FatOS

Generated by Doxygen 1.8.11

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 FileSystem Struct Reference

Structure describing a FatOS filesystem. This allows for multiple filesystems to be supported (with multiple drives) and handled as a linked list.

```
#include <fs.h>
```

Collaboration diagram for FileSystem:

Public Attributes

- int drive_id
- char ∗ FATpointer
- struct FileSystem ∗ next
- char ∗ fileList
- int fileListSize

3.1.1 Detailed Description

Structure describing a FatOS filesystem. This allows for multiple filesystems to be supported (with multiple drives) and handled as a linked list.

3.1.2 Member Data Documentation

3.1.2.1 int FileSystem::drive_id

The drive ID holding the fs

3.1.2.2 char∗ FileSystem::FATpointer

pointer to the FAT for that filesystem

3.1.2.3 char∗ FileSystem::fileList

a pointer to the fileList loaded into memory

3.1.2.4 int FileSystem::fileListSize

Length in bytes of the root filelist

3.1.2.5 struct FileSystem∗ FileSystem::next

allows for easy chaining of filesystems

The documentation for this struct was generated from the following file:

- kernel/IO/fs.h

3.2 IDT_Descriptor Struct Reference

Public Attributes

- unsigned short **limit**
- unsigned int **base**

The documentation for this struct was generated from the following file:

- kernel/interruptions/interruption.h

3.3 IDT_Entry Struct Reference

Public Attributes

- unsigned short **base_lower**
- unsigned short **segment_selector**
- unsigned char **always_0**
- unsigned char **flags**
- unsigned short **base_higher**

The documentation for this struct was generated from the following file:

- kernel/interruptions/interruption.h

3.4 MemoryHeader Struct Reference

Collaboration diagram for MemoryHeader:

Public Attributes

- int **length**
- unsigned int **flags**
- struct MemoryHeader ∗ **previous**
- struct MemoryHeader ∗ **next**

The documentation for this struct was generated from the following file:

- kernel/memory/memorymanager.h

3.5 Memorymap_entry Struct Reference

Public Attributes

- unsigned long long **base_address**
- unsigned long long **length**
- unsigned int **type**

The documentation for this struct was generated from the following file:

- kernel/IO/memorymap.h

Chapter 4

File Documentation

4.1 kernel/IO/fs.c File Reference

contains the implementation for the filesystem functions described in fs.h

```
#include "fs.h"
```
Include dependency graph for fs.c:

4.2 kernel/IO/fs.h File Reference

Contains all functions for filesystem handling.

```
#include "floppy.h"
#include "string.h"
#include "memory.h"
```
Include dependency graph for fs.h: This graph shows which files directly or indirectly include this file:

Classes

- struct FileSystem

  Structure describing a FatOS filesystem. This allows for multiple filesystems to be supported (with multiple drives) and handled as a linked list.

Macros

- #define FAT_SECTORS 10

  the numbers of sectors in the FAT
- #define FAT_FIRST 1

  the id of the sector at which FAT is located (in LBA)
- #define FAT_ENTRYAT(fatPtr, id) (∗((unsigned short∗)((int) fatPtr + 2∗id)))

  returns a pointer to a FAT entry of given FAT and entry id

Typedefs

- typedef struct FileSystem **FileSystem**

Functions

- unsigned short filesystem_getNextSector (unsigned short sector, FileSystem ∗fs)

    Gets sector after a given sector in the FAT.
- unsigned short filesystem_getNextSectorRaw (unsigned short sector, FileSystem ∗fs)

    Gets sector after a given sector in the FAT, without striping the 'used' flag (highest bit)
- unsigned short filesystem_getNthSector (unsigned short firstSector, int n, FileSystem ∗fs)

    gets the Nth sector after a given sector in FAT. Indexing starts at 0.
- unsigned short filesystem_getLastSector (unsigned short firstSector, FileSystem ∗fs)

    gets the last sector of a chain of sectors in the FAT
- void filesystems_init ()

    initializes all filesystems accessible (one per detected floppy drive). SO FAR : ONLY boot filesystem
- void filesystem_init (FileSystem ∗fs)

    init file system (especially load file table at the given address)
- unsigned short filesystem_getFileSectors (unsigned short firstSector, FileSystem ∗fs)

    computes number of sector a file actual uses (based on FAT, not on advertised size). Based on the first sector. Reads it from FAT loaded into memory.
- int filesystem_readbytesByFirstSector (unsigned short firstSector, int startByte, int length, char ∗buffer, File←System ∗fs)

    Reads the given amount of bytes from file given at first sector into provided buffer.
- int filesystem_writebytesByFirstSector (unsigned short firstSector, int startByte, int length, char ∗buffer, File←System ∗fs)

    Writes the given amount of bytes from given buffer at sector indicated returns -1 upon error, 0 upon success.
- FileSystem ∗ getFirstFileSystem ()

    Gets the first filesystem handled by the filesystem handler (boot filesystem)
- void filesystem_LoadFileList (FileSystem ∗fs)

    Loads the filetable for a given filesystem.
- FileSystem ∗ getFileSystemByDriveId (int id)

    Returns a pointer to the filesystem of given drive ID.
- void filesystem_list (char ∗fileListData, int length)

    Prints the content of a given fileList (loaded at given position in memory, with given size)
- unsigned short filesystem_findFirstSector (char ∗filename, char ∗filelist, int size)

    Find first sector of entry of specified name in table loaded in buffer, of given size.
- void filesystem_setNextSector (unsigned short start, unsigned short next, FileSystem ∗fs)

    Sets the next sector in the FAT loaded in memory (does not rewrite it in drive)
- unsigned short filesystem_findEmptySector (unsigned short first, FileSystem ∗fs)

    This finds an empty sector to write to, after a given first one (initial value, set to 0 if all drive)
- int filesystem_appendSectors (unsigned short first, int count, int rewrite, FileSystem ∗fs)

    This adds sectors at the end of chained list of sectors in FAT. Rewrites FAT into drive if specified so.
- int filesystem_appendBytes (FileSystem ∗fs, char ∗fileListData, int size, char ∗filename, int bytesToAppend)

    Appends bytes to the selected file.
- int filesystem_getBytes (char ∗fileListData, int length, char ∗filename)

    Returns the number of bytes of a file in a given filelist.
- int filesystem_setBytes (char ∗fileListData, int length, char ∗filename, int newSize)

    sets the number of bytes of file in a given filelist.
- int filesystem_loadSubFileList (FileSystem ∗fs, char ∗fileListData, int size, char ∗subFileListName, char ∗∗resBuffer)

loads a subfilelist into an allocated buffer. This buffer needs to be freed after using sys_free.

- int filesystem_subList (FileSystem ∗fs, char ∗fileListData, int size, char ∗subListName)

  Lists through a path ex. /subdir/ recursively until destination reached.

- int filesystems_driveList (char ∗fullPath)

  Lists the contents of a given directory. End user function, supports complete path, including drive id. Supports 1-digits drives ids (0-9)

Variables

- struct FileSystem **__attribute__**

4.2.1 Detailed Description

Contains all functions for filesystem handling.

Author

Anton CLAES

Date

2017

4.2.2 Macro Definition Documentation

4.2.2.1 #define FAT_ENTRYAT(*fatPtr, id*) (∗((unsigned short∗)((int) fatPtr + 2∗id)))

returns a pointer to a FAT entry of given FAT and entry id

Parameters

fatPtr	a pointer to the FAT in memory
id	the id of the entry (indexed start at 0)

4.2.3 Function Documentation

4.2.3.1 int filesystem_appendBytes (FileSystem ∗ *fs,* char ∗ *fileListData,* int *size,* char ∗ *filename,* int *bytesToAppend*)

Appends bytes to the selected file.

Parameters

fs	the filesystem on which to operate changes
fileListData	the filelist on which to operate changes
size	the length in bytes of the filelist
filename	the name of the files to add bytes to
bytesToAppend	the amount of bytes to be appened modifies FAT and filelist where the file is identified

TODO HERE

4.2.3.2 int filesystem_appendSectors (unsigned short *first*, int *count*, int *rewrite*, FileSystem ∗ *fs*)

This adds sectors at the end of chained list of sectors in FAT. Rewrites FAT into drive if specified so.

Parameters

first	a sector in a chain at the end of which to add sectors
count	the number of sectors to add
rewrite	: 1 to rewrite FAT to disk, 0 to just update memory
fs	the filesystem on which to operate changes

Returns

0 upon success

4.2.3.3 unsigned short filesystem_findEmptySector (unsigned short *first*, FileSystem ∗ *fs*)

This finds an empty sector to write to, after a given first one (initial value, set to 0 if all drive)

Parameters

first	to first sector after which to look for an empty one
fs	the filesystem on which to perform operations

Returns

the sector found (0 upon failure)

4.2.3.4 unsigned short filesystem_findFirstSector (char ∗ *filename*, char ∗ *filelist*, int *size*)

Find first sector of entry of specified name in table loaded in buffer, of given size.

Parameters

filename	a string representing the filename in the filelist
filelist	the filelist loaded in memory
size	the size of the filelist

Returns

the first sector of the file as advertised in the filelist

4.2.3.5 int filesystem_getBytes (char ∗ *fileListData,* int *length,* char ∗ *filename*)

Returns the number of bytes of a file in a given filelist.

Parameters

fileListData	the filelist into which to look for the file
length	the length of the filelist
filename	the name of the file

Returns

-1 if not found, the number of bytes of the files as indicated by the filelist otherwise

4.2.3.6 unsigned short filesystem_getFileSectors (unsigned short *firstSector,* FileSystem ∗ *fs*)

computes number of sector a file actual uses (based on FAT, not on advertised size). Based on the first sector. Reads it from FAT loaded into memory.

Parameters

firstector	the firstsector of the file. If the sector is not the first one it will return the number of sectors until the end of the file
fs	the filesystem to read from

Returns

the id of the last sector of the file

4.2.3.7 unsigned short filesystem_getLastSector (unsigned short *firstSector,* FileSystem ∗ *fs*)

gets the last sector of a chain of sectors in the FAT

Parameters

first	sector any sector in the chain to start with
fs	the filesystem to get the FAT from

Returns

the id of the last sector in the chain

4.2.3.8 unsigned short filesystem_getNextSector (unsigned short *sector,* FileSystem ∗ *fs*)

Gets sector after a given sector in the FAT.

Parameters

sector	the current sector
fs	the filesystem to get the FAT from

Returns

the next sector's id in LBA, 0 if no next sector

4.2.3.9 unsigned short filesystem_getNextSectorRaw (unsigned short *sector*, FileSystem ∗ *fs*)

Gets sector after a given sector in the FAT, without striping the 'used' flag (highest bit)

Parameters

sector	the current sector
fs	the filesystem to get the FAT from

Returns

the next sector's id in LBA, 0 if no next sector, with the used flag not striped

4.2.3.10 unsigned short filesystem_getNthSector (unsigned short *firstSector*, int *n*, FileSystem ∗ *fs*)

gets the Nth sector after a given sector in FAT. Indexing starts at 0.

Parameters

firstSector	the firstSector to seek from (not necessarely the first sector of the file)
n	the number of sectors to skip (n=0 returns the firstSector)
fs	the filesystem to get the FAT from

Returns

the id of the Nth sector

4.2.3.11 void filesystem_init (FileSystem ∗ *fs*)

init file system (especially load file table at the given address)

Parameters

fs	the filesystem to initialize

4.2.3.12 void filesystem_list (char ∗ *fileListData,* int *length*)

Prints the content of a given fileList (loaded at given position in memory, with given size)

Parameters

fileListData	the filelist (loaded in memory)
length	the length of the filelist

4.2.3.13 void filesystem_LoadFileList (FileSystem ∗ *fs*)

Loads the filetable for a given filesystem.

Parameters

fs	the filesystem for which to load the root filelist (located at sector 11)

4.2.3.14 int filesystem_loadSubFileList (FileSystem ∗ *fs,* char ∗ *fileListData,* int *size,* char ∗ *subFileListName,* char ∗∗ *resBuffer*)

loads a subfilelist into an allocated buffer. This buffer needs to be freed after using sys_free.

Parameters

fs	the filesystem from which to load the sub filelist
fileListData	the current filelist into which to look for the subFileList (works recursively)
size	the size of the current filelist in bytes
subFileListName	the name of the subFileList(directory to look for)
resBuffer	that will be allocated and into which the subfilelist will be loaded into

Returns

the size of the loaded subfilelist in bytes, -1 if failed

4.2.3.15 int filesystem_readbytesByFirstSector (unsigned short *firstSector,* int *startByte,* int *length,* char ∗ *buffer,* FileSystem ∗ *fs*)

Reads the given amount of bytes from file given at first sector into provided buffer.

Parameters

firstSector	the firstSector of the file
startByte	the firstByte to read
length	the number of bytes to read
a	buffer to read into
fs	the filesystem to read from

Returns

-1 upon error, 0 upon success

4.2.3.16 int filesystem_setBytes (char ∗ *fileListData*, int *length*, char ∗ *filename*, int *newSize*)

sets the number of bytes of file in a given filelist.

Parameters

fileListData	the filelist into which to look for the file
length	the length of the filelist
filename	the name of the file
newSize	the size to be set to the file

Returns

-1 if not found

4.2.3.17 void filesystem_setNextSector (unsigned short *start*, unsigned short *next*, FileSystem ∗ *fs*)

Sets the next sector in the FAT loaded in memory (does not rewrite it in drive)

Parameters

start	the sector to set the next one to
next	the sector to be written as following the "start" one
fs	the filesystem on whose FAT to operate changes on

4.2.3.18 int filesystem_subList (FileSystem ∗ *fs*, char ∗ *fileListData*, int *size*, char ∗ *subListName*)

Lists through a path ex. /subdir/ recursively until destination reached.

Parameters

fs	the filesystem into which to look for the sublist
fileListData	the current filelist
size	the size of the current filelist
subListName	the name of the sublist to look for

Returns

-1 upon failure, 0 upon success

4.2.3.19 int filesystem_writebytesByFirstSector (unsigned short *firstSector,* int *startByte,* int *length,* char ∗ *buffer,* FileSystem ∗ *fs*)

Writes the given amount of bytes from given buffer at sector indicated returns -1 upon error, 0 upon success.

- firstSector the first sector of the file
- startByte the first byte to write too
- length the number of bytes to write
- fs the filesystem to write into

If the required data writes beyond file limit then the function will fail. It will not append sectors to the file.

Parameters

firstSector	the firstSector of the file
startByte	the firstByte to write
length	the number of bytes to write
a	buffer to read from
fs	the filesystem to write to

Returns

-1 upon error, 0 upon success

4.2.3.20 int filesystems_driveList (char ∗ *fullPath*)

Lists the contents of a given directory. End user function, supports complete path, including drive id. Supports 1-digits drives ids (0-9)

Parameters

fullPath	the absolute path to the directory

Returns

-1 upon failure

4.2.3.21 FileSystem∗ getFileSystemByDriveId (int *id*)

Returns a pointer to the filesystem of given drive ID.

Parameters

id	the drive's id

Returns

a pointer to the filesystem's structure

4.2.3.22 FileSystem∗ getFirstFileSystem ( )

Gets the first filesystem handled by the filesystem handler (boot filesystem)

Returns

a pointer a filesystem structure representing the first filesystem

4.3 kernel/kernel.c File Reference

This contains the kernel's entry point.

```
#include "IO/print.h"
#include "memory.h"
#include "interruption.h"
#include "memorymanager.h"
#include "fs.h"
```

Include dependency graph for kernel.c:

Functions

- int main ()
  OS kernel entry point.

4.3.1 Detailed Description

This contains the kernel's entry point.

Author

Anton Claes

Date

2017

4.3.2 Function Documentation

4.3.2.1 int main ( )

OS kernel entry point.

Returns

0 after console exited

Index