CSE 30341: Home Work Project 4

Assigned: Mar 23, 2007 and Due: Apr 11, 10:40AM

Late submissions will not be accepted. Group effort

File system using fuse:

In this project, we will implement a simple file system. Instead of modifying the hard disk directly, we will implement the file system inside a single file ("disk"). We will also use the user level fie system tool called *fuse* to interact with our "disk". *Fuse* lets you intercept all the system calls issued to a file system and allows you to implement them using your own code in user space (unlike kernel file systems that live inside the kernel). Of course, *fuse* is not as fast as kernel level file systems. On the other hand, *fuse* lets us build our own file systems without worrying about crashing the kernel etc. Next we describe each of these steps in detail

1. **Step 1: Develop a file system:**

First, we implement the file system inside a file. Let us call this file *disk* for the rest of this discussion. Create a large file of size 512 MB to hold this *disk*. Logically split this *disk* into 512 byte blocks. Now, implement two functions to operate on this *disk:* write_disk() and read_disk():

- write_disk(int blockno, void *data): this function will seek to blockno and write the data. The data is always 512 bytes long. The actual seek location starts at 0 and proceeds in multiples of 512 (i.e., block 0, 1, 2 … start at 0, 512, 1024, …. bytes respectively). The function will return a -1 for error conditions such as "Invalid block".
- read_disk(int blockno, void *data): this function will seek to the blockno and read 512 bytes into data. This function will return error code of -1 for conditions such as "Invalid block".

For the rest of the project, you are only allowed to operate on *disk* using these two operations. This means that, if you want to write one byte at location 10 (say), you need to read block 0, write the value at location 10 of this read buffer and then write this entire buffer back at block 0).

2. **Step 2: Initialize the *disk:***

On the newly created *disk*, add any meta-data that you require. You would likely need to first create entries for freeblocks as well as for directories. Typical file systems also maintain some headers to

help identify the disk. They also allow users to partition the disk into multiple independent areas. For simplicity, you can assume a single level directory. Document any restrictions on the size of directory entries. You should also keep track of free blocks by storing this information in the disk. You will store all these information in the disk using the functions read_disk() and write_disk(). You will create the directory information by getting free blocks, potentially updating the free block information. When new files are created, you need to update the freeblocks table, update the directory entry, etc.

3. **Step 3: Interact with this file system using *fuse*:**

Fuse intercepts the interactions with the file system and redirects them to your file system described in Step 1. *Fuse* is available on the web at http://fuse.sourceforge.net/. I have downloaded the *fuse* distribution: it is available on the expsys machines at ~surendar/fuse-2.6.3. There are many examples in the example/ directory. Start by looking at hello.c, a program that creates a file system that contains a file called hello. If you read this file, you will always get the contents "Hello World!". More information about *fuse* will be provided by the TA on Mar 30th.

4. **Step 4: Evaluation:**

You will evaluate your file system by reporting the time taken (repeat your experiments to draw statistically valid results) to create a new file of sizes, 10 KB, 100 KB, 1 MB and 10 MB. You will also report the time taken to read each of these files that are described above.

For your report, clearly describe the layout of your disk, how and where you store the freeblock information, how and where you store the directory entries and how and where you store the actual files.

5. **Extra credit opportunities:**

You can earn up to an extra 100% by building a more sophisticated file system. For example, you might support hierarchical directory structure. You can also describe your own ideas for a *cool* file system. Discuss any such ideas with the instructor before proceeding for extra credit.