리눅스 시스템 프로그래밍

day3 참고 자료

고 강 태

010-8269-3535

james@thinkbee.kr

https://www.linkedin.com/in/thinkbeekr/

http://www.facebook.com/gangtai.goh

lseek

- 지정한 파일에 대해서 읽기/쓰기 포인터의 위치를 임의로 변경한다.

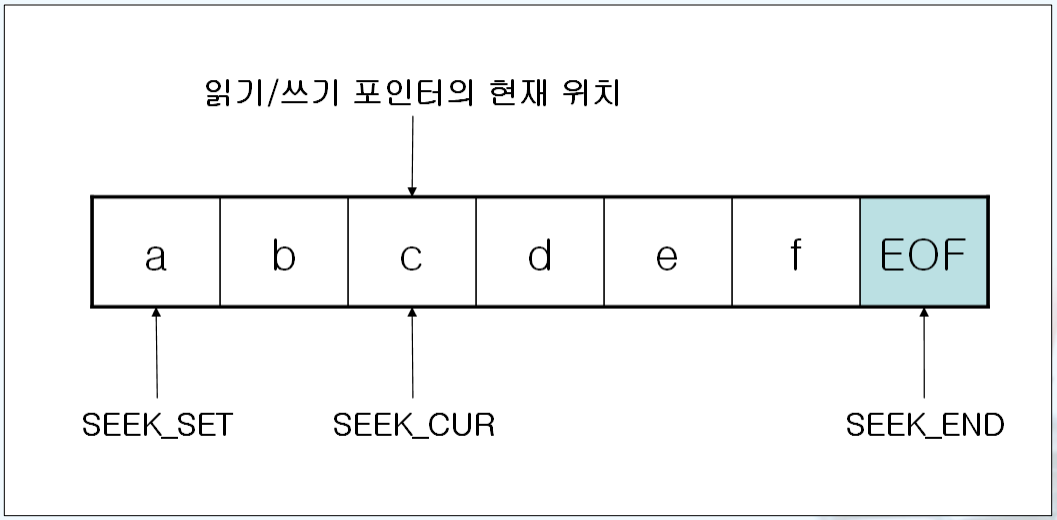
```
#include <sys/types.h>
#include <unistd.h>

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

filedes	읽기/쓰기 포인터를 변경할 파일을 지정한다.
offset	새롭게 지정할 읽기/쓰기 포인터의 위치를 의미한다. 오프셋이기 때문에 기준에 따라 음수가 될 수도 있다.
whence	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);
```

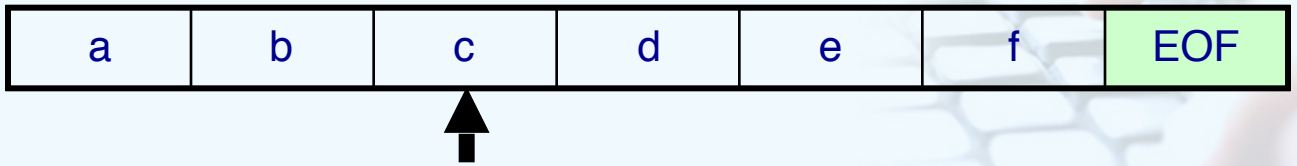
- 현재 위치에서 뒤로 2바이트만큼 옮긴다.

```
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.



실습: filecp.c

터미널에서 파일 이름을 받아 내용을 복제해 새 파일로 복사한다.

$ filecp hello.txt hello.bak

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

#define STDIN 0
#define STDOUT 1
#define STDERR 2

int main(int argc, char **argv)
{
    char *orgfile, *dstfile;
    int rfd, wfd;
    int readn;
    char buf[80];

    if (argc != 3)
    {
        printf("Usage : %s [src file] [dst file]\n", argv[0]);
        return 1;
    }

    orgfile = argv[1];
    dstfile = argv[2];

    rfd = open(orgfile, O_RDONLY);

    if (rfd < 0) // 에러 체크를 한다.
    {
        perror("org file open err :");
        return 0;
    }

    wfd = open(dstfile, O_WRONLY | O_CREAT, 0755);

    if (wfd < 0) // 에러 체크를 한다.
    {
        perror("dst file open err :");
        return 0;
    }

    while ((readn = read(rfd, buf, 80)) > 0)
    {
        write(wfd, buf, readn);
    }
    close(rfd);
    close(wfd);
    return 1;
}
```

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

void print_fileinfo(FILE *fp);

int main()
{
    getchar();
    print_fileinfo(stdin);
    getchar();
    print_fileinfo(stdin);
    getchar();
    print_fileinfo(stdin);
    getchar();
    print_fileinfo(stdin);
    return 0;
}

void print_fileinfo(FILE *fp)
{
    printf("----------------------------------------\n");
    printf("_fileno: %d\n", fp->_fileno);
    printf("_flags : %#x\n", fp->_flags);
    int dcnt = (fp->_IO_read_end - fp->_IO_read_ptr);
    printf("data count : %d\n", dcnt);
    printf("_IO_read_base : %s\n", fp->_IO_read_base);
    printf("_IO_read_ptr : %s\n", fp->_IO_read_ptr);
}
```

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

void print_fileinfo(FILE *fp);

int main()
{
    putchar('e');
    print_fileinfo(stdout);
    putchar('h');
    print_fileinfo(stdout);
    putchar('p');
    print_fileinfo(stdout);
    putchar('u');
    print_fileinfo(stdout);
    putchar('b');
    print_fileinfo(stdout);
    putchar('\n');
    return 0;
}
```

```c
void print_fileinfo(FILE *fp)
{
    printf("-----------------------------------------\n");
    printf("_fileno: %d\n", fp->_fileno);
    printf("_flags : %#x\n", fp->_flags);
    int dcnt = (fp->_IO_read_end - fp->_IO_read_ptr);
    printf("data count : %d\n", dcnt);
    printf("_IO_read_base : %s\n", fp->_IO_read_base);
    printf("_IO_read_ptr : %s\n", fp->_IO_read_ptr);
}
```

Buffering 정책

Standard I/O

교재 4장/51쪽

표준 입출력 라이브러리는 C언어 창시자인 데니스리치에 의해 1975년에 만들어졌으며 지금도 거의 개발 초기 모습을 유지하고 있습니다. 표준 입출력 라이브러리는 개발자 편의성과 효율성이 높은 파일 입출력을 제공하기 위해 만들었습니다

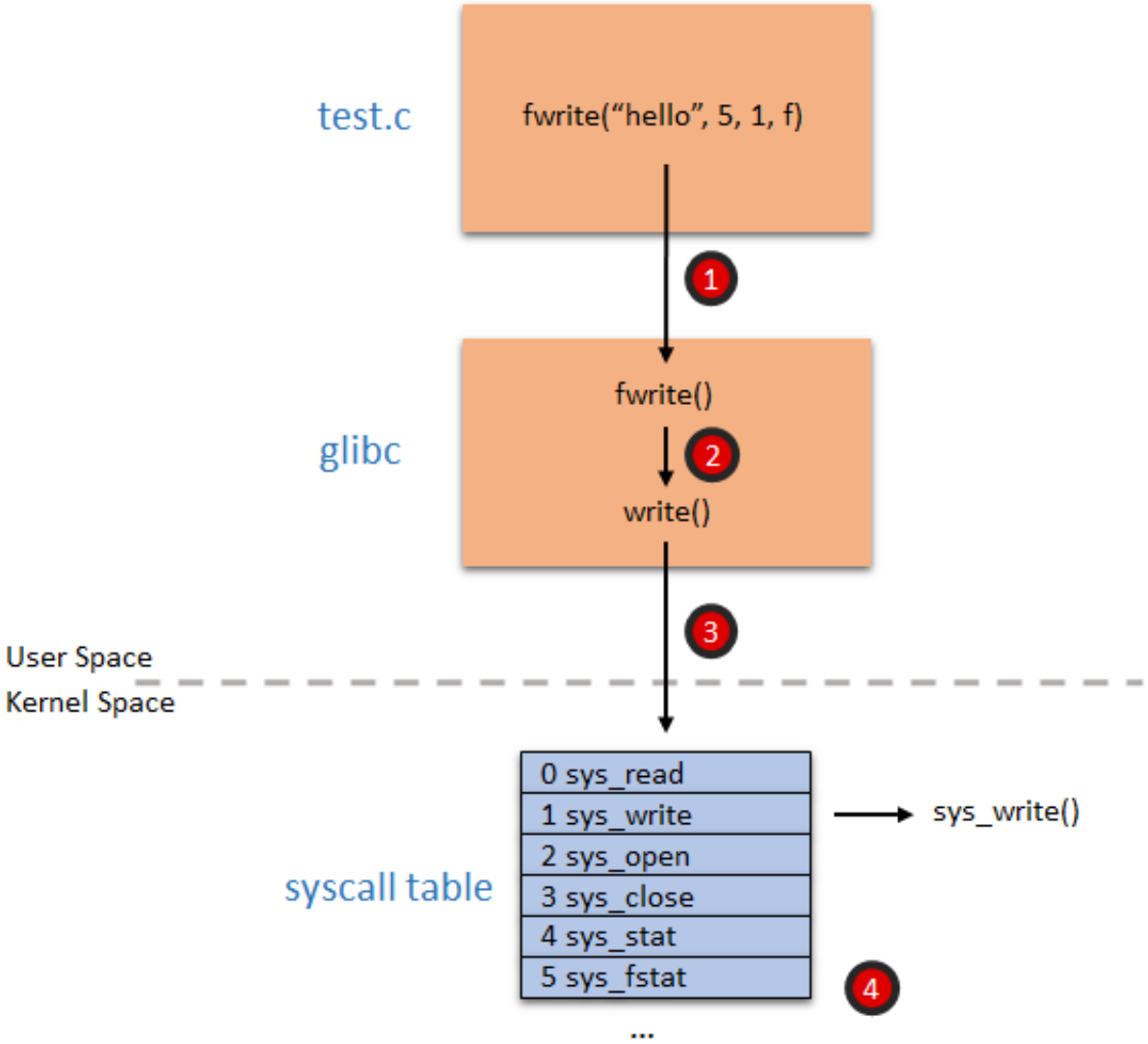
예를 들어 write 시스템 호출을 이용하여 1바이트 데이터를 100번 기록한다고 가정하면 커널은 100번의 쓰기 작업을 수행합니다. 반면 표준 입출력 라이브러리의 fwrite 함수를 이용하면 파일 스트림 버퍼에 저장하고 버퍼가 꽉 차거나 버퍼에 개행이 올 때 write 시스템 호출하여 커널에서는 1번의 쓰기 작업을 수행합니다. 물론 표준 입출력 라이브러리에서는 파일 스트림 버퍼를 어떠한 정책에 의해 동작할 것인지 설정할 수 있습니다.

버퍼링 정책

리눅스에서 제공하는 버퍼링 정책에는 버퍼가 꽉 차면 물리적인 파일에 기록하는 FULL BUFFERING과 꽉 차거나 개행문자가 오면 처리하는 LINE BUFFERING, 버퍼를 사용하지 않는 NULL BUFFERING 정책을 제공

- **디폴트 버퍼링 정책은 FULL BUFFERING**
- **char 장치은 LINE BUFFERING 정책**
- **stderr 파일 스트림은 NULL BUFFERING을 사용**

교재 4장/15쪽


test.c
fwrite("hello", 5, 1, f)
1
glibc
fwrite()
2
write()
3
User Space
Kernel Space
0 sys_read
1 sys_write
2 sys_open
3 sys_close
4 sys_stat
5 sys_fstat
...
sys_write()
syscall table
4

버퍼링 정책- ex_buffer.c

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

int main()
{
    putchar('e');
    sleep(1);
    fputc('H', stderr);

    putchar('h');
    sleep(1);
    fputc('e', stderr);

    putchar('\n');
    sleep(1);
    fputc('l', stderr);

    putchar('p');
    sleep(1);
    fputc('l', stderr);

    putchar('u');
    sleep(1);
    fputc('o', stderr);

    putchar('b');
    sleep(1);
    fputc('!', stderr);

    putchar('\n');
    sleep(1);
    fputc('\n', stderr);

    return 0;
}
```

버퍼기반 출력 - studentseek.c

```c
#define MAX_NAME_LEN 20

typedef struct _Student Student;
struct _Student {
    char name[MAX_NAME_LEN + 1];
    int age;
    int num;
};

int main(){
    store_students();
    search_student();
    return 0;
}

void store_students() {
    Student base[4] = {
        {"홍길동", 20, 3}, {"강감찬", 30, 4},
        {"김유신", 70, 1}, {"이순신", 35, 2}};
    FILE *fp = fopen("students.dat", "wb");
    fwrite(base, sizeof(Student), 4, fp);
    fclose(fp);
}
```

```c
void search_student()
{
    FILE *fp = fopen("students.dat", "rb");
    fseek(fp, 0, SEEK_END);
    int mcnt = ftell(fp) / sizeof(Student);
    rewind(fp);

    int nth = 0;
    printf("원하는 순번(1~%d):", mcnt);
    scanf("%d", &nth);

    if ((nth <= 0) || (nth > mcnt))
    {
        printf("잘못 입력하였습니다.\n");
        return;
    }

    Student student;
    fread(&student, sizeof(Student), 1, fp);
    fclose(fp);
    printf("이름:%s 번호:%d 나이:%d\n",
       student.name, student.num, student.age);
}
```

5장 - 파일 관련 몇몇 상수

매크로 상수의 의미

이름	의미
R_OK	읽기 권한을 검사한다.
W_OK	쓰기 권한을 검사한다.
X_OF	실행 권한을 검사한다. (디렉터리일 경우 탐색 권한을 검사한다.)
F_OK	대상 파일이 존재하는지 검사한다.

접근 권한의 표현

- 8진수 값
- 미리 정의된 매크로 상수

8진수 값	상수이름	의미
0400	S_IRUSR	소유자에 대한 읽기 권한
0200	S_IWUSR	소유자에 대한 쓰기 권한
0100	S_IXUSR	소유자에 대한 실행 권한
0040	S_IRGRP	그룹 사용자에 대한 읽기 권한
0020	S_IWGRP	그룹 사용자에 대한 쓰기 권한
0010	S_IXGRP	그룹 사용자에 대한 실행 권한
0004	S_IROTH	기타 사용자에 대한 읽기 권한
0002	S_IWOTH	기타 사용자에 대한 쓰기 권한
0001	S_IXOTH	기타 사용자에 대한 실행 권한

매크로 상수 사용

- 비트별 OR 연산자(|)를 사용한다

```
mode_t mode;
mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
chmod("test.txt", mode);
```

- 위는 아래와 같다.

```
chmod("test.txt", 0644);
```

- 덧셈 연산자(+)를 잘못 사용한 경우

```
mode = S_IRUSR | S_IXUR;
mode = mode | S_IRUSR;   /*  실수로 S_IWUSR 대신에 S_IRUSR를 적음 */
…
mode = S_IRUSR + S_IXUR;
mode = mode + S_IRUR;
```

※잘못된 결과를 8진수로 표현해보고 실제로 어떻게 설정되는지 확인해보라.

- 하드 링크와 소프트 링크의 비교