File and Directories

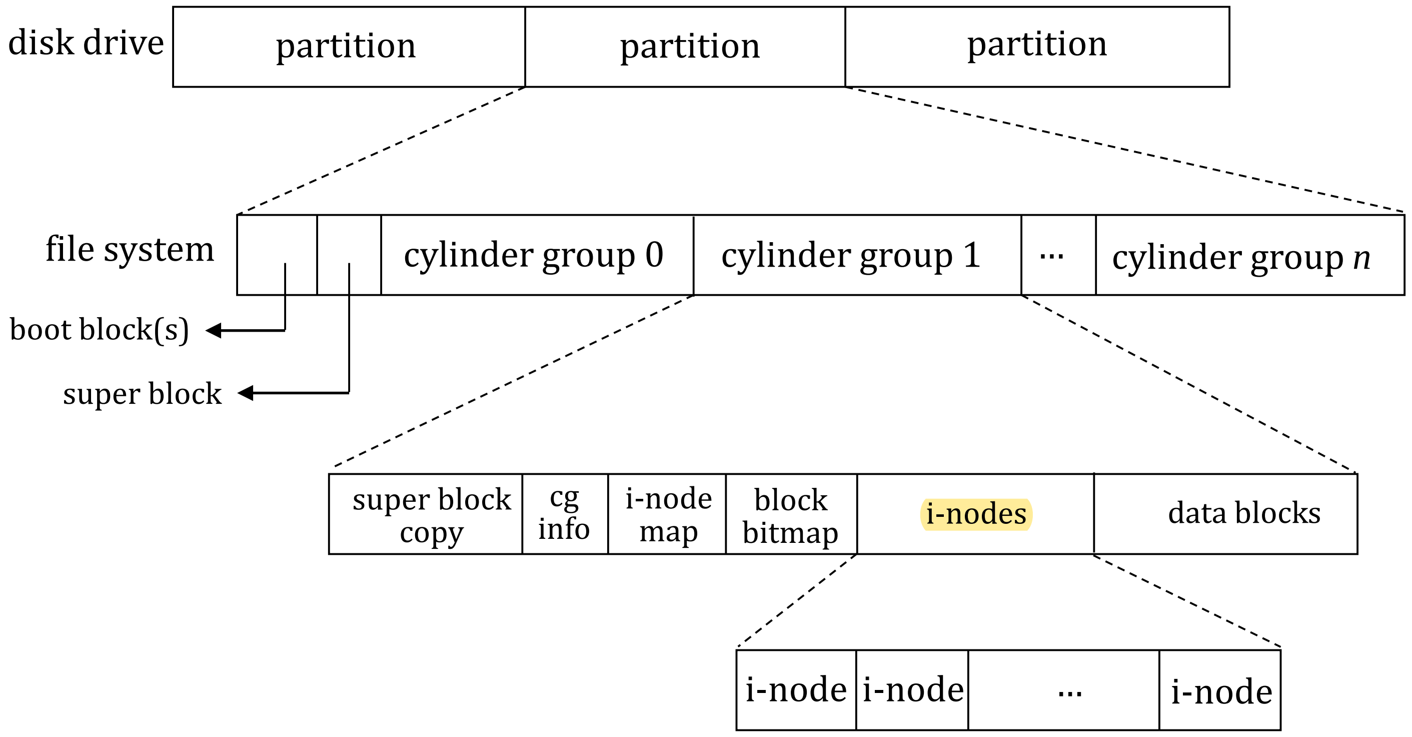
System Programming

2019 여름 계절학기

한양대학교 공과대학 컴퓨터소프트웨어학부
홍석준

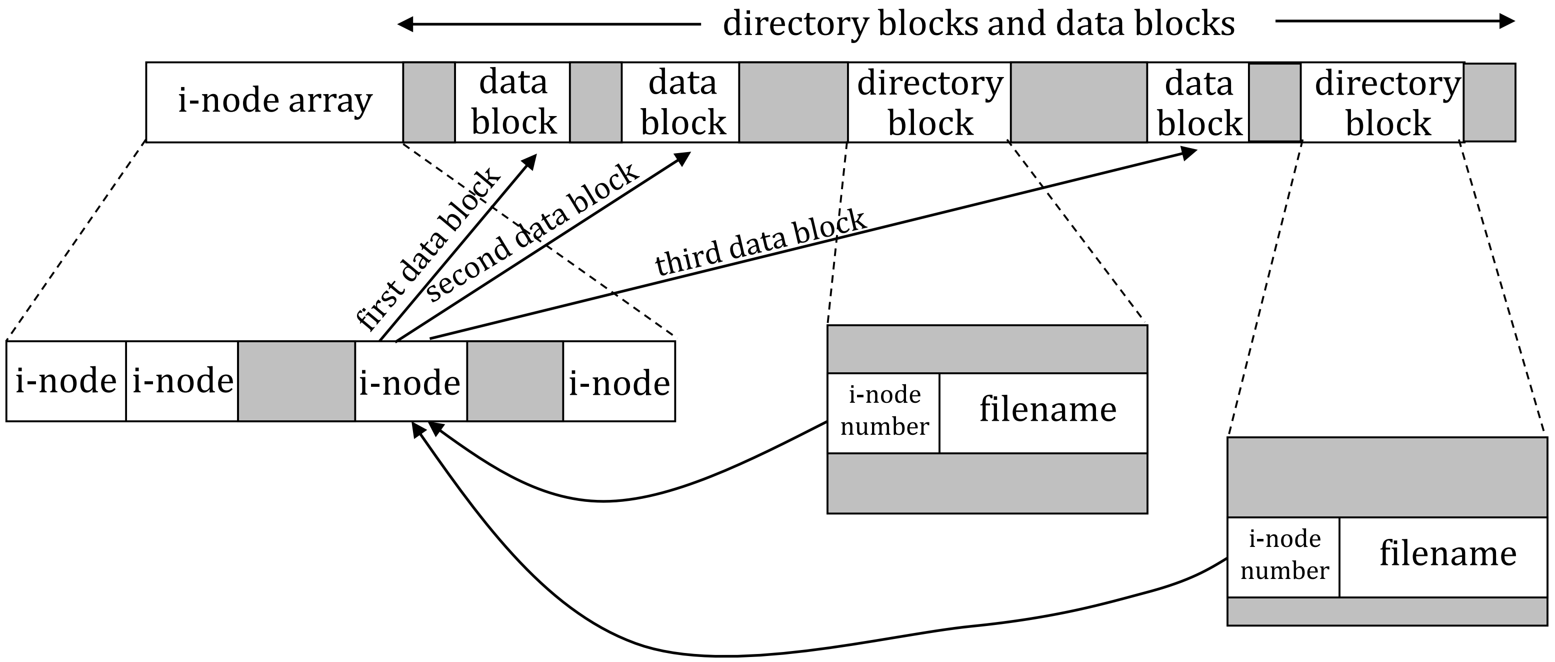
❑ Various implementations of the UNIX file system

❑ UFS



I. File Systems

- ❑ i-node contains info about the file, including file type, access permission, ref-count, size, ptrs to data blocks, and so on.
- ❑ Only two items (filename and i-node no.) are stored in the dir entry.

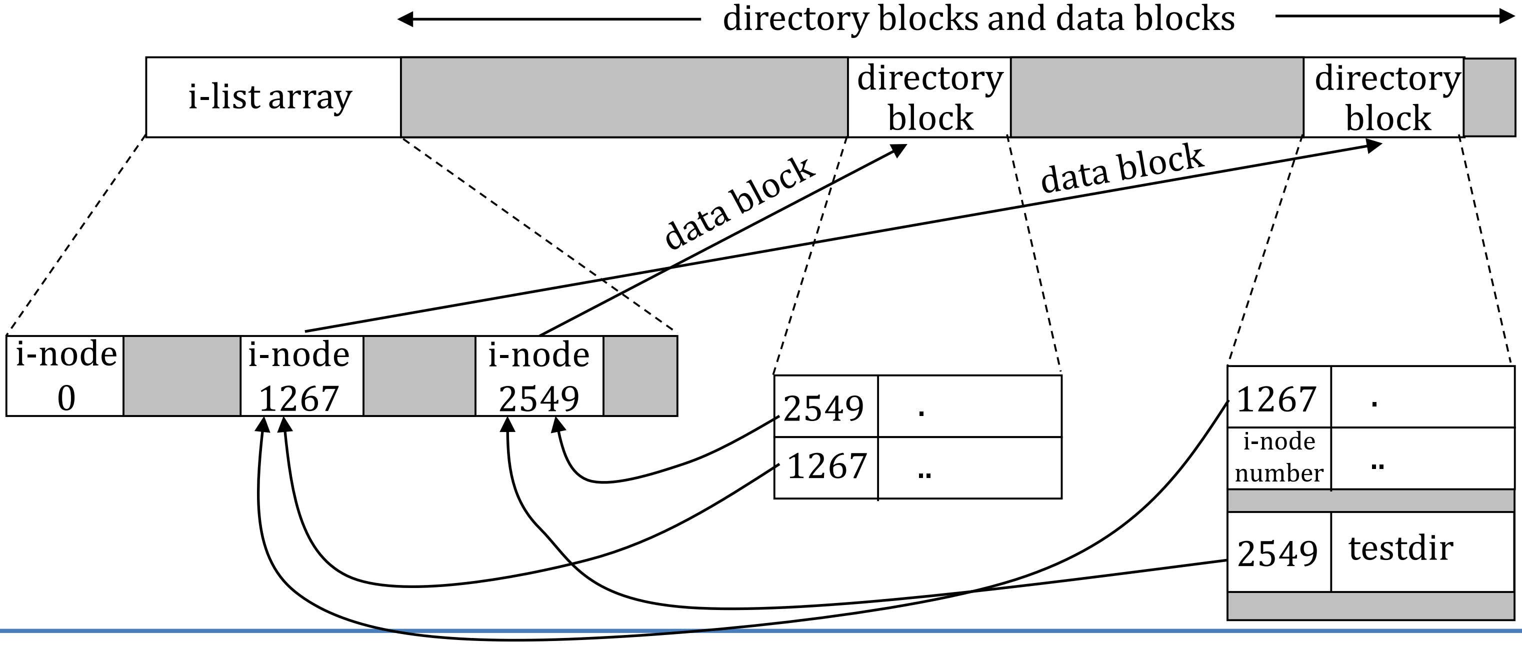


I. File Systems

- **A link count in an i-node = the number of directory entries that point to the i-node**
- **`st_nlink` in the `stat` structure**
- **Hard links vs. soft links**
  - Symbolic links (soft links)
  - The actual content of the file (the data blocks) contains the filename that the symbolic link points to.

```
lrwxrwxrwx 1 root  7 Sep 25 07:14 lib->usr/lib
```

- **No directory entry pointing to an i-node in a different file system.**



I. `link`, `unlink`, `remove`, and `rename` Functions

```
#include <unistd.h>
int link(const char *existingpath, const char *newpath);
```

- Creates a new dir entry that references the existing path (, which increments the link count.)
- Both pathnames must be on the same file system (although POSIX.1 supports linking across file systems.)
- Only a superuser can create a link to a directory.

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

- Removes the dir entry and decrements the link count (the file is deleted, when it reaches 0).
- If a symbolic link, `unlink` references the symbolic link itself.

I. link, unlink, remove, and rename Functions

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

- For a file, identical to `unlink` and, for a directory, to `rmdir`

```
#include <stdio.h>
int rename(const char *oldname, const char
    *newname);
```

I. link, unlink, remove, and rename Functions

❑ Program 4.16

```
$ ls -l tempfile
-rw-r-----   1 sar     413265408   Jan 21  07:14  tempfile
$ df /home
Filesystem  1K-blocks     Used  Available  Use% Mounted on
/dev/hda4    11021440  1956332    9056108   18% /home
$ ./a.out &
1364
$ file unlinked
ls -l tempfile
ls: tempfile: No such file or directory
$ df /home
Filesystem  1K-blocks     Used  Available  Use% Mounted on
/dev/hda4    11021440  1956332    9056108   18% /home
$ done
df /home
Filesystem  1K-blocks     Used  Available  Use% Mounted on
/dev/hda4    11021440  1552352    9469088   15% /home
```

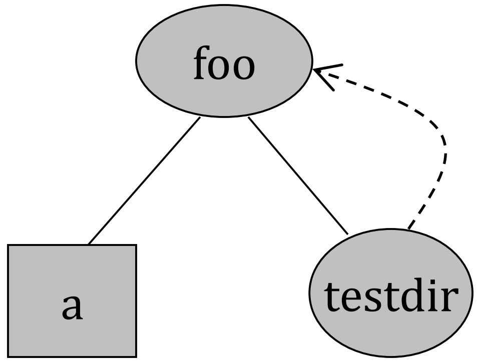
Program 4.16

```
#include "apue.h“
#include <fcntl.h>
int main(void)
{
  if (open("tempfile", O_RDWR) < 0)
    err_sys("open error");
  if (unlink("tempfile") < 0)
    err_sys("unlink error");

  printf("file unlinked\n");
  sleep(15);
  printf("done\n");

  exit(0);
}
```

I. Symbolic Links

- To get around the limitation of hard links
  - Linking across file systems
  - A hard link to a directory (only by superuser)

```
$ mkdir foo
$ touch foo/a
$ ln -s ../foo foo/testdir
$ ls -l foo
total 0
-rw-rw-r--  1 sar   0 Dec 6 06:06 a
lrwxrwxrwx  1 sar   6 Dec 6 06:06 testdir->../foo
```



Ⅰ. symlink and readlink Functions

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

`int symlink(const char *`***actualpath***`, const char *`***sympath***`);`

- ❑ A new dir entry, *sympath*, is created that points to *actual path*.

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

`ssize_t readlink(const char *`***pathname***`, char *`***buf***`, size_t` ***bufsize***`);`

- ❑ `open` follows a symbolic link, while `readlink` opens the link itself and reads the name in the link.
- ❑ Equivalent to the actions of `open`, `read`, `and` `close`.

I. File Times

Field	Description	Example	`ls`(1) option
`st_atime`	Last access time of file data	`read`	-u
`st_mtime`	Last modification time of file data	`write`	default
`st_ctime`	Last change time of i-node status	`chmod, chown`	-c

- **The modification time is when the file contents were last modified.**
- **The changed-status time indicates when the i-node was last modified, e.g., changing the file access permission, the user ID, the number of links, etc.**
- **The three times for a file/directory and its parent directory**
  - For example, creating a new file affects the containing dir, and it affects the i-node for the new file. (Figure 4.20)

I. `utime` Function

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

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

- ❑ The `utime` changes the access/modification time of a file.
- ❑ If *`times`* is `NULL`, set to current time.
  - – Effective UID must equal the real ID of the file, or write permission for the file.
- ❑ Otherwise, set to values pointed by *`times`*.
  - – Effective UID must equal the real ID of the file, or superuser privilege
- ❑ Program 4.21

Program 4.21

```
#include "apue.h“
#include <fcntl.h>
#include <utime.h>
int main(int argc, char *argv[])
{
  int   i, fd;
  struct stat  statbuf;
  struct utimbuf  timebuf;

  for (i = 1; i < argc; i++) {
    if (stat(argv[i], &statbuf) < 0) {  /* fetch current times */
      err_ret("%s: stat error", argv[i]);
      continue;
    }
    if ((fd = open(argv[i], O_RDWR | O_TRUNC)) < 0) { /* truncate */
      err_ret("%s: open error", argv[i]);
      continue;
    }
    close(fd);
    timebuf.actime  = statbuf.st_atime;
    timebuf.modtime = statbuf.st_mtime;
    if (utime(argv[i], &timebuf) < 0) {/* reset times */
      err_ret("%s: utime error", argv[i]);
      continue;
    }
  }
  exit(0);
}
```

Ⅰ. mkdir and rmdir Functions

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

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

- The `mode` is modified by the `umask` of the process.
- The user ID and group ID of the new directory.

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

`int rmdir(const char` ***pathname*****);**

- If the link count of the dir becomes 0, and no other process has the dir open, then the space occupied by the dir is freed.

I. Reading Directories

```
#include <dirent.h>

DIR *opendir(const char *pathname);
struct dirent *readdir(DIR *dp);
void rewinddir(DIR *dp);
int closedir(DIR *dp);
long telldir(DIR *dp);
void seekdir(DIR *dp, long loc);

struct dirent {
    ino_t   d_ino;                        /* i-node number */
    char    d_name[NAME_MAX + 1]; /* null-terminated fname */
}
```

- ❑ Only the kernel can write to a directory.
- ❑ Write and execute permission to create/delete files
- ❑ Program 4.22

Program 4.22

```
#include "apue.h“
#include <dirent.h>
#include <limits.h>/* function type that is called for each filename */
typedef         int                 Myfunc(const char *, const struct stat *, int);
static Myfunc                       myfunc;
 static int                         myftw(char *, Myfunc *);
static int                          dopath(Myfunc *);
static long     nreg, ndir, nblk, nchr, nfifo, nslink, nsock, ntot;

int main(int argc, char *argv[])
{
   int  ret;
   if (argc != 2)
      err_quit("usage:  ftw  <starting-pathname>");
   ret = myftw(argv[1], myfunc);                              /* does it all */
   ntot = nreg + ndir + nblk + nchr + nfifo + nslink + nsock;
   if (ntot == 0)
      ntot = 1; /* avoid divide by 0; print 0 for all counts */
   printf("regular files  = %7ld, %5.2f %%\n", nreg, nreg*100.0/ntot);
   printf("directories    = %7ld, %5.2f %%\n", ndir, ndir*100.0/ntot);
   printf("block special  = %7ld, %5.2f %%\n", nblk, nblk*100.0/ntot);
   printf("char special   = %7ld, %5.2f %%\n", nchr, nchr*100.0/ntot);
   printf("FIFOs          = %7ld, %5.2f %%\n", nfifo, nfifo*100.0/ntot);
   printf("symbolic links = %7ld, %5.2f %%\n", nslink, nslink*100.0/ntot);
   printf("sockets        = %7ld, %5.2f %%\n", nsock, nsock*100.0/ntot);
   exit(ret);
}
```

Program 4.22

```
/*
 * Descend through the hierarchy, starting at "pathname".
 * The caller's func() is called for every file. */
#define       FTW_F       1                               /* file other than directory */
#define       FTW_D       2                               /* directory */
#define       FTW_DNR     3                  /* directory that can't be read */
#define       FTW_NS      4                               /* file that we can't stat */
static char   *fullpath;            /* contains full pathname for every file */
static size_t  pathlen;
static int                          /* we return whatever func() returns */
myftw(char *pathname, Myfunc *func)
{
    fullpath = path_alloc(& pathlen); /* malloc's for PATH_MAX+1 bytes */
                                     /* ({Prog pathalloc}) */
   if(pathlen <= strlen(pathname))
  {
     pathlen = strlen(pathname) * 2;
     if((fullpath = realloc(fullpath, pathlen)) == NULL )
                 err_sys("realloc failed");
  }
   strcpy(fullpath, pathname);
   return(dopath(func));
}
```

Program 4.22

```
/*
 * Descend through the hierarchy, starting at "fullpath". If "fullpath" is anything other than a directory, we lstat() it,
 * call func(), and return.  For a directory, we call ourself recursively for each name in the directory. */
static int              /* we return whatever func() returns */
dopath(Myfunc* func) {
   struct stat  statbuf;
   struct dirent *dirp;  DIR  *dp;
   int  ret, n;
   if (lstat(fullpath, &statbuf) < 0) /* stat error */
      return(func(fullpath, &statbuf, FTW_NS));
   if (S_ISDIR(statbuf.st_mode) == 0)                      /* not a directory */
      return(func(fullpath, &statbuf, FTW_F)); /* It's a directory.  First call func() for the directory,
                                                * then process each filename in the directory. */
   if ((ret = func(fullpath, &statbuf, FTW_D)) != 0) return(ret);

   n= strlen(fullpath);
   if( n + NAME_MAX + 2 > pathlen){
      pathlen *= 2;
      if((fullpath = realloc(fullpath, pathlen)) == NULL)
          err_sys("realloc failed");
   }
   fullpath[n++] = '/';
   fullpath[n] = 0;

   if ((dp = opendir(fullpath)) == NULL) /* can't read directory */
      return(func(fullpath, &statbuf, FTW_DNR));
   while ((dirp = readdir(dp)) != NULL) {
      if (strcmp(dirp->d_name, ".") == 0  || strcmp(dirp->d_name, "..") == 0) continue; /* ignore dot and dot-dot */
      strcpy(ptr, dirp->d_name); /* append name after slash */
      if ((ret = dopath(func)) != 0) /* recursive */
         break; /* time to leave */
   }
  fullpath[n-1] = 0;

 if (closedir(dp) < 0)
      err_ret("can't close directory %s", fullpath);
   return(ret);
}
```

Program 4.22

```
static int
myfunc(const char *pathname, const struct stat *statptr, int type) {
  switch (type) {
  case FTW_F:
    switch (statptr->st_mode & S_IFMT) {
    case S_IFREG:  nreg++;  break;
    case S_IFBLK:  nblk++;  break;
    case S_IFCHR:  nchr++;  break;
    case S_IFIFO:           nfifo++;      break;
    case S_IFLNK:           nslink++;     break;
    case S_IFSOCK:  nsock++; break;
    case S_IFDIR:  err_dump("for S_IFDIR for %s", pathname);
                        /* directories should have type = FTW_D */
    }
    break;
  case FTW_D: ndir++; break;
  case FTW_DNR:  err_ret("can't read directory %s", pathname); break;
  case FTW_NS:  err_ret("stat error for %s", pathname); break;
  default: err_dump("unknown type %d for pathname %s", type, pathname);
  }
  return(0);
}
```

I. chdir, fchdir, and getcwd

```
#include <unistd.h>

int chdir(const char *pathname);

int fchdir(int filedes);

char *getcwd(char *buf, size_t size);
```

I. Special Device Files

- Every file system is known by its major/minor device numbers stored in a `dev_t` object.
- `major` and `minor` macros to access major/minor numbers.
- The `st_dev` is the dev no. of the file system containing the file.
- The `st_rdev` contains the dev no. of the character/block special files.
- Program 4.25

Program 4.25

```
#include "apue.h“
#ifdef SOLARIS
#include <sys/mkdev.h>
#endif
int main(int argc, char *argv[]) {
  int  i;
  struct stat  buf;
  for (i = 1; i < argc; i++) {
    printf("%s: ", argv[i]);
    if (stat(argv[i], &buf) < 0) {
      err_ret("stat error");
      continue;
    }
    printf("dev = %d/%d", major(buf.st_dev),  minor(buf.st_dev));
    if (S_ISCHR(buf.st_mode) || S_ISBLK(buf.st_mode)) {
      printf(" (%s) rdev = %d/%d“,
             (S_ISCHR(buf.st_mode)) ? "character" : "block",
             major(buf.st_rdev), minor(buf.st_rdev));
    }
    printf("\n");
  }
  exit(0);
}
```

Thank you for your attention !!

Q and A