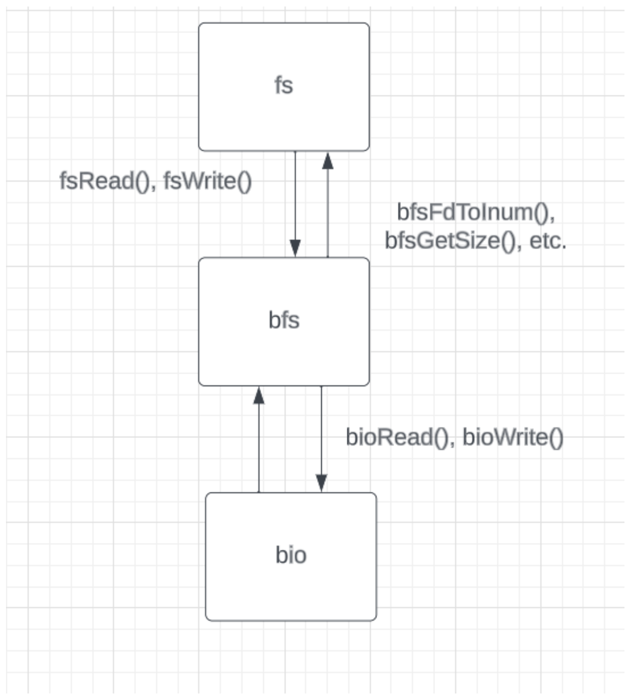
Han S. Pham
CSS 430 B Au 23: Operating Systems
Professor Dimpsey
December 8, 2023

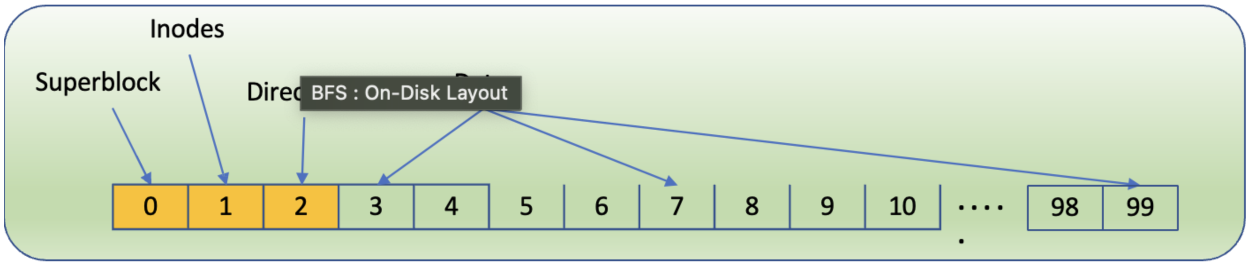
Program 5: Bothell File System Design Documentation

System Overview



In this file system architecture, the bio (Block I/O), bfs (Bothell File System), and fs (File System Interface) layers work together in a hierarchical manner. The bio layer, at the lowest level, manages direct block-level interactions with the physical disk, handling fundamental read and write operations. On top of it is the bfs layer that translates these block operations into file-level actions, managing file structures and metadata, such as inodes. Finally, the fs layer serves as the user-facing interface, providing high-level file operation functions like fsRead() and fsWrite(). This layered approach not only abstracts complexity but also ensures modularity and a clear separation of responsibilities within the filesystem.

BFS: On-Disk Layout



BFSDISK: This represents the file used by the host operating system to simulate a block storage device for the filesystem. It acts as the virtual disk on which the filesystem is stored.

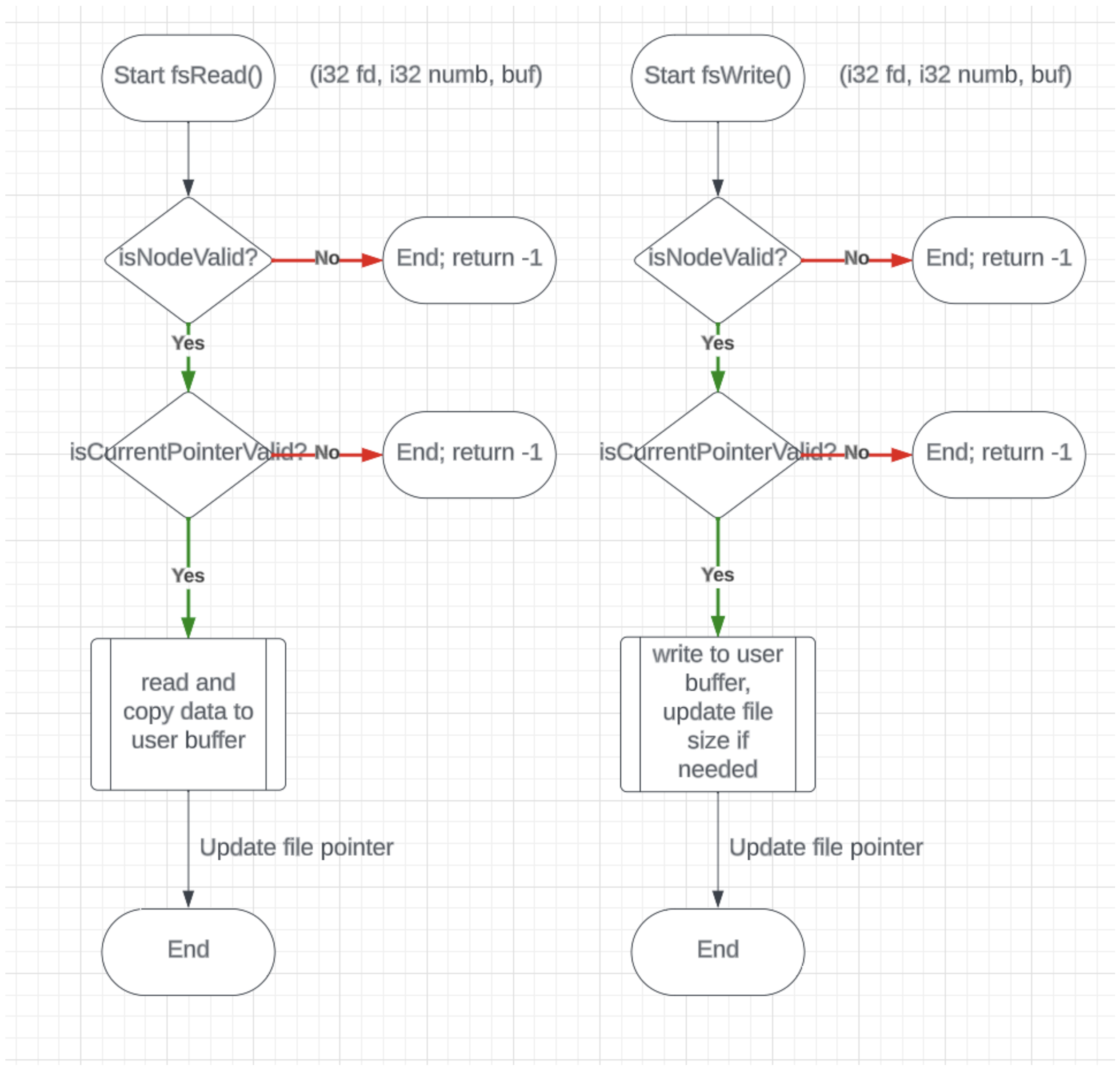
100 blocks: The BFSDISK is divided into 100 segments or "blocks," which are the basic units of storage within the filesystem. Each block can store a fixed amount of data. The first three blocks are Superblock, Inodes, and Directory blocks, and are used to hold metadata.

512 bytes per block: Each block on the BFSDISK can contain 512 bytes of data. This size determines how much information can be stored in a single block and affects how data is organized on the disk.

DBN 0 = Superblock: DBN stands for Disk Block Number. Contains information about the filesystem such as its size, the number of blocks, block size, and information on how many blocks are free or in use.

DBN 1 = 8 Inodes: DBN 1 is used to store inodes. Stores information about a file or a directory, such as its size, permissions, modification time, and disk block locations. DBN 1 contains enough space to store 8 inodes.

DBN 2 = Directory, holding 8 names: DBN 2 is used as a directory block, which holds names or entries. It can hold information on 8 different files or directories, including their names and possibly their inode numbers or pointers to their inodes, effectively acting as the mapping between human-readable file names and the system's file metadata and data location.



fsRead()

The fsRead flowchart depicts the process of reading data from a file in the filesystem. It begins with the function receiving file descriptor (fd), number of bytes to read (numb), and a buffer pointer (buf). The flowchart then proceeds to retrieve the inode number corresponding to fd and checks for validity. Following this, it calculates the start and end blocks for the read operation within the file, adjusting for the file size to ensure the read does not exceed the file boundaries. The function then allocates a buffer to temporarily store the read data. The core functionality involves looping over each file block, reading data into the buffer, and handling potential read errors. Finally, the data is copied from the read buffer to the user-provided buffer, the file pointer is updated to reflect the read operation, and the number of bytes read is returned, concluding the process.

fsWrite()

The fsWrite flowchart outlines the procedure for writing data to a file. This process starts with fsWrite receiving a file descriptor (fd), the number of bytes to write (numb), and a data buffer (buf). Initially, it retrieves the corresponding inode number and verifies its validity. The flowchart then shows the calculation of the start and end file blocks, determining if the file needs to be expanded to accommodate the write. A buffer is allocated for the write operation, into which the data from buf is copied. The flowchart details the writing of this data to the file, block by block, while checking for and handling any write errors. If the file was expanded, its size is updated. The file pointer is then moved forward by the number of bytes written, and the function concludes by returning a success indicator, marking the end of the write operation.