**fs.c**

Contained fdRead and fsWrite, as well as other functions like fsTell which called bfsTell. It also contained fsSeek which was used to move the curser. It had three different options, but we only used the SEEK\_SET option to set the curser by the offset. Other functions like fsMount and fsOpen were not needed for the read and write function.

**fsRead**

**i32 fsRead(i32 fd, i32 numb, void\* buf)**

fsRead takes a file descriptor, the number of bytes of data from the cursor, and a pointer to a buffer called buf.

**Finding numbers**:

We use the bfsTell() function to determine the current pointer and we divide the current pointer by 512 to determine the current FBN. We add the current pointer and the numb parameter together, then divide by 512 to determine the end FBN. We use the bfsFdToInum() function to determine the inum. Bytes read starts off being equal to the numb parameter. If the current Pointer plus the numb parameter is larger than the file size then we reduce bytesRead to the fileSize minus the currentPointer. And we change the end FBN to match accordingly. We also create two temporary buffers. One a smaller buffer of size 512 called loopBuf, and another larger buffer that varies in size called readBuf.

**The function**:

We use a for loop to loop from the current to the end FBN. In the loop we call bfsRead to read a particular FBN into the fixed 512 buffer. We use memcpy to transfer the contents of the smaller loopBuf into the readBuf + an offset. memcpy will copy the smaller buffer into the next block worth of the larger buffer. We then increment the offset by 512. After the for loop we call memcpy to transfer the contents of the readBuf into the buf parameter, call fsSeek to move the curser, and return the number of bytes read.

**fsWrite**

**i32 fsWrite(i32 fd, i32 numb, void\* buf)**

fsWrite uses many of the same number as fsRead does, but has a few extras. We use filesize -1 then divide by 512 to determine maxFileFbn. If this is large than end FBN then we call bfsExtend to expand the file system. We then set a bool named expandingWrite, which is set to false by default, to true.

**The function**:

We read the first FBN to a 512 buffer by calling bfsRead. We then use memcpy to copy the blockBuf to the writeBuf. This saves the starting block. This is necessary because we may not overwrit the entire block, so we need to save the part we do not over write. If we are writing to more than one block we also call bfsRead and memcpy, but for the last FBN, for the same reason. We then call memcpy and write the buffer parameter into our writeBuffer

We then make a for loop that increments from the startFbn to the endFbn. In that we call memcpy where we take the writeBuf plus an offset and copy that to teh blockBuf. We then call bfsFbnToDbn to find the current DBN. Next we call birWrite and write teh contents of the blockBuf to the correct DBN. Lastly we increment the offset.

When the for loop is done we check to see if expandingWrite is true. If it is that means we need to call bfsSetSize to increase the size of the file to the currentPointer + numb.

Then we call fsSeek to move the curser and return 0.

**bfs.c**

Most of the functions we used for this project were from the bfs.c file. Some bfs functions like bfsAllocBlock we did not need to call by us because they were not part of bfsRead or bfsWrite.

**bio.c**

This file had two functions bioRead and birWrite. These were used to read and write directly to the BFSDISK file that was pretending to be a Hard Disk Drive. We called bioWrite in a for loop in fsWrite. We did not use birReead directly in fsRead.

**main.c**

Only calles two functions bfsInitOFT which initiates the open file table and p5test which starts running the tests.

**p5test.c**

This contained 6 tests as well as some functions like check and checkCursor that helped the tests not duplicate code.

**errors.c and errors.h**

Errors.h defined 21 possibl error types and errors.c used a switch to print out the correct error type.

**alias.h**

This defined the i8, i16 and i31 integer types that we used for the program. It also defined unsigned ints, but we did not use them.