Assignment 4
CMPS 111

Forrest Kerslager, Nick Noto, David Taylor, Kevin Yeap, Connie Yu

May 29, 2014

Purpose

The purpose of this assignment is to extend the MINIX filesystem to include a special metadata area for each file. This extra space can be used to store notes and information about the file that are separate from the normal file contents.

1 Design

1.1 Metadata

Metadata regions are attached to files that are accessed and edited using the new metaread() and metawrite() functions. This region will be a 1024 byte block that we must point to using a file's i-node structure. In order to maintain backwards compatibility we must repurpose unused fields in the i-node structure to signify the presence and location of a metadata region. In the i-node structure we will reuse two preexisting fields to keep track of a file's metadata. The sticky bit is set to 0 by default for all files. We will reuse the sticky bit by setting it to 1 if metadata is available for a file. Since the sticky bit is only used for directories this should not present any conflicts with the preexisting file system. If the sticky bit is 1 we will access the block number of the metadata in the i-node field i_zone[9]. Zone pointer 9 holds a triple indirect pointer which is rarely if ever used. Using i_zone[9] will only cause conflicts for the very large files that require it. We can check if i_zone[9] is unallocated and therefore available by comparing it with the macro NO_ZONE. A possbility to avoid conflict is to restrict files from being large enough to require zone[9].

1.2 System Call

New VFS system calls metaread(int fd, void *buf, size_t count) and metawrite(int fd, void *buf, size_t count) will be created that take a file descriptor, pointer to a buffer, and the number of bytes to read or write. This call resembles read() and write() except its purpose is to read or write to a file's metadata region. If a metadata block is not available or empty on a read nothing will be returned. If a metadata block is not available on a write it will create one and link the file to it. If a file is large enough to require use of i_zone[9] then metadata can not be used. This process is not done directly by VFS instead it will send a message to MFS to perform the actual work.

1.3 User Library

A user library allows a user program to invoke a system call. First a prototype of our function must be placed in a preexisting header in /usr/src/include/. If the declaration was put in a new header file then the makefile would have to be edited to include it. A source file must then be created in /usr/src/lib/libc/sys-minix/ and added to Makefile.inc in the same directory. The source file will access the system call int _syscall(int who, int syscallnr, register message *msgptr). Finally an unused call number must be assigned in /usr/src/include/minix/callnr.h.

1.4 VFS

In order to create a new system call handled by VFS we must modify a few files in /usr/src/servers/vfs. Any new system calls must be added to an unused entry in table.c. A prototype for the system call must then be added to proto.h. The functions themselves will be defined in their appropriate .c files. For the purpose of defining the new metaread() and metawrite() functions we will add definitions to read.c and write.c.

VFS does not directly handle the new system calls. VFS will use message passing to communicate with a FS, a specific instance of MFS, that can handle the system call. When it receives one of the new system calls it will use the file descriptor to look up a v-node. Using the v-node it can find the reference to the i-node in the correct FS. A message will then be built that requests a read or write to the metadata connected to that i-node.

1.5 Message Passing

Messages are used to allow communication between VFS and MFS. When VFS begins a metaread() or metawrite() it will send a message to the instance of MFS that handles the i-node it is interested in. A field to pass an i-node reference already exists. However, a message type must be added to /usr/include/minix/vfsif.h that signals a read or write to the metadata. The message type will act as an index for MFS to run our handler for metadata.

1.6 MFS

When MFS receives a message from VFS it will use the m_type field to index into its handler table and call the metaread or metawrite handler. As discussed in the Metadata section these handlers will use the repurposed fields in the i-node to identify if metadata exists and where the address of the metadata. If the region does not exist or is empty on a read it will return nothing. If the region does not exist on a write it will create one and link the i-node to it. If a read signal is received MFS will copy the contents of the metadata block into the buffer. If a write signal is received MFS will copy the contents of the buffer into the metadata region.

1.7 Other Effects

- When a file is deleted the metadata must also be deleted if it exists.
- When a metadata region is first created it must be zeroed out.

2 Implementation

This project has not yet been implemented.

3 Testing

Testing will be able to:

- Demonstrate compatibility with the existing MINIX filesystem (either by showing existing files not being corrupted for a change in MFS, or by showing your alternate filesystem mounted alonside MFS).
- Demonstrate adding a note “This file is awesome!” to a README.txt file, and later reading back the note.

- Demonstrate that changing the regular file contents does not change the extra metadata.
- Demonstrate that changing the metadata does not change the regular file contents.
- Demonstrate that copying a file with metadata copies the metadata.
- Demonstrate that changing the metadata on the original file does not modify the metadata of the copied file.
- Demonstrate that creating 1000 files, adding metadata to them, then deleting them does not decrease the free space on the filesystem.

For testing purposes the sticky bit can be toggled on a file using the command

```
chmod +t filename
```