Overview

In this task you will be adding the following into your file system from Task 1:

- A Global File Table with support for multiple processes
- A Local Per Process File Table
- File Manipulation Functions:
  - Open
  - Read
  - Write
  - Close

Motivation of File Tables

These file tables represent the idea of giving processes quick access to information of files they have open rather than needing to traverse memory looking for them every time because in real systems memory access is very slow. In real systems (not actually implemented in this project) memory access is much slower than the global file table access, and the global file table access is much slower than the local per process file table access. Each level saves traversal time.

Global File Table

The Global File Table contains info for every file currently open by at least one process. Essentially it serves as a hash-table directory of all the files in memory. Its interactions with the file manipulation functions will be described in their corresponding sections.

Its structure:

```
typedef struct open_file_global_type
{
   unsigned short fd; // reference to the file descriptor node
   unsigned short data; // reference to the data or index node (depending on the size
)
   mode_t access; // access rights for the file
   unsigned short size; // size of the file
   unsigned short reference_count; // Number of processes using this file.
   struct open_file_global_type *next;
} OPEN_FILE_GLOBAL_TYPE;
```

Its declaration:

```
#define GLOBAL_TABLE_SIZE 65521 // prime number for hashing
OPEN_FILE_GLOBAL_TABLE global_table[GLOBAL_TABLE_SIZE];
```

Hashing

In order to enter the file into the global file table, I'd recommend using some simple hashing function based perhaps on the file name, then using linear probing to find the correct entry or an empty space for insertion. The implementation is up to the student but this would be the minimum required for full credit.

Note on Multiple Processes and the Global File Table

While the global versus per process tables implemented in this task seem to imply that students will be running this task with multiple processes interacting with the file system, they will not. I am not sure why other than possibly the amount of code that would be necessary to do so? They are simply implementing a structure that CAN scale to multiple processes in this task. In Task 5, where they integrate their file system with FUSE, this functionality will be actually tested with multiple processes.

Per Process File Table

The Per Process File Table contains information for every file open by the local process. Because Task 2 only involves one process, this will just will nominally be "every file open" but that is not the case when it will be tested in Task 5. Essentially it provides quick reference to detailed file information in the Global file table so that a search & some instances of lookup don't need to be performed.

Structure and Declaration:

```
typedef struct open_file_local_type // a node for a local list of open files (per pro
cess)
{
   mode_t access_rights; // access rights for this process
   unsigned_short global_ref; // reference to the entry for the file in the global ta
ble
} OPEN_FILE_LOCAL_TYPE;

#define MAX_OPEN_FILES_PER_PROCESS 16
OPEN_FILE_LOCAL_TYPE local_table[MAX_OPEN_FILES_PER_PROCESS];
```

File Manipulation Functions:

Open

```
If the access mode checks out given the request:
    If file is not in the global file table:
        Add to global file table with a reference count of 1
        Add to per process table for the requesting process
    If already in the global file table:
        Check that access mode of currently open files doesn't conflict with new requ
est (ie read ->write):
            Increment global file table reference count for that file
            Add to per process file table
```

Read

Gets all of the content of the file by traversing any index and or data node(s) and loads it into a buffer. File size is very important for this function.

Write

Appends more data to a file might need to create more data nodes or an index node if the file was previously short. Copies from buffer to file.

Close

Removes file from per process file table, decrements reference count in global file table, if the reference count is now 0, removes file information from the global file table.