Middle East Technical University   Department of Computer Engineering

CENG 334

Introduction to Operating Systems

Spring 2015-2016

HW3

Due date: 12th June 2016, 23:55

1 Objectives

This assignment aims to get you familiar with file system structures by implementing a file system with FUSE that uses GDBM as a backend.

2 Specifications

FUSE is an acronym for File system in USErspace, which let non privileged users to implement file systems. With FUSE, implementation of the file system resides in the user space, while kernel module of FUSE provides a bridge for the actual privileged operation access. Figure 1 shows the flow of a file system request on a FUSE system. In this figure what you will implement is the *./hello* application.

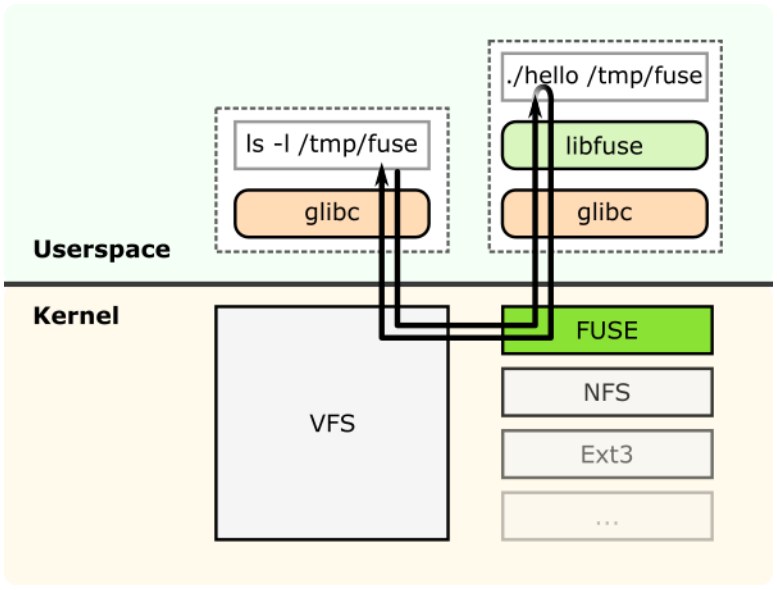

Figure 1: FUSE structure

In this assignment actual storage will be implemented on GDBM. GDBM is the GNU database manager. It stores key-data pairs in hashed files. API provides functions for searching/retrieving/deleting key-data pairs and functions for sequential access.

Table 1: Structures for three kinds of database entries

Regular File

Key	Path	
Value	R:<sizeint>:<userint,groupint>:<permissions>:<atime,mtime,ctime>	data

Directory

Key	Path	
Value	D:<sizeint>:<userint,groupint>:<permissions>:<atime,mtime,ctime>	names

Symlink

Key	Path	
Value	L:<sizeint>:<userint,groupint>:<permissions>:<atime,mtime,ctime>	actual path

2.1 GDBM Structure

Your implementation is expected to hold files, folders and symbolic links. Time, owner and group permissions are also have to be recorded. Table 1 is the structure of the key-data pairs that will be kept with GDBM. Path of the file, directory or symlink is the key for each entry. First char of the value is the type of the file, other meta data follows this fixed length header part. Rest of the value part is file content for regular file entries, names of files and directories for directory entries and the actual path for symlink entries.

Both key and values are kept as a `datum` struct which consists of a char pointer `dptr` which points to the data (key or value) and an `dsize` which recods the size of the array pointed by `dptr`. GDBM dcumentation can be accessed with this link (http://www.gnu.org.ua/software/gdbm/). Following is a small code snipped that shows GDBM usage.

```c
#include <stdio.h>
#include <string.h>
#include <gdbm.h>

main (int argc, char *argv[])
{
  GDBM_FILE dbf;
  dbf = gdbm_open ("dbFile", 0, GDBM_WRCREAT, 0666, 0);
  if (dbf){
    datum key = { "aKey", 5 };
    datum value = { "aValue", 7 }; // with terminating null

    data = gdbm_fetch (dbf, key);

    if (data.dsize > 0) {
      printf ("%s\n", data.dptr);
      free (data.dptr);
    } else {
      gdbm_store (dbf, key, value, GDBM_REPLACE); //  GDBM_INSERT returns
                                                  //      error if key exists
    }
      gdbm_close (file);
  }
}
```

2.2 FUSE Functions

Following list of FUSE functions should be implemented. For the description of each of these functions you should look at the FUSE documentation (http://libfuse.github.io/doxygen/structfuse__operations.html).

Operations: Core functions: init, destroy, getattr, access, readdir, mknod, mkdir, open, read, write. Secondary functions: symlink, chmod, chown, utime, opendir, readlink.

When your filesystem is mounted as `hw3 dbFile /mountpoint`, the user should be able to create,update,inspect and delete directories and files under this filesystem as if it was a standard system directory. You can download a skeleton file for your implementation COW.

2.3 Implementation, Compilation & Execution Details

You will need `libfuse-dev` and `libgdbm-dev` in your development system and call the compiler with `-lgdbm` and `-lfuse` flags for libraries. `-D_FILE_OFFSET_BITS=64` flag is also needed. Following `struct` is given for the value part of your GDBM entries. Although it can also be recorded in `mode_t`, first field (`char type`) of your entries have to be 'R' for regular files, 'D' for directories and 'L' for symlinks. Last field (`char content[0]`) is there to define a variable size array. For your entries, you can allocate memory similar to the example `malloc()` call given below. This way, `content` can point to contents of your file.

```
typedef struct filecontent {
   char type;
   off_t size;
   uid_t user;
   gid_t group;
   mode_t permissions;
   time_t atime;
   time_t mtime;
   time_t ctime;
   char content[0];
}filecontent;

filecontent *ptr = (filecontent*)malloc(sizeof(filecontent) + fileSize);
```

Pay attention to the directory entries. They have to be updated when a new directory or file is created under them. Also, when your system's first run, if/when there is no entry in the database, you should create an entry for "/".

Your code will be executed with `./hw3 dbfile mountPoint [options]`. FUSE library handles the optional arguments given to your executable. You can see the example usage in the skeleton file that you can download from COW. Options you should use are: `-d` to enable debugging, `-f` to run your client in foreground and `-s` to run in single threaded mode. Running multi-threaded may cause race conditions.

3 Tips & Hints

- Start with this 'hello world' implementation.
- Look at this tutorial. Compile and run the given example file system. It logs the calls to the FUSE functions. You can get an understanding of which functions are called for which console commands.
- While writing values to GDBM, size field (`dsize`) should include the null terminator for strings and `strlen()` does not count the null terminator.
- Use `-s` options to see `printf()` outputs.
- `memcpy()` can copy everything, maybe you do not need parsing.

- You can use any library, FUSE is in user space.
- You can use this small program's interactive session to look into your GDBM files.
- You should call `gdbm_reorganize()` in `init()` or `destroy()` operations of FUSE. If your DB file gets bigger than you expect, it is because GDBM does not really remove records unless it is forced to.
- Office hours will be on Thursdays 14:00-16:30.

4 Regulations

1. Your code have to be in C/C++.
2. Submission will be done via COW. Create a tar.gz file named *hw3.tar.gz* that contains all your source code files and a makefile. The executable should be named *hw3*. If your makefile does not work (or you do not have one) and therefore we will have to manually try to compile your code - there will be an automatic penalty of 5 points.
3. Your codes will be evaluated with a black-box approach.
4. Please ask your questions related to the homework on cow instead of email. Your friends, who may face with same problem, can see your questions and answers.
5. Will be graded over 110 with 10 points bonus.
    - Core functions: 80 points.
    - timestamps: 10 points
    - symbolic link: 10 points
    - owner,group and permissions: 10 points
6. Do not cheat.