# Assignment-4: Filesystem

## Description

**Prerequisites:** This assignment is to be done on FUSE (file system in user space) framework. You should use a Linux system with FUSE packages installed. To install the FUSE packages on Ubuntu, use the following commands.

```
$sudo apt-get install fuse
$sudo apt-get install libfuse-dev
```

Ensure that your system has at least 10GB free hard disk space. Play around with the attached source (along with a simple implementation in *example* folder). Read the README files to understand the usage.
Objectives of the assignment is to implement an object store with key-value semantics (with hidden file semantics) using the FUSE APIs. As part of the assignment, you are required to build an object store on the disk (emulated using `disk.img` file), provide concurrent access mechanisms to the object store and implement caching. The assignment is split into two parts where `Part B` is a superset of `Part A`. Design the solution for high performance and scalability. During evaluation, we may use disk of size up to 32 GB storing up to $10^6$ objects in the store. Assume that, maximum size of the key is 32 bytes and maximum object size is 16MB.

### Part A - The object store

In this part of the assignment, you are required to implement the object store functionalities. The template file (`objstore.c`) contains all the function definitions that you are required to implement. To understand the exact semantics, please refer to the template code and the example implementation. Below listed are some important points regarding the design constructs.

- A generic structure representing the file system (`struct objfs_state`) is declared in `lib.h`. An instance of this structure is passed to all the object store functions. One of the members of this function is a generic pointer (`objstore_data`). You can declare and initialize your own structures and assign the `objstore_data` pointer during file system mount (`objstore_init`) and cleanup during unmount (`objstore_destroy`). *Do not modify the existing structures and declarations*
- You are required to maintain a unique object ID (integer) for all the objects. Object ID 0, 1 are reserved. Object IDs are useful when reading/writing from the objects.
- Read operations read all the contents of an object and write operations overwrite the content and update the size. Note that, the requested read size can be more than the object size. In such a case read should read the contents into the buffer and return the requested size.
- You can access (read from and write to) the block device using the following functions,

```
extern int read_block(struct objfs_state *objfs, long block_offset, char *buf);
extern int write_block(struct objfs_state *objfs, long block_offset, char *buf);
```

Note that, the block offset is in multiples of 4KB (Block size of the FS). The `buf` parameter address must be 4KB aligned (see the example).

- The object store must work correctly for concurrent access scenarios. The only assured scenario is that the same object will not be concurrently accessed by multiple threads/processes.
- You can use any data structures of your choice. Make sure that, the in-memory data structures used are space efficient and are scalable.
- Use `dprintf` and check the logs in `objstore.log` file. Normal `printf` does not work.

## Part B - The object store with caching

As part of this assignment, you are required to use a cache to improve the application throughput by employing some caching scheme. The cache size is 128MB (already initialized) and accessible through the `struct objfs_state` instance (member `cache`). The cache is size is in bytes and cache is 4KB aligned. Your cache implementation should get enabled by a compile time flag `CACHE` (adding `-DCACHE` option in the `Makefile`). Refer to the `example` directory as an illustrative implementation.

## Testing and submission guidelines

All the filesystem operations are logged into the `objfs.log` file by default. While testing for performance, you may build the code by removing -DDBG CFLAG from the `Makefile`. Please check the README file inside the `test` directory. You should build your own test cases in the `test` directory apart from the existing test cases. If you want to reset the disk, recreate the disk following the procedures mentioned in the README.

- You are required to submit only `objstore.c` file. So, do not modify any other code/definitions.
- Remove/comment all print statements used for debugging.
- Your implementation will be tested with several test cases and hence should be generic.
- Write your code in a modular manner, rationalize your in-memory structures. Marks will be deducted for designs that do not scale.
- Please refer to the code comments before asking doubts regarding assignment.