CS314: Lab Report
Assignment 10 - Immediate Files

B Siddharth Prabhu
200010003@iitdh.ac.in

Devdatt N
200010012@iitdh.ac.in

27 March 2023

1 Problem Statement

In this lab, we implement immediate files in the Minix File System, for files of size up to 32 bytes. Implement only for the file system mounted at `/home`. We must consider the different functionalities required to implement such files.

- File creation: you can start by creating the file as an immediate one. When a file grows beyond 32B, then you can make it a regular file.
- File read: if it is an immediate file, you can respond with the inode structure contents. If not, you can follow the default behavior of looking up zones.
- File write: similar to read. You must take care to ensure that if you want to write to the inode structure, then the new file size is still within 32B. When a regular file shrinks to less than 32 bytes, there is no need to come back to immediate mode.
- File delete: deleting immediate files does not require any handling of zones.

2 Immediate Files: Background

Inodes contain information forming the metadata of files, such as last access time, last modified time, file size, and permissions, along with the pointers to the disk block where the data of a file is stored. These pointers either directly refer to a disk block, or they refer to a list of additional pointers to data blocks, in a single-indirect fashion. The problem with a regular file is that a complete disk block needs to be allocated even when it is very short, leading to wastage of disk space.

Immediate files allow for data to be stored directly in the inode instead of the disk. An inode in Minix3 is 64B long, and 40B are used to hold pointers to data blocks. When no data blocks are used, these 40B can be used to store the file content directly. Hence, for files of size upto 40B, immediate files can be used. Consequently, the number of disk accesses is reduced for short files, thus reducing the access time. Here, we deal with immediate files for size upto 32B.

3 Modifications Made

We have modified the following files in our implementation:

```
/minix/fs/mfs/read.c
/minix/fs/mfs/write.c
/minix/servers/vfs/link.c
/minix/servers/vfs/open.c
/minix/servers/vfs/read.c
/minix/include/minix/const.h
/minix/lib/libc/gen/fslib.c
```

4 Output Screenshots

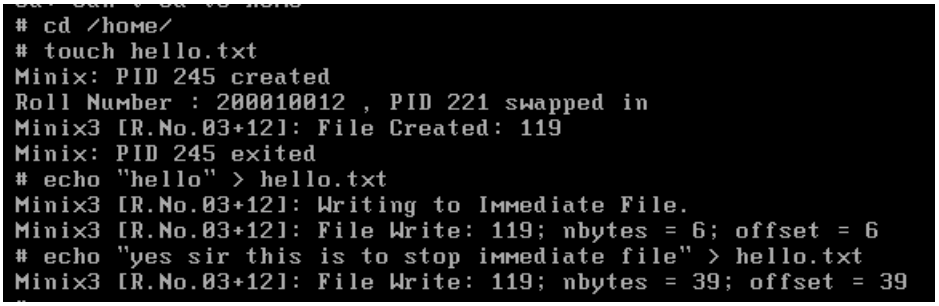

Figure 1: Output of File Operations

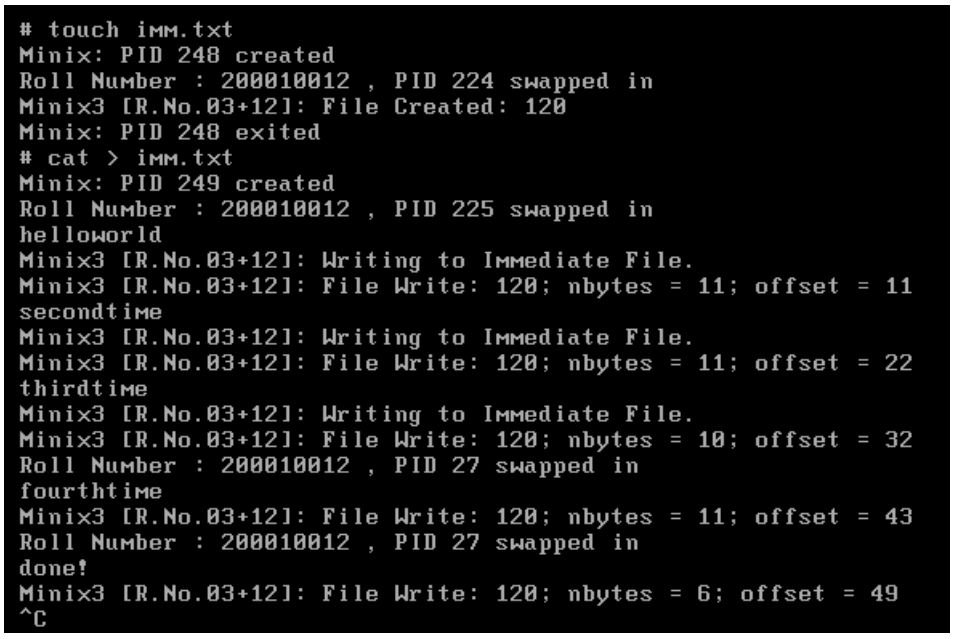

Figure 2: Output of File Operations

5 Other details

- Immediate files are implemented by setting a flag in the inode to indicate that the file is an immediate file and storing the data directly within the inode structure. The size of the data that can be stored in an immediate file is limited by the size of the inode, which is typically small and for us is 32 bytes, while the inode size is 40 bytes.
- Immediate files can improve performance for frequently-used data because they eliminate the need to read the data from disk. Instead, the file system can quickly retrieve the data from the inode and return it to the program. However, if the file is not frequently used, the overhead of storing the data in the inode may outweigh the performance benefits.
- One potential disadvantage of immediate files is that they can increase the size of the inode, which can lead to fragmentation of the file system. To address this issue, we can limit the number of immediate files that can be created, and provide a mechanism for converting immediate files to regular files if they exceed a certain size.