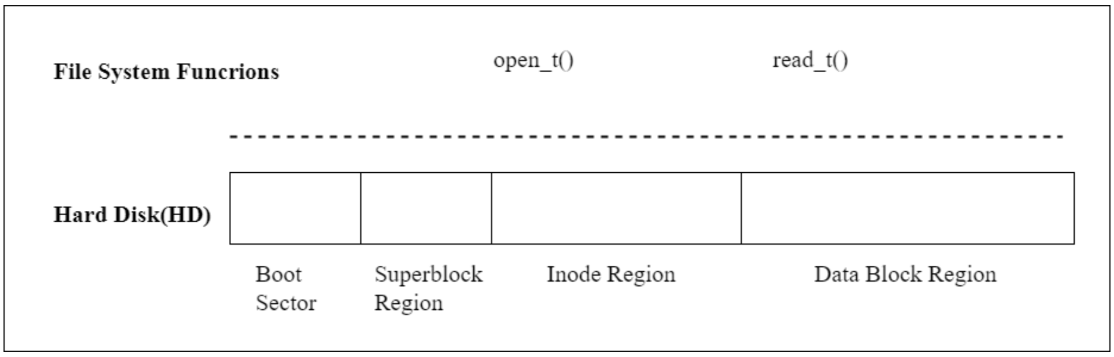
CSCI 3150 Introduction to Operating Systems
Project
Deadline: 23:55, 29 December 2019

In this project, you are required to implement a simple file system called ***SFS***. An overview of SFS is shown in the figure below.

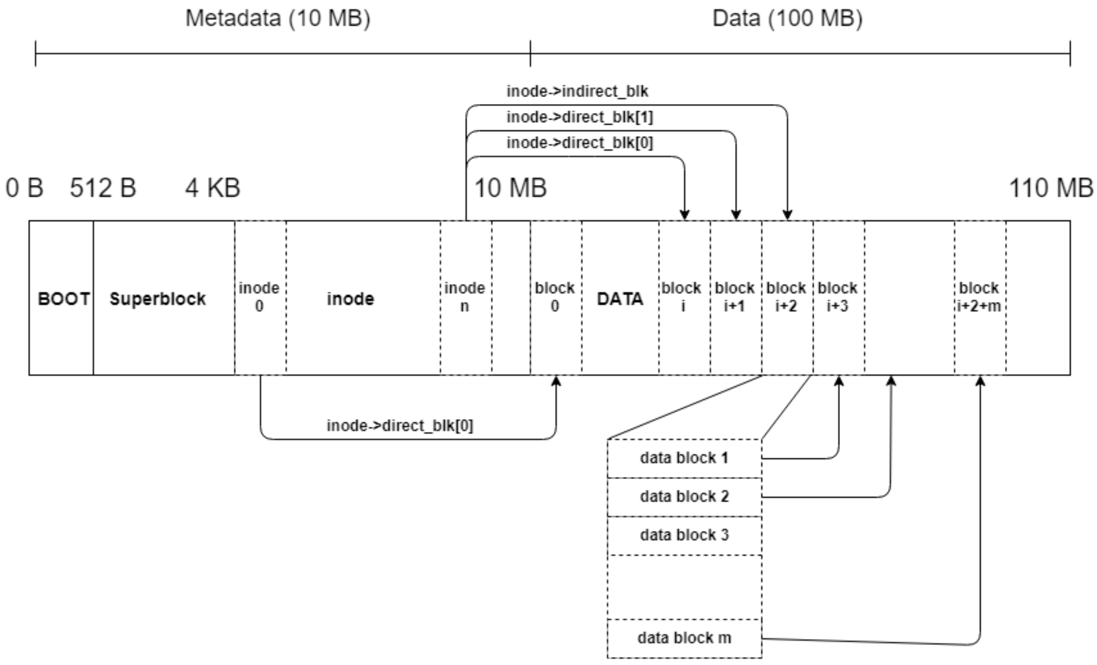


1. SFS (Simple File System)

SFS works on a file called **HD** that is a 110MB file (initialized properly) and you can find it from the zip file provided.

What you are required to do is implementing two functions **open_t()** and **read_t()**.

- These two file-system-related functions are based on the simple file system. An illustrative format on HD is shown below:



As shown above, in HD, there are two regions: the metadata and data regions. The metadata region is inside the first 10MB; it contains a boot sector (the first 512 bytes), the

superblock and inode regions. The superblock region is from 512B to 4KB, and the inode region from 4KB to 10MB.The data region is from 10 MB to 110 MB, in which it is divided into data blocks (each data block is 4 KB).
The superblock region defines the layout and its format can be found from the following structure:

```
struct   superblock          /*The key information of filesystem */
{
        int   inode_offset;        /* The start offset of the inode region */
        int   data_offset;         /* The start offset of the data region */
        int   max_inode;           /* The maximum number of inodes */
        int   max_data_blk;        /* The maximum number of data blocks */
        int   next_available_inode;   /* The index of the next free inode */
        int   next_available_blk;   /* The index of the next free block*/
        int   blk_size;              /* The size per block */
};
```

Basically, the inode region starts at 4 KB (inode_offset); the data region starts at 10 MB (data_offset), the maximum number of inodes is 100 (max_inode); the maximum number of data blocks is 25600; next_available_inode and next_available_blk are used to represent the indexes of the next free inode and the next free block, respectively; the block size is 4 KB.

The inode region contains inodes that can be retrieved based on its index in the inode region (called the inode number). An inode is used to represent a file, and is defined based on the following structure:

```
struct inode                 /* The structure of inode, each file has only one inode */
{
        int   i_number;       /* The inode number */
        time_t   i_mtime;       /* Creation time of inode*/
        int   i_type;       /* Regular file for 0, directory file for 1 */
        int   i_size;            /* The size of file */
        int   i_blocks;         /* The total numbers of data blocks    */
        int   direct_blk[2];   /*Two direct data block pointers    */
        int   indirect_blk;     /*One indirect data block pointer */
        int   file_num;         /* Number of files under a directory (0 for regular file)*/
};
```

Some related parameters can be found as follows:

```
#define SB_OFFSET     512             /* The offset of superblock region*/
```

```
#define INODE_OFFSET        4096   /* The offset of inode region */
#define DATA_OFFSET        10485760 /* The offset of data region */
#define MAX_INODE           100    /* The maximum number of inode */
#define MAX_DATA_BLK      25600  /* The maximum number of block */
#define BLOCK_SIZE      4096                    /* The size per block */
#define MAX_NESTING_DIR 10                /* The nesting number of directory */
#define MAX_COMMAND_LENGTH    50 /* The maximum command length */
```

In SFS, an inode contains two direct data block pointers and one single indirect data block pointer. There are two types of files: regular and directory files. The content of a directory file should follow the following structure:

```
typedef    struct    dir_mapping    /* Record file information in directory file */
{
        char    dir[20];          /* The file name in current directory */
        int    inode_number;      /* The corresponding inode number */
}DIR_NODE;
```

Each directory file should at least contain two mapping items, "." and "..", for itself and its parent directory, respectively.

Based on SFS, the prototypes of the three filesystem-related functions are shown as follows:

1) int   open_t(const char *pathname);

   Description: Given an absolute *pathname* for a file, open_t() returns the corresponding inode number of the file or -1 if an error occurs. The returned inode number will be used in subsequent read_t().
2) int   read_t(int inode_number, int offset, void *buf, int count);

   Description: read_t() attempts to read up to *count* bytes from the file starting at *offset* (with the inode number *inode_number*) into the buffer starting at *buf*. It commences at the file offset specified by *offset*. If *offset* is at or past the end of file, no bytes are read, and read_t() returns zero. On success, the number of bytes read is returned (zero indicates end of file), and on error, -1 is returned.

2. Requirements

In this project, you need to implement open_t() and read_t().

After unzipping this zip file, you can find the following files:

- *call.c*: **The source code for open_t() and read_t() that you should implement**. **In call.c, you are allowed to create any auxiliary functions that can help your implementation. But only "open_t()" and read_t() are allowed to call these**

auxiliary functions.

- *call.h, inode.h , superblock.h*: The header files that define the data structures and function prototypes.
- *HD*: The hard disk file, which has been initialized properly (110 MB);

This project will be graded by Mr. Chen Zizhan (Email: chenzz@cse.cuhk.edu.hk). Your programs **must be able to be compiled/run under the XUbuntu environment (in Lab One)**.

What to submit – A zip file that ONLY contains call.c.