Overview of BFS

The **BFS: “Bothell File System”** consists of 3 levels: **FS, BFS, and BIO.**

FS

FS is the highest level where the user can perform methods such as fsRead and fsWrite. This level contains methods that utilize methods from the BFS level in order to perform different tasks such as fsOpen and fsSeek. Given a file descriptor, FS will use methods from the BFS level in order to read or write a file at any offset given any number of bytes.

BFS

BFS is a level below FS and is in charge of mapping and allocating data blocks to files and is able to locate a file’s location and access it by utilizing the metablocks. For each file stored, there is a corresponding inode that contains the size of the file and the DBNs that contain the file. BFS can also read and write any file using bioRead and bioWrite at any location on the disk.

Bio

Bio is the lowest level and directly accesses the BFSDISK through bioRead and bioWrite. Bio manages the file BFSDISK by reading and writing directly to each of the 100 blocks that are each 512 bytes in size. Due to Bio being a low level simplistic system, it is only possible to read and write to a full block of 512 bytes. This limitation is a key in the implementation of FS due to having to get around the difficulty of full block writes and reads.

FSRead

The FSRead method will read a certain amount of bytes from a given file. This method supports cases where:

- Reading from one block
- Reading from multiple blocks
- Reading less bytes than amount specified
- Reading from a block with an offset

Our implementation of FSRead based on the limitations of our file system breaks down to 3 stages, **“Mid-Block Cursor Read”, “Full-Block Read”,** and **“Final Block Read”**.

Mid-Block: Mid-Block read is specifically first as it is the case that the cursor is currently in the middle of an existing block meaning that the read must start from that cursor's location.
Full-Block: Full-Block read is after Mid-Block (only if the cursor was in the middle of a block) and the number of bytes left to be read is greater than 512 (the size of a full block in our

system). This read takes a full block from BFS and reads it straight into the buffer and recounts how many bytes are left to be read.
Final Block: Final Block read is the final is after both previous stages have been completed and there are still bytes left to be read. This means that the number of bytes is greater than 0 but less than or equal to 512 which would mean that this is the final block FSRead needs to access.

In each case, we used many temporary variables to store values such as, the number of bytes left to read, the number of bytes a cursor is inside a block, and an index to tell which block we are traversing through in BFS. There is also a final check for reads that hit the end of the file and return the accurate number of bytes read.

FSWrite

The FSWrite method will write to a certain file. This method supports

- Writing to an existing file and overwriting bytes (Inside)
- Writing to an existing file and extending blocks (Overlap)
- Writing to an existing file and extending blocks and assigning gaps between start of write and end of original file to zero (Outside)

Our implementation of FSWrite based on the limitations of our file system is similar to FSRead but has some extra steps in the “Write/Read” execution. The way that we implemented the write is reading the specified block into a temporary buffer, copying bytes from the parameter buffer into the temporary buffer, possibly overwriting certain bytes, and finally using bio to write back in the modified temporary buffer back into the file system. FSWrite follows the Mid, Full, and Final block stage flow but with writes instead.