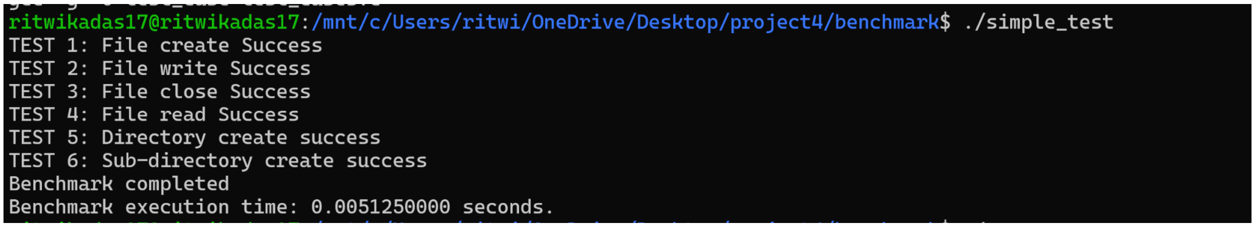
Names: Ritwika Das - rd935
Siya Vyas - sv694

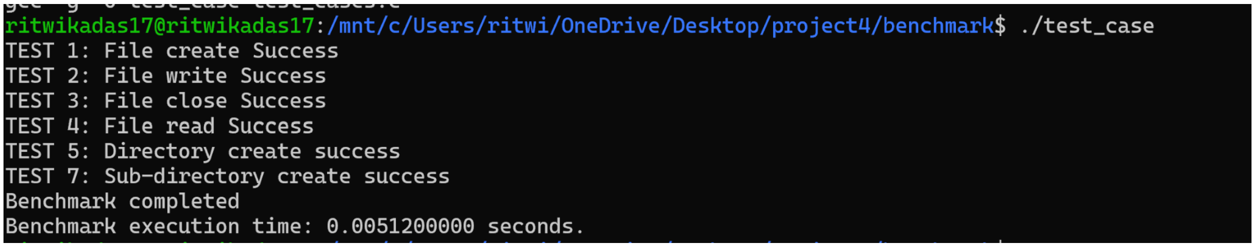
Report on RUFS Implementation

Total Number of Blocks Used and Benchmark Execution Time:

During the execution of the sample benchmark provided (sample_test.c and test_cases.c), the RUFS file system utilized a total of 130084 blocks. The benchmark execution time was approximately 0.005125 and 0.00512 seconds for both test cases.



```
ritwikadas17@ritwikadas17:/mnt/c/Users/ritwi/OneDrive/Desktop/project4/benchmark$ ./simple_test
TEST 1: File create Success
TEST 2: File write Success
TEST 3: File close Success
TEST 4: File read Success
TEST 5: Directory create success
TEST 6: Sub-directory create success
Benchmark completed
Benchmark execution time: 0.0051250000 seconds.
```



```
ritwikadas17@ritwikadas17:/mnt/c/Users/ritwi/OneDrive/Desktop/project4/benchmark$ ./test_case
TEST 1: File create Success
TEST 2: File write Success
TEST 3: File close Success
TEST 4: File read Success
TEST 5: Directory create success
TEST 7: Sub-directory create success
Benchmark completed
Benchmark execution time: 0.0051200000 seconds.
```

Implementation Overview:

The implementation of RUFS followed the provided skeleton code and project specifications closely. Key components of the implementation include:

- Initialization and Destruction: Implemented rufs_init and rufs_destroy functions to initialize and clean up RUFS state, respectively. Used dev_init and dev_close functions to manage the virtual disk file.
- File System Operations: Implemented essential file system operations such as rufs_getattr, rufs_opendir, rufs_readdir, rufs_mkdir, rufs_create, rufs_open, rufs_read, and rufs_write according to project specifications. Ensured proper error handling and validation of input parameters.
- Internal Data Structures and Operations: Implemented helper functions for block I/O operations (bio_read, bio_write), bitmap operations (set_bitmap, unset_bitmap, get_bitmap), inode operations (readi, writei), and directory operations (dir_find, dir_add, get_node_by_path).

Additional Steps for Compilation:

Check the configuration of TESTDIR in each benchmark to point your file system's mount point. We have our netid's in each benchmark, which would need to be changed.

Difficulties and Issues Faced:

- Understanding FUSE: Initially, understanding the FUSE library and its integration with RUFS posed a challenge. However, referring to FUSE documentation and tutorials provided valuable insights.
- Debugging: Debugging the file system logic, especially regarding directory operations and block management, required thorough testing and logging.
- Mounting: It took several attempts to figure out why the test cases were failing. It took several attempts to mount and remount, until it finally worked.

Collaboration and References:

During the implementation process, the following resources were consulted:

- FUSE library API documentation provided in the project resources.
- Online tutorials on FUSE-based file system development.
- Collaborated with project partners to discuss design decisions and troubleshoot issues.
- Utilized Stack Overflow for specific technical questions related to system-level programming.
- Writing a FUSE Filesystem: a Tutorial (nmsu.edu)
  - To get a general idea of where to start with implementing the FUSE-based system.
- Writing a Simple Filesystem Using FUSE in C (maastaar.net)
  - Also to have a general concept of how to write each function in the project, and to get a little more detail as to what each function does.