CS 7600: Assignment 1
Test Results

Muzammil Abdul Rehman   Vikrant Singhal

1 Results

The following results are obtained after running the compilebench, whilst enabling different parts of the homework, and specifying different disk-sizes.

Disk Size	Homework Part	Blocks Read	Blocks Written	Read Operations	Write Operations
50 MB	None	340783	329009	175794	83647
	1	407500	329009	211792	79886
	2	340615	329009	175685	83647
	3	198680	329009	159673	83678
	4	116623	174172	82347	87280
150 MB	None	341605	666377	175319	83647
	1	408770	666377	211577	79886
	2	343040	666377	176199	83647
	3	200735	666377	159888	83675
	4	119074	566964	82707	108776
500 MB	None	347321	1903393	175356	83647
	1	413146	1903393	210793	79886
	2	347262	1903393	175320	83647
	3	199331	1903393	152846	83680
	4	94234	1714541	56184	119473

2 Discussion

In this section, we will discuss the working of the caches that we implemented, and their relevance to the numbers that are mentioned in the previous section. Here is a diagram to depict the levels of caches, which we implement, with respect to the disk and the calls to the functions (that were implemented in part 1).

System call	4
Open/*Opendir*-Cache	3
$\{Parent\ Inode\ Number \times Child\ Name\} \longrightarrow Child\ Inode\ Number$	2
Write-Back-Cache	1
Disk	0

Read-Cache

The *open* function helps to cache the inode number of the file being opened, so that every subsequent read-call to that file doesn't involve fetching its inode number from the disk. This prevents some blocks to be read. Similarly, the *opendir* function is called on a directory, and helps to cache the inode number of that directory, along with the *dirent*s of the children of that directory. This way, the reads on children become more efficient as their inode numbers don't have be searched for in the disk (owing to the stored contexts).

If *open*/*opendir* don't find the right context, that level of caches searches through the next level, which is basically a mapping from pairs of directories' inode numbers, and their children's names, to the respective children's inode numbers. This is in the cache that we created in memory for part 3 of this assignmemt. This further reduces the number of blocks read.

Finally, when this layer of cache fails to locate the inodes, then it searches in the lowest layer of caches (the write-back-cache), which is closest to the hard disk. Note that it only searches through the valid locations on the write-back-cache. Here, the search takes place through both the dirty and the clean caches to find the inodes. This also reduces the number of blocks read.

Write-Cache

The only cache enabling writes is the write-back-cache. When writes are made to something in the clean cache, those pages are copied to the dirty cache (and the necessary flushing takes place in the dirty cache to accommodate the new pages). Writes are also often made to the valid pages of the dirty cache. Therefore, writes to disk are delayed as much as possible (before being finally written from the dirty cache), reducing the number of blocks written.

Consistency with Results

We implement caches incrementally from higher level to lower level. Each time a level is added, there should be some improvement only in the reads, except when we add the write-back-cache, which improves both read and write efficiencies. This is exactly what happens in the results that we mention in the first section. Note that the number of reads remains the similar for all sizes of the disk because the compilebench does not actually read the file. The writes increase becasue of we write back all the inodes and bitmaps, but since only the metadata is read, the number of blocks read remains similar.