Linux System Programming #1

Spring, 2020

Soongsil University, Seoul, Korea

Jiman Hong

II. 파일 입출력

- **파일 입출력 관련 시스템호출 함수**
- **파일**
  - 키보드에서 데이타를 입력하고 터미널로 데이타를 출력하는 것을 포함하여 데이터를 입력하고 출력하는 모든 대상
- **파일 입출력을 다루는 방법 :**
  - 저수준 파일 입출력
    - 시스템의 커널에서 제공하는 시스템호출 함수를 호출하는 것
    - 시스템호출을 이용하기 때문에 고수준 파일 입출력보다 좀 더 빠르게 파일을 다룰 수 있음
    - 바이트 단위로 파일을 다루기 때문에 일반 파일 뿐만 아니라 특수 파일(데이터 전송 및 디바이스 접근에 사용하는 파일)을 다룰 수 있음
    - 기본적으로 오픈한 파일을 다룰 때 open()의 리턴 값인 파일 디스크립터(file descriptor, 파일 기술자)를 사용
  - 고수준 파일 입출력 방법
    - IV장에서 상세하게 설명

파일과 파일디스크립터

Linux·Unix에서 파일

- 데이터를 읽을 수 있거나 데이터를 쓸 수 있는 모든 객체
- 디스크 등의 스토리지에 저장되어 있는 파일 뿐만 아니라 모든 디바이스도 파일로 취급
- Linux·Unix에서는 이러한 파일을 바이트 단위의 순차적인 스트림으로 처리

Linux·Unix에서 파일의 종류

- 사용자 프로그램/시스템 유틸리티 프로그램에 의해 입력된 정보를 포함하는 일반 파일(정규 파일)
- 일반 파일을 조직하고 접근할 수 있는 정보를 포함하는 디렉토리 파일
- 터미널이나 프린터와 같이 입출력 디바이스의 접근을 위해 사용되는 특별(special) 파일

파일 디스크립터

- 파일을 시스템 프로그래밍 차원에서 바이트 단위의 입출력으로 다룰 수 있게 하고, 커널 내부의 자료 구조들과의 연결 통로 역할
- 파일 디스크립터 = Windows의 파일 핸들(file handle)

파일과 파일디스크립터

파일 디스크립터

- 파일을 시스템 프로그래밍 차원에서 바이트 단위의 입출력으로 다룰 수 있게 하고, 커널 내부의 자료 구조들과의 연결 통로 역할
- 파일 디스크립터 = Windows의 파일 핸들(file handle)
- open(), creat(), dup()의 (음이 아닌 int형) 리턴 값
  - 리턴 값은 read(), write(), fsync(), fdatasync()에서 첫 번째 인자로 사용
- 파일 디스크립터 0은 프로세스의 표준 입력
- 파일 디스크립터 1은 표준 출력
- 파일 디스크립터 2는 표준 에러로 정의
- POSIX.1(Portable Operating System Interface, 첫 번째 POSIX 버전) 표준의 <unistd.h>에서 정의한 바와 같이 파일 디스크립터 0, 1, 2는 STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO 등 심볼릭 상수(symbolic constant) 사용

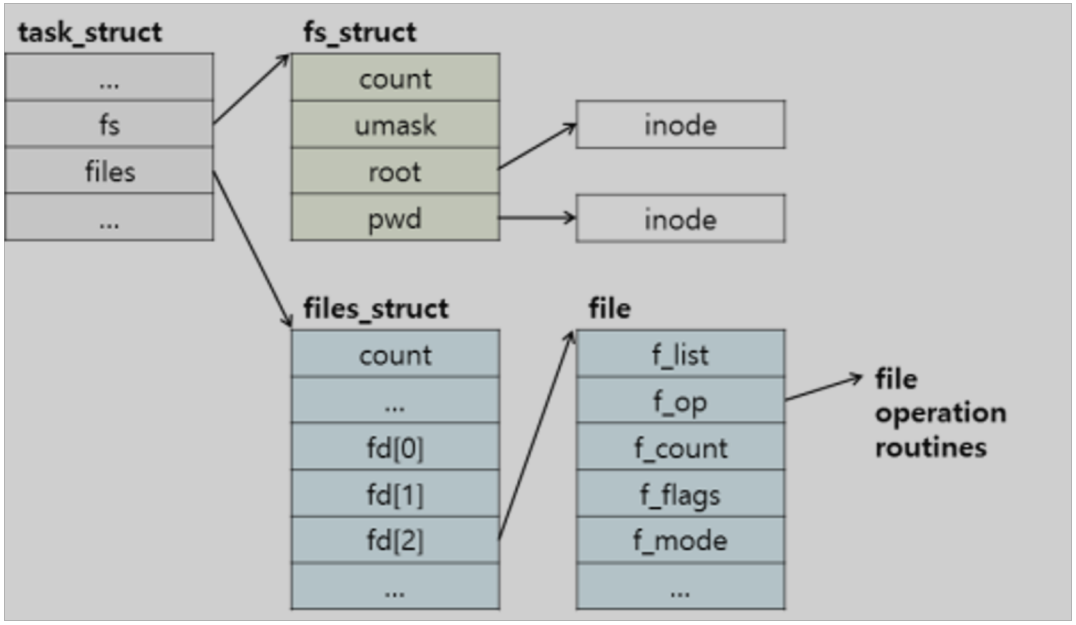
파일 디스크립터를 위한 자료 구조 => 파일 디스크립터 테이블 = 오픈 파일 테이블

- open() 함수를 호출하여 리턴 받는 파일 디스크립터 번호는 프로세스가 오픈한 파일의 목록을 관리하는 파일 디스크립터 테이블의 인덱스
- 0/1/2 파일 디스크립터는 프로세스가 생성되면 시스템에서 자동으로 할당

파일과 파일디스크립터

- **서로 다른 프로세스들이 open() 등을 호출하여 비록 서로 다른 파일을 오픈한다고 해도 동일한 파일 디스크립터 번호를 리턴 받을 수 있음**
  - 파일 디스크립터 테이블은 프로세스마다 하나씩 할당
  - 프로세스마다 하나의 파일 디스크립터 테이블을 가지고 있음 => 로컬 구조

파일과 파일디스크립터



□ 프로세스와 오픈한 파일을 연결하기 위한 커널의 자료구조.

- task_struct는 Linux에서 프로세스(또는 태스크)를 추상화한 구조체
- task_struct 내 멤버 변수 중 files_struct 에 파일 디스크립터들이 포함
- files_struct는 파일 테이블, file 구조체를 가르키는 포인터를 갖는 배열
- files_struct : fd[0]는 표준 입력(stdin), fd[1]은 표준 출력(stdout), fd[2]는 표준 에러(stderr)
- 프로세스가 생성되면 커널이 그 프로세스를 위한 파일 관련 자료구조들을 만들면서 기본적으로 fd[0], fd[1], fd[2]를 생성
- fd[3]부터는 프로세스가 파일을 오픈할 때마다 할당

open()

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int oflag);
int open(const char *pathname, int oflag, mode_t mode);

리턴 값 : 성공 시 파일디스크립터, 에러 시 -1 (errno 설정)
```

- **파일을 오픈하거나 생성할 때 사용하는 시스템호출**
- **파일의 사용을 준비하는 기능을 수행하기 때문에 모든 파일은 사용되기 전 반드시 open()를 호출**
- **Unix의 초기 버전에서 open()는 이미 존재하는 파일을 오픈하는 기능만 가지고 있었기 때문에 새로운 파일을 생성하기 위해 별도의 함수인 creat()를 사용**
  - 나중에 open()의 기능이 확장되어 새로운 파일을 생성할 경우에도 open()를 사용 => 이미 존재하는 파일을 오픈하는 경우나 새로운 파일을 생성할 경우에도 creat()와 같은 함수를 사용할 필요 없이 open()를 사용

open()

- **open() 호출**
  - 커널은 먼저 동일한 파일이 시스템에 존재하고 있는지 확인 => 파일에 대한 올바른 접근 권한을 검사 후 파일 디스크립터를 리턴
- **pathname 인자**
  - 상대경로/절대경로 모두 사용 => 현재 작업 디렉토리와 상관없이 이미 존재하고 있는 파일을 오픈하거나 새로운 파일 생성 가능
- **oflag 인자 <표 2-1> <표 2-2>**
  - open()의 작동 방식. 오픈할 파일의 특성
  - 해당 파일을 어떤 용도로 사용할지 결정
  - <fcntl.h>에 정의된 상수 중 하나/여러 개를 논리합(OR) 연산 사용 결합
- **Mode 인자 <표 2-4>**
  - 오픈할 파일의 접근 모드(권한)
  - 파일의 읽기, 쓰기, 실행 모드를 지정
  - <sys/stat.h>에 정의
  - III 장에서 상세하게 설명

- **주의할 점 : 파일을 읽기만 할 것이라면 읽기 전용으로 오픈**

open()

□ 반드시 사용해야 하는 파일 접근 모드 플래그 (oflag) <표 2-1>

- 상호배타적으로 사용

oflag	내용
O_RDONLY	파일을 읽기 전용으로만 오픈하며, 이 플래그로 오픈한 파일에 쓰기를 시도하면 에러 발생
O_WRONLY	파일을 쓰기 전용으로만 오픈하며, 이 플래그로 오픈한 파일에 읽기를 시도하면 에러 발생
O_RDWR	파일을 읽기와 쓰기가 모두 가능하도록 오픈

open()

□ 선택적으로 파일 플래그 (oflag) <표 2-2>

플래그	내용
O_APPEND	파일에 기록 시 해당 파일의 오프셋 위치를 자동으로 파일의 끝(EOF, End of File)으로 이동시킴. 파일의 원래 데이터를 그대로 유지하면서 새로운 데이터를 그 파일의 끝에 추가하고자 할 때 편리하게 사용하는 플래그임. 이 플래그 없이 O_RDWR 또는 O_WRONLY만으로 파일을 오픈했을 경우 파일에 새로운 데이터를 추가하기 위해서는 파일에 쓰기를 시도하기 직전에 lseek()를 이용해 오프셋 위치를 파일의 끝으로 옮기는 작업 필요. 그러나 일반적으로 이러한 절차는 두 프로세스가 동시에 동일한 파일에 쓰기 작업을 하는 상황에서는 에러를 만들어 낼 수 있음
O_CREAT	파일이 존재하지 않을 경우 같은 이름의 파일을 새로 생성. 이 플래그는 open()의 3번째 인자인 파일의 mode를 함께 지정해야 함. mode는 8진수 형태로 나타내며 파일을 읽기, 쓰기, 실행할 수 있는 사용자의 권한을 지정함. open()는 인자를 가변적으로 사용할 수 있는 함수이기 때문에 이 플래그가 없을 때는 3번째 인자를 제외하고 2개의 인자만 지정해도 됨. O_CRAET 플래그는 O_EXCL 또는 O_TRUNC 플래그와 함께 사용될 수 있음
O_EXCL	O_CREAT와 함께 지정되며 단독으로는 쓰이지 않음. 만일 파일이 이미 존재한다면 open()는 실패하고 에러를 리턴함. 이 플래그 없이 동일한 동작을 위해서는 open()를 이용해서 해당 파일이 존재하는지를 확인한 후 존재하지 않는다면 creat()를 호출해야 함. 그러나 이 때 두 작업 사이에 다른 프로세스에 의한 끼어들기가 발생할 수 있기 때문에 예상치 못한 에러가 발생할 수 있음. O_EXCL 플래그를 쓰면 이 두 가지 작업을 원자적으로 실행하여 하나의 작업으로 만들 수 있음
O_TRUNC	해당 파일이 이미 존재할 경우 그 파일은 길이가 0이 되면서 이전 데이터를 전부 잃게 됨. 단, 이것은 파일의 접근 권한이 O_RDWR, 또는 O_WRONLY로 지정된 경우에 한함
O_NOCTTY	파일이 터미널 장치일 경우, 해당 장치를 이 프로세스의 제어 터미널로 배정하지 않음
O_NONBLOCK	해당 파일이 블록 특수 파일이거나 문자 특수 파일 또는 FIFO(Named Pipe) 등 입출력을 시도한 시점에서 블록 상태가 될 수 있는 장치라면 이후의 입출력 작업들에 대해 논블록 모드로 동작함. 블록 상태가 될 상황에 입출력 함수들이 에러 코드를 만들고 리턴함

open()

□ 생성할 파일의 접근 모드(권한)을 나타내는 mode 인자 <표 2-4>

플래그	내용
S_IRWXU	파일 소유자의 읽기, 쓰기, 실행 권한을 지정. 00700을 사용해도 됨
S_IRUSR	파일 소유자의 읽기 권한을 지정. 00400을 사용해도 됨
S_IWUSR	파일 소유자의 쓰기 권한을 지정. 00200을 사용해도 됨
S_IXUSR	파일 소유자의 실행 권한을 지정. 00100을 사용해도 됨
S_IRWXG	그룹 사용자의 읽기, 쓰기, 실행 권한을 지정. 00070을 사용해도 됨
S_IRGRP	그룹 사용자의 읽기 권한을 지정. 00040을 사용해도 됨
S_IWGRP	그룹 사용자의 쓰기 권한을 지정. 00020을 사용해도 됨
S_IXGRP	그룹 사용자의 실행 권한을 지정. 00010을 사용해도 됨
S_IRWXO	다른 사용자의 읽기, 쓰기, 실행 권한을 지정. 00007을 사용해도 됨
S_IROTH	다른 사용자의 읽기 권한을 지정. 00004을 사용해도 됨
S_IWOTH	다른 사용자의 쓰기 권한을 지정. 00002을 사용해도 됨
S_IXOTH	다른 사용자의 실행 권한을 지정. 00001을 사용해도 됨

(참고) sys/stat.h

- stat()로 리턴되는 구조체

```
dev_t     st_dev     ID of device containing file
ino_t     st_ino     file serial number
mode_t    st_mode    mode of file (see below)
nlink_t   st_nlink   number of links to the file
uid_t     st_uid     user ID of file
gid_t     st_gid     group ID of file
dev_t     st_rdev    device ID (if file is character or block special)
off_t     st_size    file size in bytes (if file is a regular file)
time_t    st_atime   time of last access
time_t    st_mtime   time of last data modification
time_t    st_ctime   time of last status change
```

- st_mode 파일의 접근 권한 => 파일 모드 타입 (표 2-4, p. 65)
- https://pubs.opengroup.org/onlinepubs/007908775/xsh/sysstat.h.html
- https://pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/types.h.html (sys/types.h : _t 데이타 타입)
- /usr/include/linux/sys/stat.h
- /usr/include/x86_64-linux-gnu/sys
- % sudo ln -s /usr/include/x86_64-linux-gnu/sys/types.h /usr/include/sys/types.h
- % sudo ln -s /usr/include/x86_64-linux-gnu/sys/stat.h /usr/include/sys/stat.h

(참고) fcntl.h

- 파일 콘트롤 인자/cmd/flag 정의 - fcntl()/open()
- https://pubs.opengroup.org/onlinepubs/009695399/basedefs/fcntl.h.html
- open()에서 파일을 새로 생성시 사용되는 플래그 <표 2-2>
  - O_CREAT, O_EXCE, O_NOCTTY, O_TRUNC
- open()과 fcntl()에서 모두 사용되는 파일 상태 플래그 <표 2-2>
  - O_APPNED, O_DSYNC, O_NONBLCOK, O_RSYNC …
- open()과 fcntl()에서 모두 사용되는 파일 접근 모드 플래그 <표 2-2>
  - O_RDONLY, O_RDWR, O_WRONLY (open()에서는 필수 플래그)
- fcntl()에서만 사용되는 cmd
  - F_DUPFD, F_GETFD, F_SETFD, F_GETFL, F_SETFL, F_GETLK, F_SETLK, F_SETLKW… (fcntl()에서 반드시 사용)
- fcntl()에서만 사용되는 파일 디스크립터 플래그
  - FD_CLOSEXEC
- fcntl()에서만 레코드 단위 락을 걸 때 사용되는 l_type 값
  - F_RDLCK, F_UNLCK, F_WRLCK

(참고) sys/types.h

- blksize_t, ssize_t [-1~SSIZEMAX], pid_t // signed integer
- clock_t // long integer(GNU), floating-point
- dev_t // 32 bits integer (12-bits - Major, 20 bits - Minor)
- mode_t //32 bits integer 중 일부분만 접근 권한으로 사용
- nlink_t // unsigned short or unsigned long
- off_t // signed integer (off64_t : 2^63 bytes)
- pthread_attr_t, pthread_cond_t // sturture pointer
- size_t //unsigned integer, 0 또는 0 보다 큰 integer 만 가능
- time_t // signed integer(UNIX, POSIX),1970, 1, 1월 1일 0시 0분 0초을 0 기준 시간(Epoch time) => 2038년 문제.
- useconds_t // unsigned integer[0~1,000,000]

(참고) sys/types.h의 mode_t의 이해

```
#include <sys/stat.h>
#include <stdbool.h>
#include <stdio.h>
enum class { CLASS_OWNER, CLASS_GROUP, CLASS_OTHER };
enum permission { PERMISSION_READ, PERMISSION_WRITE, PERMISSION_EXECUTE };
const mode_t EMPTY_MODE = 0;
mode_t perm(enum class c, enum permission p) { return 1 << ((3-p) + (2-c)*3); }
bool mode_contains(mode_t mode, enum class c, enum permission p) { return mode & perm(c, p); }
mode_t mode_add(mode_t mode, enum class c, enum permission p) { return mode | perm(c, p); }
mode_t mode_rm(mode_t mode, enum class c, enum permission p) { return mode & ~perm(c, p); }

// buf must have at least 10 bytes
void strmode(mode_t mode, char * buf) {
            const char chars[] = "rwxrwxrwx";
            for (size_t i = 0; i < 9; i++)
            {
                        buf[i] = (mode & (1 << (8-i))) ? chars[i] : '-';
            }
            buf[9] = '\0';
}

int main(void) {
            char buf[10];
            mode_t examples[] = { 0, 0666, 0777, 0700, 0100, 01, 02, 03, 04, 05, 06, 07 };
            size_t num_examples = sizeof(examples) / sizeof(examples[0]);
            for (size_t i = 0; i < num_examples; i++)
            {
                        strmode(examples[i], buf);
                        printf("%04o is %s\n", examples[i], buf);
            }
return 0;
}
```

close()

```
#include <unistd.h>
int close(int filedes);

리턴 값 : 성공 시 0, 에러 시 -1 (errno 설정)
```

- 오픈한 파일을 닫을 때 사용하는 시스템호출
- 사용이 끝난 파일은 close()를 사용하여 바로 close => 사용 중인 파일 디스크립터가 너무 많으면 실수로 의도하지 않은 파일에 데이터를 덮어 쓰는 것 방지 및 시스템 자원 낭비 방지
- 한 프로세스가 오픈할 수 있는 파일의 개수 및 시스템 내 오픈할 수 있는 파일 수는 제한적 (/proc/sys/fs/file-max 에서 정의)
- % ulimit -a
- % cat /proc/sys/fs/file-nr

(참고) unistd.h

- **access()**
  - R_OK, W_OK, X_OK, F_OK 등의 심볼릭 상수 사용 시
- **lseek(), fcntl()**
  - SEEK_SET, SEEK_CUR, SEEK_END 등 파일의 위치(오프셋)을 지정하는 심볼릭 상수 사용 시
- **lockf()**
  - F_LOCK, F_UNLOCK, F_TEST, F_TLOCK 등 파일의 락을 지정하는 심볼릭 상수 사용 시
- **표준입출력 파일 디스크립터**
  - STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO 등 표준입출력 fd 0, 1, 2 를 위한 심볼릭 상수 사용시
- **다음 함수들 사용 시**
    - access(), brk(), chdir(), chown(), close(), dup(), dup2(), execl(), execle(), execlp(), execv(), execve(), execvp(), _exit(), fchown(), fchdir(), fork(), fpathconf(), fsync(), getcwd(), getegid(), geteuid(), getgid(), getpid(), getpgid(), getsid(), getuid(), lchown(), link(), lseek(), nice(), pathconf(), pipe(), pread(), pwrite(), symlink(), sync(), sysconf(), unlink(), read(), write()….

open()예제 (p.68)

<ssu_test.txt>

```
Linux System Programming!
Unix System Programming!
Linux Mania
Unix Mania
```

<ssu_open.c>

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

int main(void)
{
    char *fname = "ssu_test.txt";
    int fd;

    if ((fd = open(fname, O_RDONLY)) < 0) {
        fprintf(stderr, "open error for %s\n", fname);
        exit(1);
    }
    else
        printf("Success!\nFilename : %s\nDescriptor : %d\n", fname, fd);

    exit(0);
}
```

>>>실행결과

```
root@localhost:/home/oslab# ./ssu_open
Success!
Filename : ssu_test.txt
Descriptor : 3
```