Contents

1 Totest list

1. Bitmaniplation BIT

   (a) Bit maniplation of Inode INODE

Test Stragity	Test Number	Descritpion	Input	Expected Output
			Stream of 114 bits Inode object	Create an inode object, by the write as a stream of 114 bits

   (b) Bit maniplation of Directory DIR

Test Stragity	Test Number	Descritpion	Input	Expected Output
			Stream of 4096 bits directory object	Create an directory object, by write as a stream of 4096 bits

2. Setting up the drives

   (a) File system scope FS Error MEssages are sent to osErrMsg

      i. FS_{Boot}

Test Stragity	Test Number	Descritpion	Input	Expected Output
			Success NoDisk	The external disk does not ex
			Success Disk	The external disk does exist,
			Failure Disk	The external disk does exist,

      ii. FS_{Sync}

Test Stragity	Test Number	Descritpion	Input
		Saves the workign disk to external disk	Working disk and ex

      iii. FS_{Reset}

Test Stragity	Test Number	Descritpion	Input
			File System is unavaible to write till fs_{BOOT}
			Attmpt to access filesystem before/after FS_{Reset}
			Attmpt to access filestyem before/after FS_{Boot}

   (b) Disk Scope DS Error MEssages are sent to diskErrMsg

      i. $DISK_{SETUP}$ and Save

Test Stragity	Test Number	Descritpion	Input	Expected Output
			int $\text{DISK}_{\text{INIT}}()$	Run before disk ops take plac
			int $\text{Disk}_{\text{Load}}$	When booting a Disk, you ser
			int $\text{Disk}_{\text{Save}}$	Called by FS_{Sync}, send Worki

ii. DISK Write

Test Stragity	Test Number	Descritpion	Input	Expected
			Buffer is of $\text{SIZE}_{\text{SEctor}}$	If not, tha
			Buffer is nonNull	If not, tha
			Sector paramter is out of boudns	retrun E_{W}
			Sector has data being written to it	Write the

iii. DISK Read

Test Stragity	Test Number	Descritpion	Input	Expecte
			Buffer is of $\text{SIZE}_{\text{SEctor}}$	If not, t
			Buffer is nonNull	If not, t
			Sector paramter is out of boudns	retrun I
			Sector has data being read from to it	Write t

3. Setting up Directories and files

(a) getFilePath(string path) and getDirPath(string path)

Test Stragity	Test Number	Descritpion	Input	Expected Output
			String path	given path, get the inode assoicated w

(b) DIR — DIR

i. Directory Create

Test Stragity	Test Number	Descritpion	Input	Expecte
			Go to parent path File does not exist	Create
			Go to parent path Failure Exist	Get the
			Go to parent path Failure Bad Path	Get the
			Path exceeds 256 characters	If the s

ii. Directory Size/Read

Test Stragity	Test Number	Descritpion	Input
			Return the number of bytes in a path with Dir
			$\text{DIR}_{\text{SIZE}}()$ works
			DIR_{READ} Success
			DIR_{READ} Faiure Size too small
			DIR_{READ} Faiure Dir no eixst

(c) Files — FILE

i. Create/Open and Close

Test Stragity	Test Number	Descritpion	Input	Ex
			$\text{File}_{\text{Create}}$(string File) Success	Cr
			$\text{File}_{\text{Create}}$(string File) Failure already exist	Fa
			$\text{File}_{\text{Create}}$(string File) Failure max file size	Th
			Path exceeds 256 characters	If
			$\text{File}_{\text{Open}}$ success	Fil
			$\text{File}_{\text{Open}}$ Fialure noExist	Fil
			$\text{File}_{\text{Open}}$ Fialure alreadyOpen	Fil
			$\text{File}_{\text{Open}}$ Fialure too many open files	Fil
			$\text{File}_{\text{CLose}}$(int fd) Success	Cl
			$\text{File}_{\text{CLose}}$(int fd) Failure	Fil

ii. File Read, write

Test Stragity	Test Number	Descritpion	Input
			$\text{File}_{\text{Read}}$(int fd, string fuffer, int size) Success
			$\text{File}_{\text{Read}}$(int fd, string fuffer, int size) Failure no
			$\text{File}_{\text{Write}}$(int fd, string fuffer, int size) Success
			$\text{File}_{\text{Write}}$(int fd, string fuffer, int size) Failure no
			$\text{File}_{\text{Write}}$(int fd, string fuffer, int size) Failure no
			$\text{File}_{\text{Write}}$(int fd, string fuffer, int size) Failure ma

4. Seek and unLink

   (a) $\text{File}_{\text{Seek}}$ and Dir/$\text{File}_{\text{UnLink}}$ FILE:DIR

Test Stragity	Test Number	Descritpion	Input	E
			$\text{File}_{\text{Unlink}}$(String File)	R
			$\text{File}_{\text{Unlink}}$(String File) but no such file	F
			$\text{File}_{\text{Unlink}}$(String File) but file is already opened	F
			$\text{Dir}_{\text{Unlink}}$(String File) Directory is empty	R
			$\text{Dir}_{\text{Unlink}}$(String File) Directory is not empty	R
			$\text{Dir}_{\text{Unlink}}$(String File) Direcotry is root	re
			$\text{Dir}_{\text{Unlink}}$(String File) Direcotry does not exist	re
			$\text{File}_{\text{Seek}}$(int fd, int offset)	F
			$\text{File}_{\text{Seek}}$(int fd, int offset) Out of bounds	O
			$\text{File}_{\text{Seek}}$(int fd, int offset) bad fd	F

2 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.

```
#include<iostream>
#include<bitset>
using namespace std;
// Note in babel mode this will be incorrect

void readBitDataInode(string Inode){
// Type // Size // 10*10 of which bits are allociated to it.

    // This little test is used to demonstrate values used to finding where to
    /*
    string test= "11111NNNNN22222NNNNN33333NNNNN44444NNNNN55555NNNNN66666NNNNN7
    cout << test.substr(0,100) << endl; //Which are allcoiated
    cout << test.substr(100,13) << endl; //Size
    cout << test.substr(113,1) << endl; //Type
    for(int i=0; i<10; i++){
cout << test.substr(i*10,10) << endl;  // used to show how to splti the functio
    }
```

```
    */


uint alloc[10];
for(int i=0; i<10; i++){
bitset<10> temp(Inode.substr(i*10, 10));
cout << temp << '\t' << temp.to_ulong() << endl;
alloc[i]=temp.to_ulong();
}
uint size;
bitset<13> temp2(Inode.substr(100,13));
size=temp2.to_ulong();

bool type;
bitset<1> temp3(Inode.substr(113,1));
cout << temp3 << endl;
type=temp3.to_ulong();

cout << size << '\t' << type << endl;
}

int main(){
string test= "00000000001111111111000000000011111000000000011111101010100100000
readBitDataInode(test);
cout << "WOW";
}
```

(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<64> inode(TestString.substr(0,4));
    cout << inode.to_ulong() << endl;
    char temp[10];
```

```
    for(int i=0; i<8; i++){
    bitset<16> temp(TestString.substr(4+i*16,16));
    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("1111100000000000000010000000000000001000000000000000100000000000000
readDir("1111000000000100000100000000010000100000000001000011000000000100010000
}
```

(c) bitmap of indoe/datablock

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

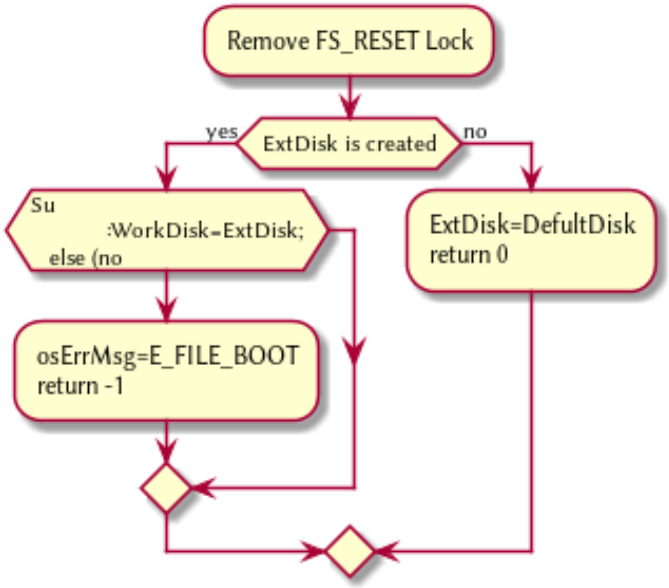
(d) Sector/Root Inode

- 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];
```

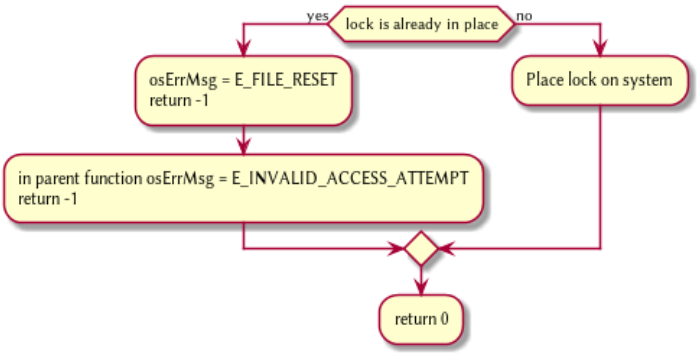
3. File System FS

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



FS_{Sync} Copys the working disk to external disk

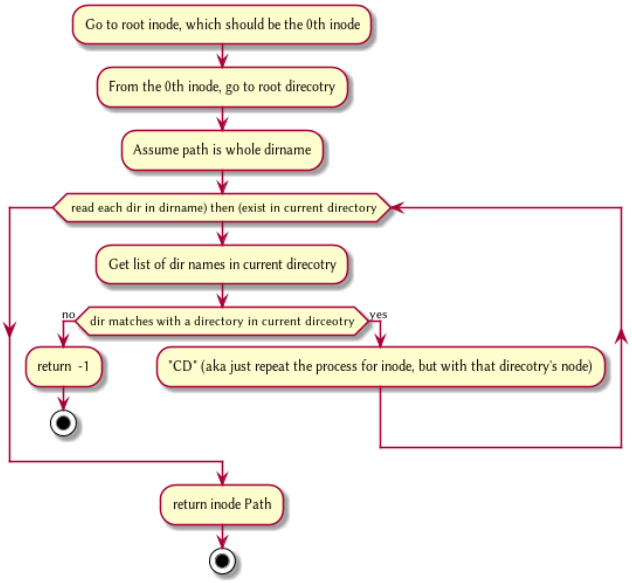
$FS_{RESET}()$ Stops the filesystem from ebing access, by placing a lock on it.



4. File Access FILE

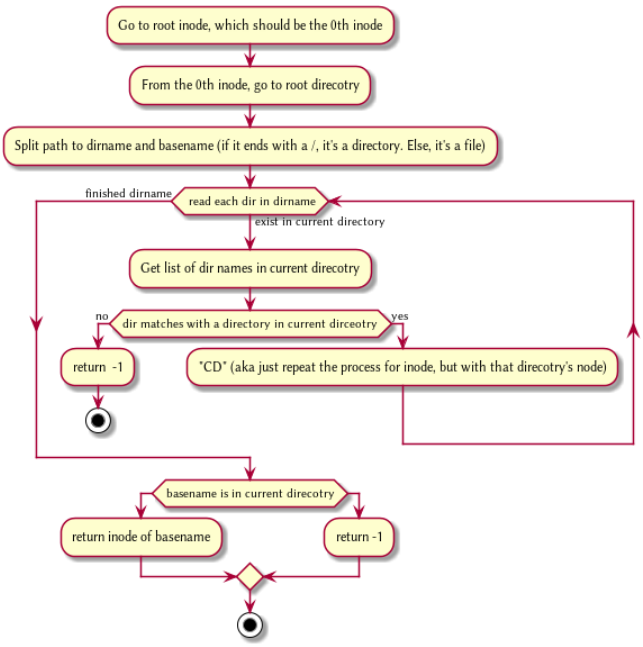
int getDirPath(string path) Helper function, used to get the directory given a path.

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

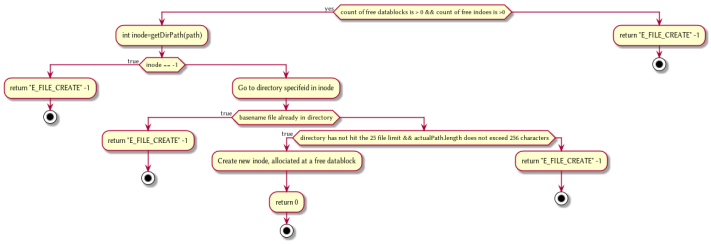


int getFilePath(string path) Helper function, used to get the file given a path.

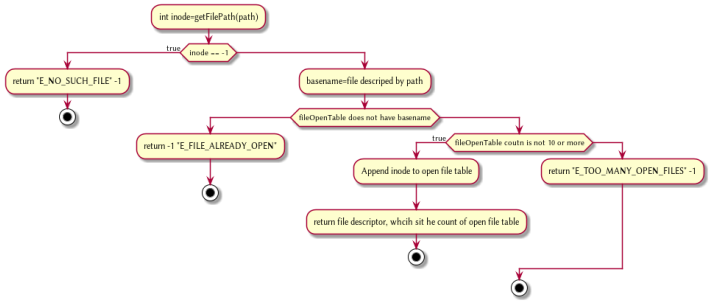
Ouptut inode number of where it is, or -1 if it's not found.



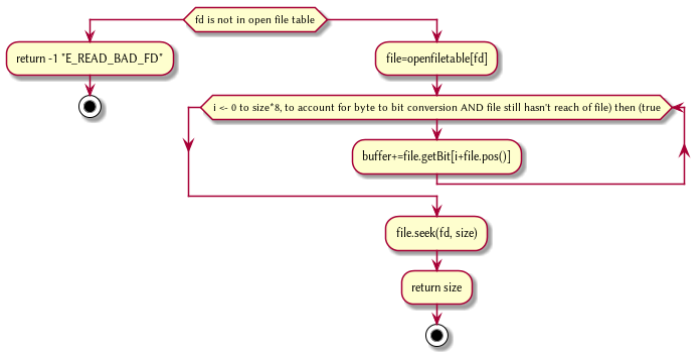
$\text{File}_{\text{Create}}$(string path) Create a new file at path. There is a check to see if that file already exist, and if there's a free datablock for it.



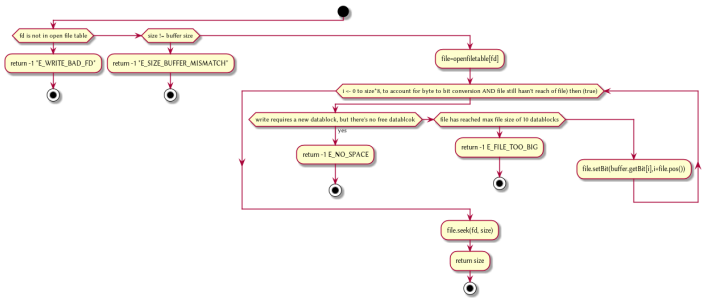
$\text{File}_{\text{Open}}$(string path) returns the file descriptor of the file, which can be used to read and write to it.



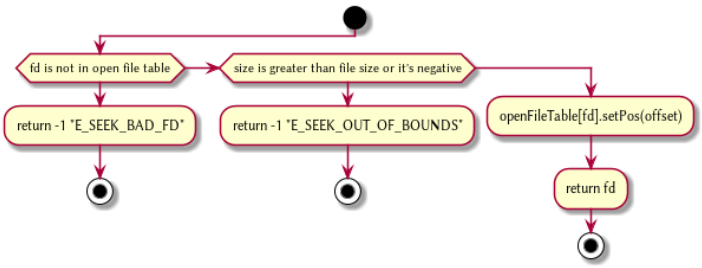
$\text{File}_{\text{Read}}$(int fd, string buffer, int size IN BYTES) Buffer reads size from the file in fd. Note the file in open file table shuold move by size



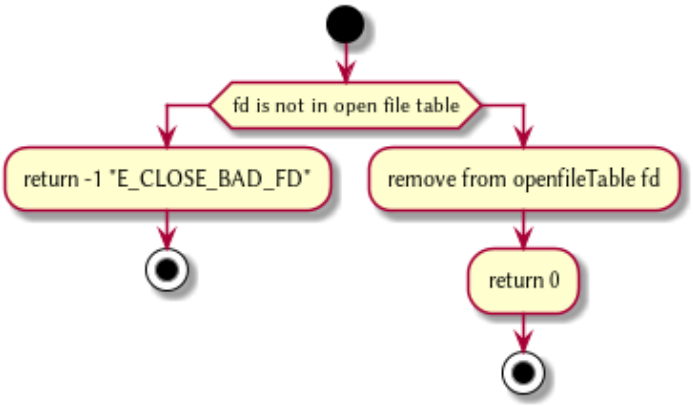
$\text{File}_{\text{Write}}$(int fd, string buffer, int size IN BYTES) Write from buffer to the file. NOTE SIZE HAS TO BE CONSISNET. If it's not, stop the program



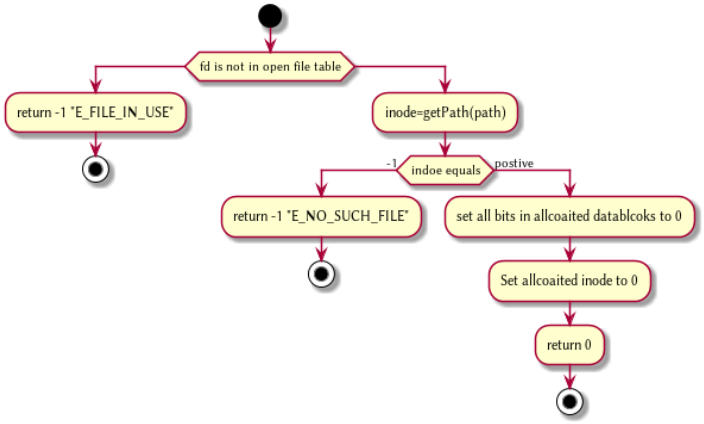
$\text{File}_{\text{Seek}}$(int fd, int offset) move the file forward by offset.



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

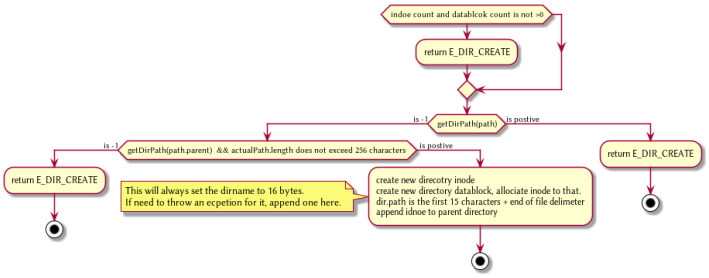


$\text{File}_{\text{UnLink}}$(string path) Delete file from the filesystem.



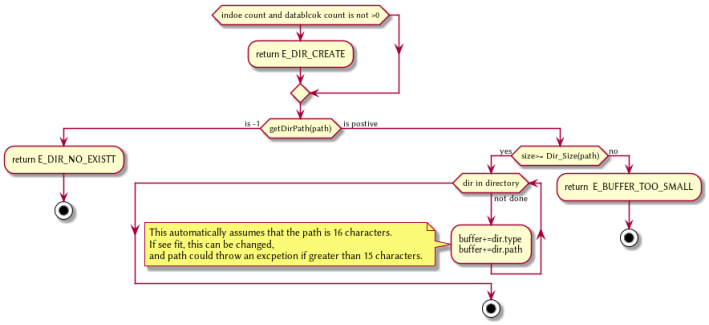
5. Directory DIR

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

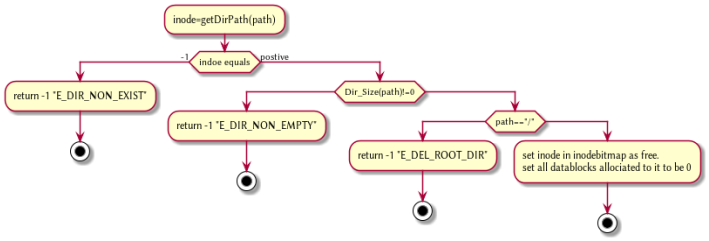


@startuml

Dir_{Read}(string path, string buffer, itn size) Read the contents of a directory.

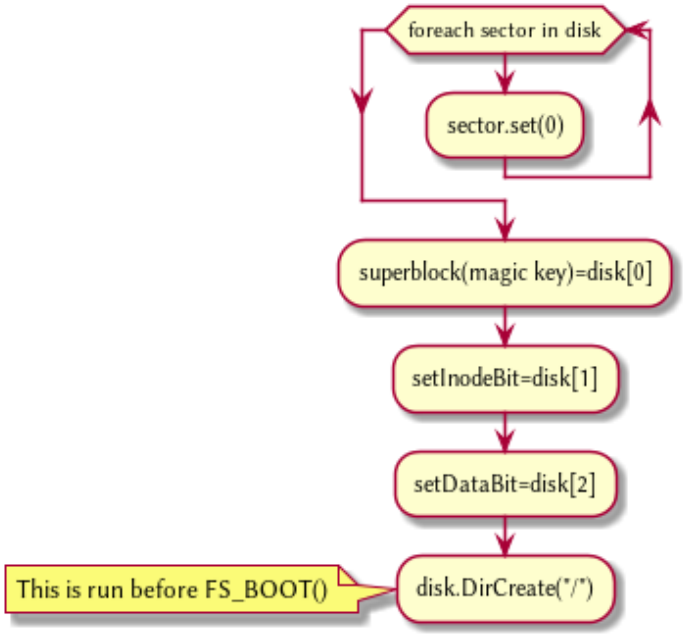


$\text{Dir}_{\text{Unlink}}$(string path) Remove file from drive

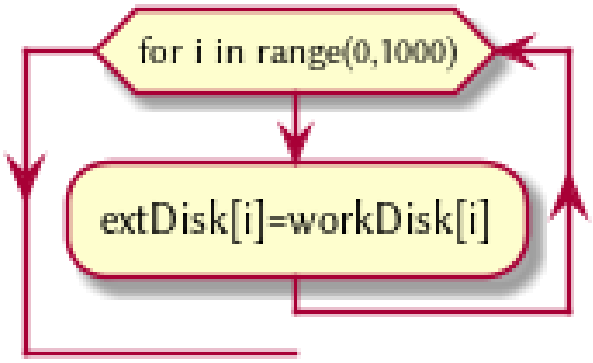


6. Disk DISK

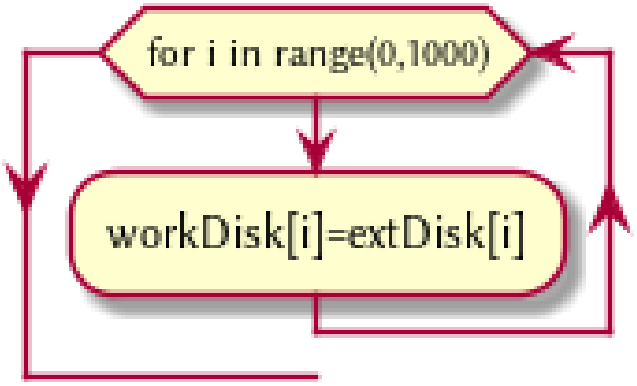
$\text{DISK}_{\text{INIT}}$() Set all the data in the disk to be 0



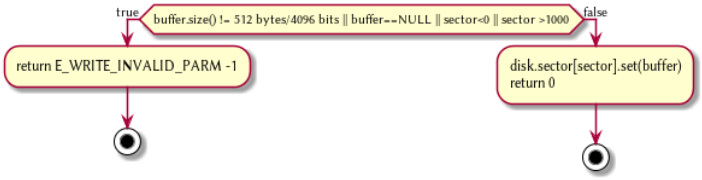
$\text{DISK}_{\text{LOAD}}()$ Save external disk to workign disk. Done when booting.



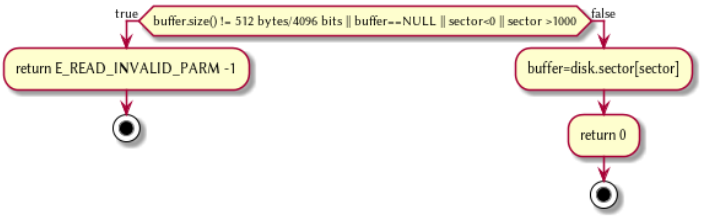
$\text{DISK}_{\text{SAVE}}()$ Save working disk to loading. Called by $\text{FS}_{\text{SYNC}}()$



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



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


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