Contents

1 Algorithm/Code **1**

1 Algorithm/Code

1. Whoel Porgram decompsiostion This is an outline/code of how the whole program will be.

2. Bit Parsing/Data Strucutre BIT

   - 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 datablcoks = 1000
   - the sequence of indoes will ahve 3 sectors, due to each indoe being able to represtn 35 inodes.
   - The rest of the space, 994 sectors, are for teh databock block.

   (a) inode

   writeBitStream() Write teh type, size and allociation, by reversing the blwo opeariton

   readBitStream() read the type, size and allcioation by folowing the following processess

   There are 4 indoes within a inode sector. The makeup totals to 114 bits.

   1 bit for which type of inode this is.

   13 bits (or 1.625 bytes) for representing the size of datablocks

   100 bits 10 seqeunces of 10 bits for reprsenting the location. note that all 1s mean that this is not allociated

   This results of 106 of useless data, and 3990 of useful data. Since there are 35 inodes in a sector, we split it up into an array, with each piece being a substr of 114 bits.

   The function below is a method ofreading it. Note it doesn't return anything. Maybe i'll try to do that thing where i have an inlnie function and do it there.

   Anotehr note: there'll be 35 inodes withn a sector, so the spliting of that by 114 is left to futrue zak.

   Writing it to bitstream is simple. if need be write a funciton for it.

   (b) datablock

   - Datablocks are disgshiustehd by two types: file and directory
   - the type of the datablock is denoted by teh inode, not the directory.

- For directory, tehre 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 mroe of 15 characters

  4 byte/32 bits inode that shows which file/driectory this is.
- This means that dictionaries cna have 25 files in a a sector, but 250 files/directories overall.
- This doesn't have the case, of half a directoriy's infroamtion being in one datablcok, and the other half being in another datablock. THat isn't consdiered.

```
using namespace std;
void readDir(string TestString){
    bitset<4> inode(TestString.substr(0,4));
    cout << inode.to_ulong() << endl;
    char temp[10];
    for(int i=0; i<16; i++){
    bitset<8> temp(TestString.substr(4+i*8,8));
    cout << (char)temp.to_ulong() << endl;

    }

}

int main(){
/*
string temp1="iiii1111111111111111222222222222222233333333333333334444444444444
for(int i=0; i<8; i++){
    cout << temp1.substr(4+i*16,16) << endl;
}
*/
/*readDir("11111000000000000000100000000000000010000000000000001000000000000000
       readDir("111101000001010000100100001101000100010001010100011001000111010
       /*
       111101000001010000100100001101000100010001010100011001000111010010000100
}
```

(c) bitmap of indoe/datablock

- this is just a bitmap, used to keep trakc of which indoes are allociated and which datablocks are allociated.

(d) Sector/Root Inode

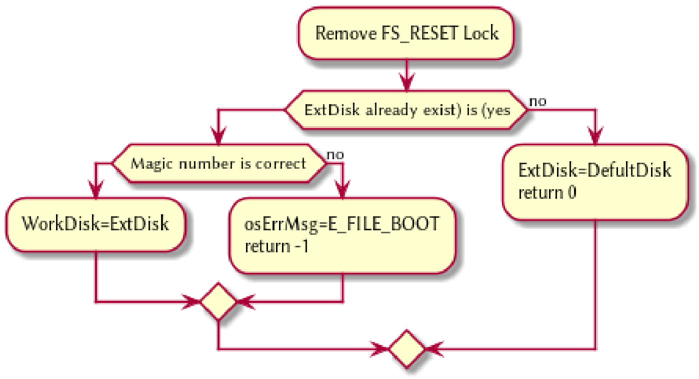
- A sector is a collection of a superblock, bitmaps for in use indoes and datablocks, a sqeunce of indoes, and a sequence of datablocks. However, this information HAS TO BE CONVERETD to that. Otehrwise, a sector is just an array of bitsets of 4096 bits.

- However, the sector converts it's concats to usuable datasturcutres. After each file/directory operation, it saves the stuff to workign directory. Than, working directory saves it stuff to external disk when $\mathrm{FS}_{\mathrm{SYNC}}()$ is made.
- The disks are just a bitset array of 4096 bits, with 1000 elements in each.
- The root inode is the indoe that represtns nothing. This is a special variable, as to not have to find out what it is on disk tediously.

```
std::bitset<4096> ExtDisk[1000];
std::bitset<4096> WorkDisk[1000];
```

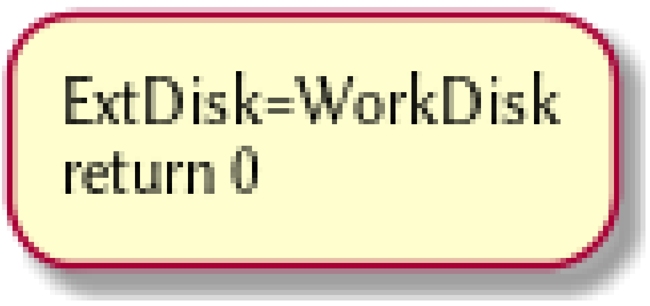
3. File System FS

$\mathrm{FS}_{\mathrm{BOOT}}()$ Called when booting filesystem/after a $\mathrm{FS}_{\mathrm{RESET}}()$

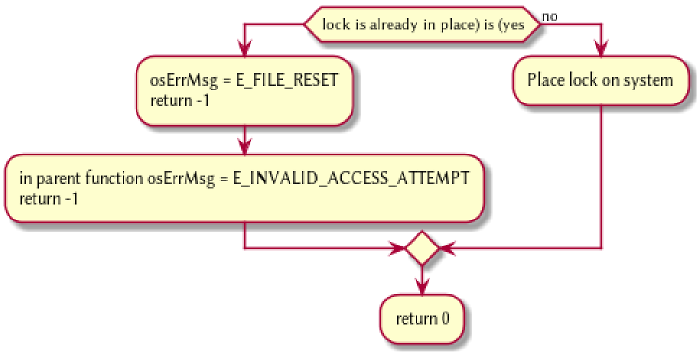


]

$\mathrm{FS}_{\mathrm{Sync}}$ Copys the working disk to external disk



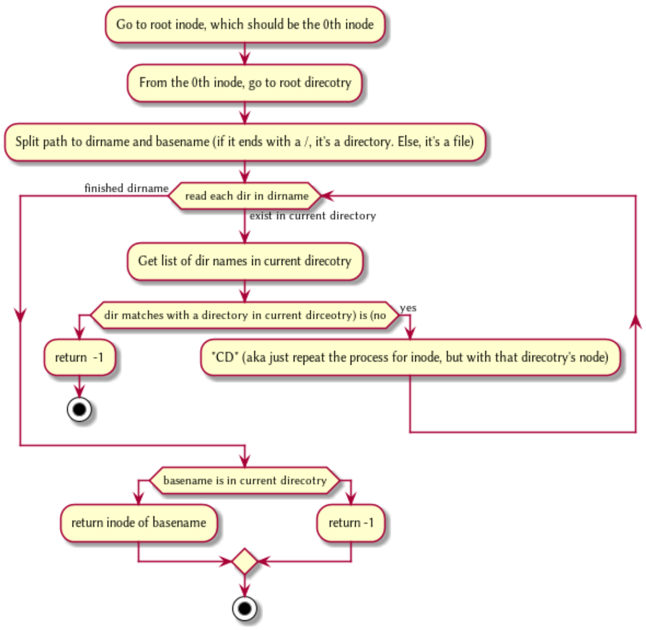
$\mathrm{FS}_{\mathrm{RESET}}()$ Stops the filesystem from ebing access, by placing a lock on it.


lock is already in place) is (yes
no
osErrMsg = E_FILE_RESET
return -1
Place lock on system
in parent function osErrMsg = E_INVALID_ACCESS_ATTEMPT
return -1
return 0

4. File Access FILE

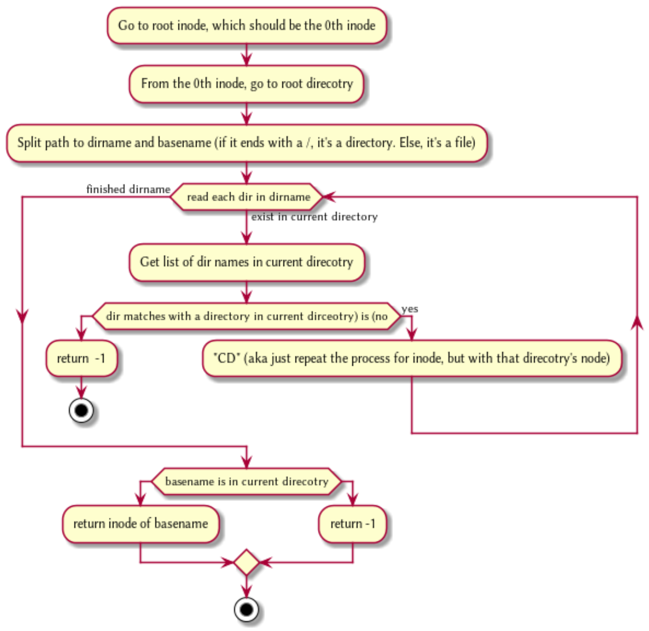
int getInode(string path) Helper function, used to get the inode given a path.

Ouptut inode number of where it is, or -1 if it's not found.

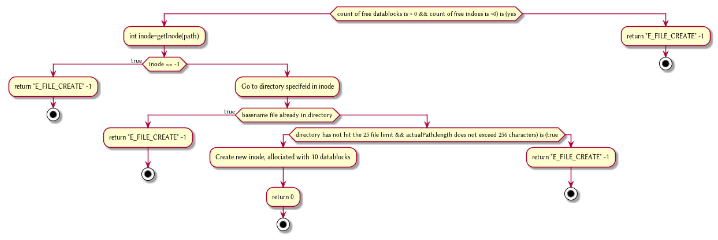

Go to root inode, which should be the 0th inode
From the 0th inode, go to root direcotry
Split path to dirname and basename (if it ends with a /, it's a directory. Else, it's a file)
finished dirname
read each dir in dirname
exist in current directory
Get list of dir names in current direcotry
dir matches with a directory in current dirceotry) is (no
yes
return -1
"CD" (aka just repeat the process for inode, but with that direcotry's node)
basename is in current direcotry
return inode of basename
return -1

int getInode(string path) Helper function, used to get the file given a path.

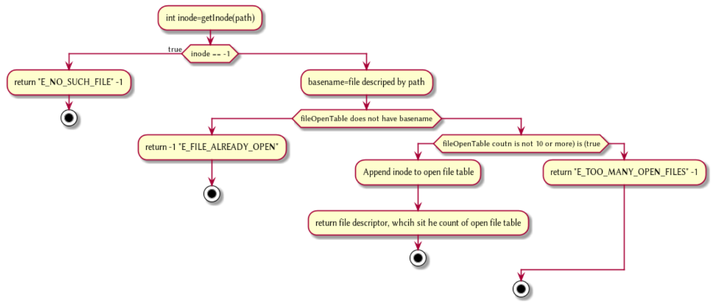
Ouptut inode number of where it is, or -1 if it's not found.


Go to root inode, which should be the 0th inode
From the 0th inode, go to root direcotry
Split path to dirname and basename (if it ends with a /, it's a directory. Else, it's a file)
finished dirname
read each dir in dirname
exist in current directory
Get list of dir names in current direcotry
dir matches with a directory in current dirceotry) is (no
yes
return -1
"CD" (aka just repeat the process for inode, but with that direcotry's node)
basename is in current direcotry
return inode of basename
return -1

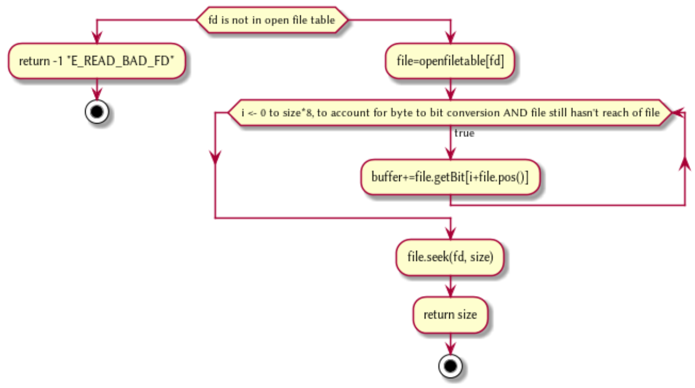
File$_{\text{Create}}$(string path) Create a new file at path. There is a check to see if that file already exist, and if there's a free datablock for it.


count of free datablocks is > 0 && count of free indoes is >0) is (yes
int inode=getInode(path)
return "E_FILE_CREATE" -1
true
inode == -1
return "E_FILE_CREATE" -1
Go to directory specifeid in inode
true
basename file already in directory
return "E_FILE_CREATE" -1
directory has not hit the 25 file limit && actualPath.length does not exceed 256 characters) is (true
Create new inode, allociated with 10 datablocks
return "E_FILE_CREATE" -1
return 0

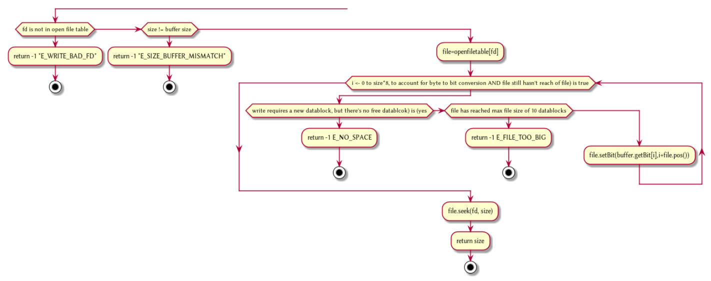
File$_{\text{Open}}$(string path) returns the file descriptor of the file, which can be used to read and write to it.


int inode=getInode(path)
true
inode == -1
return "E_NO_SUCH_FILE" -1
basename=file descriped by path
fileOpenTable does not have basename
return -1 "E_FILE_ALREADY_OPEN"
fileOpenTable coutn is not 10 or more) is (true
Append inode to open file table
return "E_TOO_MANY_OPEN_FILES" -1
return file descriptor, whcih sit he count of open file table

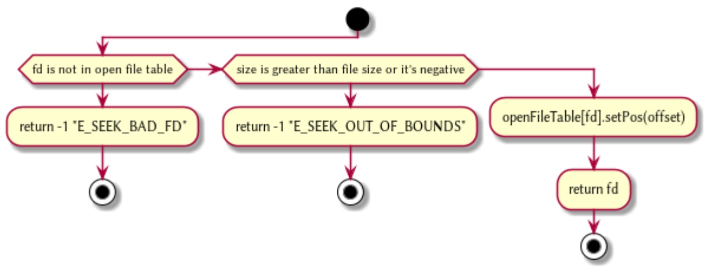
File$_{\text{Read}}$(int fd, string buffer, int size IN BYTES) Buffer reads size from the file in fd. Note the file in open file table shuold move by size


fd is not in open file table
return -1 "E_READ_BAD_FD"
file=openfiletable[fd]
i <- 0 to size*8, to account for byte to bit conversion AND file still hasn't reach of file
true
buffer+=file.getBit[i+file.pos()]
file.seek(fd, size)
return size

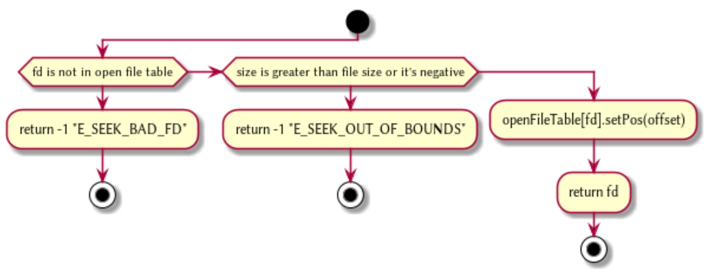
File$_{\text{Write}}$(int fd, string buffer, int size IN BYTES) Write from buffer to the file. NOTE SIZE HAS TO BE CONSISNET. If it's not, stop the program


fd is not in open file table
size != buffer size
return -1 "E_WRITE_BAD_FD"
return -1 "E_SIZE_BUFFER_MISMATCH"
file=openfiletable[fd]
i <- 0 to size*8, to account for byte to bit conversion AND file still hasn't reach of file) is true
write requires a new datablock, but there's no free datablcok) is (yes
file has reached max file size of 10 datablocks
return -1 E_NO_SPACE
return -1 E_FILE_TOO_BIG
file.setBit(buffer.getBit[i],i+file.pos())
file.seek(fd, size)
return size

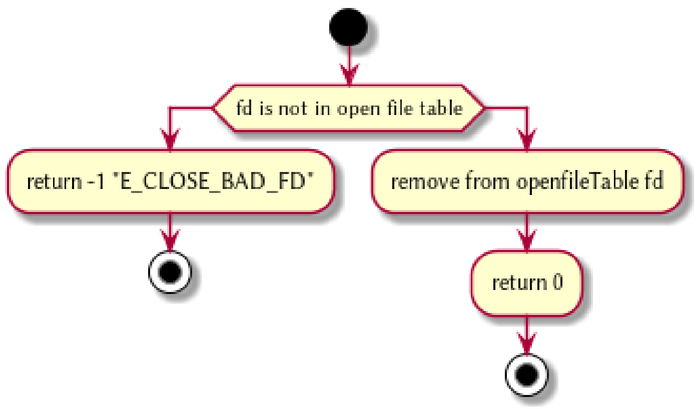
File$_{\text{Seek}}$(int fd, int offset) move the file forward by offset.


fd is not in open file table
size is greater than file size or it's negative
return -1 "E_SEEK_BAD_FD"
return -1 "E_SEEK_OUT_OF_BOUNDS"
openFileTable[fd].setPos(offset)
return fd

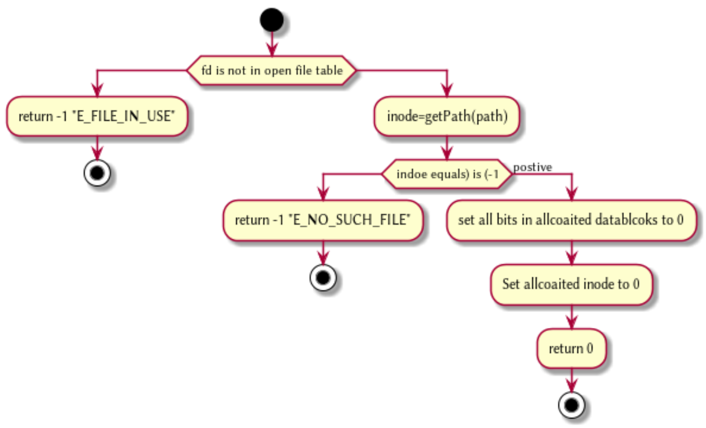
@startuml


fd is not in open file table
size is greater than file size or it's negative
return -1 "E_SEEK_BAD_FD"
return -1 "E_SEEK_OUT_OF_BOUNDS"
openFileTable[fd].setPos(offset)
return fd

File$_{\text{Close}}$(int fd) Remove file from table

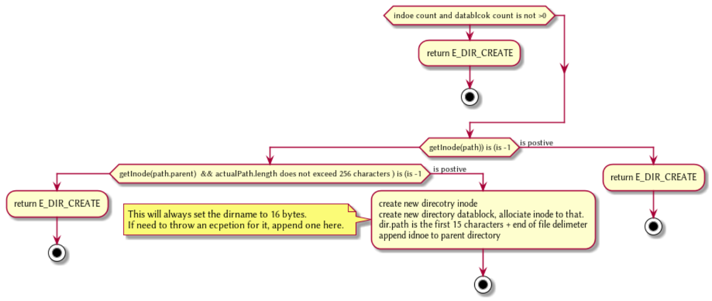

fd is not in open file table
return -1 "E_CLOSE_BAD_FD"
remove from openfileTable fd
return 0

File$_{\text{UnLink}}$(string path) Delete file from the filesystem.


fd is not in open file table
return -1 "E_FILE_IN_USE"
inode=getPath(path)
indoe equals) is (-1
postive
return -1 "E_NO_SUCH_FILE"
set all bits in allcoaited datablcoks to 0
Set allcoaited inode to 0
return 0

5. Directory DIR

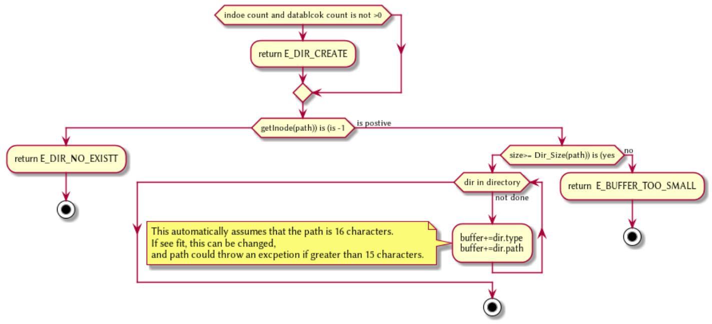
Dir$_{\text{Create}}$(string path) Create directory at path



indoe count and datablcok count is not >0
return E_DIR_CREATE
getInode(path)) is (is -1
is postive
getInode(path.parent) && actualPath.length does not exceed 256 characters) is (is -1
is postive
return E_DIR_CREATE
return E_DIR_CREATE
This will always set the dirname to 16 bytes.
If need to throw an ecpetion for it, append one here.
create new direcotry inode
create new directory datablock, allociate inode to that.
dir.path is the first 15 characters + end of file delimeter
append idnoe to parent directory

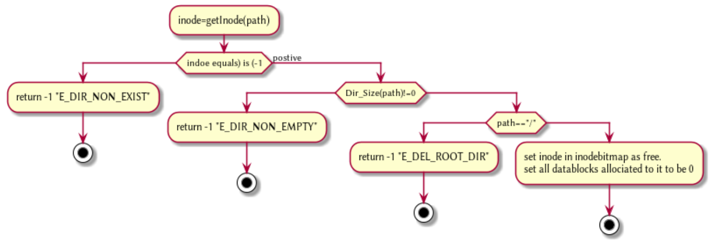
@startuml

Dir$_{\text{Read}}$(string path, string buffer, itn size) Read the contents of a directory.



indoe count and datablcok count is not >0
return E_DIR_CREATE
getInode(path)) is (is -1
is postive
return E_DIR_NO_EXISTT
size>= Dir_Size(path)) is (yes
no
dir in directory
return E_BUFFER_TOO_SMALL
not done
This automatically assumes that the path is 16 characters.
If see fit, this can be changed,
and path could throw an excpetion if greater than 15 characters.
buffer+=dir.type
buffer+=dir.path

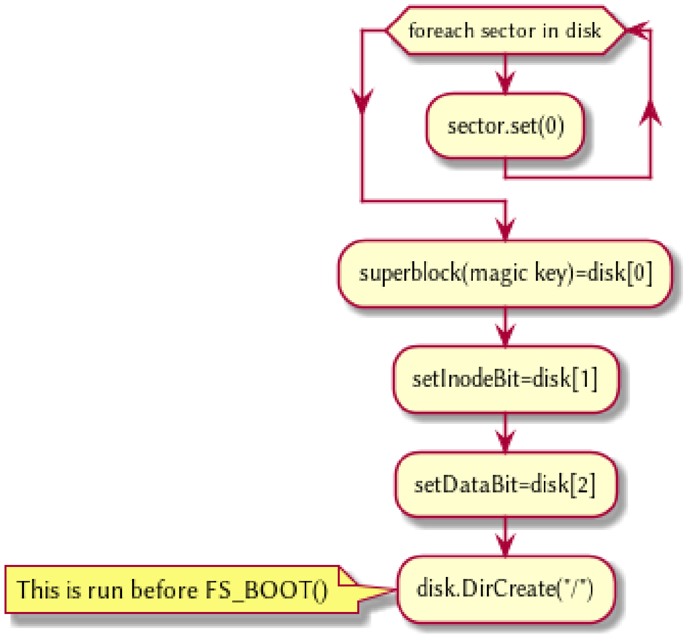
Dir$_{\text{Unlink}}$(string path) Remove file from drive



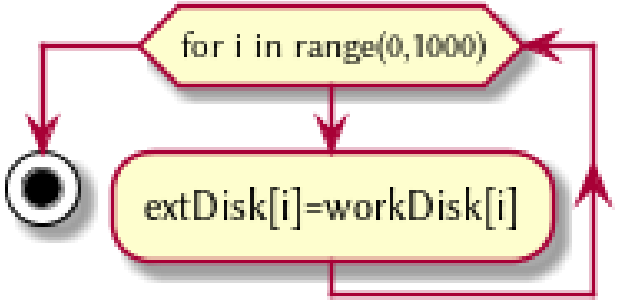
inode=getInode(path)
indoe equals) is (-1
postive
return -1 "E_DIR_NON_EXIST"
Dir_Size(path)!=0
return -1 "E_DIR_NON_EMPTY"
path=="/"
return -1 "E_DEL_ROOT_DIR"
set inode in inodebitmap as free.
set all datablocks allociated to it to be 0

6. Disk DISK

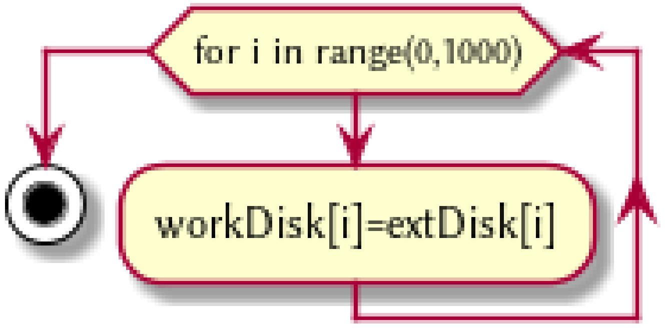
DISK$_{\text{INIT}}$() Set all the data in the disk to be 0



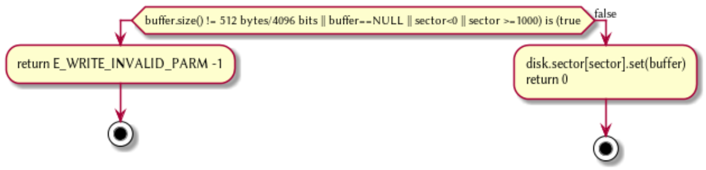
$\text{DISK}_{\text{LOAD}}()$ Save external disk to workign disk. Done when booting.



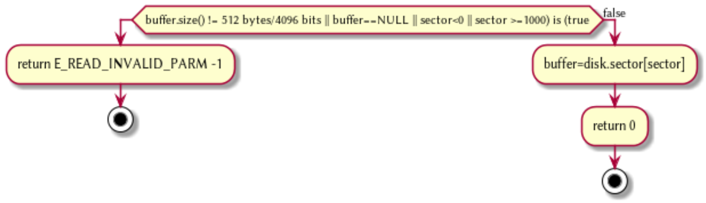
$\text{DISK}_{\text{SAVE}}()$ Save working disk to loading. Called by $\text{FS}_{\text{SYNC}}()$



DISK$_{\text{WRITE}}$(int sector, string buffer) Write from buffer to disk.


buffer.size() != 512 bytes/4096 bits || buffer==NULL || sector<0 || sector >=1000) is (true
false
return E_WRITE_INVALID_PARM -1
disk.sector[sector].set(buffer)
return 0

DISK$_{\text{Read}}$(int sector, string buffer) read from sector to buffer


buffer.size() != 512 bytes/4096 bits || buffer==NULL || sector<0 || sector >=1000) is (true
false
return E_READ_INVALID_PARM -1
buffer=disk.sector[sector]
return 0