Zero-blocks optimisation in MFS

Uladzislau Sobal
the University of Warsaw
ws374078@students.mimuw.edu.pl

ABSTRACT

This paper introduces and explains a simple optimisation for the Minix File System (MFS) that improves efficiency of storage for blocks filled with zeroes.

1. INTRODUCTION

About Minix File System and ext. Both Minix File System and

Some files contain a lot of zeroes, so a natural question to ask would be if we can be more efficient around these zeroes. To answer this question, let's get our hands dirty with the workings of MFS (and ext, since it's similar).

2. MINIX FILE SYSTEM

Both MFS and ext use i-nodes. I-nodes is a structure used to store data. It's basically an array of pointers. The first 12 pointers are direct pointers to data. The next 256 pointers are pointers to pointers to data (indirect blocks). After that there are doubly indirect blocks (pointers to pointers to pointers) and trebly indirect blocks (pointers to pinters to pointers to pointers). What's interesting is if we are trying to read one of these blocks and it's no initialised, the system just acts as if it was filled with zeroes. You can test this behaviour by creating a file, and dd'ing into it some random numbers with huge seek. Something like: dd if=/dev/random bs=512 count=1 of=somefile seek=100000000 You will get a huge file that isn't even supposed to fit on the hard drive.

However, the system doesn't do anything of the kind when writing zeroes, it simply initialises a block and writes zeros. This is what I changed.

3. CHANGING MFS SOURCE CODE

To test the optimisation I altered standard MFS implementation. The file that I changed was /usr/src/minix/fs/mfs/read.c. There is a huge multi-purpose function called rw_chunk. I altered so that it first checks the data it is about to write, and, if the data is all zeroes, doesn't create the block.

SYSTOR '17 Haifa, Israel
© 2017 ACM. ISBN 123-4567-24-567/08/06.
DOI: 10.475/123_4

4. OPTIMISATION RESULTS

To check how much we gain from this optimisation, I did 3 tests on different types of data - code, books and movies. I copied to a clean mfs and the patched one a folder containing code in different languages, a folder containing pdf, epub and doc books and documents, and a folder containing several episodes of a TV-series. Here are the results:

5. CONCLUSION

As you can see from the results, the optimisation is not always useful. It does improve space usage a little bit, but it also takes more time, and the tradeoff is not good enough to include the patch into the standard implementation. However, if you have a special case where data has a lot of zeroes, and time is not critical for you, you could use a patch that writes zeroes more effectively. Since the data written in this way can be read by the unpatched version of the file system, the most sensible way to use the optimisation is to estimate how much you gain and run the patched write only when needed.