Linux Programming
6장. 파일 시스템 (1/2)

sisong@ut.ac.kr
한국교통대학교 컴퓨터공학전공
송석일

1

6.1 파일 시스템 구현

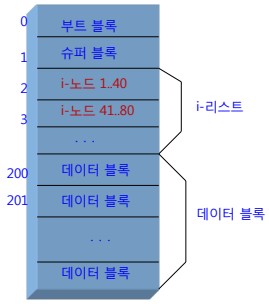
2

리눅스 파일 시스템 구조

- 유닉스 파일 시스템의 전체적인 구조
  - 부트 블록, 슈퍼 블록, i-리스트, 데이터 블록으로 구성
- ext 파일 시스템
  - 리눅스는 유닉스 파일 시스템을 확장한 ext 파일 시스템 사용
- 현재 리눅스에서 사용되는 ext4 파일 시스템
  - 1EB(엑사바이트, 1EB=1024 × 1024TB) 이상 볼륨 크기
  - 16TB 이상 파일 크기

3

파일 시스템 구조



- 부트 블록(Boot block)
  - 파일 시스템 시작부에 위치하고 보통 첫 번째 섹터를 차지
  - 부트스트랩 코드가 저장되는 블록
- 슈퍼 블록(Super block)
  - 전체 파일 시스템에 대한 정보를 저장
  - 총 블록 수, 사용 가능한 i-노드 개수, 사용 가능한 블록 비트 맵, 블록의 크기, 사용 중인 블록 수, 사용 가능한 블록 수 등
- i-리스트(i-list)
  - 각 파일을 나타내는 모든 i-노드들의 리스트
  - 한 블록은 약 40개 정도의 i-노드를 포함
- 데이터 블록(Data block)
  - 파일의 내용(데이터)을 저장하기 위한 블록들

- 실제 ext4 파일 시스템을 단순화한 구조

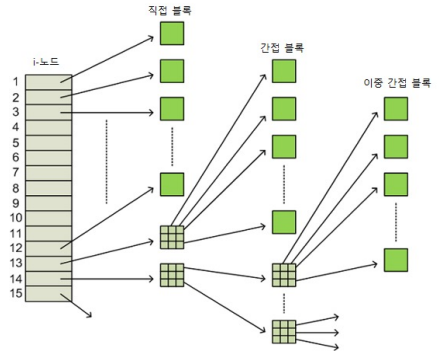
4

i-노드(i-node)

- 하나의 파일은 하나의 i-노드
- 파일에 대한 모든 정보 (메타 데이터)
  - 파일 타입: 일반 파일, 디렉터리, 블록 장치, 문자 장치 등
  - 파일 크기
  - 접근권한
  - 파일 소유자 및 그룹
  - 접근 및 갱신 시간
  - 데이터 블록에 대한 포인터(주소) 등

5

i-노드와 블록 포인터



6

i-노드와 블록 포인터

- 데이터 블록에 대한 포인터
  - 파일의 내용을 저장하기 위해 할당된 데이터 블록의 주소
- 하나의 i-노드 내의 블록 포인터
  - 직접 블록 포인터 12개
  - 간접 블록 포인터 1개
    - 4096byte 블록, 4 byte 포인터 => 1024 블록 포인터 가능
  - 이중 간접 블록 포인터 1개
  - 삼중 간접 블록 포인터 1개
- 최대 몇 개 블록 가능 ? (4096byte 블록, 4 byte 포인터)

7

6.2 파일 입출력 구현

8

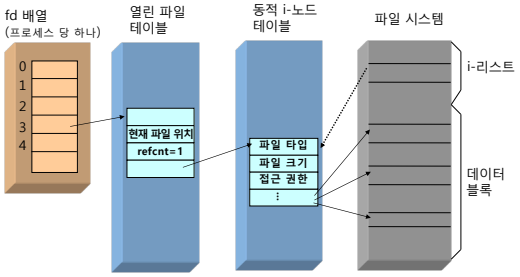
파일 열기 및 파일 입출력 구현

- 파일 열기 open()
  - 커널 내에 파일을 사용할 준비
- 파일 입출력 구현을 위한 커널 내 자료구조
  - 파일 디스크립터 배열(Fd array)
  - 열린 파일 테이블(Open File Table)
  - 동적 i-노드 테이블(Active i-node table)

9

파일을 위한 커널 자료 구조

- fd = open("file", O_RDONLY);



10

파일 디스크립터 배열(Fd Array)

- 프로세스 당 하나
- 파일 디스크립터 배열
  - 열린 파일 테이블의 엔트리를 가리킴
- 파일 디스크립터
  - 파일 디스크립터 배열의 인덱스
  - 열린 파일을 나타내는 번호 → FILE 구조체에 포함됨

11

열린 파일 테이블(Open File Table)

- 파일 테이블 (file table)
  - 커널 자료구조
  - 열려진 모든 파일 목록
  - 열려진 파일 → 파일 테이블의 항목
- 파일 테이블 항목 (file table entry)
  - 파일 상태 플래그 : read, write, append, sync, nonblocking,...
  - 파일의 현재 위치(current file offset)
  - i-node에 대한 포인터

12

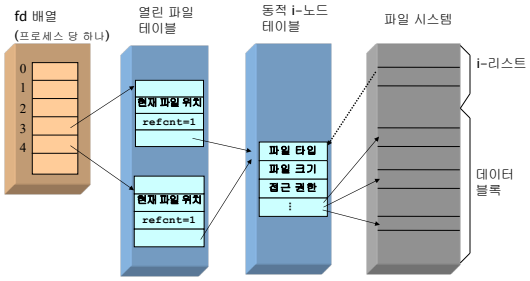
동적 i-노드 테이블(Active i-node table)

- 동적 i-노드 테이블
  - 커널 내의 자료 구조
  - Open 된 파일들의 i-node를 저장하는 테이블
- i-노드
  - 하드 디스크에 저장되어 있는 파일에 대한 자료구조
  - 한 파일에 하나의 i-node
  - 하나의 파일에 대한 정보 저장
    - 소유자, 크기
    - 파일이 위치한 장치
    - 파일 내용 디스크 블럭에 대한 포인터
- i-node table vs. i-node

13

한 파일을 두 번 열기 구현

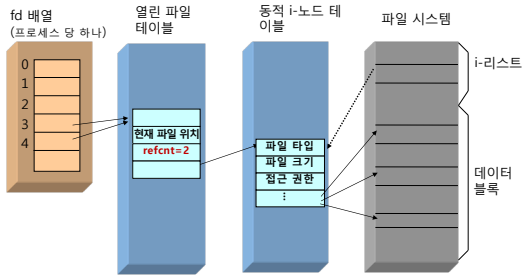
- fd1 = open("file", O_RDONLY);
- fd2 = open("file", O_RDONLY);



14

dup() 시스템 호출 구현

- fd = dup(fd1);



15

6.3 파일 상태 정보

16

파일 상태(file status)

- 파일 상태
  - 파일에 대한 모든 정보
  - 블록수, 파일 타입, 사용 권한, 링크수, 파일 소유자의 사용자 ID,
  - 그룹 ID, 파일 크기, 최종 수정 시간 등

예

```
$ ls -l hello.c
 4          -rw-r--r--   1          lect       lect       617          4월 17 15:53       hello.c
블록수      접근권한     링크수     사용자ID   그룹ID     파일 크기    최종 수정 시간     파일이름
```

파일 타입

17

상태 정보: stat()

- fstat(), lstat(), stat() 시스템 호출

```
#include <sys/types.h>
#include <sys/stat.h>
int stat (const char *filename, struct stat *buf);
int fstat (int fd, struct stat *buf);
int lstat (const char *filename, struct stat *buf);
```

- 파일의 상태 정보를 가져와서 stat 구조체 buf에 저장
- 성공하면 0, 실패하면 -1 반환

- 파일 하나당 하나의 i-노드
- i-노드 내에 파일에 대한 모든 상태 정보 저장

18

stat 구조체

```
struct stat {
  mode_t st_mode;   // 파일 타입과 접근권한
  ino_t st_ino;     // i-노드 번호
  dev_t st_dev;     // 장치 번호
  dev_t st_rdev;    // 특수 파일 장치 번호
  nlink_t st_nlink; // 링크 수
  uid_t st_uid;     // 소유자의 사용자 ID
  gid_t st_gid;     // 소유자의 그룹 ID
  off_t st_size;    // 파일 크기
  time_t st_atime;  // 최종 접근 시간
  time_t st_mtime;  // 최종 수정 시간
  time_t st_ctime;  // 최종 상태 변경 시간
  long st_blksize;  // 최적 블록 크기
  long st_blocks;   // 파일의 블록수
};
```

19

파일 타입

파일 타입	설명
일반 파일	데이터를 저장하고 있는 텍스트 파일 또는 이진 파일
디렉터리 파일	파일의 이름들과 파일 정보에 대한 포인터를 포함하는 파일
문자 장치 파일	문자 단위로 데이터를 전송하는 장치를 나타내는 파일
블록 장치 파일	블록 단위로 데이터를 전송하는 장치를 나타내는 파일
FIFO 파일	프로세스 간 통신에 사용되는 파일로 이름 있는 파이프
소켓	네트워크를 통한 프로세스 간 통신에 사용되는 파일
심볼릭 링크	다른 파일을 가리키는 포인터 역할을 하는 파일

20

파일 타입 검사 함수

- 파일 타입을 검사하기 위한 매크로 함수

파일 타입	매크로 함수
일반 파일	S_ISREG()
디렉터리 파일	S_ISDIR()
문자 장치 파일	S_ISCHR()
블록 장치 파일	S_ISBLK()
FIFO 파일	S_ISFIFO()
소켓	S_ISSOCK()
심볼릭 링크	S_ISLNK()

21

파일 타입 출력: ftype.c

```
1.  #include <sys/types.h>
2.  #include <sys/stat.h>
3.  #include <stdio.h>
4.  #include <stdlib.h>
5.  #include <unistd.h>
6.  /* 파일 타입을 검사한다. */
7.  int main(int argc, char *argv[])
8.  {
9.    int i;
10.   struct stat buf;
11.   for (i = 1; i < argc; i++) {
12.     printf("%s: ", argv[i]);
13.     if (lstat(argv[i], &buf) < 0) {
14.       perror("lstat()");
15.       continue;
16.     }
```

```
1.      if (S_ISREG(buf.st_mode))
2.        printf("%s \n", "일반 파일");
3.      if (S_ISDIR(buf.st_mode))
4.        printf("%s \n", "디렉터리");
5.      if (S_ISCHR(buf.st_mode))
6.        printf("%s \n", "문자 장치 파일");
7.      if (S_ISBLK(buf.st_mode))
8.        printf("%s \n", "블록 장치 파일");
9.      if (S_ISFIFO(buf.st_mode))
10.       printf("%s \n", "FIFO 파일");
11.     if (S_ISLNK(buf.st_mode))
12.       printf("%s \n", "심볼릭 링크");
13.     if (S_ISSOCK(buf.st_mode))
14.       printf("%s \n", "소켓");
15.   }
16.   exit(0);
17. }
```

22

파일 접근권한(File Permissions)

- 각 파일에 대한 권한 관리
  - 각 파일마다 접근권한
  - 소유자(owner)/그룹(group)/기타(others)로 구분해서 관리
- 파일에 대한 권한
  - 읽기 r
  - 쓰기 w
  - 실행 x
- 파일의 접근권한 가져오기
  - stat() 시스템 호출
- 파일의 접근권한 변경하기
  - chmod() 시스템 호출

23

chmod(), fchmod()

- chmod(), fchmod() 시스템 호출

```
#include <sys/stat.h>
#include <sys/types.h>
int  chmod (const char *path, mode_t mode );
int  fchmod (int fd, mode_t mode );
```

  - 파일의 접근권한(access permission)을 변경
  - 리턴 값
    - 성공하면 0, 실패하면 -1
  - mode
    - 8진수 접근권한
    - 예: 0644

24

접근권한 변경: fchmod.c

```
1.  #include <sys/types.h>
2.  #include <sys/stat.h>
3.  #include <stdio.h>
4.  #include <stdlib.h>
5.  /* 파일 접근권한을 변경한다. */
6.  main(int argc, char *argv[])
7.  {
8.    long strtol( );
9.    int newmode;
10.   newmode = (int) strtol(argv[1], (char **) NULL, 8);
11.   if (chmod(argv[2], newmode) == -1) {
12.     perror(argv[2]);
13.     exit(1);
14.   }
15.   exit(0);
16. }
```

```
$ fchmod 664 you.txt
$ ls -asl you.txt
4 -rw-rw-r-- 1 lect lect 518 4월  8  19:06 you.txt
```

25

접근 및 수정 시간 변경: utime()

- utime() 시스템 호출

```
#include <sys/types.h>
#include <utime.h>
int utime (const char *filename, const struct utimbuf *times );
```

- 파일의 최종 접근 시간과 최종 변경 시간을 조정
- times가 NULL 이면, 현재시간으로 설정
- 리턴 값
  - 성공하면 0, 실패하면 -1

```
struct utimbuf {
    time_t  actime;     /* access time */
    time_t  modtime;    /* modification time */
}
```

- 각 필드는 1970-1-1 00:00 부터 현재까지의 경과 시간을 초로 환산

26

접근 및 수정 시간 변경: touch.c

```
1.  #include <utime.h>
2.  #include <stdio.h>
3.  #include <stdlib.h>

4.  int main(int argc, char *argv[])
5.  {
6.    if (argc < 2) {
7.      fprintf(stderr, "사용법:  touch file1 \n");
8.      exit(-1);
9.    }
10.   utime(argv[1], NULL);
11. }
```

27

접근 및 수정 시간 복사: cptime.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <utime.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  struct stat buf;      // 파일 상태 저장을 위한 변수
  struct utimbuf time;

  if (argc < 3) {
    fprintf(stderr, "사용법: cptime file1 file2\n");
    exit(1);
  }

  if (stat(argv[1], &buf) <0) { // 상태 가져오기
    perror("stat()");
    exit(-1);
  }

  time.actime = buf.st_atime;
  time.modtime = buf.st_mtime;

  if (utime(argv[2], &time))  // 접근, 수정 시간 복사
    perror("utime");
  else exit(0);
}
```

```
$ ls –asl a.c b.c
4 -rw-rw-r--. 1 lect lect 0 3월 18 12:13 a.c
4 -rw-rw-r--. 1 lect lect 5 3월 18 13:30 b.c

$ cptime a.c b.c
$ ls –asl a.c b.c
4 -rw-rw-r--. 1 lect lect 0 3월 18 12:13 a.c
4 -rw-rw-r--. 1 lect lect 5 3월 18 12:13 b.c
```

28

파일 소유자 변경 chown()

- chown() 시스템 호출

```
#include <sys/types.h>
#include <unistd.h>
int   chown (const char *path, uid_t owner, gid_t group );
int   fchown (int filedes, uid_t owner, gid_t group );
```

- 파일의 user ID와 group ID를 변경
- lchown()은 심볼릭 링크 자체를 변경
- super-user만 변환 가능

- 리턴
  - 성공하면 0, 실패하면 -1

29