File Systems

Week 10 – Lab

Linking Files – Hard Links (1/2)

- A directory may contain several filenames that all map to the same i-node number and thus to the same file in the file system. Unix call these names pointers or links to the file.
- Hard links are new names for the same i-node. Link count in the i-node keeps track of how many directories contain a name number mapping for that i-node
- Hard links cannot be made to a directory. This restriction means that every subdirectory has one and only one parent directory

Linking Files – Hard Links (2/2)

- Command for creating a hard link:

  $ ln filename linkname

- A hard link and data it links to, must always exist in the same file system.
- Command to find i-node number of a file:

  $ ls -i filename

Exercise 1

- Create *_ex1.txt*
- Link it to *_ex1_1.txt* and *_ex1_2.txt*
- Check i-node numbers of all the files and save the output to the file *ex1.txt*

Exercise 2

- Create *file.txt* in *week01* directory and access this file from *week10* directory via **$ link <source> _ex2.txt**
- Trace all links to *file.txt*:
  $ find <path> –inum *inodenumber*
- Remove all links from *file.txt*
  $ find <path> –inum *inodenumber* -exec rm {} \;
- Save output of all the steps to file *ex2.txt*

Linking Files – Soft Links

- A soft link or symbolic link contains **a path** to another file or directory and may point to any file or directory
- Can cross file systems
- Created by

  $ ln –s <source> <target>

File Permissions

- Read (r): with read permission we can see the contents of the file
- Write (w): allows us to change the file such as add to a file, overwrite it etc.
- Execute (x): with execute permission we can ask the operating system to run the program

Directory Permissions

- Read (r): list the contents of the directory
- Write (w): add, rename and move files in the directory
- Execute (x): list information about the files in the directory (sometimes called search permission)

Exercise 3

- Make a file *_ex3.txt* and try the following:
- Remove execute permission for everybody
- Grant all permissions to owner and others (not group)
- Make group permissions equal to user permissions
  - What does 660 mean for *ex3.txt*?
  - What does 775 mean for *ex3.txt*?
  - What does 777 mean for *ex3.txt*?
- After each step save the output/answer to the *ex3.txt*

chmod()

- The read, write and execute permissions are stored in three different places called Owner, Group and Other.
- Display permissions: **$ ls -l**
- There are three sets of *rwx* determined by 9 bits of i-node information. Usage:
  - **chmod u=rwx filename**
  - **chmod g=rwx filename**
  - **chmod o=rwx filename**
  - **chmod a=rwx filename**

stat() system call (1/2)

- The stat() function obtains information about the file pointed to by path. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.

  Syntax (**$ man 2 stat**):

```
int stat(const char *restrict path,
         struct stat *restrict buf);
```

stat() system call (2/2)

Structure stat:

```
struct stat {
  dev_t    st_dev;    /* device inode resides on */
  ino_t    st_ino;    /* inode's number */
  mode_t   st_mode;   /* inode protection mode */
  nlink_t  st_nlink;  /* number of hard links to the file */
  uid_t    st_uid;    /* user-id of owner */
  gid_t    st_gid;    /* group-id of owner */
  dev_t    st_rdev;   /* device type, for special file inode */
  struct timespec st_atimespec; /* time of last access */
  struct timespec st_mtimespec; /* time of last data modification */
  struct timespec st_ctimespec; /* time of last file status change */
  off_t    st_size;   /* file size, in bytes */
  quad_t   st_blocks; /* blocks allocated for file */
  u_long   st_blksize;/* optimal file sys I/O ops blocksize */
  u_long   st_flags;  /* user defined flags for file */
  u_long   st_gen;    /* file generation number */
};
```

opendir() function

- The opendir() function opens the directory named by filename, associates a directory stream with it and returns a pointer to be used to identify the directory stream in subsequent operations
- Syntax (**$ man opendir)**:

```
DIR *opendir(const char *filename);
```

readdir() function

- The readdir() function returns a pointer to the next directory entry. It returns NULL upon reaching the end of the directory or on error
- Syntax (**$ man readdir)**:

```
struct dirent *readdir(DIR *dirp);
```

opendir() + readdir() Example

```
dirp = opendir(".");
if (dirp == NULL) { return (ERROR); }
len = strlen(name);

while ((dp = readdir(dirp)) != NULL) {
   if (dp->d_namlen == len &&
    strcmp(dp->d_name, name) == 0) {
      (void)closedir(dirp);
      return (FOUND);
   }
}

(void)closedir(dirp);
return (NOT_FOUND);
```

Exercise 4

- Create *tmp* directory with two empty files (*file1*, *file2*)
- Create one hard link named *link1* to *file1*
- Write a program that scans *tmp* directory, locates all i-nodes with a hard link count of two or more
- For each such file it should display **together** all file names that point to the file
- Save the output of the program to *ex4.txt* and also submit the code *ex4.c*

Exercise 5 (Optional)

- Implement a simulated file system that will be fully contained in a single regular file stored on the disk. This disk file will contain directories, i-nodes, free-block information, file data blocks, etc. Choose appropriate algorithms for maintaining free-block information and for allocating data blocks (contiguous, indexed, linked). Your program will accept system commands from the user to create/delete directories, create/delete/open files, read/write from/to a selected file, and to list directory contents