



3장 파일 다루기

- 서론
- 예제 프로그램
- 파일 기술자와 읽기/쓰기 포인터
- 함수
 - open
 - creat
 - close
 - read, write
 - lseek
 - unlink, remove

한빛미디어(주)

● 시스템 호출 (system call), 표준 라이브러리 함수 (Standard Library Function)

사용자가 작성한 프로그램이 커널 수준의 서비스를 사용할 수 있다.

● 파일을 다루기 위한 시스템 호출/표준 라이브러리 함수

함수	의미
open	이미 존재하는 파일을 읽기 또는 쓰기용으로 열거나, 새로운 파일을 생성하여 연다.
creat	새로운 파일을 생성하여 연다.
close	open 또는 creat로 열려진 파일을 닫는다.
read	열려진 파일로부터 데이터를 읽어 들인다.
write	열려진 파일에 데이터를 쓴다.
lseek	파일 안에서 읽기/쓰기 포인터를 지정한 바이트 위치로 이동한다.
unlink/remove	파일을 삭제한다.

【예제 3-1】 ex03-01.c

```
01 /* program ex03-01.c */
02 #include <sys/types.h>
03 #include <fcntl.h>
04 #include <unistd.h>
05
06 main()
07 {
08     int filedес, fdnew1, fdnew2;
09     ssize_t nread;
10     off_t newpos;
11
12     char buffer[1024];
13     char content[] = "Hello my
14 friend!!\n";
15     filedес = open("data.txt", O_RDWR);
16
17     nread = read(filedес, buffer, 1024);
18     printf("%s", buffer);
```

08 파일 기술자로 사용된다.

12 읽어 들인 파일의 내용을 저장한
배열

15 data.txt 파일을 연다

17 buffer 크기만큼 파일을 읽는다
18 buffer의 내용을 출력한다



【예제 3-1】 ex03-01.c

```
19  /* 코드 계속됨 */
20
21  write(filedes, content, strlen(content));
22
23  newpos = lseek(filedes, (off_t)0, SEEK_SET);
24  nread = read(filedes, buffer, 1024);
25  printf("%s", buffer);
26
27  close(filedes);
28
29  fdnew1 = open("newdata1.txt", O_RDWR |
               O_CREAT, 0644);
30  fdnew2 = creat("newdata2.txt", 0644);
31
32  close(fdnew1);  close(fdnew2);
33  unlink("newdata2.txt");
34 }
```

21 배열 content의 내용을 파일에 적는다.

23 읽기/쓰기 포인터를 이동한다

24 파일의 내용을 읽는다

27 개방된 파일을 닫는다

29 새로운 파일을 생성하여 연다

30 새로운 파일을 생성하여 연다

32 개방된 파일을 닫는다

33 파일을 삭제한다.



● data.txt의 내용

```
$ cat data.txt  
Hello my world!!  
$
```

- (1) data.txt 파일의 크기는 몇 바이트인가?
- (2) data.txt 파일의 내용을 ASCII 코드 값으로 읽을 수 있는가?

● 실행 결과

```
$ ex03-01  
Hello my world!!  
Hello my world!!  
Hello my friend!!  
$
```

- (1) 실행 결과의 각 라인의 예제 프로그램의 몇 번째 라인에 의한 결과인지 생각해보자.



● 파일을 다루는 프로그램을 작성할 때 반드시 알아야 하는 개념

● 파일 기술자(file descriptor)

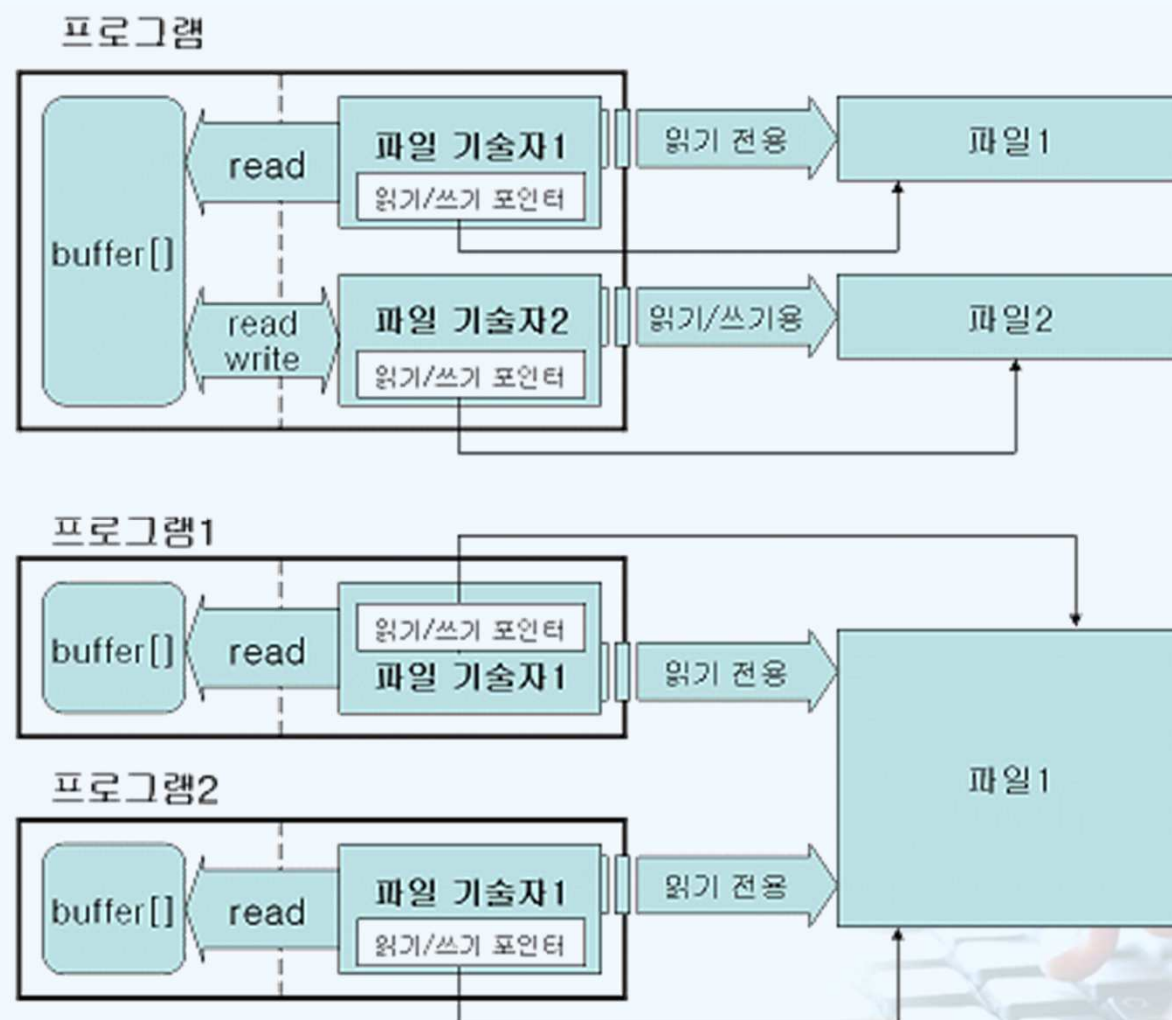
- 실행중인 프로그램과 하나의 파일 사이에 연결된 개방 상태
 - 음수가 아닌 정수형 값으로 시스템이 결정
 - 프로그램 작성 시 실제 값이 무엇인지 알 필요 없음
 - 파일 개방이 실패하면 -1이 됨
 - 커널에 의해서 관리
-
- 하나의 프로그램은 동시에 여러 개의 파일을 개방할 수 있다.
 - 여러 개의 프로그램이 동시에 하나의 파일을 개방할 수 있다.
 - 어떤 경우든 커널에 의해서 각 개방상태가 유일하게 식별되어 관리된다.

● 읽기/쓰기 포인터 (read/write pointer)

- ➔ 개방된 파일 내에서 읽기 작업이나 쓰기 작업을 수행할 바이트 단위의 위치
- ➔ 특정 위치를 기준으로 한 상대적인 위치를 의미
 - ▶ 그래서 오프셋(offset)이라고 한다.
- ➔ 파일을 개방한 직후에 읽기/쓰기 포인터는 0
 - ▶ 파일의 첫 번째 바이트를 가리킨다.
 - ▶ 파일의 내용을 읽거나 파일에 새로운 데이터를 작성하면 그 만큼 증가한다.
- ➔ 파일 기술자마다 하나씩 존재
 - ▶ 서로 다른 프로그램이 동일한 파일을 개방해도 파일 기술자가 다르기 때문에 마찬가지로 서로 다른 읽기/쓰기 포인터를 가진다.
 - ▶ 즉, 서로의 작업이 상대에게 영향을 주지 않는다.



Section 02 파일 기술자와 읽기/쓰기 포인터 IT CookBook



[그림3-1] 파일 기술자와 읽기쓰기 포인터

● 기존의 파일을 개방하거나 새로운 파일을 생성한 후에 개방한다.

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

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

<i>pathname</i>	개방할 파일의 경로 이름을 가지고 있는 문자열의 포인터이다.
<i>flags</i>	파일의 개방 방식을 지정한다.
<i>mode</i>	대부분의 경우 생략할 수 있는 값으로 새롭게 생성하는 파일의 초기 접근 권한을 지정한다.
반환값	정상적으로 파일을 개방하게 되면 파일 기술자를 반환하게 된다. 파일 개방이 실패할 경우 -1을 반환한다.

- ➔ 이미 존재하는 파일을 개방하려면 반드시 open을 사용한다.
새로운 파일을 생성할 때는 open이나 creat를 사용한다.

● pathname

개방하려는 파일의 경로 이름

- ▶ 상대 경로, 절대 경로
- ▶ 문자열 상수, 문자열형의 포인터

```
/* 절대 경로로 지정 */  
filesdes = open("/home/lsp/data.txt", O_RDONLY);  
/* 상대 경로로 지정 */  
filesdes = open("data.txt", O_RDONLY);  
...  
/* 문자열을 담고 있는 배열 */  
char pathname[] = "data.txt";  
/* 배열의 포인터로 경로를 지정 */  
filesdes = open(pathname, O_RDONLY);
```

● flags

파일을 개방하고 난 후의 접근 방식

▶ 읽기 전용, 쓰기 전용, 읽기 쓰기용

O_RDONLY	읽기만 가능한 상태로 접근한다.
O_WRONLY	쓰기만 가능한 상태로 접근한다.
O_RDWR	읽기, 쓰기 모두 가능한 상태로 접근한다.

● mode

- ➔ 파일의 접근 권한을 의미한다.
- ➔ 이미 존재하는 파일을 개방하여 사용할 경우 생략한다.
- ➔ 새로운 파일을 생성할 때 적용한다.



● 반환값

- ➔ 호출이 성공하면 음이 아닌 정수형의 값을 반환한다.
 - ▶ 파일 기술자
- ➔ 프로그램 내에서 파일 기술자는 중복되지 않는다.
 - 0, 1, 2는 반환값으로 나타나지 않는다.
 - ▶ 0, 1, 2는 각각 표준 입력, 표준 출력, 표준 에러로 사용된다.
- ➔ 반환값이 -1이면 파일 개방에 실패했다.
 - ▶ 지정한 파일이 존재하지 않는다.
 - ▶ 접근 권한이 없다.



● open 호출 실패에 대한 대비

- ➔ open의 반환값으로 호출의 성공 여부를 확인할 수 있다.
- ➔ 호출 실패시 적절한 대응이 있어야 한다.

시스템 호출이 실패했을 경우를 고려한 프로그램 코드

```
...
filesdes = open("data.txt", O_RDONLY);
/* filesdes가 -1이면 open 호출이 실패했다. */
if(filesdes == -1)
{
    printf("file open error!\n");
    exit(1);
}
```

● 개방된 상태의 파일을 닫는다

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

```
int close(int filedes);
```

filedes

이전에 open이나 creat에 의해 개방된 파일의 파일 기술자이다.

반환값

작업이 성공할 경우 0이 반환되며, 실패할 경우 -1이 반환된다.

- ➔ open이나 creat에 의해 개방된 파일을 닫는다.
 - ▶ 할당 받은 파일 기술자를 반환한다.
- ➔ 개방된 파일을 사용이 끝나면 반드시 닫아주어야 한다.
 - ▶ 하나의 프로세스가 동시에 개방할 수 있는 파일의 수는 제한되어 있음
 - ▶ 시스템 차원에서 동시에 개방할 수 있는 파일의 수는 제한되어 있음

```
/* 파일을 읽기 전용 상태로 개방한다. */  
filesdes = open("data.txt", O_RDONLY);  
...  
/* data.txt를 사용하는 코드가 위치 */  
...  
close(filesdes);
```

- ➔ 개방된 파일은 사용이 끝난 후 반드시 닫아준다.
- ➔ 개방된 파일을 닫지 않고 프로그램이 종료한 경우
 - ▶ 프로그램이 종료할 때 개방된 파일은 커널에 의해 자동으로 닫힌다.
 - ▶ 이런 사실을 알고 있더라도 사용된 파일은 마지막에 닫아주는 것이 좋다.

● **O_CREAT** 플래그

- **open**으로 새로운 파일을 생성할 때 사용한다.
- `filesdes = open("/tmp/tmpfile.txt", O_RDWR | O_CREAT, 0644);`

▶ **O_RDWR**

- 새롭게 생성된 파일은 읽기와 쓰기가 가능하다.

▶ **O_CREAT**

- /tmp/tmpfile.txt가 존재하지 않을 경우 새롭게 생성한다.
- 새로운 파일을 생성할 때 반드시 사용해야 한다.

▶ **0644**

- 새롭게 생성된 파일의 접근 권한은 0644이다.
- `rw-r--r--`



Section 05 **open**으로 새로운 파일 생성하기

IT CookBook

```
01 #include <stdlib.h>
02 #include <fcntl.h>
03
04 int main()
05 {
06     int filedес;
07     char pathname[] = "temp.txt";
08
09     if((filedes = open(pathname, O_CREAT |
10         O_RDWR, 0644)) == -1)
11     {
12         printf("file open error!\n");
13         exit(1);
14     }
15     close(filedes);
16 }
```

표준입력 스트림



● **O_EXCL** 플래그

- ➔ 이미 존재하는 파일을 **O_CREAT** 플래그를 사용하여 개방할 때 **O_EXCL** 플래그가 파일의 개방을 막는다. (즉 개방이 실패한다.)
- ➔ 이런 경우 사용한다..
 - ▶ 이미 존재하는 파일을 **O_CREAT** 플래그를 사용하여 개방하면 기존의 내용을 수정하는 실수를 범할 수 있다.
 - ▶ **O_EXCL** 플래그를 사용하여 개방 자체를 실패하게 한다.

```
files = open(pathname, O_CREAT | O_RDWR | O_EXCL, 0644);
```

- ▶ **pathname**에 해당하는 파일이 존재하지 않으면 새롭게 생성한 후에 읽기와 쓰기가 가능한 상태로 만들고 접근 권한은 0644이다.
- ▶ 이미 존재하면 **open**이 실패한다.

Section 05 **open**으로 새로운 파일 생성하기

IT CookBook

● **open**에서 자주 사용하는 플래그

파일 개방에 사용되는 플래그	
O_RDONLY	파일을 읽기 전용으로 개방한다. 읽기 이외의 다른 작업을 수행할 수 없다.
O_WRONLY	파일을 쓰기 전용으로 개방한다. 쓰기 이외의 다른 작업을 수행할 수 없다.
O_RDWR	파일을 읽기와 쓰기가 동시에 가능한 상태로 개방한다.
O_CREAT	지정한 경로의 파일이 존재하지 않으면 새롭게 생성한 후 개방한다. 지정한 경로의 파일이 존재하면 지정한 상태로 개방한다.
O_EXCL	지정한 경로의 파일이 존재하지 않으면 새롭게 생성하나, 지정한 경로의 파일이 존재하면 open 호출을 실패한다. (※O_CREAT 플래그와 함께 사용해야 한다.)
O_APPEND	파일을 개방한 직후에 읽기/쓰기 포인터의 위치를 파일 내용의 마지막 바로 뒤로 이동한다.
O_TRUNC	파일을 개방한 직후에 읽기/쓰기 포인터의 위치를 파일 내용의 첫 부분으로 이동한다.

Section 05 **open**으로 새로운 파일 생성하기

IT CookBook

```
filesdes = open(filename, O_RDWR);
```

filename으로 지정한 파일을 읽기와 쓰기가 가능한 상태로 개방

```
filesdes = open(filename, O_RDONLY | O_CREAT);
```

-filename으로 지정한 파일이 존재할 경우 읽기 전용으로 개방

-만약 파일이 존재하지 않으면 새롭게 생성한 후 읽기 전용으로 개방
(파일의 권한을 생략했기 때문에 기본 값으로 초기권한이 설정)

```
filesdes = open(filename, O_WRONLY | O_CREAT, 0644);
```

-filename으로 지정한 파일이 존재할 경우 쓰기 전용으로 개방

-만약 파일이 존재하지 않으면 새롭게 생성한 후 쓰기 전용으로 개방
-초기 권한이 0644로 설정

```
filesdes = open(filename, O_WRONLY | O_CREAT | O_EXCL, 0644);
```

-filename으로 지정한 파일이 존재하면 open의 수행이 실패

-만약에 파일이 존재하지 않으면 초기 권한 0644로 새롭게 파일을 생성
(쓰기 전용으로 개방)

● 새로운 파일을 생성한다

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

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

<i>pathname</i>	개방할 파일의 경로 이름을 가지고 있는 문자열의 포인터이다.
<i>mode</i>	새롭게 생성하는 파일의 초기 접근 권한을 지정한다. creat는 존재하지 않는 파일을 새롭게 생성하는 것이 사용 목적이기 때문에 생략할 수 없다.
반환값	정상적으로 파일을 개방하게 되면 파일 기술자를 반환하게 된다. 파일 개방이 실패할 경우 -1을 반환한다.

➡ creat는 open을 다음의 플래그와 함께 사용하는 것과 같다.

O_WRONLY, O_CREAT, O_TRUNC

● 이미 존재하는 파일을 지정하여 creat를 사용하는 경우

- ➔ 해당 파일을 개방함과 동시에 파일이 가지고 있는 데이터를 모두 삭제한다

```
filesdes = creat(pathname, 0644);  
...  
filesdes = open(pathname, O_WRONLY | O_CREAT | O_TRUNC, 0644);
```

- ➔ .creat에서 지정한 파일이 함부로 변경되어서는 안될 경우

creat보다는 open을 O_EXCL 플래그와 함께 사용한다.



예제 프로그램

```
01 #include <fcntl.h>
02
03 int main()
04 {
05     int filedес1, filedес2;
06
07     filedес1 = open("data1.txt", O_WRONLY |
08         O_CREAT | O_TRUNC, 0644);
09
10     close(filedес1);
11     close(filedес2);
12 }
```



● **read**는 파일 기술자로 지정한 파일에서 데이터를 읽어온다.

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

```
ssize_t read(int filedes, void *buf, size_t count);
```

<i>filedes</i>	읽기 작업을 수행할 파일에 대한 기술자이다.
<i>buf</i>	파일로부터 읽어 들인 내용을 저장하기 위한 공간이다. 일반적으로 배열을 사용하게 되는데 배열의 데이터 형식은 어느 것이라도 상관없다.
<i>count</i>	읽어 들일 파일 내용의 크기를 지정한다. 바이트 단위로 기술한다.
반환값	<p>파일로부터 읽기 작업이 성공할 경우</p> <ol style="list-style-type: none"> 1) 읽어 들인 파일 내용의 바이트 크기가 반환 (1 이상의 값) 2) 읽어 들인 내용이 없을 경우 (EOF일 경우) 0을 반환 <p>읽기 작업이 실패할 경우</p> <ol style="list-style-type: none"> 1) -1을 반환

● write는 파일 기술자로 지정한 파일로 데이터를 저장한다.

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

```
ssize_t write(int filedes, const void *buf, size_t count);
```

<i>filedes</i>	쓰기 작업을 수행할 파일에 대한 기술자이다.
<i>buf</i>	파일로 쓰려고 하는 내용이 저장되어 있는 공간이다. 일반적으로 배열을 사용하게 되면 배열의 데이터 형식은 어느 것이라도 상관없다.
<i>count</i>	buf에 있는 데이터 중에 실제로 파일로 저장할 데이터의 크기이다.
반환값	파일로 쓰기가 성공한 데이터의 크기이다. 대부분의 경우 count에서 지정한 값과 동일한 값이 반환된다. 만약 count의 값과 반환값이 다르다면 쓰기 작업이 실패한 것이다.

● read/write를 사용할 수 있는 파일의 개방 상태

함수	파일 개방 상태	
read	O_RDONLY	O_RDWR
write	O_WRONLY	

● 함수 호출의 성공 여부 판단

➔ read 함수

- ▶ 대부분의 경우 세 번째 인수 count로 지정한 값이 반환됨
- ▶ 파일의 마지막 부분을 읽을 경우 count보다 작은 값이 반환됨
- ▶ 반환값이 0일 경우 읽기/쓰기 포인터가 EOF(end-of-file)에 있음

➔ write 함수

- ▶ 모든 경우에서 반환값은 세 번째 인수 count로 지정한 값이 반환됨
- ▶ 반환값이 count로 지정한 값이 아닌 경우 쓰기 작업이 실패함

● read/write의 성공 여부를 검사하는 예제 코드

```
/* read가 정상적으로 수행되었는지 검사 */  
if((nread = read(filedes, buf, BUFSIZE)) > 0)  
    ...
```

- ➔ read는 0 이상의 값을 반환할 때 호출이 성공한 것이다.
- ➔ read가 0을 반환하면 파일의 내용을 모두 읽었기 때문에 더 이상 읽을 것이 없음을 의미한다.

```
/* write가 정상적으로 수행되었는지 검사 */  
if((write(filedes, buf, nread) < nread)  
    ...
```

- ➔ write는 항상 세 번째 인수로 지정한 것과 같은 값이 반환되어야 호출이 성공한 것이다.
- ➔ 호출이 실패할 경우 반환값은 세 번째 인수의 값보다 작다.

【예제 3-4】 ex03-04.c

temp1.txt 파일을 읽어서 temp2.txt로 복사한다.

```
01 #include <unistd.h>
02 #include <fcntl.h>
03
04 int main()
05 {
06     int fdin, fdout;
07     ssize_t nread;
08     char buffer[1024];
09
10     fdin = open("temp1.txt", O_RDONLY);
11     fdout = open("temp2.txt", O_WRONLY | O_CREAT
        | O_TRUNC, 0644);
```



【예제 3-4】 ex03-04.c

```
12  /* 정상적으로 읽어 들인 내용이 1바이트 이상인 동안 반복문 수행 */
13  while((nread = read(fdin, buffer, 1024)) > 0)
14  {
15      /* write가 비정상적으로 수행되었다. (실패) */
16      if(write(fdout, buffer, nread) < nread)
17      {
18          close(fdin);
19          close(fdout);
20      }
21  }
22
23  /* 프로그램이 정상적으로 수행되었다. */
24  close(fdin);
25  close(fdout);
26 }
```



●지정한 파일에 대해서 읽기/쓰기 포인터의 위치를 임의로 변경한다.

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

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

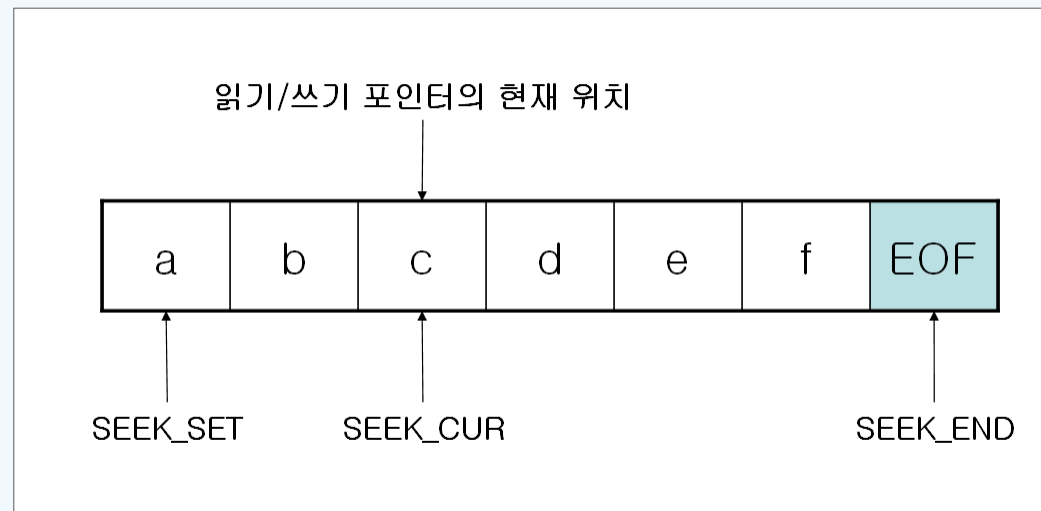
```
off_t lseek(int filedes, off_t offset, int whence);
```

<i>filedes</i>	읽기/쓰기 포인터를 변경할 파일을 지정한다.
<i>offset</i>	새롭게 지정할 읽기/쓰기 포인터의 위치를 의미한다. 오프셋이기 때문에 기준에 따라 음수가 될 수도 있다.
<i>whence</i>	offset의 기준이 된다. 파일의 맨 처음(SEEK_SET), 현재 포인터의 위치(SEEK_CUR), 파일의 맨 마지막(SEEK_END) 등 세 가지가 있다.
반환값	작업이 성공하면 파일의 첫 부분을 기준으로 한 포인터의 오프셋을 반환한다. 작업이 실패할 경우 (off_t)-1이 반환된다.

● 세 번째 인수 whence

offset의 기준

SEEK_SET	파일의 첫 번째 바이트를 시작점으로 한다.
SEEK_CUR	읽기/쓰기 포인터의 현재 위치를 시작점으로 한다.
SEEK_END	파일의 끝(end-of-file)을 시작점으로 한다.



● lseek의 사용 예

```
off_t newpos;  
newpos = lseek(filedes, (off_t)0, SEEK_SET);
```

➔ 파일의 첫 번째 바이트로 읽기/쓰기 포인터를 옮긴다.

```
newpos = lseek(filedes, (off_t)2, SEEK_CUR);
```

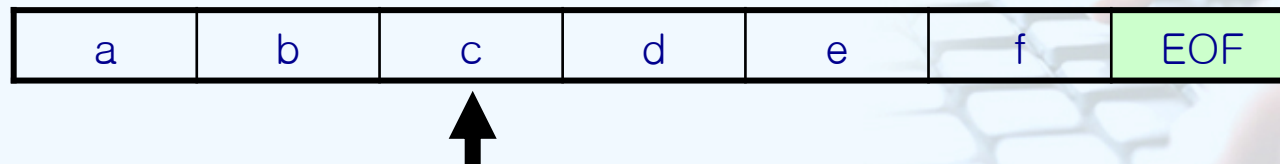
➔ 현재 위치에서 뒤로 2바이트만큼 옮긴다.

```
newpos = lseek(filedes, (off_t)0, SEEK_END);
```

➔ 파일의 마지막 바이트 바로 뒤(EOF)로 옮긴다.

```
newpos = lseek(filedes, (off_t)-1, SEEK_END);
```

➔ 파일의 마지막 바이트로 옮긴다.



【예제 3-5】 ex03-05.c

```
01 /* lseek을 사용하여 파일의 크기를 계산한다. */
02 #include <sys/types.h>
03 #include <unistd.h>
04 #include <fcntl.h>
05
06 int main()
07 {
08     int filedес;
09     off_t newpos;
10
11     filedес = open("data1.txt", O_RDONLY);
12
13     /* 읽기/쓰기 포인터를 EOF로 이동한다. */
14     newpos = lseek(filedes, (off_t)0, SEEK_END);
15
16     printf("file size : %d\n", newpos);
17 }
```



● 경로명으로 지정한 파일을 삭제한다

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

<i>pathname</i>	삭제할 파일의 경로 이름이다.
반환값	작업이 성공할 경우 0이 반환되며, 실패할 경우 -1이 반환된다.

- ➔ **pathname**으로 지정한 파일을 삭제한다.
- ➔ 비어 있는 디렉터리는 **remove**만 삭제할 수 있다. (**unlink**는 불가능)
 - ↔ 비어 있지 않은 디렉터리는 둘 모두 삭제할 수 없다.