@author - Patrick Royal, Jonathan Schmidt

DFS - Top Layer

Contains list of all dfiles on the disk.

- Max file size is at least 50 blocks

DFS.init is called by each test program as it runs

- Check that each DFile has exactly one Inode, the size of each DFile is a legal value, the block maps of all DFiles have a valid block number for every block in the DFile, and no data block is listed for more than one DFile
- Build the list of DFiles on the disk by scanning the *i*node region for valid *i*nodes
- Build a list of all allocated and free blocks on the VirtualDisk in order to allocate free blocks to new files

This is the layer that is accessed by programs that need to store information on the disk.

- Create dFiles
    - Searches for a set of free space large enough to hold the new file. If none can be found, then use the largest free space first and repeat the algorithm for the remainder of the file's blocks
    - A newly-created file is loaded immediately into the cache
- Destroy dFiles
    - Adds all of the blocks in the dFile to the list of free blocks
- Read dFiles - starts at offset 0
    - Goes through each block in the dFile map and gets a DBuffer for that block from the DBufferCache.
    - Reads the data from the DBuffers in the correct order and copies it into the byte array.
- Write dFiles - starts at offset 0
    - Gets the DBuffers for each block in the dFile map from the DBufferCache
    - Copies the information in order from the byte array into the DBuffer byte array
    - Releases the blocks through the DBufferCache so that they can be accessed again
    - The maximum write is equal to the file size

At most one program can run with a given VDF at a given time. Client access to the underlying disk is synchronized

Reading from and writing to a dFile is synchronized. A client cannot read a file from the DFS while another client is writing the dFile.

DBufferCache - Middle Layer

Stores a fixed number of DBuffer objects that represent a block of data from the hard disk.

Makes use of Least Recently Used caching policies.

- One LRU bit is maintained for each block
- The items in the cache are placed into a binary search tree

- When searching for an item, the LRU bit tells the traverser to “go left for an LRU element” or “go right for an LRU element”
- When accessing an item, flip all of the bits of the items encountered along the way to denote the opposite of the direction taken
- This strikes an efficient balance between finding *the* least-recently used element and saving time on the number of elements whose metadata must be changed with each access
- Whenever the LRU block is removed from the cache, it is written to the disk if dirty

Allows threads to find the DBuffer associated with a given block, if it is in the cache, or allocate a new block for it otherwise
All reads/writes from test programs are stored here, and then written to the virtualdisk according to caching algorithms.

- Individual DBuffers are synchronized, so only one thread can access them at a time

Calling sync() writes all “dirty” blocks in the cache back to the disk without removing them from the cache

```
For block b : cache
    if isBusy
        fail
    else if checkValid and not checkClean
        startPush
        waitClean
```

VirtualDisk - Lowest Layer

Made up of an array of blocks each of fixed size that is a power of two.

- 0th Block contains metadata
- 1-*i*th Blocks contain INode information representing the different files
- *i*-end Blocks contain the actual data blocks

Supports reads and writes of blocks based on their block number

- Each low-level read/write operation affects exactly one block
- The VDF may be grown by writing blocks at successively higher block numbers with no maximum

Each operation on the VirtualDisk is asynchronous. The VirtualDisk will call an iocomplete() method on the DBuffer once it has finished.
Every I/O operation that is created using the startRequest method will put the operation onto a queue which the VirtualDisk will process in a FCFS order.

- If a request to the queue would call multiple I/O operations on a single DBuffer before the first is done, then the request is moved to the back of the queue. If all requests in the queue would violate this rule, then it fails

Other Components

DBuffer

Describes the contents and status of the buffer.
Status information should include whether or not device I/O is in progress on the block (to facilitate locking).
Each block is associated with at most one buffer/DBuffer pair at a time.
The DBuffer allows threads to initiate asynchronous read/write operations on the block.
The DFS can check the status of the DBuffer and wait until it is in the correct status before it continues writing or reading from it.
When a DBuffer is written to by the DFS, the DBuffer will be marked as dirty.
When a DBuffer is written to the VDF or read from the VDF, the DBuffer will be marked as clean.
If an operation is happening on the DBuffer, then it will be marked as busy.
Fetching a block from the volume first checks to ensure that no other block from the file is currently in use. This supports file atomicity

Process for using the DBuffer

```
if not DBuffer.isBusy
    if DBuffer.checkValid
        if writing to DBuffer
            write information to the DBuffer
        else reading from DBuffer
            read information from the buffer
    else
        startFetch // starts fetching the data from the VirtualDisk
        waitValid // calling thread blocks for the data to be returned
else
    will fail
```

DFile

- Represented as a LinkedList of blocks
    - The maximum number of blocks for a DFile is at least 50
- Contains a DfileID, whose value is a positive, unique integer (assigned by the DFS layer)
- Max number of DFiles is 512