# **Problem Statement**

**Build a file system API, that has exception handling.**

# **Requirements**

# **Assumptions**

- Each sector in the disk takes up 512 bytes

- There are 1000 sectors in each disk

- the sequence of blocks in the drive are superblock, indoe bitmap, data block bitmap, 3 indoes, and the rest are data block

- Each directory will have 20 bytes, 16 for name and 4 for inode

- datablock come in two types:

- Buffer is 512 bytes

- only 10 files can be opend at a time

- only 256 characters for path name

# Specifciations

- FS\_BOOT() :: Check if external disk is made and if not, make it. Copy it to wrokign disk. If external disk exist BUT it's superblcok si wrong, osErrMsg=E\_FILE\_BOOT [FS]

- FS\_SYNC() :: Copys the working disk to the external disk. [FS]

- FS\_RESET() :: Call FS\_SYNC(), than make file system unavaible till FS\_BOOT(). If the system already is unavaible, return E\_FILE\_RESET osErrMsg [FS]

- File\_Create(string path) :: create a file to the path. if this fails, due to there already being a file in the path, return with the E\_FILE\_CREATE errorr [FILE]

- File\_Open(string path) :: Put that file in the open file table [FILE]

- File\_Read(int fd, string buffer, int size) [FILE]

- File\_Create(string path) :: create a file to the path. if this fails, due to there already being a file in the path, return with the E\_FILE\_CREATE errorr [FILE]

- File\_Open(string path) :: Put that file in the open file table [FILE]

- File\_Read(int fd, string buffer, int size) [FILE]

- File\_Write(int fd, string buffer, int size IN BYTES) :: Write from buffer to the file. NOTE SIZE HAS TO BE CONSISNET. If it's not, stop the program [FILE]

- File\_Seek(int fd, int offset) :: move the file forward by offset. [FILE]

- File\_Close(int fd) :: Remove file from table [FILE]

- File\_UnLink(string path) :: Delete file from the filesystem. [FILE]

- Dir\_Create(string path) :: Create directory at path [DIR]

- Dir\_Read(string path, string buffer, itn size) :: Read the contents of a directory. [DIR]

- Dir\_Unlink(string path) :: Remove file from drive [DIR]

- DISK\_INIT() :: Set all the data in the disk to be 0 [DISK]

- DISK\_LOAD() :: Save external disk to workign disk. Done when booting. [DISK]

- DISK\_SAVE() :: Save working disk to loading. Called by FS\_SYNC() [DISK]

- DISK\_WRITE(int sector, string buffer) :: Write from buffer to disk. [DISK]

- DISK\_Read(int sector, string buffer) :: read from sector to buffer [DISK]

# Test Cases

## Bit manipulation [BITSTREAM]

### Bit manipulation of Inode [INODE]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | Read in a stream of bits, and convert to an inode object | Stream of 114 bits | Create an inode object, by the specified 114 bits. |  |  |
| valid | 2 | Write as a stream of bits from inode object | Inode object | write as a stream of 114 bits |  |  |

### Bit manipulation of Directory [DIR]

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 3 | Read in a stream of bits, and convert to an directory object | Stream of 4096 bits | Create an directory object, by the specified 4096 bits. |  |  |
| valid | 4 | Write as a stream of bits from directory object | directory object | write as a stream of 4096 bits |  |  |

## Setting up the drives

### File system scope [FS]

Error Messages are sent to osErrMsg

1. FSBoot

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 4 | The external disk isn’t made, and so create one and rerun this function again | Success NoDisk | The external disk does not exist, creates it |  |  |
| Valid | 5 | the external disk exist and has correct magical number | Success Disk | The external disk does exist, clone it to working test and verify |  |  |
| Invalid | 6 | the external disk exist and has incorrect magical number | Failure Disk | The external disk does exist, clone it to working test and verify, but fails, resulting in EFILEBOOT |  |  |

1. FSSync

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 7 | Saves the working disk to external disk | Working disk and external disk | Copy working disk to external disk |  |  |

1. FSReset

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| valid | 8 | Attempt to access file system after FSRESET() but before FSBOOT() | FSReset() and no FSBoot() | EINVALIDACCESSATTEMPT |  |  |
| invalid | 9 | Attempt to access file system after FSRESET() and after FSBOOT() | FSReset() and FSBoot() | File system boots |  |  |

### Disk Scope [DS]

Error Messages are sent to diskErrMsg

1. DISKSETUP and Save

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 10 | Set all the bits to the disk to 0, and creates root directory | int DISKINIT() | Run before disk ops take place. All sectors are all zeros, and you create he superblock and the root directory. Called during FSBoot |  |  |
| Valid | 11 | External disk saves to working disk | int DiskLoad | When booting a Disk, you send External Disk -> working Disk. Called during FSBoot |  |  |
| Valid | 12 | Working disk saves to external disk | int DiskSave | Called by FSSync, send Working Disk -> External Disk. |  |  |

1. DISK Write

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Invalid | 13 | DiskWrite(), but size too small. EXCEPTION | Buffer is less than SIZESector | return EWRITEINVALIDPARAM |  |  |
| Invalid | 14 | DiskWrite(), but buffer is NULL. EXCEPTION | Buffer is NULL | If not, than return EWRITEINVALIDPARAM |  |  |
| Invalid | 15 | DiskWrite(), but sector parameter is not valid. EXCEPTION | Sector parameter is out of bounds | return EWRITEINVALIDPARM |  |  |
| Valid | 16 | DiskWrite(), and it’s correct size, buffer is not null, and sector parameter is within [0,100) ] | Sector has data being written to it | Write the Buffer to Sector. |  |  |

1. DISK Read

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Invalid | 17 | DiskRead(), but size too small. EXCEPTION | Buffer is less than SIZESector | return EREADINVALIDPARAM |  |  |
| Invalid | 18 | DiskRead(), but buffer is NULL. EXCEPTION | Buffer is NULL | If not, than return EREADINVALIDPARAM |  |  |
| Invalid | 19 | DiskRead(), but sector parameter is not valid. EXCEPTION | Sector parameter is out of bounds | return EREADINVALIDPARM |  |  |
| Valid | 20 | DiskRead(), and it’s correct size, buffer is not null, and sector parameter is within [0,100) ] | Sector has data being written to it | write the sector to buffer. |  |  |

## Setting up Directories and files

### DIR [DIR]

1. Directory Create

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| valid | 23 | DirCreate(path) and it doesn’t already exist, the parent does not exist, and does not exceed 256 characters | Go to parent path File does not exist | Create directory at parent directory |  |  |
| invalid | 24 | DirCreate(path) but path already exist | Go to parent path Failure Exist | Get the file of the parent path, and once you do that, you see the file Path already exist. Than, you return EDIRCREATE |  |  |
| invalid | 25 | DirCreate(path) but the parent does not exist | Go to parent path Failure Bad Path | Get the file of the parent path, but you the parent path doesn’t exist. Than, you return EDIRCREATE |  |  |
| invalid | 26 | DirCreate(path) but path does not exist | Path exceeds 256 characters | If the supposed path is greater than 256 characters, or 2048 bits, return ETOOLONGPATH |  |  |

1. Directory Size/Read

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| valid | 27 |  | DIRSIZE() returns value. (NOTE THIS FUNCTION ONLY IS CALLED BY DIRREAD AND SHOULD ONLY WORK WHEN DIRREAD DETERMINED THE PATH IS VALID) | Gets a size of the directory, and sets hate buffer to appropriate size. Note that each entry in a directory is 20 bytes |  |  |
| valid | 28 |  | DIRREAD Success | The buffer is of appropriate size, and as such, you can write each inode to it. |  |  |
| invalid | 29 |  | DIRREAD Failure Size too small | The buffer is of too small size, return EBUFFERTOOSMALL |  |  |
| invalid | 30 |  | DIRREAD Failure Dir no exist | directory does not exist, return EDIRNOEXIST |  |  |

### Files [FILE]

1. Create/Open and Close

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
|  | 31 |  | FileCreate(string File) Success | Create a file at the path, with size 0. |  |  |
|  | 32 |  | FileCreate(string File) Failure already exist | Failed at creating a file, due to there already being a file. Return EFILECREATE |  |  |
|  | 33 |  | FileCreate(string File) Failure max file size | The files system reaches max files. Return EFILECREATE |  |  |
|  | 34 |  | Path exceeds 256 characters | If the supposed path is greater than 256 characters, or 2048 bits, return ETOOLONGPATH |  |  |
|  | 35 |  | FileOpen success | File doe exist and doesn’t’ exceed files open limit |  |  |
|  | 36 |  | FileOpen Failure noExist | File does not exist, return ENOSUCHFILE |  |  |
|  | 37 |  | FileOpen Failure already Open | File already is opened. return EFILEALREADYOPEN |  |  |
|  | 38 |  | FileOpen Failure too many open files | File does exist, but there’s too many open files. return ETOOMANYOPENFILES |  |  |
|  | 39 |  | FileClose(int fd) Success | Close file in open file table, with fd. |  |  |
|  | 40 |  | FileClose(int fd) Failure | File is not in open file table, return ECLOSEBADFD |  |  |

1. File Read, write

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
|  | 41 |  | FileRead(int fd, string buffer, int size) Success | check Open file, go to current position in file, than read from current position to size/end of file. Than have current file position |  |  |
|  | 42 |  | FileRead(int fd, string buffer, int size) Failure not open | File is not in open file table, and thus, return EREADBADFD. |  |  |
|  | 43 |  | FileWrite(int fd, string buffer, int size) Success | Write to a file from buffer, from the buffer. Current file offset should be by size. |  |  |
|  | 44 |  | FileWrite(int fd, string buffer, int size) Failure not open | Write to a file from buffer, from the buffer. Current file offset should be by size. Return EWRITEBADFD |  |  |
|  | 45 |  | FileWrite(int fd, string buffer, int size) Failure no space left | While writing, if you find there’s no more space in the disk, return ENOSPACE |  |  |
|  | 46 |  | FileWrite(int fd, string buffer, int size) Failure maximum file size | While writing, if you find there’s the file takes up more than 10 data blocks, return EFILETOOBIG |  |  |

## Seek and UnLink

### FileSeek and Dir/FileUnLink [FILE : DIR]

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| valid | 47 | Remove file | FileUnlink(String File) | Remove file in inode block, and freeing up any data blocks/indoes the file used. |  |  |
| invalid | 48 | Remove file BUT THE FILE DOES NOT EXIST | FileUnlink(String File) but no such file | File does not exist. Return ECLOSEBADFD |  |  |
| invalid | 49 | Remove file but file is in openpagetable | FileUnlink(String File) but file is already opened | File is currently opened. Return EFileINUse |  |  |
| valid | 50 | remove directory | DirUnlink(String File) Directory is empty | Remove file from parent inode pointers, and than free up the inode/data blocks. |  |  |
| invalid | 51 | remove directory but it’s empty | DirUnlink(String File) Directory is not empty | Return EDIRNOTEMPTY |  |  |
| invalid | 52 | remove directory but it’s root | DirUnlink(String File) Directory is root | return EDELROOTDIR |  |  |
| invalid | 53 | remove directory but directory never existed | DirUnlink(String File) Directory does not exist | return EDIRNONEXIST |  |  |
| valid | 54 | file seek and it has a offset that’s within 0 to file size | FileSeek(int fd, int offset) | File’s position is changed by the offset |  |  |
| invalid | 55 | file seek but it has a offset that’s without 0 to file size | FileSeek(int fd, int offset) Out of bounds | Offset is negative/exceeds file size. Return ESEEKOUTOFBOUNDS |  |  |
| invalid | 56 | file seek and it’s fd is not in page table | FileSeek(int fd, int offset) bad fd | File isn’t open. Return out of bounds |  |  |

# Algorithm/Code

## Program General code

* Create the two disk, with each disk having a set of functions regulating to disk.
* File system has the two disk, and it has all the functionality of FS.
* OpenFileTable keeps track of each file, and it’s current position
* As we are writing bits, we have to format the disk to be able to read and write bits.
* SUPERBLOCK | indoebitmap | datablock bitmap | sequence of indoes | sequence of datablock = 1000
* the sequence of indoes will have 3 sectors, due to each Inode being able to represent 35 indoes.
* The rest of the space, 994 sectors, are for the datablock block.

## Bit Parsing/Data Strucutre [BITSTREAM]

### inode

1. writeBitStream()
   1. Write teh type, size and allocation, by reversing the blow operation
2. readBitStream()
   1. read the type, size and allocation by following the following processes
3. Indoe structure
   1. There are 4 indoes within a inode sector. The makeup totals to 114 bits.
      1. 1 bit
         1. for which type of inode this is.
      2. 13 bits
         1. (or 1.625 bytes) for representing the size of data blocks
      3. 100 bits
         1. 10 sequences of 10 bits for representing the location. note that all 1s mean that this is not allocated
4. Fragmentation
   1. This results of 106 of useless data, and 3990 of useful data. Since there are 35 indoes in a sector, we split it up into an array, with each piece being a substr of 114 bits.

### directory

* Data blocks are disgusted by two types: file and directory
* the type of the data block is denoted by the inode, not the directory.
* For directory, there is a 20 bytes/160 bits, which are
  + 16 bytes/128 bits
    - file name. 15 characters PLUS 1 for end of string, so it’s More of 15 characters
  + 4 byte/32 bits
    - inode that shows which file/directory this is.
* This means that dictionaries Can have 25 files in a a sector, but 250 files/directories overall.
* This doesn’t have the case, of half a directory’s information being in one data block, and the other half being in another data block. That isn’t considered.

### bitmap of Indoe/data block

* this is just a bitmap, used to keep track of which indoes are allocated and which data blocks are allocated.

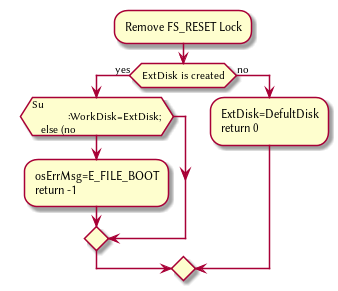
### Sector/Root Inode

* A sector is a collection of a superblock, bitmaps for in use indoes and data blocks, a sequence of indoes, and a sequence of data blocks. However, this information HAS TO BE CONVERTED to that. Otherwise, a sector is just an array of bit sets of 4096 bits.
* However, the sector converts it’s con cats to usable data structures. After each file/directory operation, it saves the stuff to working directory. Than, working directory saves it stuff to external disk when FSSYNC() is made.
* The disks are just a bitset array of 4096 bits, with 1000 elements in each.
* The root inode is the Indoe that represents nothing. This is a special variable, as to not have to find out what it is on disk tediously.

### 

## File System (note I couldn’t get outline to work here) [FS]

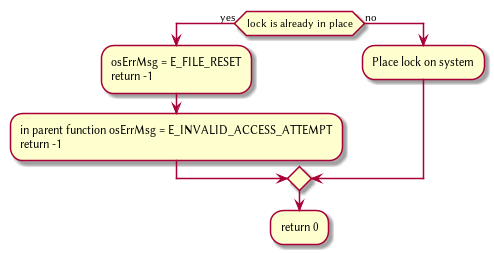
1. FSBOOT()
   1. Called when booting filesystem/after a FSRESET()



1. FSSync
   1. Copys the working disk to external disk

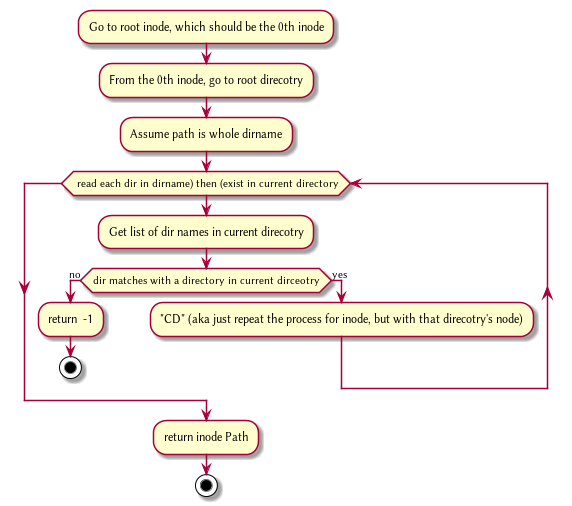


1. FSRESET()
   1. Stops the filesystem from ebing access, by placing a lock on it.

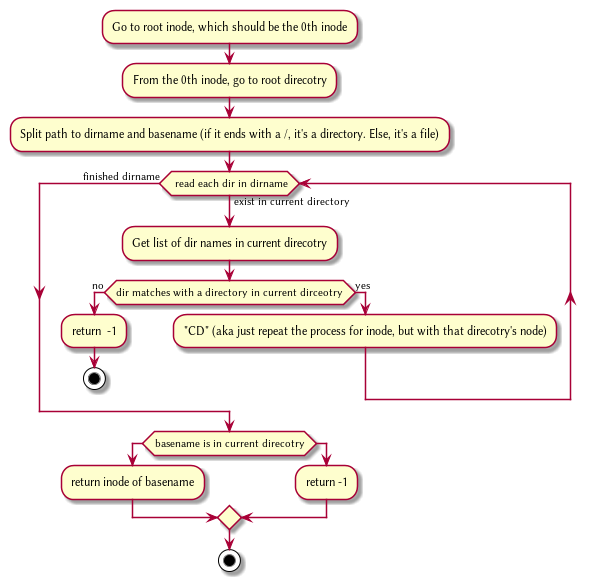


## File Access [FILE]

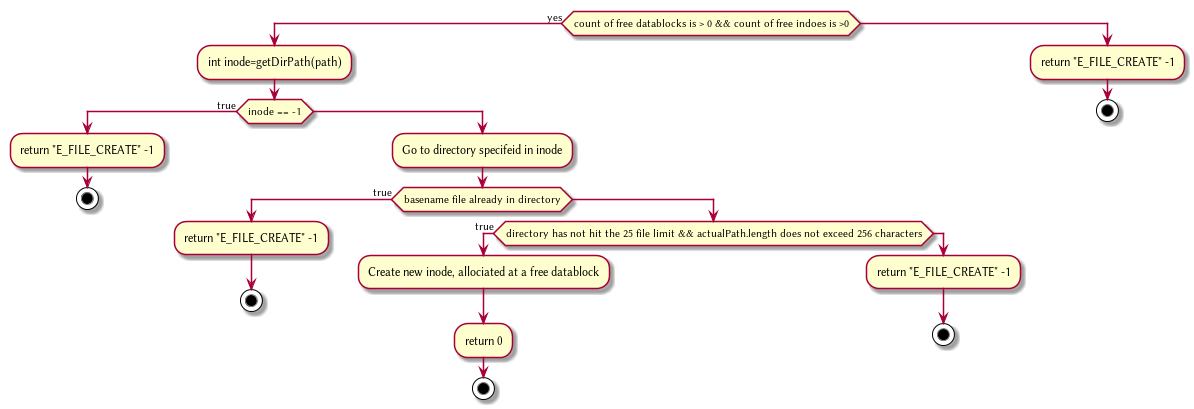
1. int getDirPath(string path)
   1. Helper function, used to get the directory given a path.
      1. Ouptut
         1. inode number of where it is, or -1 if it’s not found.



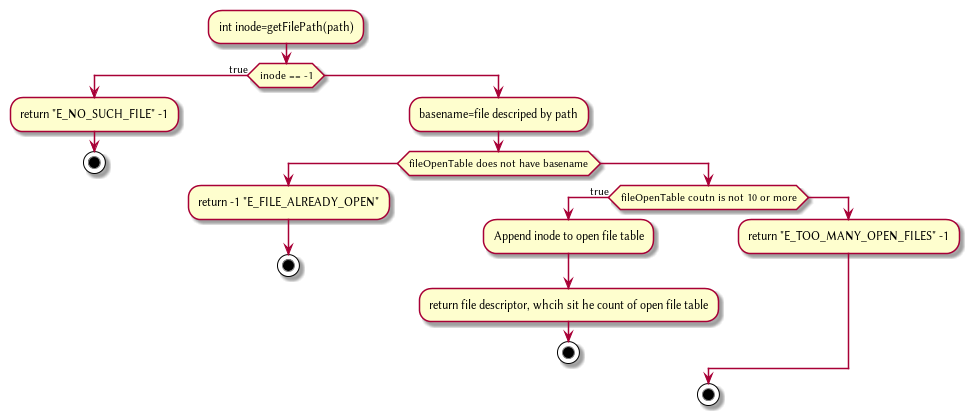
1. int getFilePath(string path)
   1. Helper function, used to get the file given a path.
      1. Ouptut
         1. inode number of where it is, or -1 if it’s not found.



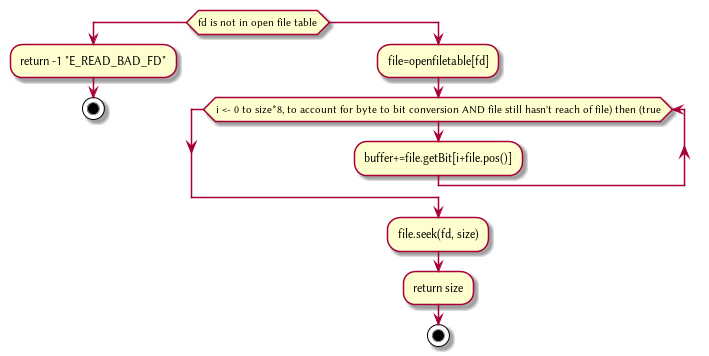
1. FileCreate(string path)
   1. Create a new file at path. There is a check to see if that file already exist, and if there’s a free datablock for it.



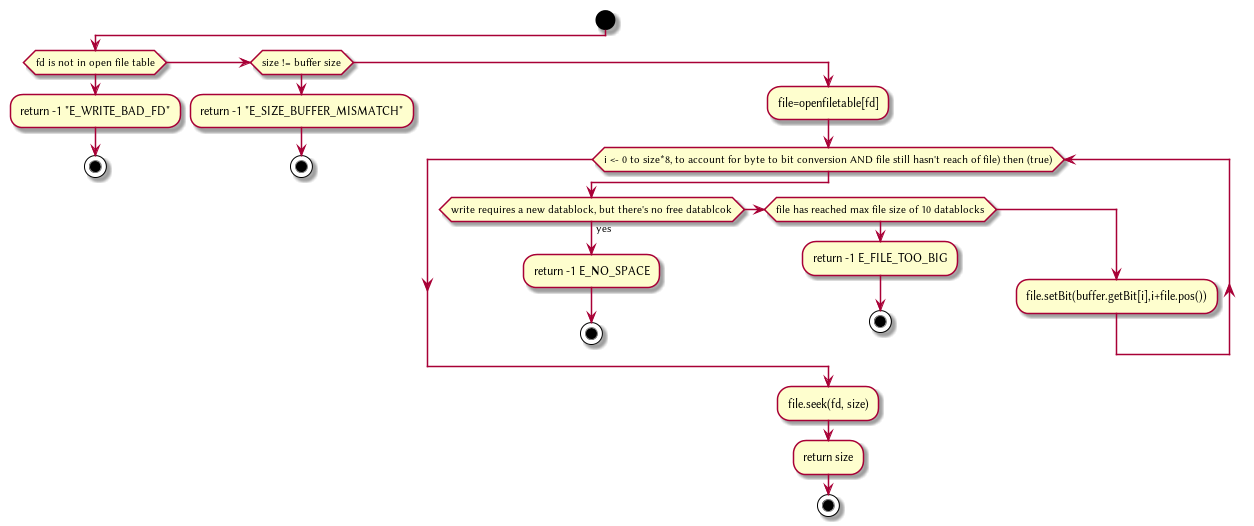
1. FileOpen(string path)
   1. returns the file descriptor of the file, which can be used to read and write to it.



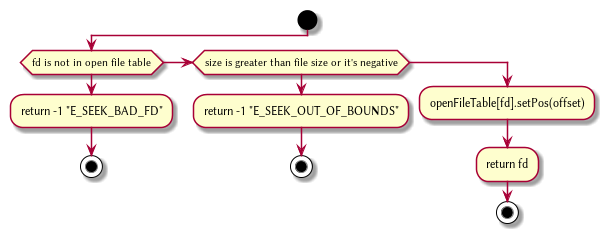
1. FileRead(int fd, string buffer, int size IN BYTES)
   1. Buffer reads size from the file in fd. Note the file in open file table shuold move by size



1. FileWrite(int fd, string buffer, int size IN BYTES)
   1. Write from buffer to the file. NOTE SIZE HAS TO BE CONSISNET. If it’s not, stop the program

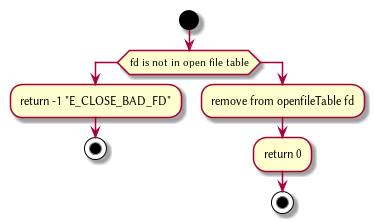


1. FileSeek(int fd, int offset)
   1. move the file forward by offset.

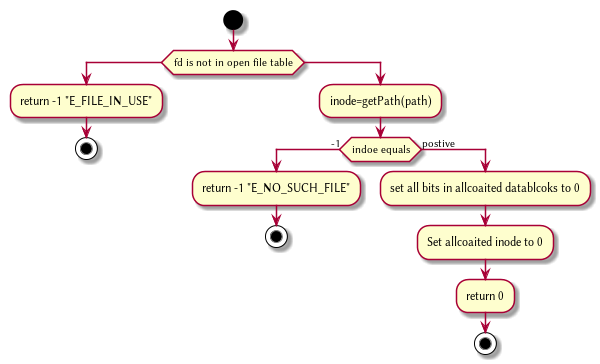


@startuml

1. FileClose(int fd)
   1. Remove file from table

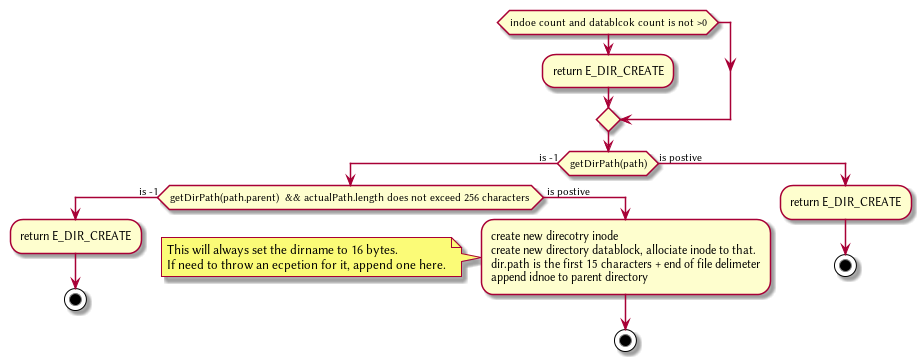


1. FileUnLink(string path)
   1. Delete file from the filesystem.



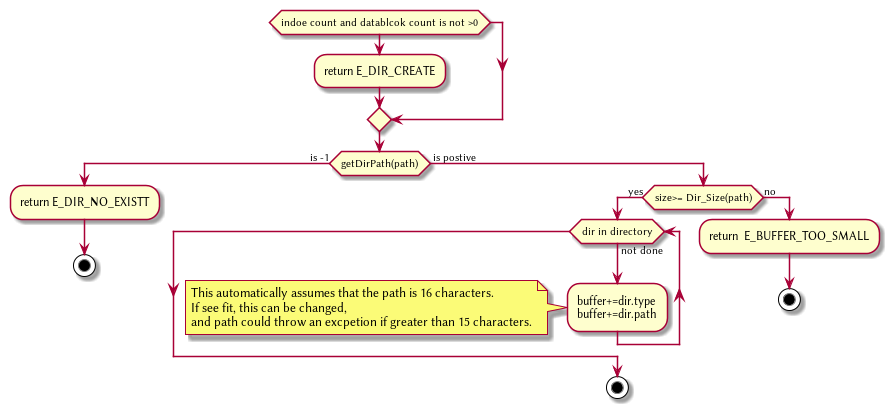
## Directory [DIR]

1. DirCreate(string path)
   1. Create directory at path

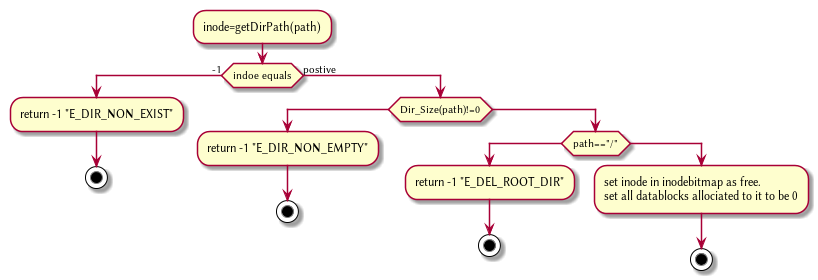


@startuml

1. DirRead(string path, string buffer, itn size)
   1. Read the contents of a directory.

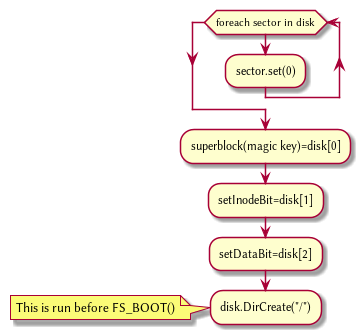


1. DirUnlink(string path)
   1. Remove file from drive

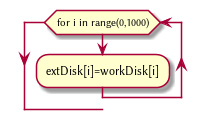


## Disk [DISK]

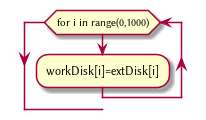
1. DISKINIT()
   1. Set all the data in the disk to be 0



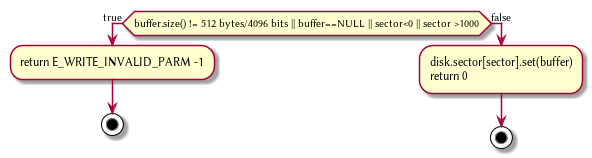
1. DISKLOAD()
   1. Save external disk to workign disk. Done when booting.



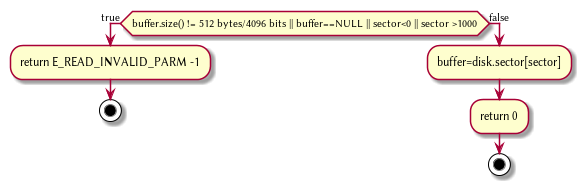
1. DISKSAVE()
   1. Save working disk to loading. Called by FSSYNC()



1. DISKWRITE(int sector, string buffer)
   1. Write from buffer to disk.



1. DISKRead(int sector, string buffer)
   1. read from sector to buffer



# Test Cases 2

## Bit manipulation [BITSTREAM]

### Bit manipulation of Inode [INODE]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | Read in a stream of bits, and convert to an inode object | Stream of 114 bits | Create an inode object, by the specified 114 bits. |  |  |
| valid | 2 | Write as a stream of bits from inode object | Inode object | write as a stream of 114 bits |  |  |

### Bit manipulation of Directory [DIR]

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 3 | Read in a stream of bits, and convert to an directory object | Stream of 4096 bits | Create an directory object, by the specified 4096 bits. |  |  |
| valid | 4 | Write as a stream of bits from directory object | directory object | write as a stream of 4096 bits |  |  |

## Setting up the drives

### File system scope [FS]

Error Messages are sent to osErrMsg

1. FSBoot

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 4 | The external disk isn’t made, and so create one and rerun this function again | Success NoDisk | The external disk does not exist, creates it |  |  |
| Valid | 5 | the external disk exist and has correct magical number | Success Disk | The external disk does exist, clone it to working test and verify |  |  |
| Invalid | 6 | the external disk exist and has incorrect magical number | Failure Disk | The external disk does exist, clone it to working test and verify, but fails, resulting in EFILEBOOT |  |  |

1. FSSync

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 7 | Saves the working disk to external disk | Working disk and external disk | Copy working disk to external disk |  |  |

1. FSReset

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| valid | 8 | Attempt to access file system after FSRESET() but before FSBOOT() | FSReset() and no FSBoot() | EINVALIDACCESSATTEMPT |  |  |
| invalid | 9 | Attempt to access file system after FSRESET() and after FSBOOT() | FSReset() and FSBoot() | File system boots |  |  |

### Disk Scope [DS]

Error Messages are sent to diskErrMsg

1. DISKSETUP and Save

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 10 | Set all the bits to the disk to 0, and creates root directory | int DISKINIT() | Run before disk ops take place. All sectors are all zeros, and you create he superblock and the root directory. Called during FSBoot |  |  |
| Valid | 11 | External disk saves to working disk | int DiskLoad | When booting a Disk, you send External Disk -> working Disk. Called during FSBoot |  |  |
| Valid | 12 | Working disk saves to external disk | int DiskSave | Called by FSSync, send Working Disk -> External Disk. |  |  |

1. DISK Write

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Invalid | 13 | DiskWrite(), but size too small. EXCEPTION | Buffer is less than SIZESector | return EWRITEINVALIDPARAM |  |  |
| Invalid | 14 | DiskWrite(), but buffer is NULL. EXCEPTION | Buffer is NULL | If not, than return EWRITEINVALIDPARAM |  |  |
| Invalid | 15 | DiskWrite(), but sector parameter is not valid. EXCEPTION | Sector parameter is out of bounds | return EWRITEINVALIDPARM |  |  |
| Valid | 16 | DiskWrite(), and it’s correct size, buffer is not null, and sector parameter is within [0,100) ] | Sector has data being written to it | Write the Buffer to Sector. |  |  |

1. DISK Read

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Invalid | 17 | DiskRead(), but size too small. EXCEPTION | Buffer is less than SIZESector | return EREADINVALIDPARAM |  |  |
| Invalid | 18 | DiskRead(), but buffer is NULL. EXCEPTION | Buffer is NULL | If not, than return EREADINVALIDPARAM |  |  |
| Invalid | 19 | DiskRead(), but sector parameter is not valid. EXCEPTION | Sector parameter is out of bounds | return EREADINVALIDPARM |  |  |
| Valid | 20 | DiskRead(), and it’s correct size, buffer is not null, and sector parameter is within [0,100) ] | Sector has data being written to it | write the sector to buffer. |  |  |

## Setting up Directories and files

### getFilePath(string path) and getDirPath(string path)

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Invalid | 21 | path is not in system | String path | -1 |  |  |
| Valid | 22 | path is in system | String path | the inode that’s associated with this file/directory |  |  |

### DIR [DIR]

1. Directory Create

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| valid | 23 | DirCreate(path) and it doesn’t already exist, the parent does not exist, and does not exceed 256 characters | Go to parent path File does not exist | Create directory at parent directory |  |  |
| invalid | 24 | DirCreate(path) but path already exist | Go to parent path Failure Exist | Get the file of the parent path, and once you do that, you see the file Path already exist. Than, you return EDIRCREATE |  |  |
| invalid | 25 | DirCreate(path) but the parent does not exist | Go to parent path Failure Bad Path | Get the file of the parent path, but you the parent path doesn’t exist. Than, you return EDIRCREATE |  |  |
| invalid | 26 | DirCreate(path) but path does not exist | Path exceeds 256 characters | If the supposed path is greater than 256 characters, or 2048 bits, return ETOOLONGPATH |  |  |

1. Directory Size/Read

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| valid | 27 |  | DIRSIZE() returns value. (NOTE THIS FUNCTION ONLY IS CALLED BY DIRREAD AND SHOULD ONLY WORK WHEN DIRREAD DETERMINED THE PATH IS VALID) | Gets a size of the directory, and sets hate buffer to appropriate size. Note that each entry in a directory is 20 bytes |  |  |
| valid | 28 |  | DIRREAD Success | The buffer is of appropriate size, and as such, you can write each inode to it. |  |  |
| invalid | 29 |  | DIRREAD Failure Size too small | The buffer is of too small size, return EBUFFERTOOSMALL |  |  |
| invalid | 30 |  | DIRREAD Failure Dir no exist | directory does not exist, return EDIRNOEXIST |  |  |

### Files [FILE]

1. Create/Open and Close

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
|  | 31 |  | FileCreate(string File) Success | Create a file at the path, with size 0. |  |  |
|  | 32 |  | FileCreate(string File) Failure already exist | Failed at creating a file, due to there already being a file. Return EFILECREATE |  |  |
|  | 33 |  | FileCreate(string File) Failure max file size | The files system reaches max files. Return EFILECREATE |  |  |
|  | 34 |  | Path exceeds 256 characters | If the supposed path is greater than 256 characters, or 2048 bits, return ETOOLONGPATH |  |  |
|  | 35 |  | FileOpen success | File doe exist and doesn’t’ exceed files open limit |  |  |
|  | 36 |  | FileOpen Failure noExist | File does not exist, return ENOSUCHFILE |  |  |
|  | 37 |  | FileOpen Failure already Open | File already is opened. return EFILEALREADYOPEN |  |  |
|  | 38 |  | FileOpen Failure too many open files | File does exist, but there’s too many open files. return ETOOMANYOPENFILES |  |  |
|  | 39 |  | FileClose(int fd) Success | Close file in open file table, with fd. |  |  |
|  | 40 |  | FileClose(int fd) Failure | File is not in open file table, return ECLOSEBADFD |  |  |

1. File Read, write

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
|  | 41 |  | FileRead(int fd, string buffer, int size) Success | check Open file, go to current position in file, than read from current position to size/end of file. Than have current file position |  |  |
|  | 42 |  | FileRead(int fd, string buffer, int size) Failure not open | File is not in open file table, and thus, return EREADBADFD. |  |  |
|  | 43 |  | FileWrite(int fd, string buffer, int size) Success | Write to a file from buffer, from the buffer. Current file offset should be by size. |  |  |
|  | 44 |  | FileWrite(int fd, string buffer, int size) Failure not open | Write to a file from buffer, from the buffer. Current file offset should be by size. Return EWRITEBADFD |  |  |
|  | 45 |  | FileWrite(int fd, string buffer, int size) Failure no space left | While writing, if you find there’s no more space in the disk, return ENOSPACE |  |  |
|  | 46 |  | FileWrite(int fd, string buffer, int size) Failure maximum file size | While writing, if you find there’s the file takes up more than 10 data blocks, return EFILETOOBIG |  |  |

## Seek and UnLink

### FileSeek and Dir/FileUnLink [FILE : DIR]

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| valid | 47 | Remove file | FileUnlink(String File) | Remove file in inode block, and freeing up any data blocks/indoes the file used. |  |  |
| invalid | 48 | Remove file BUT THE FILE DOES NOT EXIST | FileUnlink(String File) but no such file | File does not exist. Return ECLOSEBADFD |  |  |
| invalid | 49 | Remove file but file is in openpagetable | FileUnlink(String File) but file is already opened | File is currently opened. Return EFileINUse |  |  |
| valid | 50 | remove directory | DirUnlink(String File) Directory is empty | Remove file from parent inode pointers, and than free up the inode/data blocks. |  |  |
| invalid | 51 | remove directory but it’s empty | DirUnlink(String File) Directory is not empty | Return EDIRNOTEMPTY |  |  |
| invalid | 52 | remove directory but it’s root | DirUnlink(String File) Directory is root | return EDELROOTDIR |  |  |
| invalid | 53 | remove directory but directory never existed | DirUnlink(String File) Directory does not exist | return EDIRNONEXIST |  |  |
| valid | 54 | file seek and it has a offset that’s within 0 to file size | FileSeek(int fd, int offset) | File’s position is changed by the offset |  |  |
| invalid | 55 | file seek but it has a offset that’s without 0 to file size | FileSeek(int fd, int offset) Out of bounds | Offset is negative/exceeds file size. Return ESEEKOUTOFBOUNDS |  |  |
| invalid | 56 | file seek and it’s fd is not in page table | FileSeek(int fd, int offset) bad fd | File isn’t open. Return out of bounds |  |  |

# Test Cases Part 2

## Bit manipulation [BITSTREAM]

### Bit manipulation of Inode [INODE]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | Read in a stream of bits, and convert to an inode object | Stream of 114 bits | Create an inode object, by the specified 114 bits. | Works | Pass |
| valid | 2 | Write as a stream of bits from inode object | Inode object | write as a stream of 114 bits | Works | Pass |

### Bit manipulation of Directory [DIR]

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 3 | Read in a stream of bits, and convert to an directory object | Stream of 4096 bits | Create an directory object, by the specified 4096 bits. | Works | Pass |
| valid | 4 | Write as a stream of bits from directory object | directory object | write as a stream of 4096 bits | Works | Pass |

## Setting up the drives

### File system scope [FS]

Error Messages are sent to osErrMsg

1. FSBoot

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 4 | The external disk isn’t made, and so create one and rerun this function again | Success NoDisk | The external disk does not exist, creates it | Works | Pass |
| Valid | 5 | the external disk exist and has correct magical number | Success Disk | The external disk does exist, clone it to working test and verify | Works | Pass |
| Invalid | 6 | the external disk exist and has incorrect magical number | Failure Disk | The external disk does exist, clone it to working test and verify, but fails, resulting in EFILEBOOT | Works | Pass |

1. FSSync

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 7 | Saves the working disk to external disk | Working disk and external disk | Copy working disk to external disk | Works | Pass |

1. FSReset

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| valid | 8 | Attempt to access file system after FSRESET() but before FSBOOT() | FSReset() and no FSBoot() | EINVALIDACCESSATTEMPT | Work | Pass |
| invalid | 9 | Attempt to access file system after FSRESET() and after FSBOOT() | FSReset() and FSBoot() | File system boots | Works | Pass |

### Disk Scope [DS]

Error Messages are sent to diskErrMsg

1. DISKSETUP and Save

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 10 | Set all the bits to the disk to 0, and creates root directory | int DISKINIT() | Run before disk ops take place. All sectors are all zeros, and you create he superblock and the root directory. Called during FSBoot | Works | Pass |
| Valid | 11 | External disk saves to working disk | int DiskLoad | When booting a Disk, you send External Disk -> working Disk. Called during FSBoot | Works | Pass |
| Valid | 12 | Working disk saves to external disk | int DiskSave | Called by FSSync, send Working Disk -> External Disk. | Works | Pass |

1. DISK Write

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Invalid | 13 | DiskWrite(), but size too small. EXCEPTION | Buffer is less than SIZESector | return EWRITEINVALIDPARAM | Works | Pass |
| Invalid | 14 | DiskWrite(), but buffer is NULL. EXCEPTION | Buffer is NULL | If not, than return EWRITEINVALIDPARAM | Works | Pass |
| Invalid | 15 | DiskWrite(), but sector parameter is not valid. EXCEPTION | Sector parameter is out of bounds | return EWRITEINVALIDPARM | Works | Pass |
| Valid | 16 | DiskWrite(), and it’s correct size, buffer is not null, and sector parameter is within [0,100) ] | Sector has data being written to it | Write the Buffer to Sector. | Works | Pass |

1. DISK Read

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Invalid | 17 | DiskRead(), but size too small. EXCEPTION | Buffer is less than SIZESector | return EREADINVALIDPARAM | Works | Pass |
| Invalid | 18 | DiskRead(), but buffer is NULL. EXCEPTION | Buffer is NULL | If not, than return EREADINVALIDPARAM | Works | Pass |
| Invalid | 19 | DiskRead(), but sector parameter is not valid. EXCEPTION | Sector parameter is out of bounds | return EREADINVALIDPARM | Works | Pass |
| Valid | 20 | DiskRead(), and it’s correct size, buffer is not null, and sector parameter is within [0,100) ] | Sector has data being written to it | write the sector to buffer. | Works | Pass |

## Setting up Directories and files

### DIR [DIR]

1. Directory Create

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| valid | 23 | DirCreate(path) and it doesn’t already exist, the parent does not exist, and does not exceed 256 characters | Go to parent path File does not exist | Create directory at parent directory | Works | Pass |
| invalid | 24 | DirCreate(path) but path already exist | Go to parent path Failure Exist | Get the file of the parent path, and once you do that, you see the file Path already exist. Than, you return EDIRCREATE | Works | Pass |
| invalid | 25 | DirCreate(path) but the parent does not exist | Go to parent path Failure Bad Path | Get the file of the parent path, but you the parent path doesn’t exist. Than, you return EDIRCREATE | Works | Pass |
| invalid | 26 | DirCreate(path) but path does not exist | Path exceeds 256 characters | If the supposed path is greater than 256 characters, or 2048 bits, return ETOOLONGPATH | Works | Pass |

1. Directory Size/Read

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| valid | 27 |  | DIRSIZE() returns value. (NOTE THIS FUNCTION ONLY IS CALLED BY DIRREAD AND SHOULD ONLY WORK WHEN DIRREAD DETERMINED THE PATH IS VALID) | Gets a size of the directory, and sets hate buffer to appropriate size. Note that each entry in a directory is 20 bytes | Works | Pass |
| valid | 28 |  | DIRREAD Success | The buffer is of appropriate size, and as such, you can write each inode to it. | Works | Pass |
| invalid | 29 |  | DIRREAD Failure Size too small | The buffer is of too small size, return EBUFFERTOOSMALL | Works | Pass |
| invalid | 30 |  | DIRREAD Failure Dir no exist | directory does not exist, return EDIRNOEXIST | Works | Pass |

### Files [FILE]

1. Create/Open and Close

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 31 | Create the file successfully | FileCreate(string File) Success | Create a file at the path, with size 0. | Works | Pass |
| Invalid | 32 | Create the file but the file already exist | FileCreate(string File) Failure already exist | Failed at creating a file, due to there already being a file. Return EFILECREATE | Works | Pass |
| Invalid | 33 | Create the file but reaches maximum file size | FileCreate(string File) Failure max file size | The files system reaches max files. Return EFILECREATE | Works | Pass |
| Invalid | 34 | Create the path but exceeds 256 characters | Path exceeds 256 characters | If the supposed path is greater than 256 characters, or 2048 bits, return ETOOLONGPATH | Works | Pass |
| Valid | 35 | Opens file successful | FileOpen success | File doe exist and doesn’t exceed files open limit | Works | Pass |
| Invalid | 36 | Open file but it does not exist | FileOpen Failure noExist | File does not exist, return ENOSUCHFILE | Works | Pass |
| Invalid | 37 | Open file but it’s already open | FileOpen Failure already Open | File already is opened. return EFILEALREADYOPEN | Works | Pass |
| Invalid | 38 | Open file but there’s max file open has been reached | FileOpen Failure too many open files | File does exist, but there’s too many open files. return ETOOMANYOPENFILES | Works | pass |
| Valid | 39 | File close successfuly | FileClose(int fd) Success | Close file in open file table, with fd. | Works | Pass |
| Invalid | 40 | File close but file is not in table | FileClose(int fd) Failure | return ECLOSEBADFD | Works | pass |

1. File Read, write

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| Valid | 41 | Read the file sucessfully | FileRead(int fd, string buffer, int size) Success | check Open file, go to current position in file, than read from current position to size/end of file. Than have current file position | Works | Pass |
| Invalid | 42 | Read the file but the file is not opened | FileRead(int fd, string buffer, int size) Failure not open | File is not in open file table, and thus, return EREADBADFD. | Works | Pass |
| Valid | 43 | Write the file with the buffer successfuly | FileWrite(int fd, string buffer, int size) Success | Write to a file from buffer, from the buffer. Current file offset should be by size. | Works | Pass |
| Valid | 44 | Write the file but it’s not opend | FileWrite(int fd, string buffer, int size) Failure not open | Write to a file from buffer, from the buffer. Current file offset should be by size. Return EWRITEBADFD | Works | Pass |
| Valid | 45 | Write the file but there’s no space | FileWrite(int fd, string buffer, int size) Failure no space left | While writing, if you find there’s no more space in the disk, return ENOSPACE | Works | Pass |
| Valid | 46 | Write the file but max file size has been reached | FileWrite(int fd, string buffer, int size) Failure maximum file size | While writing, if you find there’s the file takes up more than 10 data blocks, return EFILETOOBIG | Works | Pass |

## Seek and UnLink

### FileSeek and Dir/FileUnLink [FILE and DIR]

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- | --- | --- |
| valid | 47 | Remove file | FileUnlink(String File) | Remove file in inode block, and freeing up any data blocks/indoes the file used. | Works | Pass |
| invalid | 48 | Remove file BUT THE FILE DOES NOT EXIST | FileUnlink(String File) but no such file | File does not exist. Return ECLOSEBADFD | Works | Pass |
| invalid | 49 | Remove file but file is in openpagetable | FileUnlink(String File) but file is already opened | File is currently opened. Return EFileINUse | Works | Pass |
| valid | 50 | remove directory | DirUnlink(String File) Directory is empty | Remove file from parent inode pointers, and than free up the inode/data blocks. | Works | Pass |
| invalid | 51 | remove directory but it’s empty | DirUnlink(String File) Directory is not empty | Return EDIRNOTEMPTY | Works | Pass |
| invalid | 52 | remove directory but it’s root | DirUnlink(String File) Directory is root | return EDELROOTDIR | Works | Pass |
| invalid | 53 | remove directory but directory never existed | DirUnlink(String File) Directory does not exist | return EDIRNONEXIST | Works | Pass |
| valid | 54 | file seek and it has a offset that’s within 0 to file size | FileSeek(int fd, int offset) | File’s position is changed by the offset | Works | Pass |
| invalid | 55 | file seek but it has a offset that’s without 0 to file size | FileSeek(int fd, int offset) Out of bounds | Offset is negative/exceeds file size. Return ESEEKOUTOFBOUNDS | Works | Pass |
| invalid | 56 | file seek and it’s fd is not in page table | FileSeek(int fd, int offset) bad fd | File isn’t open. Return out of bounds | Works | Pass |

# Algorithm/Code

## Program General code

* Create the two disk, with each disk having a set of functions regulating to disk.
* File system has the two disk, and it has all the functionality of FS.
* OpenFileTable keeps track of each file, and it’s current position
* As we are writing bits, we have to format the disk to be able to read and write bits.
* SUPERBLOCK | indoebitmap | datablock bitmap | sequence of indoes | sequence of datablock = 1000
* the sequence of indoes will have 3 sectors, due to each Inode being able to represent 35 indoes.
* The rest of the space, 994 sectors, are for the datablock block.

## Bit Parsing/Data Strucutre [BITSTREAM]

### inode

1. writeBitStream()
   1. Write teh type, size and allocation, by reversing the blow operation
2. readBitStream()
   1. read the type, size and allocation by following the following processes
3. Indoe structure
   1. There are 4 indoes within a inode sector. The makeup totals to 114 bits.
      1. 1 bit
         1. for which type of inode this is.
      2. 13 bits
         1. (or 1.625 bytes) for representing the size of data blocks
      3. 100 bits
         1. 10 sequences of 10 bits for representing the location. note that all 1s mean that this is not allocated
4. Fragmentation
   1. This results of 106 of useless data, and 3990 of useful data. Since there are 35 indoes in a sector, we split it up into an array, with each piece being a substr of 114 bits.

### directory

* Data blocks are disgusted by two types: file and directory
* the type of the data block is denoted by the inode, not the directory.
* For directory, there is a 20 bytes/160 bits, which are
  + 16 bytes/128 bits
    - file name. 15 characters PLUS 1 for end of string, so it’s More of 15 characters
  + 4 byte/32 bits
    - inode that shows which file/directory this is.
* This means that dictionaries Can have 25 files in a a sector, but 250 files/directories overall.
* This doesn’t have the case, of half a directory’s information being in one data block, and the other half being in another data block. That isn’t considered.

### bitmap of Indoe/data block

* this is just a bitmap, used to keep track of which indoes are allocated and which data blocks are allocated.

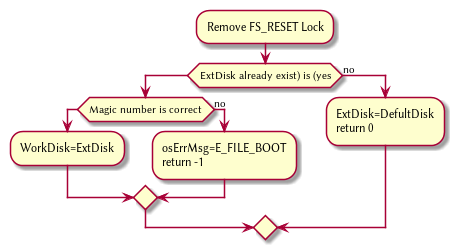
### Sector/Root Inode

* A sector is a collection of a superblock, bitmaps for in use indoes and data blocks, a sequence of indoes, and a sequence of data blocks. However, this information HAS TO BE CONVERTED to that. Otherwise, a sector is just an array of bit sets of 4096 bits.
* However, the sector converts it’s con cats to usable data structures. After each file/directory operation, it saves the stuff to working directory. Than, working directory saves it stuff to external disk when FSSYNC() is made.
* The disks are just a bitset array of 4096 bits, with 1000 elements in each.
* The root inode is the Indoe that represents nothing. This is a special variable, as to not have to find out what it is on disk tediously.

### 

## File System (note I couldn’t get outline to work here) [FS]

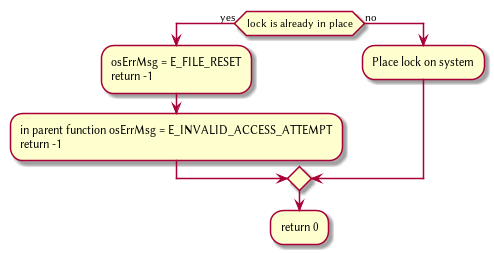
1. FSBOOT()
   1. Called when booting filesystem/after a FSRESET()



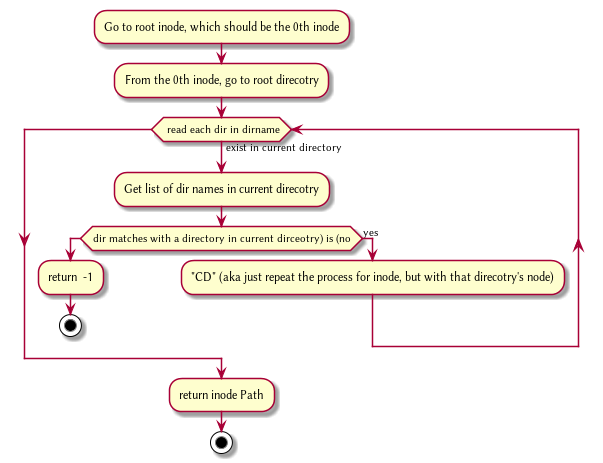
1. FSSync
   1. Copys the working disk to external disk



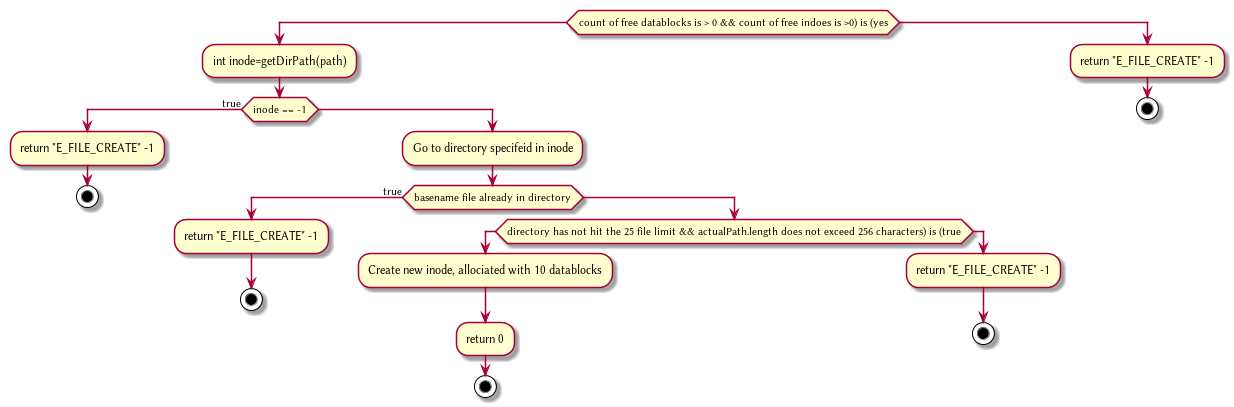
1. FSRESET()
   1. Stops the filesystem from ebing access, by placing a lock on it.



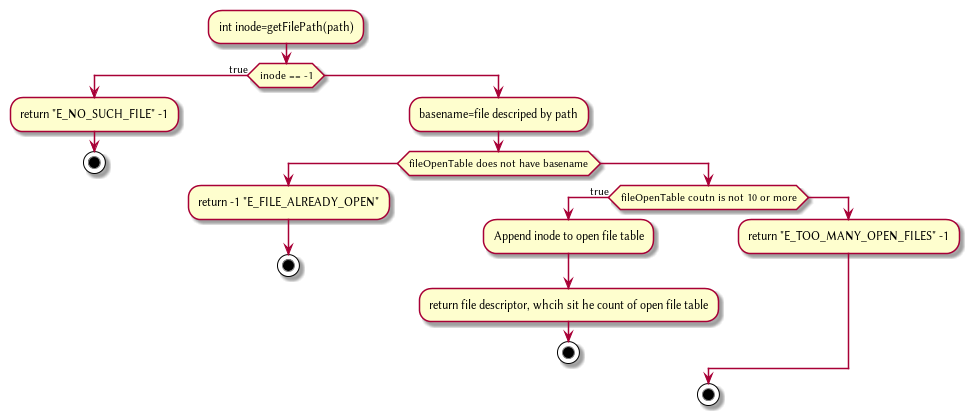
## File Access [FILE]

1. int getInode(string path)
   1. Helper function, used to get the directory given a path.
      1. Ouptut
         1. inode number of where it is, or -1 if it’s not found.
         2. 

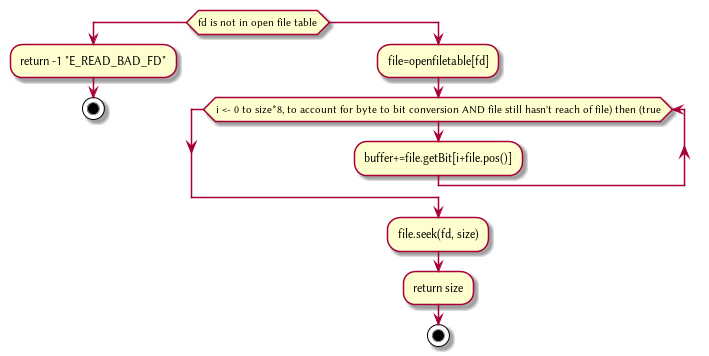
1. FileCreate(string path)
   1. Create a new file at path. There is a check to see if that file already exist, and if there’s a free datablock for it.



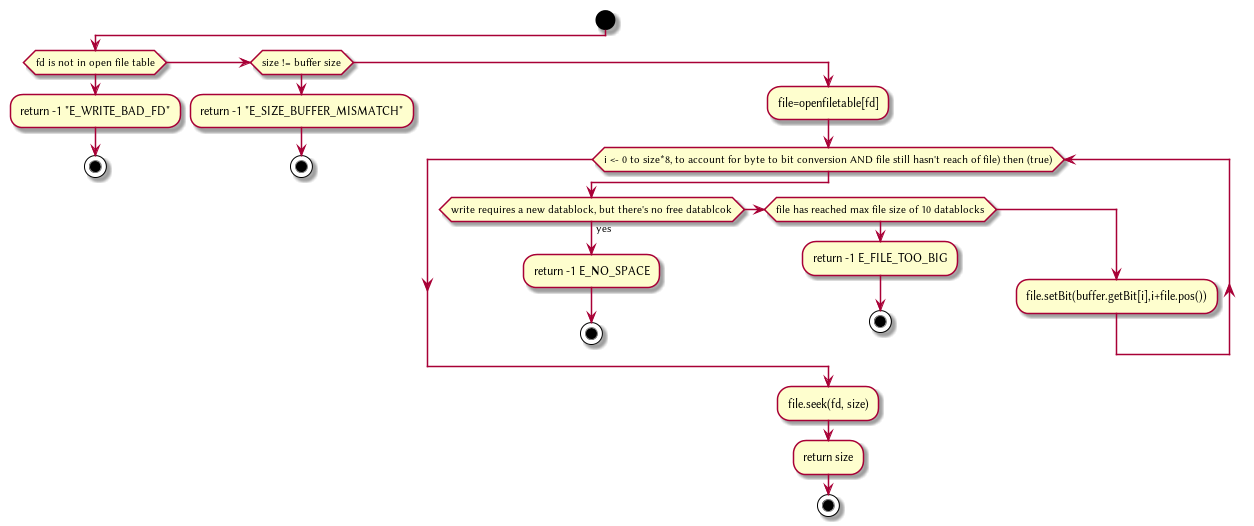
1. FileOpen(string path)
   1. returns the file descriptor of the file, which can be used to read and write to it.



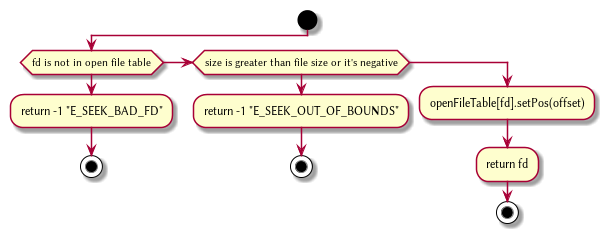
1. FileRead(int fd, string buffer, int size IN BYTES)
   1. Buffer reads size from the file in fd. Note the file in open file table shuold move by size



1. FileWrite(int fd, string buffer, int size IN BYTES)
   1. Write from buffer to the file. NOTE SIZE HAS TO BE CONSISNET. If it’s not, stop the program

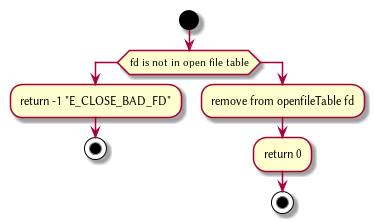


1. FileSeek(int fd, int offset)
   1. move the file forward by offset.

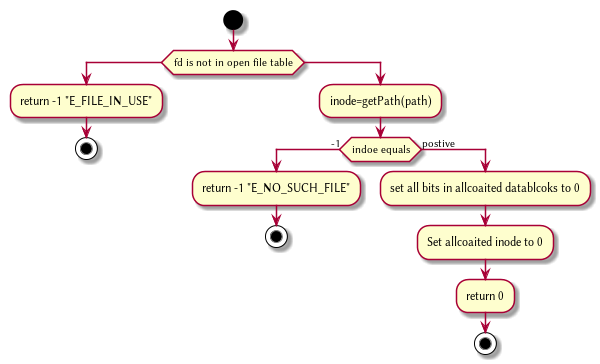


@startuml

1. FileClose(int fd)
   1. Remove file from table

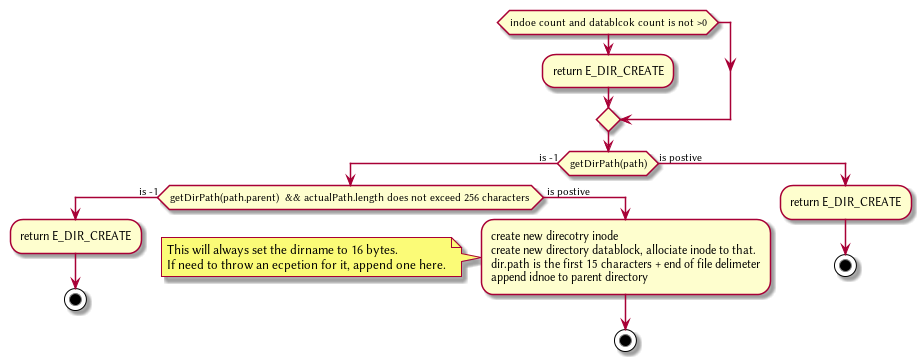


1. FileUnLink(string path)
   1. Delete file from the filesystem.



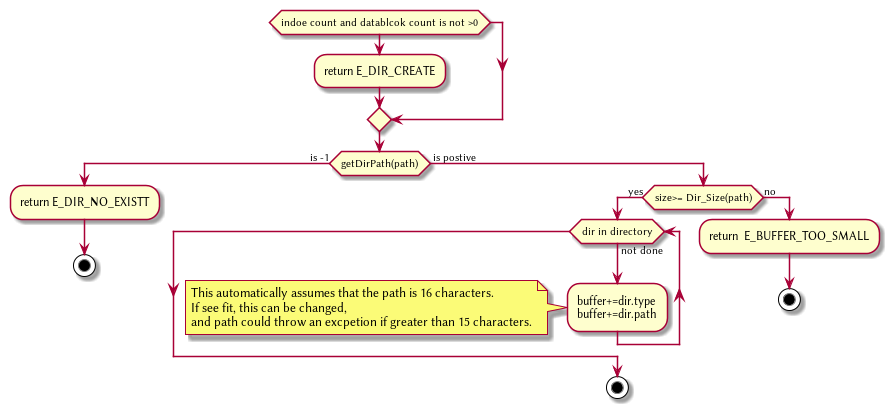
## Directory [DIR]

1. DirCreate(string path)
   1. Create directory at path

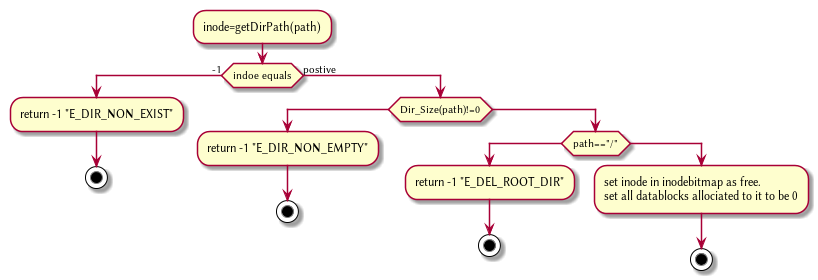


@startuml

1. DirRead(string path, string buffer, itn size)
   1. Read the contents of a directory.

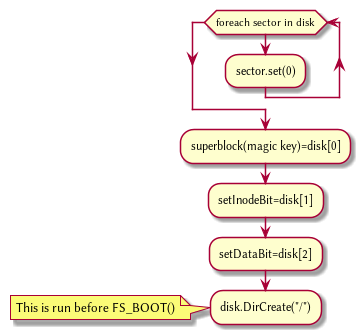


1. DirUnlink(string path)
   1. Remove file from drive

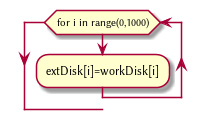


## Disk [DISK]

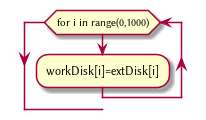
1. DISKINIT()
   1. Set all the data in the disk to be 0



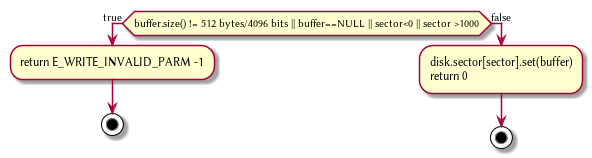
1. DISKLOAD()
   1. Save external disk to workign disk. Done when booting.



1. DISKSAVE()
   1. Save working disk to loading. Called by FSSYNC()



1. DISKWRITE(int sector, string buffer)
   1. Write from buffer to disk.



1. DISKRead(int sector, string buffer)
   1. read from sector to buffer

