Project 4: Extending a Filesystem
Design Document
5/28/2014
Matt Caruano, Taylor Caswell, Eric Evans, Mesuilame Mataitoga, Radhika Mitra

Purpose

The purpose of this project is to extend the MINIX file system to allocate a special metadata area for each file. This extra space can be used to store notes about the file that are separate from the normal file contents. Tools must also be designed and implemented to allow users convenient access to this metadata.

Design Plans

Overall implementation

At a high level, we will expand upon the existing MINIX file system by adding the ability to store metadata with every file. This will be done using file inodes, and additionally will only use existing inode components, thus maintaining backwards compatibility. Users will be able to read and write file metadata through the use of two provided C programs *metacat* and *metatag*.

Metadata internals

- Our system will allow users to store up to 1024 bytes of metadata for each file. What the user wishes to store here is at his/her own discretion.
- We will need to be able to keep track of a binary value to indicate if a given file has metadata or not. This will need to be kept track of it in the inode, together with the block number where the metadata will be stored.
    - We will keep track of the address of the metadata information block in a rarely-used "Zone 9" pointer. This pointer is only used for excessively large files, and should not interfere for the purposes of this project
    - A value of 0 in the Zone 9 pointer of the inode will be used to indicate that a file has no metadata.
- We will also be sure that the appropriate free() call gets made to the metadata of a file that has metadata at the time it is deleted from the filesystem.

Metadata interaction
- We will need to implement user space functions metadata_read() and metadata_write() so users can read and write file metadata. In order to accomplish this, we will have to implement m_read() and m_write() syscalls to VFS, which will in turn be called by our user space functions.
  - m_write() will allocate 1024 bytes before storing the data if the file currently has no metadata field
  - m_read() will return an error message if the file contains no metadata
  - The arguments to m_read() and m_write() will be XYZ
- This will require us to define a new message type from VFS → MFS, which we will define alongside the existing message types in */usr/include/minix/vfsif.h*
- Once the user-level functions have been defined, we will then write C programs *metacat* and *metatag*, which will call our user-level functions metadata_read() metadata_write(), respectively.

Testing
- Testing will be done incrementally throughout the project to ensure that functionality is working for all parts of the project up to the part currently being implemented.
  - A simple function to be called by the VFS that prints debug statements after handling our own syscall. This will ensure we are correctly making a syscall being caught by the VFS.
  - A couple of message sent from the VFS to the MFS. These will each handle a function that re-implements either the read() or write() calls, along with a debug() function. This will ensure that we are correctly message passing.
  - To the redundant write() call, include a call to allocate the 1024 bytes block for the meta if it is not already allocated, and do something with it to ensure that the block is properly allocated, such as copying characters there and then printing them out within a separate call. This will ensure that the allocated block for the metadata is properly being allocated.
  - Within the redundant read() call, add a code to read the data in the block, and print it out. This will ensure that the block that will contain the metadata is properly storing its data, and that it can be access by our own read() call.
- After the program is finished, we will make test scripts that demonstrate the required abilities of our program as layout out in the project 4 specifications.