰기/실행 권한
S_IRWXO	0007	기타 사용자 읽기/쓰기/실행 권한

더 있으니 필요한 경우 찾아보기

+ 권한

0	소유자	그룹	기타
---	-----	----	----

소유자권한, 그룹권한, 기타사용자권한 자리에는 각각 7, 4, 2, 1이 들어갈 수 있다.

4는 읽기권한, 2는 쓰기권한, 1은 실행권한, 7은 읽기·쓰기·실행 권한을 다 준거다.

ex1) 만약 어떤 파일의 권한을 0644로 주었다면 **소유자**에게는 6=4+2 👉 읽기, 쓰기 권한을 준거고 **그룹 사용자**에게는 4 👉 쓰기 권한을 **기타 사용자**에게는 4 👉 쓰기 권한을 준거다.

ex2) 만약 권한을 0711로 주었다면 **소유자**에게는 7=4+2+1 👉 읽기, 쓰기, 실행 권한을 준거고 **그룹 사용자**에게는 1 👉 실행 권한만, **기타 사용자**에게도 1 👉 실행 권한만 준거다.

oflag와 마찬가지로 OR 연산자(|)로 연결해 여러 개를 지정할 수 있다.

ex) 만약 0644 권한을 주고 싶다면 OR 연산자를 사용해 이렇게 묶으면 된다. S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;

파일 닫기 (close)

```
#include <unistd.h>
int close(int fildes); // fildes는 파일 기술자이다
```

파일 입출력 작업을 모두 완료하면 반드시 close 메소드로 파일을 닫아야한다.

close 메소드는 파일을 성공적으로 닫을 경우 0을 리턴한다. 파일 닫기에 실패할 경우 -1을 리턴한다. (이를 이용해 조건문을 사용해서 파일이 제대로 닫혔는지 확인할 수도 있다 (errno 변수에 오류 코드를 저장하므로 왜 오류가 났는지도 확인 가능하다))

예제 2-2

파일 open, close 예제

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

int main()
{
    int fd;

    fd = open("unix.txt", O_CREAT | O_EXCL, 0644); // 파일이 없는 경우 0644
    권한으로 생성하고, 있는 경우 파일을 생성하지 않고 오류 메시지를 출력(O_EXCL)
    if (fd == -1){ // open 메소드는 파일 열기 실패시 -1을 리턴하므로, 조건문을 사용하여
    파일이 제대로 열렸는지 확인할 수 있다
        perror("Exc1");
        exit(1); // 에러 발생시 프로그램 종료
    }

    close(fd);

    return 0;
}
```

예제 2-3

파일 기술자 재할당 예제

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

int main()
{
    int fd;

    close(0); // 파일 기술자 0을 닫음 -> 키보드

    fd = open("unix.txt", O_RDWR); // 현재 파일 기술자의 가장 빠른 번호는 0번이므로,
    // 파일 open시 0번에 할당됨
    if(fd == -1){
        perror("Exc1");
        exit(1);
    }

    printf("unix.txt: fd = %d\n", fd); // 0번에 할당됐는지 출력 (출력 결과:
    // unix.txt: fd = 0)
    close(fd);

    return 0;
}
```

파일 읽기 (read)

```
#include <unistd.h>

ssize_t read(int fildes, void *buf, size_t nbytes);
```

read 메소드는 파일 기술자(fildes)가 가리키는 파일에서 지정한 크기(nbytes)만큼 바이트를 읽어 buf로 지정한 메모리에 저장하는 메소드이다.

실제로 읽어온 bytes 수를 리턴하며 오류 발생시 -1을 리턴한다. 만약 파일을 모두 읽어 끝에 도달한 경우 0을 리턴한다.

예제 2-4

파일 읽기 예제

```
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h> // exit 메소드
#include <stdio.h>

int main()
{
    int fd, n;
    char buf[10]; // 10bytes (char가 1byte니까)

    fd = open("unix.txt", O_RDONLY);
    if(fd == -1){ // open 메소드 오류 발생시 종료
        perror("Open");
        exit(1);
    }

    n = read(fd, buf, 6); // 읽어야될 파일(fd), 읽어서 저장할 임시 버퍼(buf), 읽어올
    // 바이트 수(6)
    if(n == -1){ // read 메소드 오류 발생시 종료
        perror("Read");
    }
}
```

```

    exit(1);
}

buf[n] = '\0'; // string으로 만들기 위해선 마지막에 '\0'를 붙여줘야함
printf("n=%d, buf=%s\n", n, buf); // n=6, buf=Unix S
close(fd);

return 0;
}

```

파일 쓰기 (write)

```

#include <unistd.h>
ssize_t write(int fildes, const void *buf, size_t nbytes);

```

read 함수와 인자 구조는 같지만 동작 방식이 다르다.

파일 기술자(fildes)는 쓰기를 수행할 파일을 가리키고, buf는 파일에 기록할 데이터를 저장하고 있는 메모리 영역을 가리킨다. (쓸 내용이라고 생각해도 될 듯)

buf가 가리키는 메모리에서 nbytes로 지정한 크기만큼 읽어 파일에 쓴다.

예제 2-5

파일 읽고 쓰기 예제

```

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

int main()
{
    int rfd, wfd, n;

```

```

char buf[10];

rfd = open("unix.txt", O_RDONLY); // read only로 파일 open
if(rfd == -1){ // read 파일 open 실패시 실행 종료
    perror("Open unix.txt");
    exit(1);
}

wfd = open("unix.bak", O_CREAT | O_WRONLY | O_TRUNC, 0644); // 파일이
없다면 0644 권한으로 생성하고(O_CREAT), 있다면 모든 내용을 지우고(O_TRUNC) write
only로 open
if(wfd == -1){
    perror("Open unix.bak");
    exit(1);
}

while((n = read(rfd, buf, 6)) > 0) // 파일의 끝이 될 때까지 읽음 (파일이 끝난
경우 0을 반환하므로)
    if(write(wfd, buf, n) != n) // n은 바로 위 while문에서 읽어온 bytes의 길이
가 담겨 있음 -> 파일 쓰거나 쓰여진 bytes의 길이와 다르다면 어딘가 덜 쓰여진 것이므로 에러를
발생하도록 함
        perror("Write");

if(n == -1) // read 메소드에서 -1을 리턴할 경우 에러가 발생한 것이므로 오류 메세지
출력
    perror("Read");

// 파일을 다 썼으면 파일 기술자는 꼭 닫아줍시다!
close(rfd);
close(wfd);

return 0;
}

```


파일 오프셋 지정

파일의 내용을 읽거나 쓸 경우 현재 읽을 위치(또는 쓸 위치)를 알려주는 오프셋(offset)이 자동으로 변경된다.

오프셋은 파일의 시작 지점에서 현재 위치까지의 바이트 수이다. 파일을 처음 열 경우 오프셋이 파일의 시작인 0이다.

! 한 파일에 파일 오프셋은 하나이다. 만약 파일을 읽기/쓰기 모드로 열었을 경우 하나의 오프셋으로 읽고, 써야한다.

lseek

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
```

lseek 메소드는 파일 기술자가 가리키는 파일에서 offset으로 지정한 크기만큼 오프셋을 이동시킨다.

이 때, whence 값을 기준으로 해석한다.

▪ whence

값	설명
SEEK_SET	파일의 시작을 기준으로
SEEK_CUR	현재 위치를 기준으로
SEEK_END	파일의 끝을 기준으로

ex1) lseek(fd, 5, SEEK_SET): 파일의 시작에서 5번째 위치로 이동

ex2) lseek(fd, 0, SEEK_END); 파일의 끝에서 0번째로 이동 (파일의 끝으로 이동)

+ 만약 파일의 현재 offset 위치를 알고싶은 경우 다음과 같이 현재 위치를 기준으로 0만큼 이동한 값을 구하면 됨

```
cur_offset = lseek(fd, 0, SEEK_CUR);
```

✚ 반대 방향으로 이동하려면 음수 값을 지정하면 됨

lseek 메소드는 실행에 성공하면 새로운 offset(off_t 자료형 - <sys/types.h>에 정의돼 있음)을 리턴하고, 실패하면 -1을 리턴한다.

예제 2-6

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

int main()
{
    int fd, n;
    off_t start, cur; // off_t는 <sys/types.h>에 정의된 자료형
    char buf[256];

    fd = open("unix.txt", O_RDONLY); // read only로 파일 open
    if(fd == -1){ // 파일 open 실패시 종료
        perror("Open unix.txt");
        exit(1);
    }

    start = lseek(fd, 0, SEEK_CUR); // 파일을 처음 읽기 때문에 오프셋 값이 0임
    (start = 0)
    n = read(fd, buf, 255);
    /* 255bytes까지만 읽는 이유?
    1. 마지막에 '\0'을 넣으려고
    + 만약 read하는 파일에 10bytes만 저장돼 있다면, 10bytes만 읽음 (245bytes는 노는
    거 ..)
    */
```

```

    buf[n] = '\0'; // 마지막에 '\0'을 넣어야 string이 됨
    printf("Offset start=%d, Read Str=%s, n=%d\n", (int)start, buf, n);
    // lseek는 long type을 반환해서 int로 type casting

    cur = lseek(fd, 0, SEEK_CUR); // 현재 오프셋 위치 출력
    printf("Offset cur=%d\n", (int)cur);

    start = lseek(fd, 5, SEEK_SET); // offset을 5로 변경
    n = read(fd, buf, 255);
    buf[n] = '\0';
    printf("Offset start=%d, Read Str=%s", (int)start, buf);

    close(fd);

    return 0;
}

```

파일 기술자 복사 (dup)

파일 기술자를 복사해 여러 변수가 같은 파일을 가리키도록 할 수 있다.

dup

```

#include <unistd.h>
int dup(int fildes); // fildes는 파일 기술자

```

dup 메소드는 기존 파일 기술자를 인자로 받아 새로운 파일 기술자를 리턴한다.

이때 새로 할당되는 파일 기술자는 **현재 할당할 수 있는 파일 기술자 중 가장 작은 값**으로 자동 할당된다.

보통 입출력을 전환할 때 사용한다.

예제 2-7

파일 기술자 변환 예제

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

int main()
{
    int fd, fd1;

    fd = open("tmp.aaa", O_CREAT | O_WRONLY | O_TRUNC, 0644); // 파일 기술자
3
    if(fd == -1){
        perror("Creat tmp.aaa");
        exit(1);
    }

    close(1); // 파일 기술자 1을 닫음 (표준 출력)

    fd1 = dup(fd); // tmp.aaa의 파일 기술자를 복사 -> 1번이 가장 낮은 번호이기 때문에
1에 할당

    // 원래는 프롬프트(표준 출력)에 출력돼야 하는데, 파일 기술자를 위에서 닫은 뒤 tmp.aaa로
재할당 했기 때문에 프롬프트에 출력되지 않고, tmp.aaa에 출력된다.
    printf("DUP FD=%d\n", fd1);
    printf("Standard Output Redirection\n");

    close(fd);

    return 0;
}
```

dup2

```
#include <unistd.h>

int dup2(int fildes, int fildes2);
```

dup2 메소드는 복사하는 파일 기술자의 번호를 지정할 수 있다.

예제 2-8

예제 2-7과 동작이 동일한 코드 (dup2를 사용)

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

int main()
{
    int fd;

    fd = open("tmp.bbb", O_CREAT | O_WRONLY | O_TRUNC, 0644);
    if(fd == -1){
        perror("Create tmp.bbb");
        exit(1);
    }

    dup2(fd, 1); // 1번 파일 기술자에 tmp.bbb를 할당

    printf("DUP2 : Standard Output Redirection\n");

    close(fd);

    return 0;
}
```

파일 삭제 (unlink)

```
#include <unistd.h>
int unlink(const char *path);
```

unlink 메소드는 path에 지정된 파일의 inode에서 링크 수를 감소시킨다.

❗ 이 메소드를 사용하기 위해서는 해당 프로세스가 해당 파일이 위치한 디렉토리 쓰기 권한이 있어야 한다.

+inode란?

inode는 유닉스에서 파일을 식별하는 유일한 식별자다.

만약 다음과 같이 cp명령어로 b를 복사할 경우, a와 b 안에 있는 내용은 동일하지만 **inode가 달라 서로 다른 파일로 취급된다.**

```
cp a b
```

추후에 배우겠지만, 링크 명령인 ln을 사용할 경우, a와 b안에 있는 내용도 동일하고 **inode도 동일해 서로 같은 파일이다.**

```
ln a b
```