5: Directories and I-Nodes

CSCI 4547 / 6647 Systems Programming
Fall 2017

1 Goals

1. To work with Unix directories and I-nodes.
2. To read and process directory entries and I-nodes.
3. To sort the data in a vector using a comparison function.

2 Instructions

Write a C++ program to analyze the entries in a disk directory. You will run this program from a Unix command shell, using command-line arguments. Your main function should accept argc and argv from the command shell, and process them as follows:

2.1 Class `Stats`

All Unix systems define `struct stat`, but the definitions are not all identical: field sizes change. Here is the definition on my machine.

```
    #include <sys/stat.h>
    struct stat {            /* when _DARWIN_FEATURE_64_BIT_INODE is NOT defined */
     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 */
};
```

Define a wrapper class for Unix standard type `struct stat`. Create a class called Stats by deriving it from `struct stat`, by private derivation. (This closes off outside access to the underlying C struct.) (Yes, you can derive a class from a struct.) Within this class, define the following functions:

- `void print( ostream& out )` Send (using <<) the inode number, nlink, and size to the output stream parameter.
- An accessor named `inode()` to return the inode number.
- An accessor named `size()` to return the file size.
- An accessor named `links()` to return the number of hard links to this inode.

Please write all accessors as 1-line, inline functions. If you don't know what I mean, ask, but please do not ignore this request.

2.2 `Class Direntry`

Define a wrapper class for UNIX standard type `struct dirent`. Create the class, Direntry, by deriving it from `struct dirent` using private derivation. Here is the definition of `struct dirent` on my machine. Yours may be slightly different:

```
struct dirent { /* when _DARWIN_FEATURE_64_BIT_INODE is NOT defined */
    ino_t      d_ino;                 /* inode number of entry */
    __uint16_t d_reclen;              /* length of this record */
    __uint8_t  d_type;                /* file type, see below */
    __uint8_t  d_namlen;              /* length of string in d_name */
    char       d_name[255 + 1];       /* name must be no longer than this */
};
```

Within this class, define the following functions:

- `void print( ostream& out )` Send (using <<) the file type, the inode number, and the name to the output stream named by the parameter.
- An accessor named `name()` to return the name (type char*).
- An accessor named `inode()` to return the inode number (type ino_t).
- An accessor named `type()` to return the file type, a small unsigned integer.

Please write all accessors as 1-line, inline functions. If you don't know what I mean, ask, but please do not ignore this request.

2.3 The FileID Class

Define a class named `FileID` with three private data members: the pathname (a C++ string), the I-node number, and the file length. Within this class, define the following functions:

- A constructor, with one parameter for each data field.
- `void print( ostream& out )` Send (using <<) all fields to the output stream. Use tabs to make neat columns.
- A static comparison function for sorting in ascending order by inode number.
- A static comparison function for sorting in ascending order by file size.

Write accessors if you need them. I don't think you do. Let me know if you do not know how to write a comparison function. I will put together a lesson about that.

2.4 Sweeper and the main function

main() Accept command-line arguments. Print a welcome message on the screen. Construct an instance of the Sweeper class and pass the command-line-arguments to the Sweeper constructor. Call the `sweep()` function.

Sweeper Your `Sweeper` class will become the file-duplicate-cleanup application. All functionality will go into this class, not into the main function. You already have a constructor and a run() function..

2.5 Review: The Sweeper Constructor.

You will process the following switches, as in P4:

- `-b` or `--verbose`: provide verbose debugging feedback.
- `--delete`: Delete the duplicate files
- `--o` filename: open and use the named file for output.
- A small integer (optional) that represents a file size, in kilobytes. (That is, 5 represents 5K bytes.)
- A pathname that denotes the directory at which to start sweeping (a C-string, as a required, non-switch argument).

2.6 In the run() function.

Preparation: A relative pathname. If the pathname in the Params object is a relative pathname, you need to do some string manipulation to convert it to an absolute pathname. If it starts with ./ remove those two characters. Then use `getcwd()` to get the current working directory. Then append a slash to it, followed by the pathname-argument-string. Now you have an absolute pathname. Store it in `Sweeper::path`.

Use `opendir()` to open the directory named by the pathname in Params::name. Read each entry in it. If verbose is on, print the file name and the I-node number in the entry. If the verbose switch is not on, ignore all entries EXCEPT the regular files. (In P6, we will also process subdirectories.) Process the regular files thus:

- Call `lstat()` to get the stats on this file.
- Convert the file name to an absolute pathname by appending it, with a leading slash, to `path`.
- Create a `FileID` object using the absolute pathname, the I-node number, and the file length. Push the object onto the back of a `vector<FileID>`.

When all of the directory entries have been read, use `sort()` to sort your vector in order of inode number. Use the static comparison function in the `FileID` class. Then print the `FileID`'s, in groups, as follows:

- On the first line of each group, print the I-node number and the number of hard links to that I-node.
- Below that, indented, print one line for each filename that shares this I-node.

Example. Suppose the working directory is ∼`/cs647` and it contains these entries:

```
-rw------- 1 alice staff 3680 Jan 9 20:29 Sysfirst.dat
-rw-r--r-- 3 alice staff 760 Jan 9 11:39 dummy1.html
drwx------ 7 alice staff 238 Jan 12 12:31 modules
lrwx------ 1 alice staff 16 Jan 14 11:49 mytemp.txt
drwxr-xr-x 12 alice staff 408 Jan 10 14:23 options
-rwx------ 1 alice staff 8712 Jan 10 11:54 preDemo
-rw-r--r-- 1 alice staff 9140 Jan 10 11:54 preprocess.c
-rw-r--r-- 3 alice staff 760 Jan 9 11:39 syllabus.html
-rw-r--r-- 1 alice staff 3041 Jan 8 21:24 sysprog.1
```

The verbose output would start with:

```
file       3451716 Sysfirst.dat
file       3451367 dummy1.html
directory  3441725 modules
link       3461619 mytemp.txt
directory  3453742 options
file       3452164 preDemo
file       3443578 preprocess.c
file       3451367 syllabus.html
file       3457298 sysprog.1
```

The output (verbose or not) would end with:

```
I-node 3443578 links 1
   ~/cs647/preprocess.c
I-node 3451367 links 2
    ~/cs647/syllabus.html
    ~/cs647/dummy1.html
I-node 3451716 links 1
    ~/cs647/Sysfirst.datdummy1.html
I-node 3452164 links 1
    ~/cs647/preDemo
I-node 3457298 links 1
    ~/cs647/sysprog.1
```

Other instructions: Due Oct 9

- Don't hesitate to ask for help or clarification.
- Test for error codes returned by the library functions. Do not let an error crash your program. Otherwise, don't do anything special about handling errors; that will come in the next phase of this program.
- Create a test folder. In it, put a few files of varied lengths and a subdirectory that contains a couple of files. Now add some links, both hard and soft, to files in the same directory and in the subdirectory. Do `ls -l` on your test directory before the tests and put the results in your output file.
- Test all of the command-line options except `--delete` and capture the results from all tests in your same output file.