REPORT

FILE SYSTEM SIMULATION

Submitted by,

COSC 519 Project – Files & Directories

Primary Functions

Our simulation includes these operations:

Files

Create
Delete
Open
Close
Read
Write

Directories

Create
Delete
List
Initialize

Both files & directories

Rename
Move
GetInfo
SetInfo

Language used C

Basic data structures

File system object info (FSObjectInfo) is the primary metadata structure. It contains data on size, time of creation, time last modified, permissions (readable/writable), name, and whether it is a directory or file.

Another struct FSObject contains this data for files: pointer to the parent, data, pointers to siblings (next and previous children of the parent), and open status. The same struct is used for the root directory.

Design decisions:

- Multi-Level Directory Structure. This is more functionality than required by Dr. Karne, who suggested a single level file system structure. Because the multi-level programs were so far along – already in the final debugging phase -- when Dr. Karne made his suggestion, the attached programs reflect the multilevel design; thus, for example, rather than just a create root directory program, we have included programs that create and delete directories at different levels.

- Hash Table: ASCII values of all characters are added to compute the key value for the Dir/File name. The key value is used as an index to place the file in a particular location. The children are added using a linked list implementation.
- Attributes (size & timestamps) For directories, size represents the number of immediate children.
- Permissions – read & write permissions.
- Simultaneous access – only one file can be opened at a time. This can be easily expanded to use a file table to allow an arbitrary number of files to be opened simultaneously.

Attachment A: This attachment contains documentation for some of the programs. The actual programs that we are also submitting have documentation in the form of comments incorporated into the programs.

CreateDir:

Creates a directory at the path specified by the user

Input: User specifies the directory name to be created along with the complete path.

Process:

FindParent() returns the parent of the directory by using the path specified by the user.

Failure Conditions**:**

If any of the below conditions is true, the operation fails and an appropriate error response is sent.

a) No path corresponding to the parent exists in the file system

b) Parent does not have write permission (since the new file cannot be created under it)

c) If another directory already exists in file system with the same name specified by the user.

Otherwise, the directory is created is created successfully at the specified path and linked to its parents. Appropriate attributes are set and SUCCESS code is sent back.

==========

DeleteDirectory:

Deletes a directory at the path specified by the user

Process:

FindNode() returns the particular node (directory) to be deleted.

Failure Conditions**:**

If any of the below conditions is true, the operation fails and an appropriate error response is sent.

a) Attempt to delete root directory.
b) User specified directory is not in our file system.
c) Node returned by FindNode() is a file. (DeleteDir is invalid on files)
d) The particular directory has subdirectories
e) Node is not writable.
f) Node's parent is not writable.

Otherwise, the directory is delinked from its parent and the attributes are set to appropriate values. A success code is sent back.

==========

CreateFile:

Creates a file at the path specified by the user

Input: User specifies the file name to be created along with the complete path.

Process:

FindParent() returns the parent of the file specified by the user.

Failure Conditions**:**

If any of the below conditions is true, the operation fails and an appropriate error response is sent.

a) No path corresponding to the parent exists in the file system

b) Parent does not have write permission (since the new file cannot be created under it)

Otherwise, hash function computes the key for the file name.

Appropriate location is indexed using the key value. If there are no children in the particular index, a new file will be created (Linked List Implementation)

If the indexed location has children already, the linked list is traversed to find a match for the file name specified by the user. On finding a match, a failure message is returned since there can't be files with duplicate names. If there is no match, new file entry is created at the particular index and the file is linked to the parent as necessary. Appropriate values for the attributes are set and success code is returned.

DeleteFile:

Deletes a file at the path specified by the user

Input: User specifies the name of the file to be deleted along with the complete path.

Process:

FindNode() returns the node corresponding to the file specified by user.

Failure Conditions:

If any of the below conditions is true, the delete operation fails and an appropriate error response is sent.

a) File/node is not present in the file system.

b) Node is a directory (DeleteFile becomes an invalid option for directories)

c) Node is not modifiable.

d) Delete a file which is in open status.

Otherwise, hash value for the file name is computed. Appropriate location is indexed to locate the file. The file is then delinked from its parent. The attributes are updated appropriately and a success code is returned

---

OpenFile:

Opens a file for either Read or Write operation

Input: User specifies the path of the file to be opened and the access mode (read/write)

Process:

If a file is already open, the operation fails & an appropriate error message is returned (Assumption: Only one file can be open at a time)

Otherwise, the file has to be located in the file system.

Failure Conditions:

If any of the below conditions is true, open operation fails and an appropriate error response is sent.

a) File doesn't exist in file system.

b) The required node is a directory instead of a file.
(DeleteFile is invalid for directories)

Otherwise file can be opened in one of the following access modes:

Open for Read: If the file is readable and is not open, then the file is successfully opened for reading and status is set to READ indicating that a read operation is in progress. A success message is sent.

Open for Write: If the file has writable and is not currently open, the file is opened successfully for writing status is set to WRITE indicating that a write operation is in progress. A success message is sent.

CloseFile:

Closes a file that is currently open in the file system

Input: User issues a close request to close an open file.

Process:

Failure Condition:

If any of the below conditions is true, close operation fails and an appropriate error response is sent.

If no file is in open status, the close option becomes invalid and an error response is returned.

Otherwise, the corresponding file is closed successfully.
The open status is set to "CLOSED" indicating that there are no files open in the system.
Also, open_file which points to file that is currently open is set to null.

ReadFile:

Reads the contents of file for the length specified by the user

Input: User specifies start position, number of bytes to be read and the buffer to place the data read.

Process:

Failure Condition:

If any of the below conditions is true, read operation fails and an appropriate error response is sent.

a) No open files in the file system
b) File is open for writing and does not have read permissions enabled.

Otherwise, the contents of he file are read for the required length and the data is placed in the buffer. A Success code is sent.

WriteFile:

Writes the contents of buffer into file

Input: User specifies the data buffer, start position and the length

Failure Condition:

If any of the below conditions is true, write operation fails and an appropriate error response is sent.

c) No open files in the file system
d) File is open for reading and does not have write permission enabled.

Otherwise, the contents of data buffer are written to the file and a success code is sent.

==========

Getinfo

Gets the information about specified file system object

Input: User specifies the name of the file to be shown its information.

Process:

Failure Condition:

If any of the below conditions is true, the operation fails and an appropriate error response is sent.

a) No path to the specified object (File/Directory)

Otherwise, the node information is returned successfully.

Rename

Renames a file that is currently open in the file system

Input: User specifies the path of the file to be renamed and a new filename.

Process:

Failure Condition:
If any of the below conditions is true, rename operation fails and an appropriate error response is sent.

a) No path exists
b) Root directory is the node which has to be renamed.
c) Parent does not have write permission.

Otherwise, the corresponding file is renamed successfully.

SetInfo

Sets the attributes of the node to new values specified by the user

Input: User specifies the name of the file whose attributes is to be set.

Process:

Failure Condition
No path to the object exists.

Otherwise, the required information is updated successfully to the node.

Contributions:

- CreateDirectory: Pratibha Rajendran/Mark Borinsky
- DeleteDirectory: Pratibha Rajendran/Mark Borinsky
- ListDirectory: Kyoungha Kim
- GetInfo: Kyoungha Kim
- Rename: Mansik Kim
- CreateFile: Pratibha Rajendran/Mark Borinsky
- DeleteFile: Pratibha Rajendran/Mark Borinsky
- OpenFile: Jaydutt Shukla
- CloseFile: Jaydutt Shukla
- ReadFile: Jaydutt Shukla
- WriteFile: Jaydutt Shukla
- Hash function & Linked List: Hojin Chang
- UI: Mansik Kim
- UI & backend integration: Jaydutt Shukla