CSE 221: Homework 3

Hao-en Sung [A53204772] (wrangle1005@gmail.com)
Department of Computer Science
University of California, San Diego
San Diego, CA 92092

1 Scheduler Activations

1.1 Why is this not a concern with standard kernel threads, i.e., why do scheduler activations have to worry about this deadlock issue, but standard kernel threads implementations do not have to?

Scheduler activations implement user-level threads with the same services integration as kernel threads. According to the paper, since the kernel threads have no information about the user threads, the deadlock happens when a user thread, which holds the ready list lock, is placed into a ready list by kernel thread.

On the other hand, standard kernel threads implementation will not face this issue, because the information are all kept in kernel space.

2 LFS

2.1 Explain what are the assumptions about hardware and access patterns that LFS depends on?

Some assumptions include:

- Processor speeds and main memory sizes increase at exponential rate; disk capacity improves rapidly; while, disk access times have evolved much more slowly.
- Caches can capture most of the read accesses. Thus, disk traffic will be dominated by writes.
- Due to the increasing of memory size, cache size also increases, which can be regarded as a buffer to accumulate write requests.
- Workloads are mainly dominated by accesses to small files, or said many random disk accesses.
- Time for file creation and deletion is dominated by directoy and Inodes updates.

2.2 Explain which mechanisms in LFS depend on each assumption?

The mechanisms provided in LFS can be categorized into three groups — data reading, data writing, and segment cleaning.

2.2.1 Data Reading

This mechanism mainly based on second assumption.

To read data, LFS will first refer Inode map to get the location of corresponding Inode. Later, one can use that Inode to retrieve data. It is pretty similar to Unix system, except an additional Inode map is introduced.

2.2.2 Data Writing

This mechanism based on five mentioned assumptions.

The design of this system mainly contributes to a more efficient data writing. Since the whole system data storage is sequential, even written data is not related to each other, they are all written at once with the help of file buffer. Without the huge amount of random accesses on hard drive, the system performance is greatly improved.

2.2.3 Segment Cleaning

This mechanism mainly based on third and fourth assumptions.

Due to the usage of sequential system storage, it is crucial to scan through entire disk and clean up not-used space in a regular basis; otherwise, the whole system storage will be fully occupied easily. This can be fulfilled by frequently loading fragmented data back to disk in sequence.

2.3 At the time the LFS paper was written, non-volatile RAM (NVRAM) was not commonly available. If disks were replaced with NVRAM entirely, would the LFS design still make sense?

Since NVRAM needs not to use mechanical arm to read memory, the main motivation of using log-structured file — saving seeking time from random disk access — no longer exists. On the other hand, the cost of clean segments periodically still remains. Thus, if NVRAM is available, then there is no need to derive this log-structured file system.

2.4 In real-life the cost per byte of disk is likely to be far cheaper than NVRAM for some time. So instead, consider a situation where some NVRAM is available (e.g., 1/10th of the disk size). This NVRAM might be used for caching reads, caching writes, or storing particular meta-data. Argue which use might be most appropriate for improving the performance of LFS.

Different from Unix system, to fulfill log-structured system, an additional Inode map is required, which is used to maintains the location of all Inode blocks. Since Inode map needs to be edited when a file is updated, storing Inode map in NVRAM shall effectively improve the overall system performance.