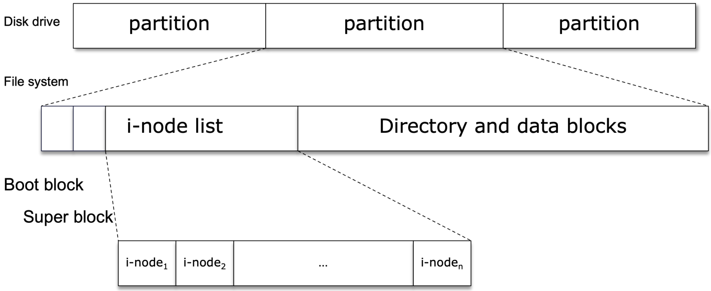
Linux File Stats

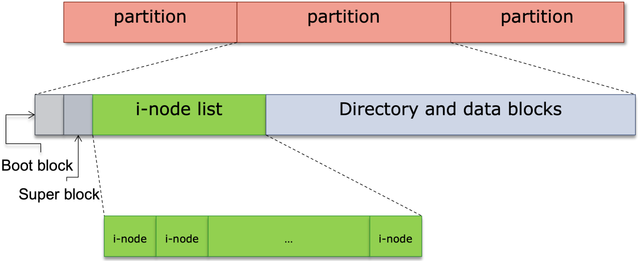
File Partition

- Various implementations of the UNIX/Linux file system are in use today.
- We can think of a disk drive being divided into one or more partitions. Each partition can contain a file system.



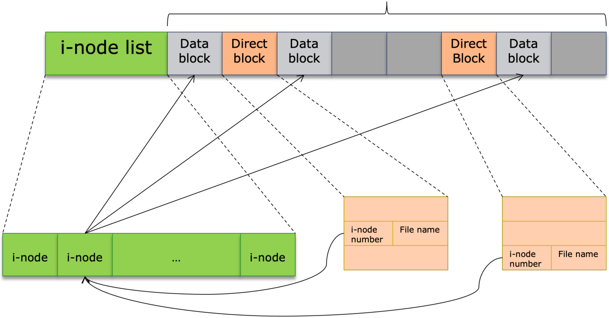
I Node List

- In Linux, almost everything is represented as a file.
- Each disk partition can be divided into disk blocks of smaller size. A block size varies in different systems.
- The Linux file system uses Index-Node (aka. I-Node) implementation method to keep track of how many blocks, and which blocks are used for a file.
- The superblock essentially records a file system's characteristics such as block size, other block properties, sizes of block groups and location of inode tables.



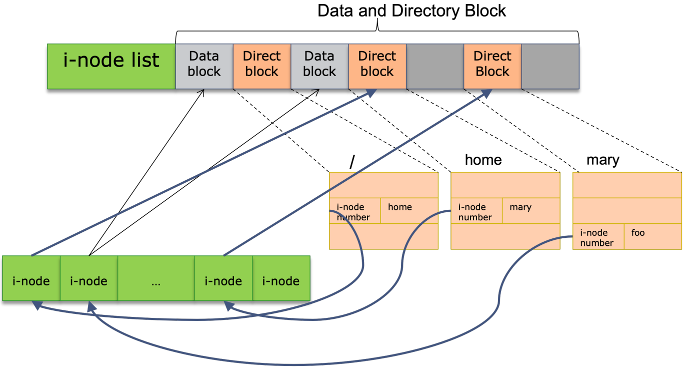
Directories and Data Blocks

- A Linux directory is a special file that acts as a container for other files and subdirectories.
- A directory entry stores information needed to locate the file disk blocks or subdirectory. Specifically, it stores i-node number, and filename/subdirectory name.
- The i-node stores file related information: file type, access permission, etc...
- Every i-node also stores the number of links from directory entries pointing to the i-node.
- A file is deleted when the number of links is 0.



Example of File Move

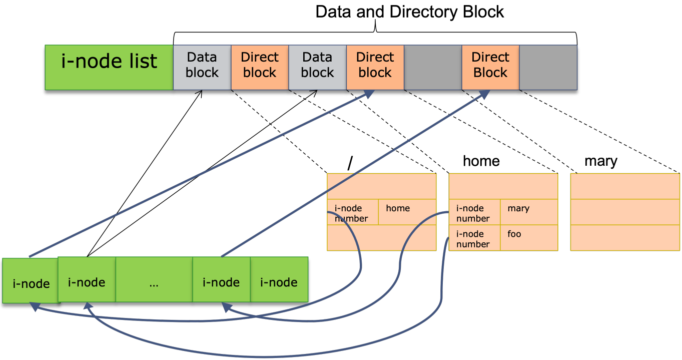
- When move a file from one directory to another directory, it only needs to change directory entry point.



Move /home/mary/foo to /home/foo

Example of File Move (Cont')

- When move a file from one directory to another directory, it only needs to change directory entry point.



Move /home/mary/foo to /home/foo

stat(), fstat(), lstat() System Calls for File Attributes

- Information/attributes related to a file can be obtained through `stat()`, `fstat()`, or `lstat()` system call.
- The `stat()` returns the structural information on a given file.
- The `fstat()` obtains attributes on a file with an open descriptor.
- The `lstat()` function is similar to `stat`, but when the named file is a symbolic link, `lstat()` returns information about the symbolic link.

Prototype of the 3 system calls:

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

// returns 0 if success, -1 on error.
int stat(const char *fname, struct stat *buf);
int fsat(int fd, struct stat *buf);
int lstat(cons char *fname , struct stat *buf);
```

Definition of stat() System Call

```
struct stat {
    mode_t st_mode;   /*file type & mode (permissions) */
    ino_t st_ino;     /* i-node number */
    dev_t st_dev;     /* device number (file system) */
    dev_t st_rdev;    /* device number for special files */
    nlink_t st_nlink; /* number of links */
    uid_t st_uid;     /* user ID of owner */
    gid_t st_gid;     /* group ID of owner */
    off_t st_size;    /* size in bytes, for regular files */
    time_t st_atime;  /* time of last access */
    time_t st_mtime;  /* time of last modification */
    time_t st_ctime;  /* time of last file status change */
    blksize_t st_blksize; /* best I/O block size */
    blkcnt_t st_blocks; /*number of 512 byte blocks allocated */
    mode_t st_attr;   /* The DOS-style attributes for this file */
};
```

The above special data types are all aliases for integer types (signed or unsigned):
https://www.gnu.org/software/libc/manual/html_node/Attribute-Meanings.html

or type `man 2 stat`

File Types

Regular file contains data of some form (text or binary)

Directory file contains the name of other files and pointers to the information on these file. Only kernel can write to the directory file

Character special file is a type of file providing unbuffered I/O access to variable-sized units to devices.

Block special file is a type of file used for disk devices.

FIFO is a type of file used for inter-process communication.

Socket is a type of file used for used for network communication.

Symbolic link is a type of file that points to another file.

How to Determine File Types

- The type of a file is encoded in the `st_mode` member of the `stat` structure.
- We can determine the file type by using macros in `<sys/stat.h>`.

```
S_ISREG(st_mode): Regular file
S_ISDIR(st_mode): Directory
S_ISCHR(st_mode): Character special
S_ISBLK(st_mode): Block special
S_ISFIFO(st_mode) : FIFO
S_ISLNK(st_mode): Symbolic link
S_ISSOCK(st_mode): Socket
```

Code Example to Display File Type

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

void err_ret(char *str){
  printf ("%s",str);
  exit (1);
}

int main (int argc, char *argv[]){
  int i;
  struct stat sb;
  char *ptr;

  if (argc != 2)
    err_ret("Argument number error.\n");
  /* get file infomation */
  if (stat(argv[1], &sb) < 0)
    err_ret("lstat Error");
  if (S_ISREG(sb.st_mode)) ptr ="regular";
  else if (S_ISDIR(sb.st_mode)) ptr ="directory";
  else if (S_ISCHR(sb.st_mode)) ptr ="character special";
  else if (S_ISBLK(sb.st_mode)) ptr = "block special";
  else if (S_ISFIFO(sb.st_mode)) ptr="fifo";
  else if (S_ISLNK(sb.st_mode)) ptr = "symbolic link";
  else if (S_ISSOCK(sb.st_mode)) ptr ="socket";
  else ptr ="unknown mode ";
  printf ("%s \n", ptr);
  exit (0);
}
```

In-depth Code Example to Display File Attributes

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h> // for ctime
//Check a file type and other attributes
int main(int argc, char *argv[]){
  struct stat sb;
  if (argc != 2) {
     fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);
     exit(1);
  }
  // get a file information and save in stat structure type
  if (stat(argv[1], &sb) == -1) {
     perror("stat");
     exit(1);
  }
  // check and print file type here ...
  // print other file attributes information
  printf("I-node number: %ld\n", (long) sb.st_ino);
  printf("Mode: %lo (octal)\n",(unsigned long) sb.st_mode);
  printf("Link count: %ld\n", (long) sb.st_nlink);
  printf("Ownership: UID=%ld GID=%ld\n",
  (long) sb.st_uid, (long) sb.st_gid);
  printf("Preferred I/O block size: %ld bytes\n", (long) sb.st_blksize);
  printf("File size: %ld bytes\n",(long) sb.st_size);
  printf("Blocks allocated: %ld\n",(long) sb.st_blocks);
  printf("Last status change: %s", ctime(&sb.st_ctime));
  printf("Last file access: %s", ctime(&sb.st_atime));
  printf("Last file modification: %s", ctime(&sb.st_mtime));
  exit(0);
}
```

Question

What’s the i-node number for root directory?

link() System Call

- Any file can have multiple directory entries pointing to its i-node.
- The way we can create a link to an existing file is with the link system call.
- The link() system call create a new directory entry `newpath` that references the existing file `existingpath`.
- The link creation and link count increase are done automatically by kernel.
- Only a superuser process can create a new link that points to a directory. (Because link system calls can cause loops in the filesystem.)

```
#include <unistd.h>

//Return 0 on success, -1 on error.
int link (const char *existingpath, const char *newpath)
```

link() System Call Example

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

void err_sys(char *str){
printf ("%s\n",str);
  exit (1);
}

int main (int argc, char *argv[]){
  struct stat buff;
  if (argc < 2)
    err_sys ("Less than two argument Error");
  /* increase the number of link by one */

  if (stat (argv[1], &buff) < 0)
    err_sys("stat error for foo");
  printf ("link count for a file %s was %d \n",argv[1], buff.st_nlink);

  if (link (argv[1], argv[2]) <0)
    err_sys("Link Error");

  if (stat (argv[1], &buff) < 0)
    err_sys("stat error for foo");
  printf ("link count for a file %s is now %d \n",argv[1], buff.st_nlink);
  return 0;
}
```

unlink() System Call

- unlink() system call removes an existing directory entry, and and decrement the link count of the file referenced by pathname.
- To unlink a file, we must have write permission and execute permission in the directory containing the directory entry.

```
#include <unistd.h>

//Return 0 on success, -1 on error.
int unlink (const char *pathname)
```

link() and unlink() System Call Example

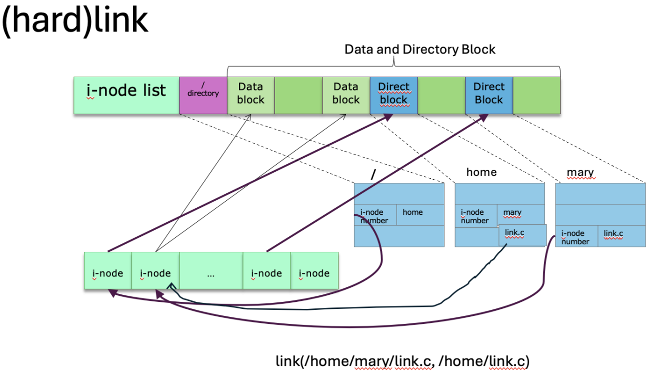
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
void err_sys(char *str){
  printf ("%s\n",str);
  exit (1);
}

int main (int argc, char *argv[]){
  struct stat buff;
  if (argc < 2)
    err_sys ("less than two argument Error");
  if (stat (argv[1], &buff) < 0)
    err_sys("stat error for foo");
  /* increase the number of link by one */
  printf ("link count for a file %s was %d \n",argv[1], buff.st_nlink);
  if (link (argv[1], argv[2]) <0)
    err_sys("Link Error");
  if (stat (argv[1], &buff) < 0)
    err_sys("stat error");
  printf ("link count for a file %s is now %d \n",argv[1], buff.st_nlink);
  sleep (10);
  /* unlink existing file */
  if (unlink(argv[1])<0)
    err_sys("unlink Eror");

  return 0;
}
```

Limitation of (Hard) Link

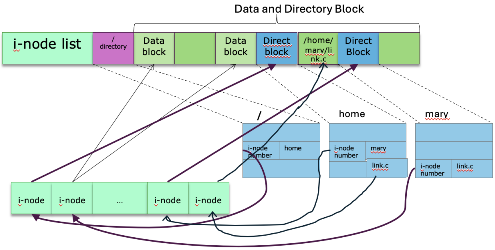
- It requires the file reside in the same file system.
- Only super user can create (hard) link to a directory.



Symbolic Links

- A symbolic link is an indirect pointer to a file or directory. It stores the path of the existing file or directory.
- Unlike (hard) link, symbolic link can be used to move a file or an entire directory hierarch to some other location on a system.
- It can also be used to execute a file.

Symbolic link



symlink(/home/mary/link.c, /home/link.c)

symlink() System Call

```
#include <unistd.h>
// return 0 if success; -1 on error.
int symlink (const char *existingpath, const char *sympath)
```

Example code to create a symbolic link to an executable file

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

void err_sys(char *str){
  printf ("%s\n",str);
  exit (1);
}
int main(int argc, char *argv[]){
  /* create a symbolic link from argv[2] to argv[1] */
  if ( symlink(argv[1], argv[2]) <0)
     err_sys("Symbolic Link Creation Error \n");

  printf("A symbolic link is created \n");

  return 0;
}
```

Time Attributes in File Stats

Three time fields are maintained for each file.

st_atime

st_mtime

st_ctime – last change time of i-node status (permission, user id, number of links, etc.)

The access time and the modification time of a file can be changed with the utime system call.

```
//Return 0 on success, -1 on error.
#include <sys/types.h>
#include <utime.h>
int utime(const char *pathname, const struct utimebuf *times);

struct utimebuf{
  time_t actime;
  time_t modtime;
}
```

utime() System Call Example Code

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <utime.h>
void err_sys(char *str) {
  printf ("%s\n",str);
  exit (1);
}
int main (int argc, const char *argv[]){
  struct stat statbuf0, statbuf1;
  struct utimbuf timebuf0, timebuf1;
  if (argc < 3)
    err_sys ("Argument Error");
  /* get first file information */
  if (stat(argv[1], &statbuf0) <0)
    err_sys("Stat Error");
  timebuf0.actime = statbuf0.st_atime;
  timebuf0.modtime = statbuf0.st_mtime;
  /* get second file information */
  if (stat (argv[2], &statbuf1) <0)
    err_sys("Stat Error");
  timebuf1.actime = statbuf1.st_atime;
  timebuf1.modtime = statbuf1.st_mtime;
  /* change time information between two file */
  if (utime (argv[1], &timebuf1) <0)
    err_sys("Time change Error1");
  if (utime (argv[2],&timebuf0) <0)
    err_sys("Time Change Error");
  exit (0);
}
```

Time and Date Service in Linux

- Linux time counts the # of seconds that has passed since the Epoch:00:00:00 Jan. 1 1970.
- The time function returns the current time and date.

```
#include <time.h>
time_t time (time_t *calptr);
```

- The two functions localtime and gmtime convert a calendar time into `tm` structure.

```
#include <time.h>
struct tm *gmtime(const time_t *timer); /*convert to UTC */
struct tm *localtime(const time_t *timer); /*convert to local time*/

struct tm {
  int tm_sec; /* seconds [0,61] */
  int tm_min; /* minutes [0,59] */
  int tm_hour; /* hour [0,23] */
  int tm_mday; /* day of month [1,31] */
  int tm_mon; /* month of year [0,11] */
  int tm_year; /* years since 1900 */
  int tm_wday; /* day of week [0,6] (Sunday = 0) */
  int tm_yday; /* day of year [0,365] */
  int tm_isdst; /* daylight savings flag */
}
```

Time and Date Service in Linux

- The asctime and ctime function produce the 26 byte formatted string such as
  `Tue Sep 23 07:07:21 2020`

  ```
  #include <time.h>
  char *asctime(const struct tm *timeptr);
  char *ctime(const time_t *clock);
  ```

- The two functions are essentially the same, but ctime tend to adjusts to the time zone and daylight saving time.

```
/* example.c displays current date and time */
#include <stdio.h>
#include <time.h>
int main() {
  time_t t1 = time(NULL); //current calendar time in sec. since 1/1/1970
  printf("current time is: %s\n",asctime(localtime(&t1)));
  return 0;
}
```