Test5 Result

A close up of text on a black background

Description automatically generated

# Assumptions and Limitations

One limitation is the size of the file system. There are only 1000 blocks of 512 bytes. So, the system can’t handle larger amounts of data. This is also limited by the fact that the inode only has one indirect and 11 direct data blocks. Another limitation is parent and child processes will share the same files so writing to a file will affect both processes. Additionally, there is no mechanism to duplicate files. The file system has to perform 8 basic system calls (). The manipulations on files can be done by multiple user threads. The file system must provide stream-oriented files for user thread programs. The file system has to maintain the list of all open files. Users cannot edit the same file simultaneously. There is no user interface that would make working with the file system much easier. This file system can only work on ThreadOS. The user won’t modify the current functions inside ThreadOS. Inability to create user permissions so that only certain users can access certain files

# Internal Design

The file system holds blocks of data. The first block is the superblock which holds data about the rest of the file system. The next 4 data blocks hold inodes which are smaller blocks of data that hold data pertaining to a single file. The 5th block holds the directory which holds data about the organization of all the files. The rest of the blocks hold the actual data of the files. The inodes have data that maps to the file.

# Directory

The directory holds info about all the files and keeps track of which files are being used. It contains two arrays, one to hold each file name size and the other one is a two-dimensional array that contains files name. First row in both holds the root of the directory “/”. Array index corresponds to iNumber, thus can be also used to keep track of available inodes. Since the directory is a file info about it is stored in inode 0 of disk block 1.

# FileSystem

FileSystem is in control of actions including open, close, read, write, append, seek, delete and format. File systems bring together all other classes, it instantiates an instance of the SuperBlock, Directory and the FileTable. It has access to FileTable, Directory and SuperBlock and through FileTable it can grab each FileTableEntry and its inode. It maintains the FileSystem by providing interface to the system calls. It provides an easy to user list of operations that users can use. This is the overarching class that will create the other classes in the FileSystem.

# FileTable

This class holds a list of FileTable entries. Each of these entries represent one file descriptor. It is responsible for allocating and removing file entries from a vector when needed. It keeps track of files (FileTableEntry) that are currently opened. It allocates a new FileTableEntry for new files to the table, inserts the new file into the directory and removes it from the FTE when a thread is finished using the file (close file, not delete). It is shared table among all threads that keeps track of all the files in the system. It consists of FileTableEntry objects which hold an inode of a specific file in addition to the mode the file is in (read-only, write-only, read & write, append).

# FileTableEntry

File​ ​table​ ​entry​ ​is​ ​actually​ ​an​ ​object​ ​presentation​ ​of​ ​an​ ​inode​ ​(file)​ ​in​ ​the​ ​memory. ​ ​​Each​ ​file descriptor​ ​should​ ​correspond​ ​to​ ​exactly​ ​ONE​ ​of​ ​FileTableEntry​ ​because​ ​each​ ​file descriptor​ ​can​ ​open​ ​its​ ​file​ ​in​ ​its​ ​individual​ ​mode. ​ ​It​ ​contains​ ​seekPtr, ​ ​inode, ​ ​iNumber, count​ ​of​ ​the​ ​threads​ ​that​ ​are​ ​working​ ​on​ ​this​ ​file​ ​entry​ ​and​ ​mode​ ​of​ ​the​ ​operation.

# Inode

Each​ ​file​ ​in​ ​the​ ​file​ ​system​ ​will​ ​be​ ​represented​ ​by​ ​an​ ​inode. ​ ​Each​ ​inode​ ​contains​ ​32-byte size​ ​of: ​ ​file​ ​size, ​ ​flag​ ​to​ ​indicate​ ​if​ ​it’s​ ​in​ ​unused/use/read/write, ​ ​number​ ​of​ ​threads​ ​that are​ ​using​ ​this​ ​file​ ​currently, ​ ​total​ ​of​ ​11​ ​direct​ ​pointers​ ​and​ ​1​ ​indirect​ ​pointer​ ​that​ ​points​ ​to blocks​ ​of​ ​Data. Each inode keeps track of one file in the system. Keeps track of information such as whether it has been used, read, or written to as well as keeping track of how many threads are accessing the file.

# Kernel

Inside​ ​Kernel​ ​had​ ​to​ ​add​ ​different​ ​cases​ ​for​ ​the​ ​given​ ​flag​ ​by​ ​SysLib. ​ ​All​ ​had​ ​to do​ ​was​ ​to​ ​check​ ​for​ ​the​ ​edge​ ​cases​ ​and​ ​call​ ​the​ ​related​ ​function​ ​to​ ​the​ ​current​ ​flag​ ​from FileSystem.

# SuperBlock

The superblock is a block of information that describes the FileSystem and its variables. SuperBlock is the class that formats the disk. The superblock is responsible for reading from the disk and making sure that there are no errors read from the disk about the FileSystem. If it detects an error, it will format the block and all the Filesystem. Responsible for allocating and deallocating blocks, setting total blocks and the available inodes for each block, and keeping track of current free blocks available at all times. This class is a strictly OS managed and no user thread is allowed to access or interact with this class. SuperBlock contains information regarding the number of disk blocks, number of inodes and block number for the head block of the free list. On disk, SuperBlock holds the first position at disk[0].

# SysLib

SysLib​ ​is​ ​the​ ​file​ ​that​ ​calls​ ​most​ ​functions​ ​for​ ​kernel. ​ ​It was​ ​added​ ​missing​ ​interrupt​ ​calls needed​ ​to​ ​run​ ​the ​file​ ​system​ ​such​ ​as​ ​format, ​ ​open, ​ ​close, ​ ​write, ​ ​read​, ​and​ ​delete.

# Performance

There are no actual numbers regarding how fast the performance is because there were too many factors that could have affected the speed. Formatting re-configures and re-initializes the disk to whatever the value of total blocks was sent in. There are ways (which are complicated) in which to more frequently update the disk in order to improve overall performance.

# Current functionality

# Current FileSystem has the ability to format and creates a blank disk data. It also is able to write read and append to a file. Files can be written to and read from any point within the data of the file. Space allocation is set for a certain size of files but can change if the file is large enough.

# Possible Extend Functionality

Possible Extends could be to have a user interface to make using the current functions easier. One possible extended functionality can be adding double index or triple index. Blocks to increase the file size that can store. Adding a GUI can be another Extended Functionality for the current FileSystem. A simple to use GUI can enhance the user experience. It could also add services for other software the O.S might encounter.