7: Finishing File Sweeper

CSCI 4547 / 6647 Systems Programming

Fall 2017

1 Goals

1. To calculate and use a cryptographic hash value.
2. To identify duplicate files in a directory tree and output their path names.

1.1 Preparation

- For P6, you created a file tree as test data. Go into that tree and add several hard and soft links. Link things from different subdirectories and different levels of the same path. Create at least one file with multiple hard and soft links leading to it. Make sure you have some files of the same length and some of different lengths.
- Add another field to your FileID class to store the fingerprint of the file: an array of unsigned char of length SHA_DIGEST_LENGTH.
- Add a comparison function called `bySize()` to the FileID class that will let you sort by the file size. This function will be called at the beginning of P7.
- Add another comparison function called `byFprint()` to the FileID class that will let you sort by the fingerprint field.
- Add a function to the FileID class that will calculate the SHA256 hash for and store it in the FileID object. It will be called in the last phase of the process. You should copy and modify my SHA demo program to implement it. Use `fread()` to read a file into a buffer in huge blocks, to minimize disk time. Suggestion if it works for your system: 10^10 bytes. For many files, this is big enough to hold the whole file.
- In the Sweeper class, define a function called `areDups()` with two pathname parameters. Its job is to compare the two files and return true if they are identical, false otherwise. This function will only be called if the file size is the same and the iNode #s are different.

2 Sweeper::run().

By the end of program 6, you had built a `vector<FileID>` that contained one FileID object for every file and every hard link in your directory tree (starting at the specified directory).

The goal of program 7 is to use that vector to discover the duplicate files, and to report on those files in enough detail that someone could actually delete them.

First, sort by size.

- In the Sweeper class, declare two iterators (`start` and `end`), for your vector. Initialize both to the slot 0 of the vector.
- Use `stable_sort()` and `bySize()` to sort your vector in order. The prototype is:

```
    void stable_sort ( Iterator first, Iterator last, Compare comp );
```

Now, enter a loop. This loop implements what some people call a “sliding window”. The top of the window is set to the first FileID of each size, in turn, and the bottom of the window is set to the first FileID that is of the next bigger size. Between are the files that could possibly be duplicates.

- Enter a loop that increments the `end` iterator, checking the file size each time. Leave the loop when the size changes. Now your iterators point to the beginning and off-board-end of a block of equal-size FileID objects. This is the correct position for calling `sort()` or `stable_sort()`
- If the difference of the iterators is only one slot, you are done with this block of objects. Set the `start` iterator = the `end` iterator and continue from the top of the loop.
- If the difference of the iterators is more than one slot, you might have a duplicate OR you might have a file that has hard links. To check, call the `checkDups()` function.
- When `checkDups()` returns, go back to the top of the loop.

Write a function `void checkDups()`. This function will do the actual output of duplicate-file pathnames.

- Because you used `stable_sort()`, the objects are still in iNode order. Check whether all the iNode #s in the block are the same,. If so, print the paths from the FileID objects in the block. For each, print “Link: ” followed by the pathname.
- If there are at least two different iNode #s in the block, calculate the SHA256 hash for each FileID and store it in the object. Do this by calling the function in the FileID class. (With a little programming effort, you can avoid calculating the hash twice for the same iNode #.)
- When hash values have been stored in all the objects in the current block, call `stable_sort()` again to sort the FileID objects by fingerprint.
- Start at the top of the block. Use control-break logic to print groups of duplicate files, that is, groups of FileID’s that have the same fingerprint. (Either these files actually are different, or we have “broken” SHA256 and have a publishable paper.)