**File System**

**Submission:**

* Deadline: Wednesday, November 29, 2023, 8:00 pm HKT.
* Answer ALL questions. Add additional pages if necessary.
* Submit this answer sheet via Canvas->Assignments->Tutorials->Tutorial 8.

**Questions**

1. Run the simulator with some different random seeds (say 17, 18, 19, 20), and see if you can ﬁgure out which operations must have taken place between each state change.

**Answer:**

**With python2 vsfs.py -n 6 –s 17, operations should be as followings:**

1. **mkdir("/u");**
2. **create("/a");**
3. **unlink("/a");**
4. **mkdir("/z");**
5. **mkdir("/s");**
6. **create("/z/x");**

**With python2 vsfs.py -n 6 –s 18, operations should be as followings:**

1. **mkdir("/f");**
2. **create("/s");**
3. **mkdir("/h");**
4. **fd=open("/s", O\_WRONLY|O\_APPEND); write(fd, buf, BLOCKSIZE); close(fd);**
5. **create("/f/o");**
6. **create("/c");**

**With python2 vsfs.py -n 6 –s 19, operations should be as followings:**

1. **create("/k");**
2. **creat("/g");**
3. **fd=open("/k", O\_WRONLY|O\_APPEND); write(fd, buf, BLOCKSIZE); close(fd);**
4. **link("/k", "/b");**
5. **link("/b", "/t");**
6. **unlink("/k");**

**With python2 vsfs.py -n 6 –s 20, operations should be as followings:**

1. **create("/x");**
2. **fd=open("/x", O\_WRONLY|O\_APPEND); write(fd, buf, BLOCKSIZE); close(fd);**
3. **create("/k");**
4. **create("/y");**
5. **unlink("/x");**
6. **unlink("/y");**
7. Now do the same, using different random seeds (say 21, 22, 23, 24), except run with the -r ﬂag, thus making you guess the state change while being shown the operation. What can you conclude about the inode and data-block allocation algorithms, in terms of which blocks they prefer to allocate?

**Answer:**

**With python2 vsfs.py -n 6 –s 21 -r, the state should changed to:**

1. **mkdir("/o");**

**inode bitmap 11000000**

**inodes [d a:0 r:3] [d a:1 r:2] [] [] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (o,1)] [(.,1) (..,0)] [] [] [] [] [] []**

1. **create("/b");**

**inode bitmap 11100000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:-1 r:1] [] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (o,1) (b,2)] [(.,1) (..,0)] [] [] [] [] [] []**

1. **create("/o/q");**

**inode bitmap 11110000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:-1 r:1] [f a:-1 r:1] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (o,1) (b,2)] [(.,1) (..,0) (q,3)] [] [] [] [] [] []**

1. **fd=open("/b", O\_WRONLY|O\_APPEND); write(fd, buf, BLOCKSIZE); close(fd);**

**inode bitmap 11110000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:2 r:1] [f a:-1 r:1] [] [] [] []**

**data bitmap 11100000**

**data [(.,0) (..,0) (o,1) (b,2)] [(.,1) (..,0) (q,3)] [m] [] [] [] [] []**

1. **fd=open("/o/q", O\_WRONLY|O\_APPEND); write(fd, buf, BLOCKSIZE); close(fd);**

**inode bitmap 11110000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:2 r:1] [f a:3 r:1] [] [] [] []**

**data bitmap 11110000**

**data [(.,0) (..,0) (o,1) (b,2)] [(.,1) (..,0) (q,3)] [m] [j] [] [] [] []**

1. **create("/o/j");**

**inode bitmap 11111000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:2 r:1] [f a:3 r:1] [f a:-1 r:1] [] [] []**

**data bitmap 11110000**

**data [(.,0) (..,0) (o,1) (b,2)] [(.,1) (..,0) (q,3) (j,4)] [m] [j] [] [] [] []**

**With python2 vsfs.py -n 6 –s 22 -r, the state should changed to:**

1. **create("/z");**

**inode bitmap 11000000**

**inodes [d a:0 r:2] [f a:-1 r:1] [] [] [] [] [] []**

**data bitmap 10000000**

**data [(.,0) (..,0) (z,1)] [] [] [] [] [] [] []**

1. **fd=open("/z", O\_WRONLY|O\_APPEND); write(fd, buf, BLOCKSIZE); close(fd);**

**inode bitmap 11000000**

**inodes [d a:0 r:2] [f a:1 r:1] [] [] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (z,1)] [q] [] [] [] [] [] []**

1. **unlink("/z");**

**inode bitmap 10000000**

**inodes [d a:0 r:2] [] [] [] [] [] [] []**

**data bitmap 10000000**

**data [(.,0) (..,0)] [] [] [] [] [] [] []**

1. **create("/y");**

**inode bitmap 11000000**

**inodes [d a:0 r:2] [f a:-1 r:1] [] [] [] [] [] []**

**data bitmap 10000000**

**data [(.,0) (..,0) (y,1)] [] [] [] [] [] [] []**

1. **link("/y", "/s");**

**inode bitmap 11000000**

**inodes [d a:0 r:2] [f a:-1 r:2] [] [] [] [] [] []**

**data bitmap 10000000**

**data [(.,0) (..,0) (y,1) (s,1)] [] [] [] [] [] [] []**

1. **create("/e");**

**inode bitmap 11100000**

**inodes [d a:0 r:2] [f a:-1 r:2] [f a:-1 r:1] [] [] [] [] []**

**data bitmap 10000000**

**data [(.,0) (..,0) (y,1) (s,1) (e,2)] [] [] [] [] [] [] []**

**With python2 vsfs.py -n 6 –s 23 -r, the state should changed to:**

1. **mkdir("/c");**

**inode bitmap 11000000**

**inodes [d a:0 r:3] [d a:1 r:2] [] [] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (c,1)] [(.,1) (..,0)] [] [] [] [] [] []**

1. **create("/c/t");**

**inode bitmap 11100000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:-1 r:1] [] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (c,1)] [(.,1) (..,0) (t,2)] [] [] [] [] [] []**

1. **unlink("/c/t");**

**inode bitmap 11000000**

**inodes [d a:0 r:3] [d a:1 r:2] [] [] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (c,1)] [(.,1) (..,0)] [] [] [] [] [] []**

1. **create("/c/q");**

**inode bitmap 11100000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:-1 r:1] [] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (c,1)] [(.,1) (..,0) (q,2)] [] [] [] [] [] []**

1. **create("/c/j");**

**inode bitmap 11110000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:-1 r:1] [f a:-1 r:1] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (c,1)] [(.,1) (..,0) (q,2) (j,3)] [] [] [] [] [] []**

1. **link("/c/q", "/c/h");**

**inode bitmap 11110000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:-1 r:2] [f a:-1 r:1] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (c,1)] [(.,1) (..,0) (q,2) (j,3) (h,2)] [] [] [] [] [] []**

**With python2 vsfs.py -n 6 –s 24 -r, the state should changed to:**

1. **mkdir("/z");**

**inode bitmap 11000000**

**inodes [d a:0 r:3] [d a:1 r:2] [] [] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (z,1)] [(.,1) (..,0)] [] [] [] [] [] []**

1. **create("/z/t");**

**inode bitmap 11100000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:-1 r:1] [] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (z,1)] [(.,1) (..,0) (t,2)] [] [] [] [] [] []**

1. **create("/z/z");**

**inode bitmap 11110000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:-1 r:1] [f a:-1 r:1] [] [] [] []**

**data bitmap 11000000**

**data [(.,0) (..,0) (z,1)] [(.,1) (..,0) (t,2) (z,3)] [] [] [] [] [] []**

1. **fd=open("/z/z", O\_WRONLY|O\_APPEND); write(fd, buf, BLOCKSIZE); close(fd);**

**inode bitmap 11110000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:-1 r:1] [f a:2 r:1] [] [] [] []**

**data bitmap 11100000**

**data [(.,0) (..,0) (z,1)] [(.,1) (..,0) (t,2) (z,3)] [y] [] [] [] [] []**

1. **create("/y");**

**inode bitmap 11111000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:-1 r:1] [f a:2 r:1] [f a:-1 r:1] [] [] []**

**data bitmap 11100000**

**data [(.,0) (..,0) (z,1) (y,4)] [(.,1) (..,0) (t,2) (z,3)] [y] [] [] [] [] []**

1. **fd=open("/y", O\_WRONLY|O\_APPEND); write(fd, buf, BLOCKSIZE); close(fd);**

**inode bitmap 11111000**

**inodes [d a:0 r:3] [d a:1 r:2] [f a:-1 r:1] [f a:2 r:1] [f a:3 r:1] [] [] []**

**data bitmap 11110000**

**data [(.,0) (..,0) (z,1) (y,4)] [(.,1) (..,0) (t,2) (z,3)] [y] [v] [] [] [] []**

**As a result, the file system simulation used a simplified approach to inode and data block allocation. In this simulation, inodes (representing files or directories) and data blocks are tracked using bitmaps, where each bit corresponds to the availability status of a resource. In the allocation process, a linear search through the bitmaps is performed to find the first available inode or data block, marking it as allocated. Unlike traditional file systems that may use linked structures or index blocks, this simulation relies on a straightforward bitmap-based mechanism for tracking availability. The code does not explicitly use pointers between blocks or dedicated index blocks, making it a simplified representation of file system allocation strategies.**

1. Now reduce the number of data blocks in the ﬁle system, to very low numbers (say two), and run the simulator for a hundred or so requests. What types of ﬁles end up in the ﬁle system in this highly constrained layout? What types of operations would fail?

**Answer:**

**With python2 vsfs.py -d 2 -n 100**

**I conducted simulations with a highly constrained file system, reducing the number of data blocks to two. The results revealed that only regular files (denoted as 'f' in the context of file creation operations) were able to exist in this layout. Notably, operations such as mkdir(), open(), write(), and close() failed under these conditions. However, create(), link(), and unlink() operations remained functional. This suggests a limitation in supporting directory-related operations and emphasizes the system's focus on basic file creation and manipulation in such a highly restricted environment.**

1. Now do the same, but with inodes. With very few inodes, what types of operations can succeed? Which will usually fail? What is the ﬁnal state of the ﬁle system likely to be?

**Answer:**

**With python2 vsfs.py -i 2 -n 100**

**In a file system with very few inodes, the investigation revealed a significant limitation — all file system operations, regardless of type, failed under these conditions. This suggests that the scarcity of inodes severely restricts the system's ability to handle file-related actions. Consequently, the final state of the file system is likely to be incomplete and inconsistent due to the inability to successfully execute essential operations such as file creation, deletion, and modification. As a result, all operations would fail.**