Operating Systems 2016/17
Assignment 13

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marc Rittinghaus

Submission Deadline: Monday, February 6th, 2017 – 23:59

A new assignment will be published every week, right after the last one was due. It must be completed before its submission deadline.

The assignments must be filled out online in ILIAS. Handwritten solutions are no longer accepted. You will find the online version for each assignment in your tutorial's directory. **P-Questions** are programming assignments. Download the provided template from ILIAS. Do not fiddle with the compiler flags. Submission instructions can be found on the first assignment.

In this assignment you will delve into the implementation of file systems.

T-Question 13.1: File System Implementation

a. What three kernel data structures are required to manage open files? Explain the meaning of each. **3 T-pt**

b. Why is the file name not stored in the inode? **1 T-pt**

c. How does the file system determine if an inode and thus the blocks allocated to the file can be deleted? **1 T-pt**

d. What is the benefit of a file allocation table (FAT) compared to chained allocation? **1 T-pt**

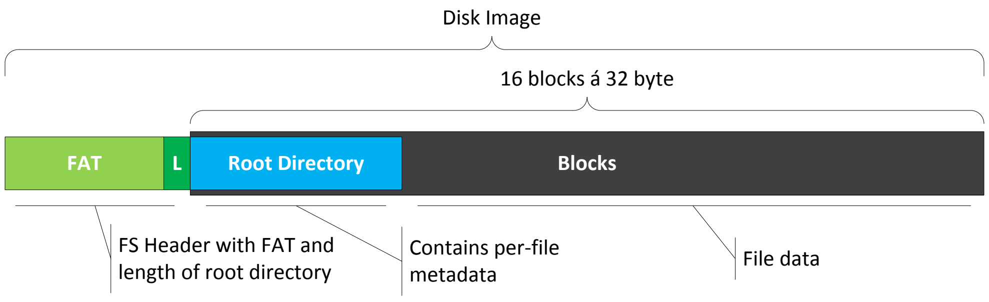
e. Explain the concept of indexed disk space allocation. What approach allows this allocation type to represent very large files while still being efficient for small files? **2 T-pt**

P-Question 13.1: File System Implementation

Download the template **p1** for this assignment from ILIAS. You may only modify and upload the file `filesystem.c`.

In this assignment you will write your own minimalistic FAT-like file system and handle open files for it. The template comes with a disk image (`test.image`) that has been formatted with the assignment's file system.

The file system is fixed to manage 16 disk blocks with 32 bytes each and supports only a single directory (the root directory). The root directory is stored within the available 16 disk blocks, the header (FAT, etc.) is stored in front of the 16 blocks. The file system uses the following on-disk organization, where the root directory starts at block 0:


Disk Image
16 blocks á 32 byte
FAT
L
Root Directory
Blocks
FS Header with FAT and length of root directory
Contains per-file metadata
File data

Per-file meta information such as the file names and sizes are stored in the root directory, whereas each entry in the directory represents a single file and has a fixed size (reserving 8 bytes for the file name). Note that the root directory may not be stored in contiguous blocks!

Open files are represented through open file handles that contain the current seek pointer (i.e., the offset of the next byte to read from the file) and the corresponding block.

You'll find all relevant data structures and sizes in the template's header file. You can use the command `hexdump -C test.image` to view a hex dump of the disk image.

a. Write a function that maps the given full disk image into the address space of your program using the `mmap` system call. Your function should provide access to the file system through the `FileSystem` structure by casting and returning the address of the image in memory after the call to `mmap`. Return `NULL` on any error. **2 P-pt**

```
FileSystem *mapFileSystem(char *diskFile);
```

b. Write a function that determines if more bytes can be read with a supplied open file handle. Return 0 if not, 1 otherwise. **1 P-pt**

```
int _hasMoreBytes(OpenFileHandle *handle);
```

c. Write a function that reads the next byte from the file specified with the given open file handle. Use the file allocation table to determine the next block, if you reach the end of a block. *Hints*: Remember to adjust the file handle appropriately. **2 P-pt**

```
char _readFileByte(OpenFileHandle *handle);
```

d. Write a function that opens a file from the file system by name. Your function should fulfill the following requirements: **3 P-pt**

- Uses the provided handle to the root directory and the `readFile` method to iterate over the root directory's `DirectoryEntry` structures.
- Searches for the requested file name (case sensitive).
- Closes the root directory handle.
- Returns a new file handle for the requested file using `_openFileAtBlock` on success, `NULL` otherwise.

Hints: When working with the root directory, consider it as a regular file whose data are directory entries.

```
OpenFileHandle *openFile(FileSystem *fs, char *name);
```

Total:
8 T-pt
8 P-pt