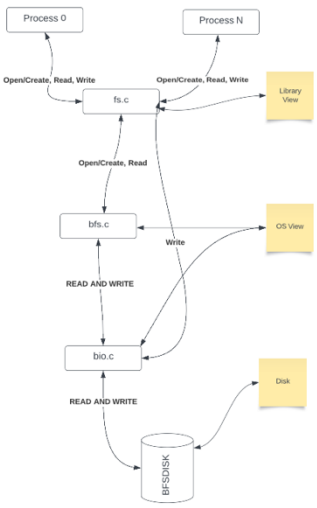
FILE SYTEM IMPLEMENTATION DESIGN

Adilet Kuroda & Nathaniel Fincham

Overview

Figure 1



This project implementation design for layered file system. This file system consists of three layers: block IO level (BIO), raw IO Level (BFS) and user level (FS). BIO level provides a functionality like read or write raw data from the disk. BFS level manages internal layout of file system including directory, inodes, open file table (OFT) and meta data (superblock) regarding entire file system. FS layer provides an API to user to interact with file system to perform open, create, read, and write operations. Please refer to layout of the in Figure 1. Disk (BFSDISK) consists of 100 blocks and each block's size is 512 bytes. First three blocks hold information about inodes, superblock, directory. Superblock holds meta data about the disk including total number of block available in given disk, number of inodes and data block number of first available (free) block. Free block implemented utilizing linked approach. Inode contains information about file: size, 5 direct access and 256 indirect access to different block which can have the size of 261*512bytes. Directory hold I node number for each file and file name.

Block IO Level (BIO)

BIO level provides two functionalities to interface with the disk: bioWrite and bioRead.
Both functionalities read and write unstructured raw data to/from the disk. Both reads from or write to entire block.

I32 bioRead (i32 dbn, void* buf):

This function ensures that there is valid data block number. Once validated, it opens the disk and find the starting position of the block and reads the entire block (512 bytes) to buffer (buf) and closes the connection to disk. On success, returns 0. In any failure, it aborts.

I32 bioWrite (i32 dbn, void* buf):

This function ensures that there is valid data block number. Once validated, it opens the disk and find the starting position of the block and writes 512 bytes from buffer to given block and closes connection to disk. On success, returns 0. In any failure, it aborts.

Raw IO level (BFS)

BFS manages internal structure of the file system. It utilizes BIO level to interact with BFSDISK. It has open file table (OFT) that keeps track of all the open files and manages super block (meta data about the file), directory and inodes. In addition, it provides functionality like creating file, reading from disk, extending existing file, allocating new blocks in a disk and updating cursor position.

Inode Management

Inode initialized by bfsInitInode function, which writes 512 bytes of I8's initialized to 0 to block 1 on the disk with bioWrite. Once initialized, we can access the inodes with bfsReadInode and update it with bfsWriteInode function, which utilizes bioRead and bioWrite to update the disk.

Directory Management

Directory is initialized with bfsInitDir function which writes 512 bytes of I8's initialized to 0 to block 2 on the disk with bioWrite (). When file is created with bfscreateFile, directory gets updated.

Super Block Management

Super block gets initialized by bfsInitSuper where it writes metadata as I mentioned on overview section will be written to block 0 utilizing bioWrite. Free block section of super block gets updated every time when there is request for free space with bsfFindFreeBlock function.

Open File Table Management

Open file table is array of structs. Each struck hold inode number (inum) of the file, reference (number of processes opened the file) and current position of the cursor. OFT get initialized by bfsInitOFT where inum sets to -1 and rest of the fields get initialized to 0. When file is opened, it gets updated by bfsRefOFT, which calls bfsFindOFT, which either find the file that is already open or add file to OTF and bfsRefOFT increments the reference to file. When file closes, bfsDerefOFT decrements the reference field and if there is not reference than it removed from OFT. Once the file open, cursor field can be access with bfsTell and updated with bsfSetCursor functions.

Creating, Reading, and Extending file

BFS section provides interface to create, read, and extend a file. These functionalities will allow FS level to interact with the disk.

I32 bfsCreate(str fname):

This function creates file by adding file name to the directory and referencing the file to OFT table.

I32 bfsExtend(i32 inum, i32 fnb):

This function allows to extend file from last file block number new parameter fnb (file block number). Every iteration, it calls bfsAllocBlock to allocate/extend memory in the disk for given file. During the allocation, initially bfsAllocBlock finds the free data block number (DBN) going into the super block and accessing and updating free block field using bfsFreeBlock function. Once DBN obtained, DBN is added to files either direct or indirect block depending on how many blocks are allocated to given file.

I32 bfsRead(I32 inum, I32 fbn, i8* buf):

This function read 512 bytes from disk utilzing bioRead and copies the content of read data into buf. Based on inum and fnb, it determines the DNB utilzing bfsFbnToDbn function.

User level file system (FS)

This level offers interface for user to interact with the disk. If user needs to format the disk, user uses fsFormat to reset all the information about the disk to default utilzing BFS layer functorialities. To open file, fsOpen provides interface, where it fetches inum from directory using bfsLookUp file. To close the file, it deference's file from OFT using bfsDerefOFT. To create file, it utilized I32 bfsCreate to add to directory and OFT. To update get the current position of cursor, fsTell is provided, which fetches cursor position from OFT utilizing bfsTell. To update the cursor fsSeek is provided to change the cursor position utilizing bsfSetCursor. In addition, user can write and read from the disk.

Reading from disk:

Refer to Figure 2 for flow diagram and below will provide functionality details.

I32 fsRead(i32 fd, i32 numb, void* buf):

This function uses BFS functionalities to read number of bytes from the disk. Initially identifies current position of the cursor and determines the size of the file. If the number requested to read is less than from current cursor position to the end, then it ensures only to read until end of the file. Based on cursor position file block number is identified and starts reading block by block from the disk starting from given position to either to end of the file or until all the requested bytes read. Read utilizes bfsRead function and since it only read one block at a time, new read data is appended(copied) into buffer to that user can use it later.

Figure 2

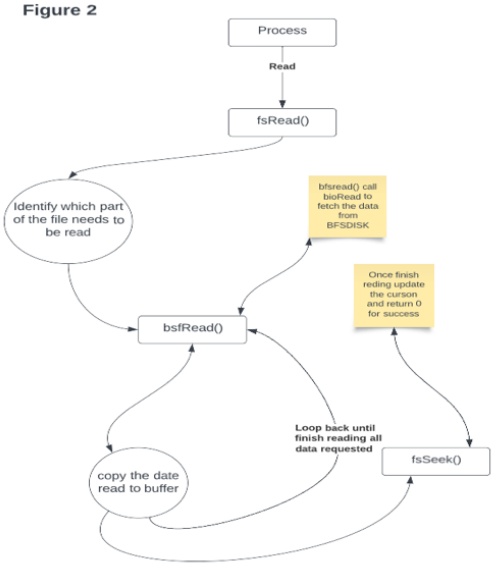
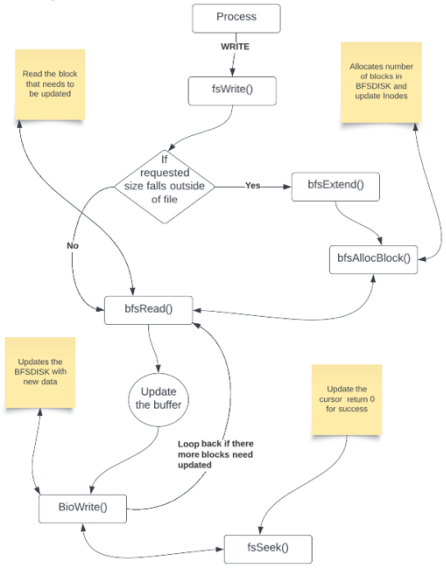


Figure 3



Writing disk:

Refer to Figure 3 for flow diagram and below will provide functionality details.

I32 fsWrite(i32 fd, i32 numb, void* buf):

Initially this function checks if from current position to end of the file is sufficient to write the data from buffer. If not, it extends the file using bfsExtend, which find and allocates memory. Refer to bsfExtend for more details. Once there is enough space to write, this function starts writing the content of buffer to disk using bioWrite. Based on different position of the file and requested size, it might only update certain part of the block and remaining information will stay same.