Lab 8 Building Your Own Filesystem Report

-- 140050002, 140050007

How to run patch :

1. Extract the compressed fuse-tutorial.tgz. This will create a folder "fuse-tutorial-2016-03-25" ; cd into this folder. Copy the patch file "FUSE.patch" into this folder.
2. Run "$ patch -p1 < FUSE.patch"
3. Now run "./configure" and then "make" to build the executables.
4. Now cd to the folder "example". Here running "$ ../src/bbfs rootdir mountdir" will mount the mountdir and the corresponding version of it will be seen in rootdir.
5. Also "$ fusermount -u mountdir" will unmount the mounted directory.

Explanation for implementation of deduplication :

We create a directory "**DataStore**" which actually contains the data blocks of size 4KB in our case. These block are named based on their corresponding hash representation(SHA1) of the content. Note that we have #include <openssl/sha.h> in our code so it needs to be present as we use the SHA library to generate the 20 B hashcode.

In bb_write, we divide the buffer which we get into parts of 4KB and for each segment we generate its hash code. If the hash code exists in DataStore directory we do not need to store this segment as it already exists. If the hash code is not present we create a file whose **name is the hash code** and its content is the data block inside the DataStore. The actual file just contains the sequence of hash codes in it.

Thus repeated data blocks are not repeatedly stored and deduplication is established.

While reading we read the corresponding hash codes required based on the offset value and the size to be read from the hash file. We get the data block corresponding to every hash string required and concatenate over it to finally return the complete string in **buf** and the buf size.

Testing our code :

After mounting **mountdir**, we first drag and drop "file1.txt" from example(folder/directory) on to "mountdir". This will generate the "*DataStore*" directory and we will also see the file1.txt in both folders viz. *mountdir* as well as *rootdir*.

“file1.txt” is made up of 4 data blocks (16KB). Opening the **rootdir** we will see 4 files in *Datastore* and if we open file1.txt we will see that it has only 80 characters. These are the hash strings corresponding to the 4 blocks of data. Also opening the hash named data blocks in *DataStore* we will see actual chunks of data.

The c++ code below makes a read request for *file1.txt*:

```
-------------------------------------------------------------------------------------------------------------------
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>

#define CHUNK 4096

using namespace std;

int main()
{

        int someFD = open("MAIN_PATH_LOCATION/example/mountdir/file1.txt", O_RDWR);
        char buf[CHUNK];
        pread(someFD, buf, CHUNK, CHUNK);

        return 0;
}
-------------------------------------------------------------------------------------------------------------------
```

Replace MAIN_PATH_LOCATION with the path of the folder of patch.
After execution of this code, open the **bbfs.log** file in *example* and search for the following “The block data :”. This shows the number of bytes read and the actual data block of characters in the **buf** variable, which was read. The code requests for the second chunk of data, as the size of request(pread) is 4KB as well as the offset is also 4KB
The **bbfs-sample.log** is an example to show the results obtained and to check the data block read is correct.
Thus we show that we can read correctly from file hash values.

Next we drag and drop “**file2.txt**” on *mountdir*. “*file2.txt”* differs from “*file1.txt*” only at the last line. Hence the first three data blocks already exists and only one new data block is added in *DataStore* directory. This ensures that our logic for *deduplication* is working correctly and we store no repeated blocks of data.

Major changes in code are in *bb_read and bb_write functions in bbfs.c*. Other definitions are present in *params.h*. *Makefile.in* is modified to implement -lcrypto and some other minor changes in another Makefile. Files of file1.txt and file2.txt is present for testing.