**Report**

**System Design:**

The file system is implemented as a static library that allows applications use it for file creation, read, write and delete. This file system is limited in terms of maximum number of files that can be created. There are 4 blocks of 4KB size available for holding the directory entries (128 bytes each). Therefore, the file system can accommodate at most 128 files ((4 \* 4096)/128). Each directory entry needs an FCB. Hence 4 blocks are reserved for holding File Control Blocks for each directory.

In terms of free block management, the empty blocks are tracked by using a 4-block bitmap. The bitmap blocks are initialized as all zeros in while formatting the file system. On each request for an empty block, the bitmap is traversed until finding an unset bit (0). Then the bit is set and the offset of the empty block is returned to the requesting application. When a file is deleted, all of it’s data blocks are marked as free by unsetting the corresponding bits in the bitmap.

Indexed allocation is used as the allocation algorithm. Each FCB holds an index node, in which all data block indices of its corresponding file are located. One level indexing is used. Therefore, each file can have at most 1024 (assuming each data block pointer is 4 bytes) blocks which limits maximum size of each block to 4MB (1024 \* 4096 = 4MB).

The system defines structures for directory entry (as d\_entry) and FCB (as fcb\_t). A directory entry holds name, index of an FCB and in integer to indicate whether it is used or not. An FCB structure contains the size of its corresponding file, an integer to indicate availability, number of allocated data blocks and the index of index block.

At most 16 files can be opened by a process. Therefore, the library holds an array of open files and tracks the files opened. This array holds structures, which include the name and mode of the file.

**Experiments:**

* **File Creation**

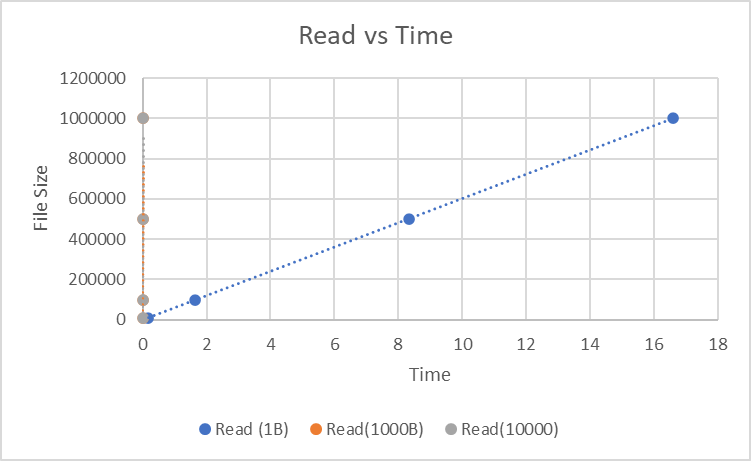
File creation takes around 1200 microseconds. However, it was slightly more (1450) for the last files (125th or 128th). FCB block allocation does not use bitmap for finding empty blocks. Hence, it can account for this small difference.

Size of the disk seems to be irrelevant when it comes to creation of files. Experiments were done on 6MB and 512MB and creation time was very close.

* **Read**

Read time turns out to be significantly affected by the number of bytes being read, in a read call, and the total size of the file.

|  |  |  |  |
| --- | --- | --- | --- |
| Total/Read | 1 | 1000 | 10000 |
| 10000 | 0.171875 | 0.000002 | 0.000000 |
| 100000 | 1.640625 | 0.000021 | 0.000000 |
| 500000 | 8.328125 | 0.011134 | 0.000022 |
| 1000000 | 16.593750 | 0.012155 | 0.000032 |



* **Write**

Write time, same as read time, seem to be mostly affected by the number of bytes being written, in a write call, and the total size of the file.

|  |  |  |  |
| --- | --- | --- | --- |
| Total/Read | 1 | 1000 | 10000 |
| 10000 | 0.218750 | 0.000004 | 0.000000 |
| 100000 | 2.218750 | 0.015625 | 0.000001 |
| 500000 | 11.359375 | 0.030125 | 0.015625 |
| 1000000 | 22.562500 | 0.031250 | 0.021621 |

**Conculusion:**

As it can be seen from the experiments, file creation takes negligible time due to the maximum 128 number of possible files. When it comes to read and write time, they are affected mainly by the number of bytes being read in a read or write call. As it can be seen from the charts, very small read or write size takes significant time i.e. for 1MB file and read size of 1 Byte, it takes around 17 and 23 seconds to read and write respectively. Moreover, the time taken largely drops to around half a second when the read and write size increases to 1000.