Operating Systems
CSCI 5806

Spring Semester 2025 — CRN 22968

Term Project — Step 6 — Directory Access

Target completion date: Friday, April 18, 2025

Goals

- Provide functions to provide access to directory entries.

Details

In this step, we provide access to a directory's entries in sequential order.

Directory Entries

A directory entry contains a file name and an inode number, plus size information. The data, in order of appearance, is:

- A four-byte inode number. If the number is zero, this entry is not used.
- A two-byte record length. This is the number of bytes used by all fields in this entry. It will always be a multiple of 4.
- A one-byte name length. This is the actual number of bytes in the file name.
- A one-byte file type. It contains information already stored in the inode, relating to the file type.
- The file name, as an array of characters.

The following structure can (should?) be used:

```
struct Dirent {
    uint32_t
        iNum;
    uint16_t
        recLen;
    uint8_t
        nameLen,
        fileType,
        name[1];
};
```

Note that the name array only has one byte; C and C++ don't check array bounds, so it is safe to point this structure to the start of a directory entry and copy *nameLen* bytes from the name array.

Also note that a directory entry will never span two blocks; an insufficiently large space at the end of a block is usually absorbed into the last entry as extra padding after the name; the *recLen* field is increased to incorporate the additional space.

Directory functions

Four functions are necessary at a minimum for reading a directory:

- **`struct Directory *openDir(uint32_t iNum)`**
  Open the directory with the given inode number. Return a pointer to a dynamically allocated structure that holds relevant information for processing.
- **`bool getNextDirent(struct Directory *d,uint32_t &iNum,char *name)`**
  Fetch the next directory entry in sequence. Fill in the inode number and name in the last two parameters. Advance file cursor to the next entry. Return true if the fetch succeeded, false if you were at the end of the directory.
- **`void rewindDir(struct Directory *d)`**
  Reset the directory's cursor to `0`.
- **`void closeDir(struct Directory *d)`**
  Close the directory. Deallocate any dynamic memory used by the directory.

The . and .. entries

The first two entries in any directory have the names `.` and `..`

They are an alias for the current directory (`.`) and the parent directory (`..`).

If you are doing a recursive traversal of the directory tree, be sure to skip these two entries, or you will end up with infinite recursion.

Erratum

The part 5 example has an error. Inode 11 — the lost+found directory — is not all zero bytes after the first two directory entries. The first entry is `.` and takes up 12 bytes as it should. The second entry is `..` and takes the remaining 1012 bytes in the first file block. Each of the remaining 11 blocks contains a single directory entry with inode number 0 and record length 1024.

Example

This is the code snippet I used to test my step 6.

```
char name[256];
uint32_t iNum;
Directory *d;

cout << "Root directory contents" << endl;
d = openDir(f,2);
while (getNextDirent(d,&iNum,name)) {
    cout << "Inode: " << iNum << "   name: [" << name << ']' << endl;
}
closeDir(d);

cout << "\n\nlost+found directory contents" << endl;
d = openDir(f,11);
while (getNextDirent(d,&iNum,name)) {
    cout << "Inode: " << iNum << "   name: [" << name << ']' << endl;
}
closeDir(d);
```

Here is the output:

```
Root directory contents
Inode: 2   name: [.]
Inode: 2   name: [..]
Inode: 11   name: [lost+found]
Inode: 12   name: [arduino-1.6.7-linux64.tar.xz]
Inode: 30481   name: [examples]
Inode: 24   name: [arduino-builder]


lost+found directory contents
Inode: 11   name: [.]
Inode: 2   name: [..]
```