MP7: Simple File System

Joshua Clapp
UIN: 930004089
CSCE410: Operating System

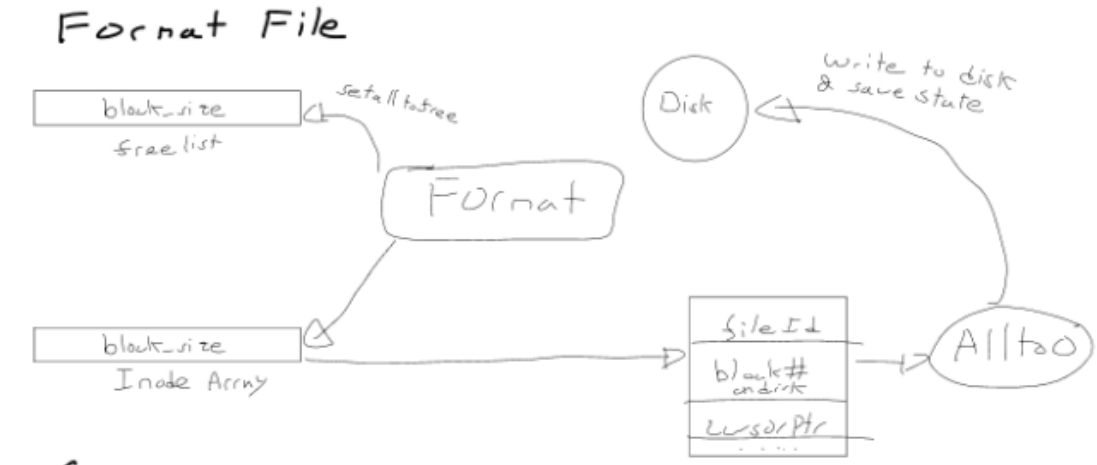
Assigned Tasks

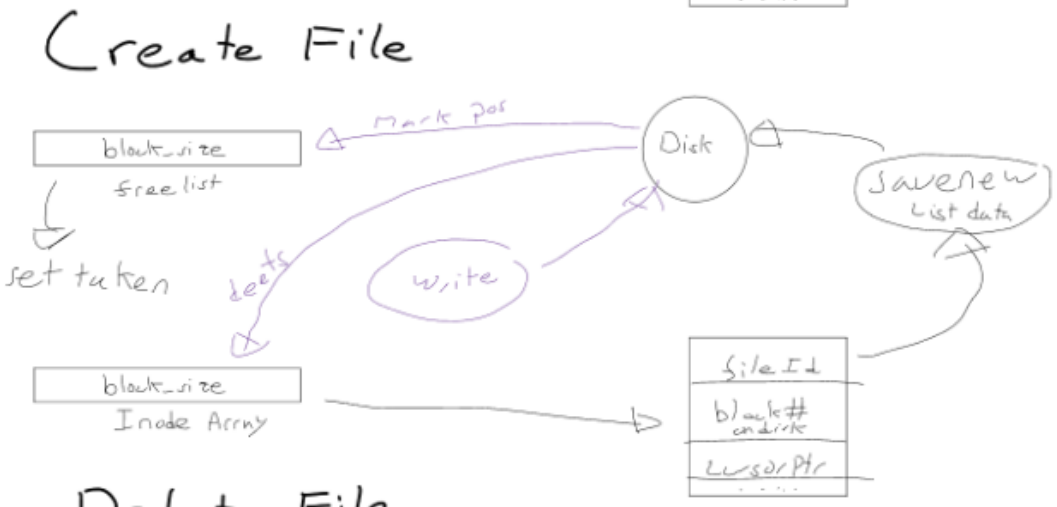
Main:
Completed.
Bonus Option 1: Disk Design
Completed
Bonus Option 2: Disk Design Implementation
Not attempted

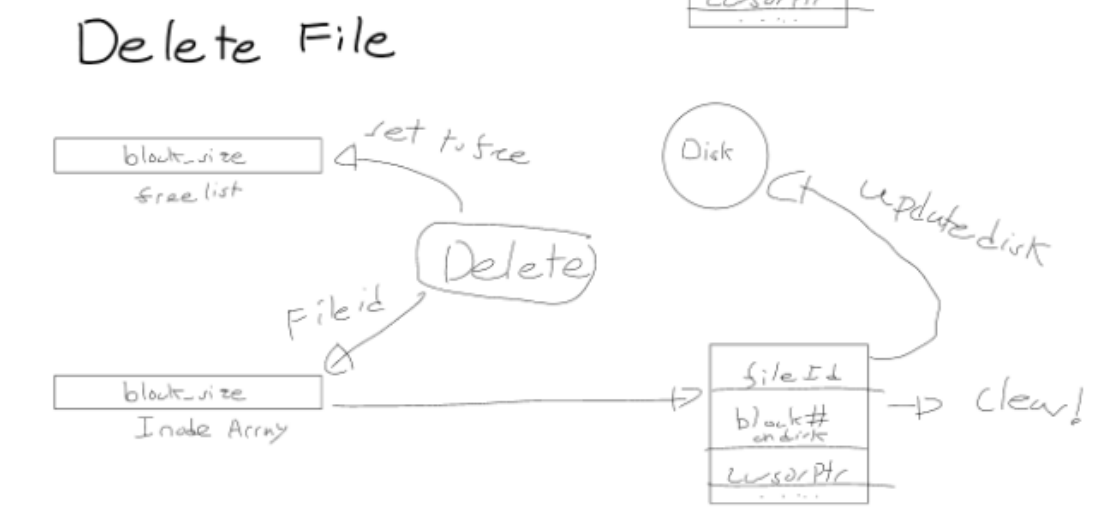
System Design

Main:

The main system is to write sequentially and allocate free blocks on disk







Bonus Option 1: Disk Design

To expand the disk to hold files up to 64KB, I would need to allocate more blocks per write a file as each file block has 512 bytes on the simple disk drive. This would require a function that finds free blocks. I would need to then account for the cursor in the file write as changing blocks would require me to hold a cursor pointer to the buffer position. Then, I need to mark multiple allocated blocks related to the same inode and the order to read in the proper order, a queue would work well here. When deleting, instead of deleting one 512 block for a file, I would need to delete multiple blocks up to 64KB, order would not matter here, this would require the same data structure that links multiple blocks per inode.

Code Description

Main:

I changed File_System.C, File_System.H, File.C, File.H, and Kernel.C.

To run this logic

Make and run it

File_System.H

The basic implementation is inodes containing file Id, the cursor pointer on the disk, and the respective block number that the disk contains the written file on.

```
class Inode
{
  friend class FileSystem; // The inode is in an uncomfortable position between
  friend class File;       // File System and File. We give both full access
                           // to the Inode.

public:

  /* You will need additional information in the inode, such as allocation
     information. */


  int fileId;
   int blockPointer;
   int blockNumberFile; // file poitner
  FileSystem *fs; // It may be handy to have a pointer to the File system.
                  // For example when you need a new block or when you want
                  // to load or save the inode list. (Depends on your
                  // implementation.)
```

```
public:
  static constexpr unsigned int MAX_INODES = SimpleDisk::BLOCK_SIZE / sizeof(Inode);
  Inode *inodeList[block_size]; // the inode list
  unsigned char  freeList[block_size];
  SimpleDisk *disk;
  FileSystem();
  /* Just initializes local data structures. Does not connect to disk yet. */

  ~FileSystem();
  /* Unmount file system if it has been mounted. */

  bool Mount(SimpleDisk *_disk);
  /* Associates this file system with a disk. Limit to at most one file system per disk.
     Returns true if operation successful (i.e. there is indeed a file system on the disk.) */

  bool Format(SimpleDisk *_disk, unsigned int _size);
  /* Wipes any file system from the disk and installs an empty file system of given size. */

  Inode *LookupFile(int _file_id);
  /* Find file with given id in file system. If found, return its inode.
     Otherwise, return null. */

  bool CreateFile(int _file_id);
  /* Create file with given id in the file system. If file exists already,
     abort and return false. Otherwise, return true. */

  bool DeleteFile(int _file_id);
  /* Delete file with given id in the file system; free any disk block occupied by the file. */
};
#endif
```

File_System.C:FileSystem()

This initializes our inodeList and sets all data to 0

```
41  FileSystem::FileSystem()
42  {
43      Console::puts("In file system constructor.\n");
44      memset(inodeList, 0, block_size);
45      // memset(freeList, 0, block_size);
46  }
47
```

File_System.C: ~FileSystem();

Write the free list data to the disk, set the memory and then copy all the inode data back into the file system.

```
48  FileSystem::~FileSystem()
49  {
50      Console::puts("unmounting file system saving data \n");
51      /* Make sure that the inode list and the free list are saved. */
52      this->disk->write(1, freeList);
53      memset(freeList, 'f', block_size);
54      // clear freelist, read form thers
55      this->disk->read(0, freeList);
56
57      for (int i = 0; i < SYSTEM_BLOCKS; i++)
58      {
59          this->disk->write(i, freeList);
60      }
61
62      // read the data back to the blokcs now
63  }
64
```

File_System.C: Mount(SimpleDisk *_disk);

Set the given disk into the Inode drive

```
68  bool FileSystem::Mount(SimpleDisk *_disk)
69  {
70      Console::puts("mounting file system from disk\n");
71      this->disk = _disk;
72      // /* Here you read the inode list and the free list into memory */
73      return true;
74      // assert(false);
75  }
76
```

File_System.C: Format(SimpleDisk *_disk, unsigned int _size);

This uses the free list as a placeholder and writes an empty inode array, then reads it back into the inode list, and writes into the block zero of the disk. It then sets up the inode array to the empty parameters. I then clear out the freelist and then read from the freelist block into the freelist array, and mark the inode, and freelist block as taken and lastly writing this data back to the disk.

```
77  bool FileSystem::Format(SimpleDisk *_disk, unsigned int _size)
78  { // static!
79      Console::puts("formatting disk\n");
80      /* Here you populate the disk with an initialized (probably empty) inode list
81         and a free list. Make sure that blocks used for the inodeList and for the free list
82         are marked as used, otherwise they may get overwritten. */
83      unsigned char veracityCheckerNodes[block_size]; // so this is for passing ablock side, I know the file descriptor is -1 block side
84      memset(veracityCheckerNodes, 0, block_size);
85
86      for (int i = 0; i < SYSTEM_BLOCKS; i++)
87      {
88          _disk->write(i, veracityCheckerNodes);
89      }
90      // read the data back to the blokcs now
91
92      _disk->read(0, veracityCheckerNodes);
93
94      // now return freelist back to its proper purpose
95
96      for (int i = 0; i < MAX_INODES; i++)
97      {
98
99          inodeList[i]->blockNumberFile = -1; // where allocated on block
100         inodeList[i]->fileId = 0;           // no disk id
101     }
102     _disk->write(0, veracityCheckerNodes);
103
104     // clear free list
105
106     memset(freeList, 'f', block_size);
107
108     _disk->read(1, freeList);
109     for (int i = 0; i < block_size; i++)
110     {
111         freeList[i] = 'f';
112     }
113     freeList[0] = 't';
114     freeList[1] = 't';
115     _disk->write(1, freeList);
116     Console::puts(" disk format complete \n");
117     return true;
118 }
```

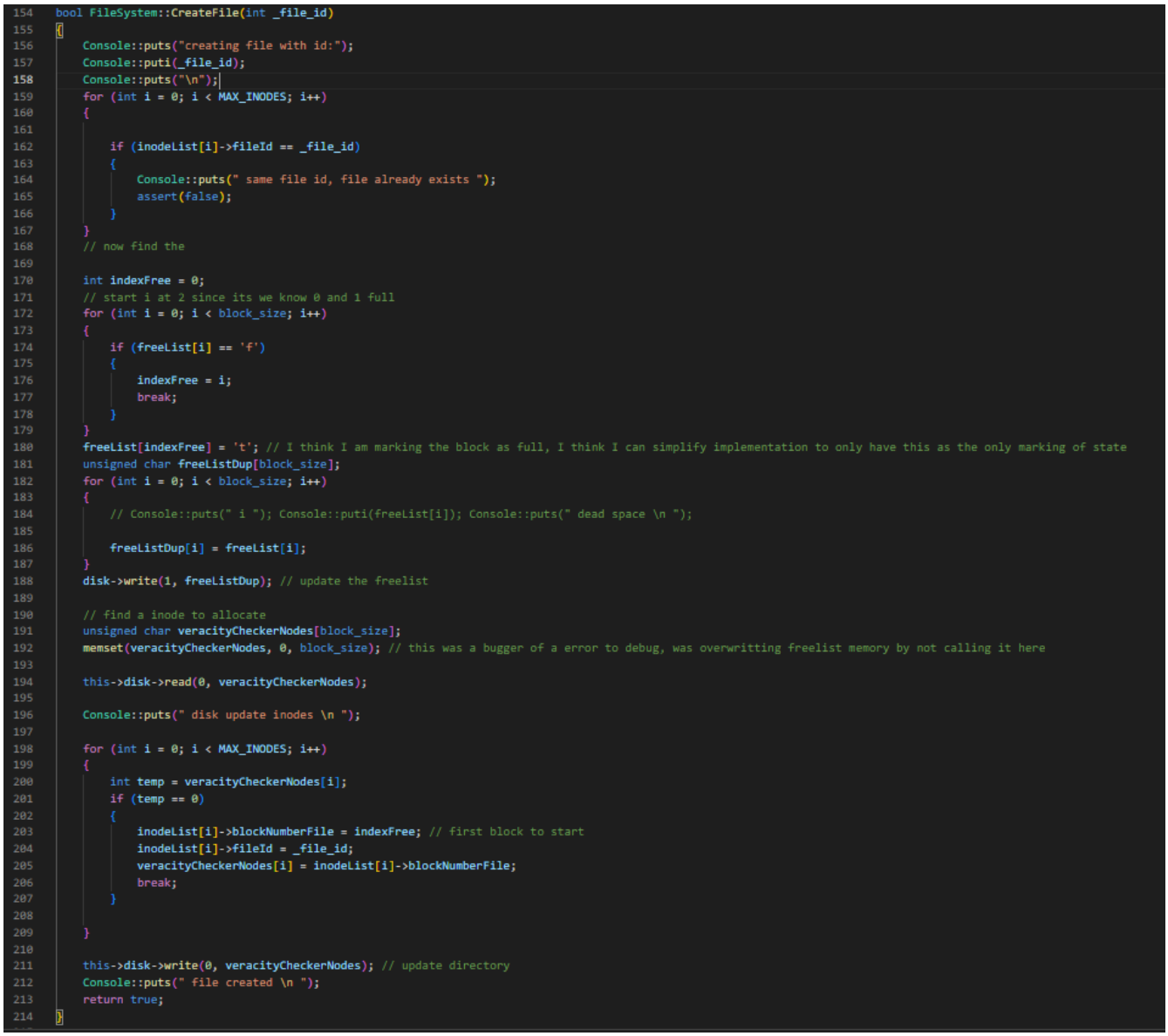
File_System.C: *LookupFile(int _file_id);

This runs through the inode array, finds a file id that matches and returns it.

```
130  Inode *FileSystem::LookupFile(int _file_id)
131  {
132      Console::puts(" looking up file with id = ");
133      Console::puti(_file_id);
134      Console::puts("\n");
135      /* Here you go through the inode list to find the file. */
136      for (int i = 0; i < MAX_INODES; i++)
137      {
138          if (this->inodeList[i]->fileId == _file_id)
139          {
140              return inodeList[i];
141          }
142      }
143      Console::puts(" Inode not found ith file id \n ");
144      assert(false);
145  }
```

File_System.C: CreateFile(int _file_id);

First error check to see if the file id does not exist before. Then check the free list and find the first free block and break sequentially. Mark the freelist as taken, and then I make a copy of the freelist to call from the disk block zero, I take a deep copy of the freelist and read from the disk. Using this data from the disk I find the first unallocated inode and assign it the proper block number, and file id. Then I write the data to the disk and return.



```
154  bool FileSystem::CreateFile(int _file_id)
155  {
156      Console::puts("creating file with id:");
157      Console::puti(_file_id);
158      Console::puts("\n");
159      for (int i = 0; i < MAX_INODES; i++)
160      {
161
162          if (inodeList[i]->fileId == _file_id)
163          {
164              Console::puts(" same file id, file already exists ");
165              assert(false);
166          }
167      }
168      // now find the
169
170      int indexFree = 0;
171      // start i at 2 since its we know 0 and 1 full
172      for (int i = 0; i < block_size; i++)
173      {
174          if (freeList[i] == 'f')
175          {
176              indexFree = i;
177              break;
178          }
179      }
180      freeList[indexFree] = 't'; // I think I am marking the block as full, I think I can simplify implementation to only have this as the only marking of state
181      unsigned char freeListDup[block_size];
182      for (int i = 0; i < block_size; i++)
183      {
184          // Console::puts(" i "); Console::puti(freeList[i]); Console::puts(" dead space \n ");
185
186          freeListDup[i] = freeList[i];
187      }
188      disk->write(1, freeListDup); // update the freelist
189
190      // find a inode to allocate
191      unsigned char veracityCheckerNodes[block_size];
192      memset(veracityCheckerNodes, 0, block_size); // this was a bugger of a error to debug, was overwritting freelist memory by not calling it here
193
194      this->disk->read(0, veracityCheckerNodes);
195
196      Console::puts(" disk update inodes \n ");
197
198      for (int i = 0; i < MAX_INODES; i++)
199      {
200          int temp = veracityCheckerNodes[i];
201          if (temp == 0)
202          {
203              inodeList[i]->blockNumberFile = indexFree; // first block to start
204              inodeList[i]->fileId = _file_id;
205              veracityCheckerNodes[i] = inodeList[i]->blockNumberFile;
206              break;
207          }
208
209      }
210
211      this->disk->write(0, veracityCheckerNodes); // update directory
212      Console::puts(" file created \n ");
213      return true;
214  }
```

File_System.C: DeleteFile(int _file_id);

Make sure the file id exists by taking a copy of the inode disk data from block zero, and error checking it for the file id using the fact that with our sequential allocation, the block number is always file id pulse one. Then go through the Inode array and at the file id clear the inode out, reset the freelist and write the data back to the disk for both freelist and inode.

```
240  bool FileSystem::DeleteFile(int _file_id)
241  {
242      unsigned char veracityCheckerNodes[block_size]; // so this is for passing ablock side, I know the file descriptor is -1 block side
243      memset(veracityCheckerNodes, 0, block_size);
244      memset(freeList, 'f', block_size);
245      this->disk->read(0, veracityCheckerNodes);
246      this->disk->read(1, freeList);
247      bool deleteCheck = false;
248
249      for (int i = 0; i < block_size; i++)
250      {
251          /* code */
252          int temp = veracityCheckerNodes[i];
253          if ( temp == _file_id+1 )
254          {
255
256              deleteCheck = true;
257              veracityCheckerNodes[i] = 0; //  delete clear out
258          }
259      }
260      if (deleteCheck == false)
261      {
262          Console::puts(" no file to delete breaking \n");
263          assert(false);
264      }
265      // kill freelist postion
266
267      // invalidate inode
268
269      for (int i = 0; i < MAX_INODES; i++)
270      {
271          if (veracityCheckerNodes[i] - 1 == _file_id)
272          {
273              // Console::puts(" do I exist ? \n");Console::puti(i); Console::puts(" dead spcae \n ");
274
275              inodeList[i]->blockNumberFile = -1; // first block to start
276              inodeList[i]->fileId = 0;
277              veracityCheckerNodes[i] = inodeList[i]->blockNumberFile;
278              break;
279          }
280          // veracityCheckerNodes[i] = inodeList[i]->blockNumberFile;
281      }
282
283
284      freeList[_file_id+1] = 'f'; // this is the position where is is free.
285      this->disk->write(0, veracityCheckerNodes); // update directory
286      this->disk->write(1, freeList); // update directory
287      return true;
288  }
```

File.H

I keep track of file id, current block on disk from the inode, and the cursor of the buffer reading to or writing frome.

```
44  class File {
45
46  private:
47      /* -- your file data structures here ... */
48     FileSystem * currentFileSystem;
49
50      /* You will need a reference to the inode, maybe even a reference to the
51         file system.
52         You may also want a current position, which indicates which position in
53         the file you will read or write next. */
54
55      unsigned char block_cache[SimpleDisk::BLOCK_SIZE];
56      /* It will be helpful to have a cached copy of the block that you are reading
57         from and writing to. In the base submission, files have only one block, which
58         you can cache here. You read the data from disk to cache whenever you open the
59         file, and you write the data to disk whenever you close the file.
60      */
61  public:
62
63     int fileId;
64     int currentBlock;
65     int cursorPointer;
66      File(FileSystem *_fs, int _id);
67      /* Constructor for the file handle. Set the 'current position' to be at the
68         beginning of the file. */
69
70      ~File();
71      /* Closes the file. Deletes any data structures associated with the file handle. */
72
73      int Read(unsigned int _n, char * _buf);
74      /* Read _n characters from the file starting at the current position and
75         copy them in _buf.  Return the number of characters read.
76         Do not read beyond the end of the file. */
77
78      int Write(unsigned int _n, const char * _buf);
79      /* Write _n characters to the file starting at the current position. If the write
80         extends over the end of the file, extend the length of the file until all data is
81         written or until the maximum file size is reached. Do not write beyond the maximum
82         length of the file.
83         Return the number of characters written. */
84
85      void Reset();
86      /* Set the 'current position' to the beginning of the file. */
87
88      bool EoF();
89      /* Is the current position for the file at the end of the file? */
90
91  };
92
93  #endif
94
```

File.C: File(FileSystem *_fs, int _id);

Save vital information like current file system, set the cursor pointer marking the buffer to zero, save the file id, and use the knowledge that a current block allocation is one more than the file id.

```
29  File::File(FileSystem *_fs, int _id)
30  {
31      Console::puts("Opening file.\n");
32      this->currentFileSystem = _fs;
33      this->cursorPointer = 0;
34      this->fileId = _id;
35      this->currentBlock = _id+1; // allocating
36      Console::puts(" current block print "); Console::puti(this->currentBlock);Console::puts(" dead space \n");
37      // assert(false);
38
39  }
40
```

File.C: ~File();

Search through the inodes and match the file id saving the fileId, cursor pointer, and the current block.

```
41  File::~File()
42  {
43      Console::puts("Closing file.\n");
44
45      Console::puts("update inodes \n");
46      auto fileIdInode = this->currentFileSystem->inodeList;
47      for (int i = 0; i < currentFileSystem->MAX_INODES; i++)
48      {
49
50          if (fileIdInode[i]->fileId == this->fileId )
51          {
52              fileIdInode[i]->fileId = this->fileId;              // no disk id
53              fileIdInode[i]->blockPointer = this->cursorPointer;
54              fileIdInode[i]->blockNumberFile = this->currentBlock;
55              Console::puts( (const char * ) currentBlock );
56          }
57      }
58
59      /* Make sure that you write any cached data to disk. */
60      /* Also make sure that the inode in the inode list is updated. */
61  }
62
```

File.C: Read(unsigned int _n, char * _buf);

Error check the _n data make sure it is 1 block, and then clear the memory in block_cache, reading from the disk into _buf.

```
67  int File::Read(unsigned int _n, char *_buf)
68  {
69      Console::puts("reading from file\n");
70      int offset = 0;
71      // unsigned char block_cache[512];
72
73
74
75      if (_n > 512)
76      {
77          _n = 512;
78      }
79      int countDown = _n;
80
81      // while (countDown > 0)
82      // {
83          // clear the junk that may have not beed writtenm this logic is from csce 313
84          memset(block_cache, 0, block_size);
85          this->currentFileSystem->disk->read(this->currentBlock, block_cache);
86
87      // deep copy required, test checks for pointers
88      for ( int i = 0 ; i < _n ; i++)
89      {
90          _buf[i] = block_cache[i];
91      }
92      // Console::puts(" read value");
93      Console::puts(_buf);
94
95      return (int) _n;
96  }
```

File.C: Write(unsigned int _n, const char * _buf);

Error check the _n block sign to make sure it is one block sign, clear the memory in the data, copy the data into block_cache and add the pointer from _buf, write the data to disk and adjust the pointer.

File.C: Reset();

```
98  int File::Write(unsigned int _n, const char *_buf)
99  {
100     Console::puts("writing to file\n");
101     if (_n > 512)
102     {
103         _n = 512;
104     }
105     int offset = 0;
106
107     // int countDown = _n;
108     memset(block_cache,0,block_size); //  clear the cache to avoid erronous writes
109     Console::puts(" the block is "); Console::puti(currentBlock); Console::puts(" dead space \n");
110     memcpy(block_cache + this->cursorPointer, _buf, _n);
111     Console::puts(" the block is "); Console::puts((const char * )block_cache+this->cursorPointer); Console::puts(" dead space \n");
112      this->currentFileSystem->disk->write(this->currentBlock, block_cache + this->cursorPointer);
113     this->cursorPointer = this->cursorPointer + _n;
114     Console::puts(" curosor positioning! ");Console::puti( this->cursorPointer );Console::puts(" dead space \n ");
115
116      Console::puts(" clear cache? "); Console::puts((const char * )block_cache); Console::puts(" dead space \n");
117     return (int) _n;
118 }
119
```

Reset the cursorPointer to zero

```
120 void File::Reset()
121     {
122         Console::puts("resetting file\n");
123         this->cursorPointer = 0;
124
125     }
126
```

File.C: EoF();

Check if the pointer is at the end of the file block and allocate another block, resetting pointer. This would be used in the design of the MP2 bonus.

```
bool File::EoF()
{

    Console::puts("checking for EoF\n");
    if(this->cursorPointer > block_size-1)
    {
        Console::puts(" handle adjusting pointer now  \n ");
        this->currentBlock++;
        this->cursorPointer = 0;
        return true;
    }
    else
    {
        return false;
    }
}
```

Testing

I added no additional tests and the coverage is quite small, for the actual block size testing, there are no files that allocate to multiple blocks or multi directory files. I relied on the test cases implemented which ensures that my system can dynamically allocate to the disk, delete and reallocate without issue. I am ignoring eviction policies of the data and other important file system operations.

Main Task: Device Driver Delete, Create, and Allocate files.

THIS RUNS FOREVER, I ADDED A STOP CONDITION OF 20 ITERATIONS SO 40 FILES ARE CREATED AND DELETED.