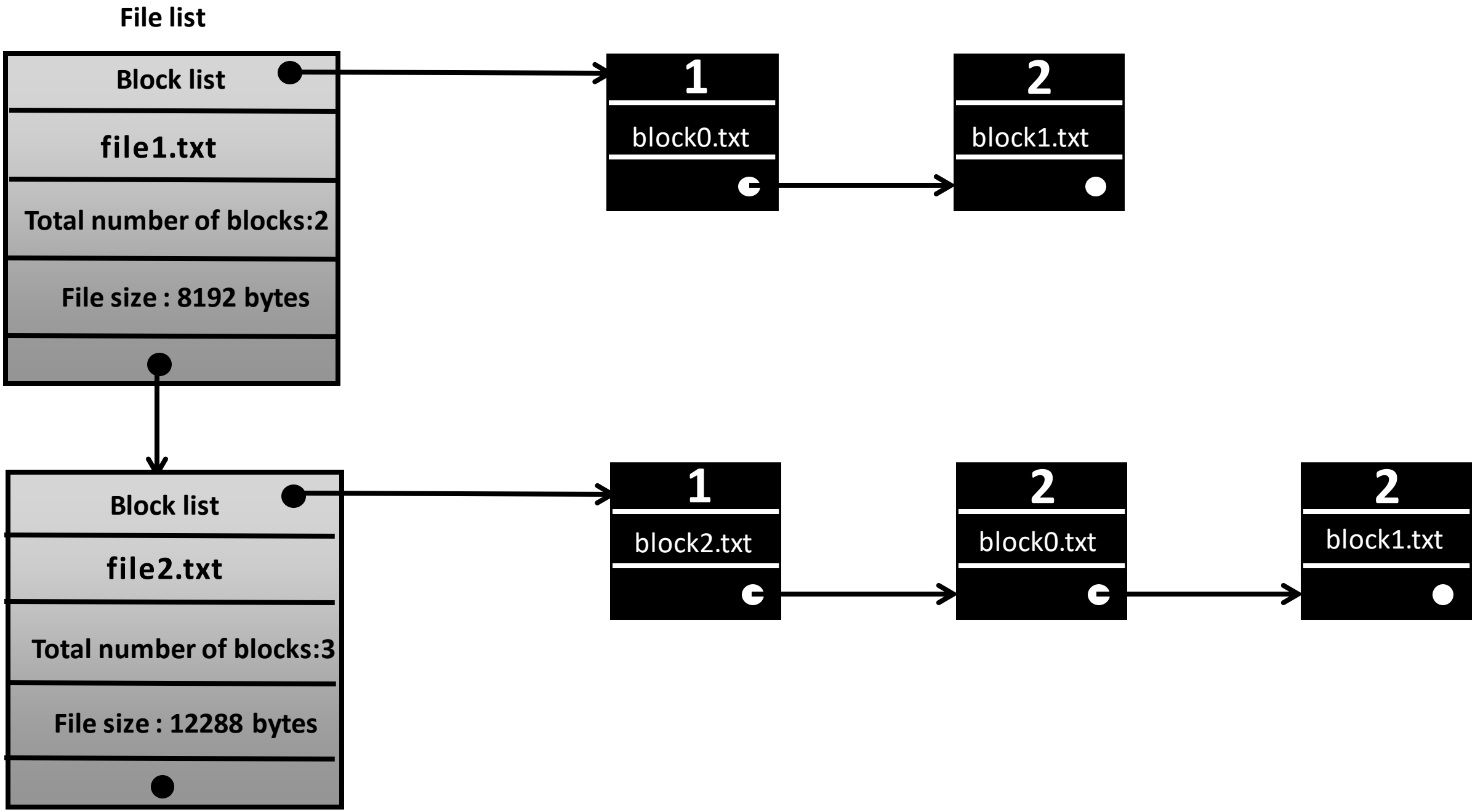
Implementation

In order to be able to control, monitor, and properly update files,we keep a list of struct file_node inside bbfs code. Each file_node struct represents a file. This structure also contains a list of the blocks that make up the particular file, the size of the file, the number of blocks the file has, and indexes to the next and the previous file in the file list.

Each block file is represented by a struct block. The struct block contains the position of this block in the file, the block name and indexes to the next and the previous block in the block list.


File list
Block list
file1.txt
Total number of blocks:2
File size : 8192 bytes
1
block0.txt
2
block1.txt
Block list
file2.txt
Total number of blocks:3
File size : 12288 bytes
1
block2.txt
2
block0.txt
2
block1.txt

Function implementation

bb_readdir: When accessing the folder, only the conventional files (file1.txt etc) are read. The block files that are used to store the data of an archive are not read during readdir function.

bb_open: We search the file list. If the file does not exist, we add a file_node that represents this file to the list.

bb_read: Each time the size of a read-call is served in pieces of 4096 bytes. First we scan the block list of the specific file_node that represents the file.If this block exists,we get the contents and write them in the buffer that user-space gave us .The process continues for the next 4096 bytes.If this block does not exist,we read the 4096 bytes from the conventional file and calculate their hash value.Then,we check the block store.If it exists there, we create this block in the list of blocks of that particular file_node by appropriately updating the corresponding information. If this block does not exist on storage, we create it , computing its hash and then adding it to the block list of the file_node as above. This process is done for size / 4096 times. Lastly, we also update the size and the total number of blocks of the corresponding file.

bb_write: Each time the size of a write-call is served in pieces of 4096 bytes.Initially, we calculate the hash of these data. Then we check if there is storage block with the same hash. If it does not exist,we create it. Then we will refresh the list of blocks of that particular file_node. The process is repeated for size / 4096 times. At the end, the file size is appropriately updated in the corresponding field of the corresponding file_node.

bb_release: We find the file_node that represents the particular file for which the close was called. We scan the list of blocks of the specific file_node and write down all the blocks in the conventional file so that the changes we have made can finally be seen.

bb_getattr: First,we call lstat.In the statbuf-> st_size field,we store the size of the file that we have kept in the file_node which represents this file. In this way, the user receives exactly the information they would receive in a conventional file system.

bb_unlink: The file becomes unlinked. It is then checked whether each block of this file is shared with blocks of other files. The non-shared blocks of this file are ultimately "destroyed".

bb_close: We delete the file node with the same path(name)from the file list.