Server

Structs:

```
Struct filenode{
        pthread_mutex_t writelock;
        pthread_mutex_t nodelock;
        pthread_cond_t doneReading;
        pthread_cond_t doneWriting;
        int status;
        char* path;
        int reads;
        int writes;
        int unrestricted;
        int exclusive;
        int transaction;
        struct filenode* next;
} filenode;
```

The filenode struct is used for tracking and managing open files. There is one per file. The two locks are for writing and reading from the file itself, and editing the contents of the node respectively. The two conditionals are for signaling other threads that they may do their own operation on the file. The status integer tracks whether a read or a write is being performed. A 0 signifies no operation, 1 signifies reading, and 2 signifies writing. Path is the full pathname of the file. Reads and writes track how many read/write permissions have been applied to the file. Unrestricted, exclusive, and transaction track how many of each mode are applied to the file.

```
Struct listnode{
        Int fd;
        Char* path;
        Char mode;
        Int r;
        Int w;
        Struct listnode* next;
} listnode;
```

The listnode struct is used for managing file descriptors. There is one per file descriptor. Fd is the file descriptor itself. Path is the full pathname the file descriptor refers to. Mode is the filemode for the client's connection. R and w track the read and write permissions of the file descriptor.

Data Structures:

```
Filenode* Table[BUCKETSIZE]
```

The Table data structure is a hash table of filenodes. Each of the nodes are hashed by the ASCII values of their file paths. This allows files with the same name to be hashed differently. Table is used primarily for determining the read and write permissions applied to a file, as well as the modes that a file has been opened with.

Listnode* List[BUCKETSIZE];

This data structure is a hashtable of listnodes. It keeps track of all the file descriptors. It's primary use is for determining the filenode* which corresponds to the file descriptor.

Functions:

Int hash(char* word)

This functions adds up the ASCII values in word, and returns the modulus with the bucketsize. It is used for hashing nodes into the Table hashtable.

Void AddtoList(char* path, int fd, int flags, char mode)

This functions adds a listnode to List using the given information.

Int AddtoTable(char* path, char mode, int flags)

This function determines whether the given path can be opened. It checks the filemode against existing filemodes, as well as existing read and write permissions. If it cannot be opened, -1 is returned. If it can be opened, the path is opened and the file descriptor is returned. If no node is found, the function tries to open path. If it can open it, a new filenode is inserted into Table and the file descriptor is returned. Otherwise, -1 is returned.

Int fileopen(char* path, char mode, int flags, int length)

This function calls AddtoTable and AddtoList. It returns either a file descriptor or -1.

Int foleclose(int fd)

This function closes the given file descriptor, removes the corresponding listnode, and updates the corresponding filenode. If, after updating the filenode, there is no client which has the corresponding file open, the filenode is deleted from Table.

Void *process (void* arg)

This function is called by each new thread to service individual clients. It unpacks the protoco l, and determines which operation to perform. This is where each thread communicates with their clients. Depending on the received message, this function will either perform an open, close, read, or write operation. If no message is received over the course of 20 seconds, the thread terminates.

Int main(int argc, char** argv)

This function initializes the server, and branches off each client connections into their own individual threads.

Client

The client uses negative file descriptors, but positive ones are sent to the server. This is done by adding 2 and multiplying by 1, as it is the simplest method to avoid a file descriptor of -1.

Protocols:

The protocol used differs depending on the operation, but each protocol share the same feature: the first character signifies the operation. Protocols are sent as character arrays of varying sizes. The protocols for each operation are listed below, there are no spaces in between entries.

Netopen

Sent: char,char,int,int,char*
The first char is 'O', signifying an open operation
The second char is the filemode
The first int is the flags
The second in is the length of the path
The char* is the path
Received: char,int
If char is 'e', int is an errno value
If char is 'f', the int is returned as the file descriptor.

Netread

Sent: char,int,size_t
The char is 'R', signifying a read operation
The int is the file descriptor
The size_t is the number of bytes to read
Received: char,int or void*
If char is 'e', int is an errno value
If char is not 'e', then void* is the data to be written

Netwrite

Sent: char,int,size_t,void*
Char is 'W', signifying a write operation
Int is the file descriptor
Size_t is the number of bytes to be written
Void* is the data to be written

Received: char,int or ssize_t
If char is 'e', int is an errno value
If char is not 'e', ssize_t is the number of bytes written

Netclose

Sent: char,char,int
The first char is 'C', signifying a close operation
The second char is the filemode
The int is the file descriptor
Received: (char,int) or "yes"
If char is 'e', int is an errno value
If char is not e, the string "yes" is received

Functions:

Int netserverinit(char* hostname, int filemode);

This function establishes a connection with the server. If the hostname or filemode are not correct, errors are thrown.

Int netopen(const char* pathname, int flags)

This function sends a message to the server to open a file. The file descriptor is returned if this was successful, otherwise errno is set and -1 is returned.

Ssize_t netread(int fildes, void *buf, size_t nbyte)

This function sends a message to the server requesting to read from a file. If the server doesn't respond, or responds with an error, -1 is returned and errno is set. Otherwise, data is copied into buf, and the number of bytes read is returned.

Ssize_t netwrite(int fildes, const void *buf, size_t nbyte)

This function sends a message to the server requesting to write to a file. If the server doesn't respond, or responds with an error, errno is set and -1 is returned. Otherwise, the number of bytes written is returned.

Int netclose(int netfd)

This function sends a message to the server requesting to close a file descriptor. If the server doesn't respond, or responds with an error errno is set and -1 is returned. Otherwise, if the file was successfully closed, 0 is returned.