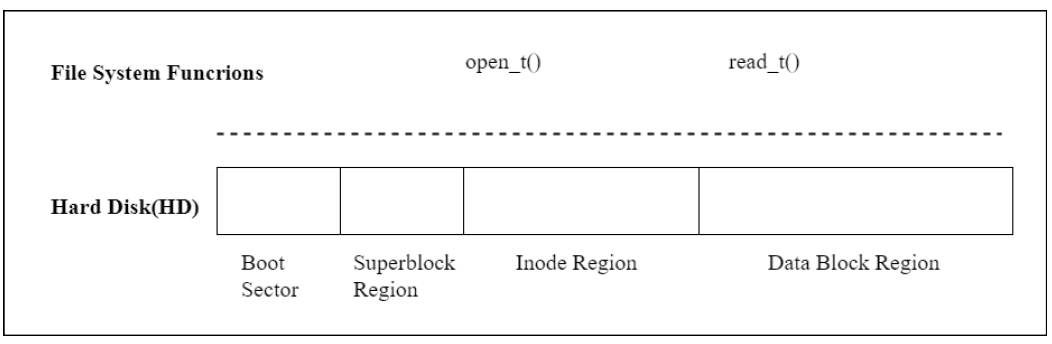
# CSCI 3150 Introduction to Operating Systems Project: Implementation of SFS

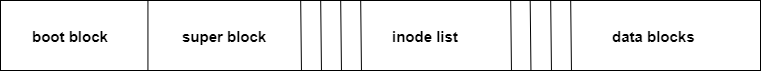
**Deadline: 23:59, Dec. 5, 2020**

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

## 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 4KB to 8KB, and the inode region from 8KB 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 \_super\_block\_ /\*The key information of filesystem \*/*

*{*

*int inode\_offset; int data\_offset; int max\_inode; int max\_data\_blk;*

*int next\_available\_inode;*

*int next\_available\_blk; int blk\_size;*

*};*

Basically, the inode region starts at 8 KB (inode\_offset); the data region starts at 10 MB (data\_offset); 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 inode\_number; /\* The inode number \*/*

*time\_t creation\_time; /\* Creation time of inode\*/*

*int file\_type; /\* Regular file for 0, directory file for 1 \*/ int file\_size; /\* The size of file \*/*

*int blk\_num; /\* 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 4096 /\* The offset of superblock region\*/ #define INODE\_OFFSET 8192 /\* The offset of inode region \*/ #define DATA\_OFFSET 10485760 /\* The offset of data region \*/ #define BLOCK\_SIZE 4096 /\* The size per block \*/

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 two 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 functions in read\_t().

* 1. 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.

## 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. GUO Yang (Email: [yangguo@cse.cuhk.edu.hk](mailto:yangguo@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.