Operating Systems – Fall 2020 Project 4

Considering the shorter semester this year, Project 4 is a small extension to project 3, but focusing on some concurrency issues.

Tasks

1. (20, extra credit)

   Fix all bugs you left from last time, either as your known bugs, or found by the TA's. You can get a max of 20 extra credit for this project, to make up your point loss from last time. If you lose more than 20 points at project 3, you can only make up for a max of 20 points here. Late penalty cannot me made up, though.

2. (30%)

   Change your FS to support multiple concurrent processes, allowing interleaving I/O's and disk accesses. With enough number of processes, you should be able to allow your CPU utilization to approach 100%, even if these processes are doing a lot of synchronized read/writes (flush each time). Note that the concurrency should not affect the capability for the file system to recover from crash failures.

   You should extend your testcases to show both the concurrent execution time, as well as monitoring the CPU utilization.

   Also, to respect the tradition of file system research, you should be able to build your source code for this project entirely on the LFS you build, and report the time takes to build.

3. (30%)

   Support a user-level write-back memory cache, and use this cache independent of system's buffer cache, i.e. everything is cached in this one, and each time you want to write to disk, you should immediately call the system's sync to force to disk, without allowing the data to buffer in kernel buffer cache. You should change all your user-level read and writes to use this cache, and sync only periodically or as user requests to the kernel.

   You can use standard malloc to manage the cache space (i.e. you do not need to implement your own memory manager), but you do need to track the cache size and limit your cache to 4MB at max. You can use any cache replacement policy, such as FIFO or RANDOM for simplicy.

4. (40%)

   Implement garbage collection to actually remove delete files and reclaim the disk space. First, you can implement a mechanism that is only *thread safe*, i.e. you can block all other file system operations while you do the garbage collection / compaction. The garbage collection should be

done in the same log file (i.e. do not create another disk file and do the copy). After the garbage collection, the old log can be rewritten without losing useful data. It worth 30 points to implement this part correctly.

Then you can implement a concurrent garbage collection mechanism, that allows *most* normal file system operations (i.e. operations that do not touch the blocks being garbage collected) to run concurrently. 10 points for this task.