Part 02. 리눅스 시스템 프로그래밍

Chapter 02. 파일 I/O(2)

진행 순서

Chapter 02_01 파일 열고 닫기 (open, close, fcntl)
Chapter 02_02 파일 읽고 쓰기 (read, wirte, lseek)
Chapter 02_03 실습 프로그램

Chapter 02_01 파일 열고 닫기 (open)

```
#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);
```

open은 시스템호출로, 파일을 열거나 생성할 때 사용한다.
성공하면 해당파일을 지시하는 int 형의 파일 디스크립터를 되돌려준다.
path_name 은 생성하거나 열고자 하는 파일이름을 나타낸다.
보통 full path 이름을 적어주며, 단지 파일이름만 적을 경우에는 현재 경로에서 찾는다.

flag 는 파일을 어떠한 모드로 열 것인지를 결정하기 위해서 사용한다.
" 읽기전용 " , " 쓰기전용 " , " 읽기/쓰기 " 모드로 열수 있다.
이들 모드 선택을 위해서 O_RDONLY, O_WRONLY, O_RDWR 이 존재 한다.

open은 mode 인자가 붙은 형식과 붙지 않은 형식 둘 다 유효하다.
mode 인자는 파일을 생성 시 파일의 권한(소유권)을 나타낸다.
파일을 생성(O_CREAT)하지 않으면 mode 인자는 무시된다.
반대로 O_CREAT으로 파일을 생성 시 파일 권한이 정의되지 않는다면 종종 골치 아픈 일을 겪는다.

정상 수행 시 파일 디스크립터 리턴, 에러 발생 시 -1 리턴(errno)

Chapter 02_01 파일 열고 닫기 (open)

flags
O_CREAT : 만약 pathname 파일이 존재하지 않을 경우 파일을 생성한다.
O_EXCL : O_CREAT 를 이용해서 파일을 생성하고자 할 때, 이미 파일이 존재한다면 에러발생
O_APPEND : 파일이 추가모드로 열린다. 파일의 위치는 파일의 끝이 된다.
O_NONBOLOCK, O_NDELAY : 파일이 비봉쇄(Nonblock) 모드로 열린다.
O_NOFOLLOW : 경로명이 심볼릭링크라면, 파일열기에 실패한다.
O_DIRECTORY : 경로명이 디렉토리가 아니라면 파일열기에 실패한다.
O_SYNC : 입출력 동기화 모드로 열린다. 모든 write 는 데이터가 물리적인 하드웨어에 기록될 때까지 호출 프로세스를 블록

mode
S_IRWXU : 00700 모드로 파일 소유자에게 읽기, 쓰기, 쓰기 실행권한을 준다.
S_IRUSR : 00400 으로 사용자에게 읽기 권한을 준다.
S_IWUSR : 00200 으로 사용자에게 쓰기 권한을 준다.
S_IXUSR : 00100 으로 사용자에게 실행 권한을 준다.
S_IRWXG : 00070 으로 그룹에게 읽기, 쓰기, 실행 권한을 준다.
S_IRGRP : 00040 으로 그룹에게 읽기권한을 준다.
S_IWGRP : 00020 으로 그룹에게 쓰기권한을 준다.
S_IXGRP : 00010 으로 그룹에게 실행권한을 준다.
S_IRWXO : 00007 으로 기타 사용자 에게 읽기, 쓰기, 실행 권한을 준다.
S_IROTH : 00004 으로 기타 사용자 에게 읽기 권한을 준다.
S_IWOTH : 00002 으로 기타 사용자 에게 쓰기 권한을 준다.
S_IXOTH : 00001 으로 기타 사용자 에게 실행 권한을 준다.

Chapter 02_01 파일 열고 닫기 (open)

errno
EEXIST : O_CREAT 와 O_EXECL 이 같이 사용되었을 경우 발생한다. 이미 경로파일이 존재할 경우 발생된다.
EACCES : 파일 접근이 거부될 경우이다. 주로 권한 문제 때문에 발생한다.
ENOENT : 경로명의 디렉토리가 없거나, 심볼릭 링크가 깨져있을 때.
ENOENT : 경로명의 디렉토리가 없거나, 심볼릭 링크가 깨져있을 때.
ENODEV : 경로명이 장치파일을 참고하고, 일치하는 장치가 없을 때.
EROFS : 경로명이 read-only 파일시스템을 참조하면서, 쓰기로 열려고 할 때.
EROFS : 경로명이 read-only 파일시스템을 참조하면서, 쓰기로 열려고 할 때.
EFAULT : 경로명이 접근할 수 없는 주소강간을 가리킬 때
ELOOP : 심볼릭 링크가 너무 많을 때.

Chapter 02_01 파일 열고 닫기 (close)

```
#include <unistd.h>

int close(int fd);
```

close()는 파일 디스크립터를 닫아서 더 이상 파일을 참조하지 않고 재사용 할 수 있도록 합니다.
프로세스와 연관되고 프로세스가 소유한 파일에 보유된 모든 레코드 잠금이 제거 됩니다.
fd가 열린 마지막 파일 디스크립터인 경우 연관된 자원이 해제됩니다.
파일 디스크립터가 unlink를 사용하여 제거된 파일에 대한 마지막 참조인 경우 파일이 삭제됩니다.

정상 수행 시 파일 디스크립터 리턴, 에러 발생 시 -1 리턴(errno)

errno
EBADF : fd가 유효한 파일 디스크립터가 아닌 경우
EINTR : 시그널 호출에 의해 인터립트된 경우
EIO : I/O 에러 발생된 경우
ENOSPC, SDQUOT : NFS에서 스토리지 공간 초과 시 write, fsync, close에 대해 보고

Chapter 02_01 파일 열고 닫기 (fcntl)

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

int fcntl(int fd, int cmd, ... /* arg */ );
```

fcntl()은 파일 디스크립터를 조작합니다.
fcntl()은 열린 파일 디스크립터 fd에 대해 아래 설명된 작업 중 하나를 수행합니다.
작업은 cmd에 의해 결정됩니다.
선택적으로 세 번째 인수를 취할 수 있으며, 필요 여부는 cmd에 의해 결정됩니다.
인수가 필요하지 않은 경우 void가 지정

Chapter 02_01 파일 열고 닫기 (fcntl)

CMD

파일 디스크립터 복제

F_DUPFD (int) : 복사된 파일 디스크립터는 잠금, 파일위치 포인터, 플래그 등을 공유한다.
lseek 등으로 위치 변경 시 복제된 모든 파일 디스크립터도 변경됨

파일 상태 플래그

F_GETFL (void) : 파일 디스크립터에 대한 플래그 값(open 호출 시 지정한 플래그)을 되돌려준다.
F_SETFL (int) : arg 에 지정된 값으로 파일 디스크립터 fd 의 플래그를 재 설정한다.
현재는 단지 O_APPEND, O_ASYNC, O_DIRECT, O_NONBLOCK 만을 설정할 수 있다.
다른 플래그들 (파일 액세스 플래그 O_WRONLY 와 같은, 파일 생성 플래그 O_CREAT 와 같은) 은 영향을 받지 않는다.

레코드 잠금

F_SETLK (struct flock *) : 잠금을 획득하거나 잠금을 풀기 위해서 사용
F_SETLKW (struct flock *) : F_SETLK 과 같은 일을 하지만, 에러 리턴하는 대신 잠금이 풀릴 때까지 대기(block)
F_GETLK (struct flock *) : 잠금이 있는지 없는지 검사한다.

```
struct flock {
    short int l_type;       /* 잠김 타입: F_RDLCK, F_WRLCK, or F_UNLCK.  */
    short int l_whence;     /* 파일의 절대적 위치 */
    __off_t l_start;        /* 파일의 offset */
    __off_t l_len;          /* 잠그고자 하는 파일의 길이 */
    __pid_t l_pid;          /* 잠금을 얻은 프로세스의 pid */
};
```

그 외 많은 기능들 존재

Chapter 02_02 파일 읽고 쓰기 (read)

```
#include <unistd.h>

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

socket() 이나 open() 등으로 열린 파일 디스크립터에서 데이터를 읽어 들인다.
fd에 읽을 데이터가 있다면 buf에 담아서 가져온다.
count는 buf에 한번에 가져올 데이터의 크기를 의미한다.

성공할 경우 0 이상의 값을 반환한다. (읽어들인 buf의 크기)
0 이라면 파일의 끝을 의미하며, 데이터를 가져오는데 성공했다면 파일 포인터의 위치는 읽은 데이터 크기만큼 이동
에러가 발생할 경우 -1 리턴 (errno)

Chapter 02_02 파일 읽고 쓰기 (write)

```
#include <unistd.h>

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

open()이나 socket() 등으로 열린 파일 디스크립터가 가리키는 파일에 쓴다.
buf는 쓸 데이터이며, count는 쓸 데이터의 크기이다.

성공할 경우 쓰여진 바이트 크기 만큼 리턴 된다.
0이면 쓰여진 것이 없음을 나타내며,
-1일 경우 에러 발생(errno)

Chapter 02_02 파일 읽고 쓰기 (lseek)

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

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

lseek()은 열린 파일 디스크립터 fd의 위치 포인터를 offset 만큼 위치를 변경한다.
위치 변경 시 whence를 이용해 기준점을 정할 수 있다.

SEEK_SET
- 파일의 처음을 기준으로 offset을 계산

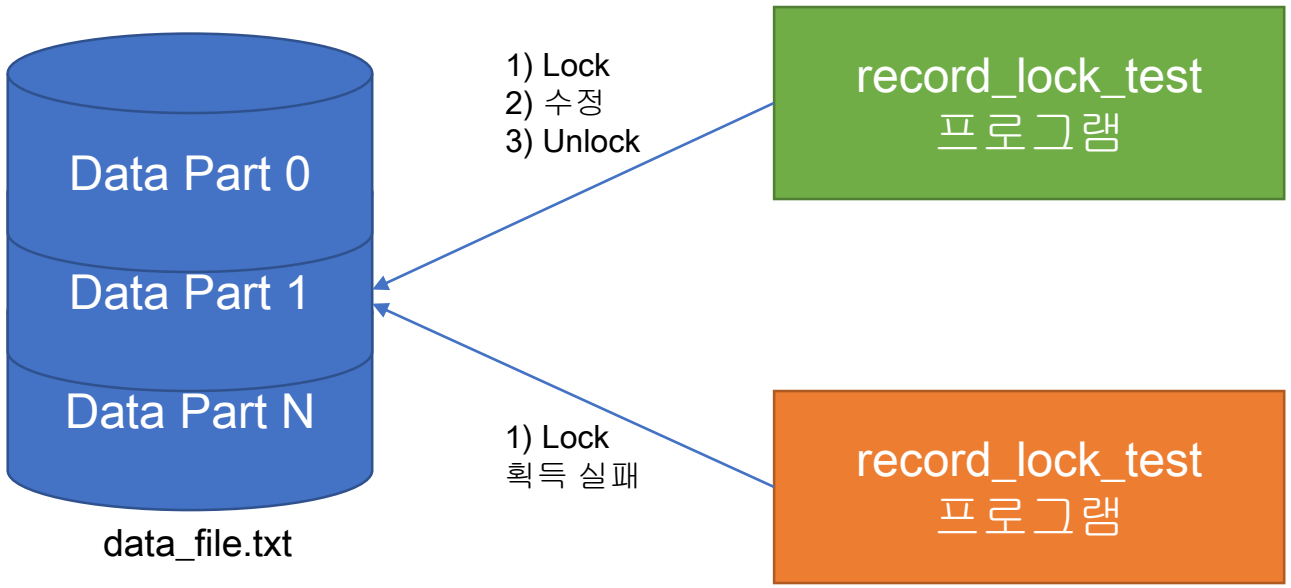
SEEK_CUR
- 파일의 현재 위치를 기준으로 offset을 계산

SEEK_END
- 파일의 마지막을 기준으로 offset을 계산

성공했을 경우 파일의 시작으로부터 멀어진 byte만큼의 offse을 리턴한다.
실패했을 경우 -1 리턴(errno)

Chapter 02_03 실습 프로그램

어떤 데이터가 저장되어 있는 파일을 열어 특정 부분의 데이터를 수정을 시도하고,
수정을 시도하는 동안 해당 파일의 수정중인 데이터 부분에 잠금을 걸어 다른 프로세스가 참조할 수 없도록 수행



Chapter 02_03 실습 프로그램

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

#define LOCK_T 0
#define UNLOCK_T 1
#define BUFSIZE 1024

int record_lock(int type, int fd, int start, int len);

int main(int argc, char **argv)
{
        int fd;
        int record_start, record_len;
        char buf[BUFSIZE] = {0};
        int i;

        if (argc < 4) {
                printf("Usage: %s [record file] [record start] [record length]\n", argv[0]);
                exit(0);
        }
```

record_lock_test.c

Chapter 02_03 실습 프로그램

```
        fd = open(argv[1], O_RDWR);
        if (fd == -1) {
                perror("file open error: ");
                exit(0);
        }

        record_start = atoi(argv[2]);
        record_len = atoi(argv[3]);
        if (record_len > BUFSIZE) {
                printf("record_len(%d) cannot over %d\n", record_len, BUFSIZE);
                exit(0);
        }

        /* record lock */
        if (record_lock(LOCK_T, fd, record_start, record_len) == -1) {
                perror("record lock error: ");
                exit(0);
        }

        /* process data */
        lseek(fd, record_start, SEEK_SET);
        if (read(fd, buf, record_len) < 0) {
                perror("read error: ");
                exit(0);
        }
        printf("record data = %s\n", buf);
```

record_lock_test.c

Chapter 02_03 실습 프로그램

```
        /* data modify */
        for (i = 0; i < record_len; i++) {
                if (buf[i] == '0' || buf[i] == '9')
                        buf[i] = 'x';
        }
        lseek(fd, record_start, SEEK_SET);
        write(fd, buf, record_len);

        /* delay 20 sec */
        sleep(20);
        printf("record lock process done\n");

        /* record unlock */
        if (record_lock(UNLOCK_T, fd, record_start, record_len) == -1) {
                perror("record unlock error: ");
                exit(0);
        }

        close(fd);

        return 0;
}
```

record_lock_test.c

Chapter 02_03 실습 프로그램

```
int record_lock(int type, int fd, int start, int len)
{
        int ret;
        struct flock lock;

        lock.l_type = (type == LOCK_T) ? F_WRLCK : F_UNLCK;
        lock.l_start = start;
        lock.l_whence = SEEK_SET;
        lock.l_len = len;

        ret = fcntl(fd, F_SETLK, &lock);
        return ret;
}
```

record_lock_test.c

```
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
```

data_file.txt

Chapter 02_03 실습 프로그램 (결과)

```
[root@localhost ch9]# ls -hl
합계 8.0K
-rw-r--r--. 1 root root  110  3월 31 13:53 data_file.txt
-rw-r--r--. 1 root root 1.7K  3월 31 13:52 record_lock_test.c

[root@localhost ch9]# gcc -g record_lock_test.c -o record_lock_test
[root@localhost ch9]# ./record_lock_test
Usage: ./record_lock_test [record file] [record start] [record length]

[root@localhost ch9]# ./record_lock_test data_file.txt 11 10 &
[1] 2310
[root@localhost ch9]# record data = 0123456789

[root@localhost ch9]# ./record_lock_test data_file.txt 11 10
record lock error: : Resource temporarily unavailable

[root@localhost ch9]# record lock process done
[1]+  Done                    ./record_lock_test data_file.txt 11 10

[root@localhost ch9]# cat ./data_file.txt
0123456789
x12345678x
0123456789
0123456789
...
```