

Department of Computer Science and Engineering
The Chinese University of Hong Kong

CSCI/CENG 3150: Introduction to Operating Systems

Lab Nine: mkfs – Format File Systems

Objectives:

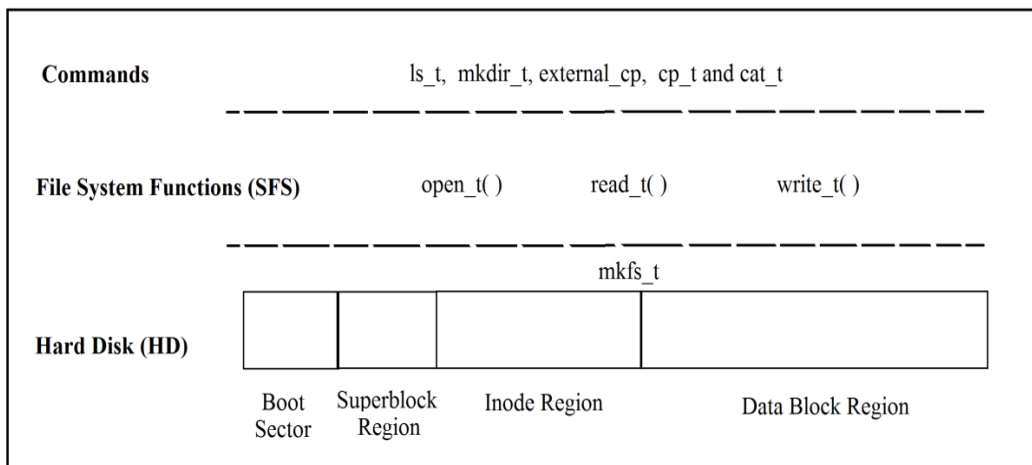
- (1) Understand three key data structures (*superblock*, *inode*, and *directory entry*) in SFS (Simple File System) that you will implement in Assignment Three.
- (2) Learn how to format a file system via a simplified version of mkfs.

Note: In Assignment Three, you will be asked to implement a simple file system, and this lab is *the first step* in order to achieve that.

1. Background

A file system is used to **organize** and **access** the data stored in storage devices (e.g. hard drives, SSDs, etc.) via files and directories. It *organizes* data via data structures such as superblock, inode, and directory entry, and provides a data *access* interface via system calls such as open(), read() and write().

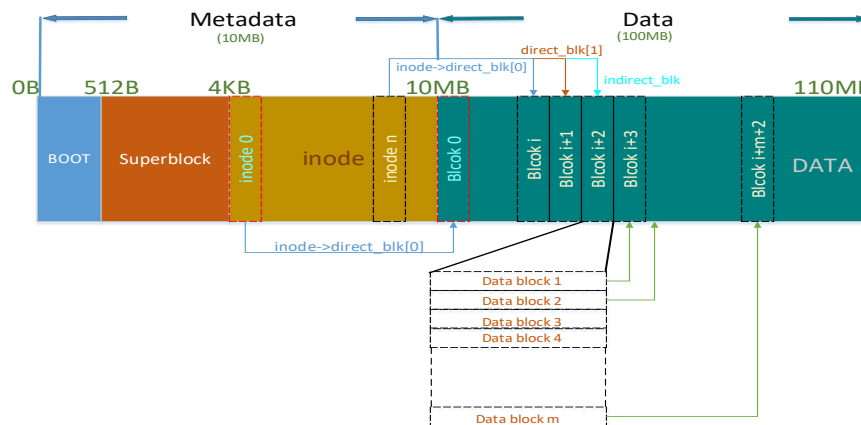
This lab is the first step to implement SFS in Assignment Three based a virtual disk (a file called HD). An overview of SFS is shown in the figure below.



Specifically, in this lab, we will study a simpler version of “mkfs” called “mkfs_t” by which we can format a file system. Basically, *mkfs_t* will do three things:

- 1) Write the parameters of SFS into the superblock region in HD;
- 2) Create the first inode (for root directory) in the inode region in HD;
- 3) Create the first two directory entries (for “.” and “..”) for the root directory in the data block region.

An illustrative format on HD after using “mkfs_t” is shown below:



1.1 Write the parameters of SFS into the superblock region

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. To make it simple, you do not need to reclaim inodes or data blocks, and **you can simply obtain the next available inode (data block) index based on next_available_inode (next_available_blk) when you create a file (allocate data blocks).**

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 */
```

1.2 Create the first inode (for root directory) in the inode region in HD

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) */
};
```

We need to create the first inode (inode 0) for the root directory based on the above data structure.

1.3 Create directory entries (“.” and “..”) for the root directory in the data block region

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. For the root directory, its data content will be written in data block 0 based on the above data structure.

2. A Simple *mkfs* Program

Next, we will learn how *mkfs_t* (a simple *mkfs*) works.

2.1 Compile and Run

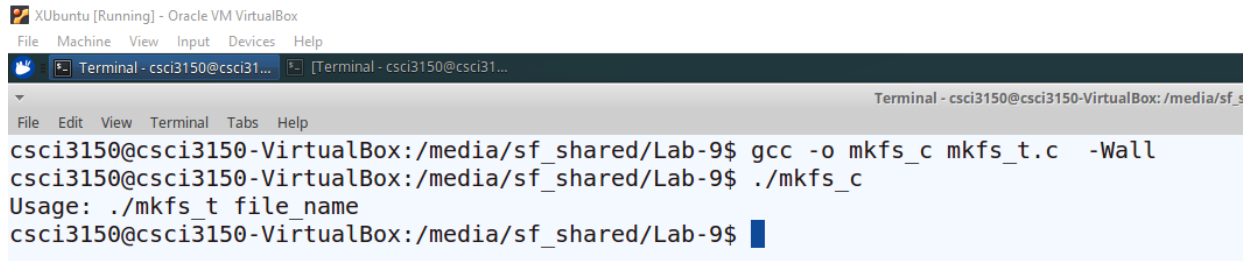
Download and copy *simple-shell.c* into your current directory and compile it as follows:

```
gcc -o mkfs_t mkfs_t.c -Wall
```

Then you can run it as follows:

```
./mkfs_t
```

You should see the following in the terminal (or something similar depending on which directory you run it):



```
XUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal - csci3150@csci3150-VirtualBox: /media/sf_shared/Lab-9$ gcc -o mkfs_c mkfs_t.c -Wall
csci3150@csci3150-VirtualBox: /media/sf_shared/Lab-9$ ./mkfs_c
Usage: ./mkfs_t file_name
csci3150@csci3150-VirtualBox: /media/sf_shared/Lab-9$
```

Basically, you need to input a file name (for the virtual disk). In our case, we use **HD** that is a 110MB file (initially empty). You can find **HD** from the zip file provided, or create it by using the command below:

```
dd if=/dev/zero of=./HD bs=4k iflag=fullblock,count_bytes count=110M
```

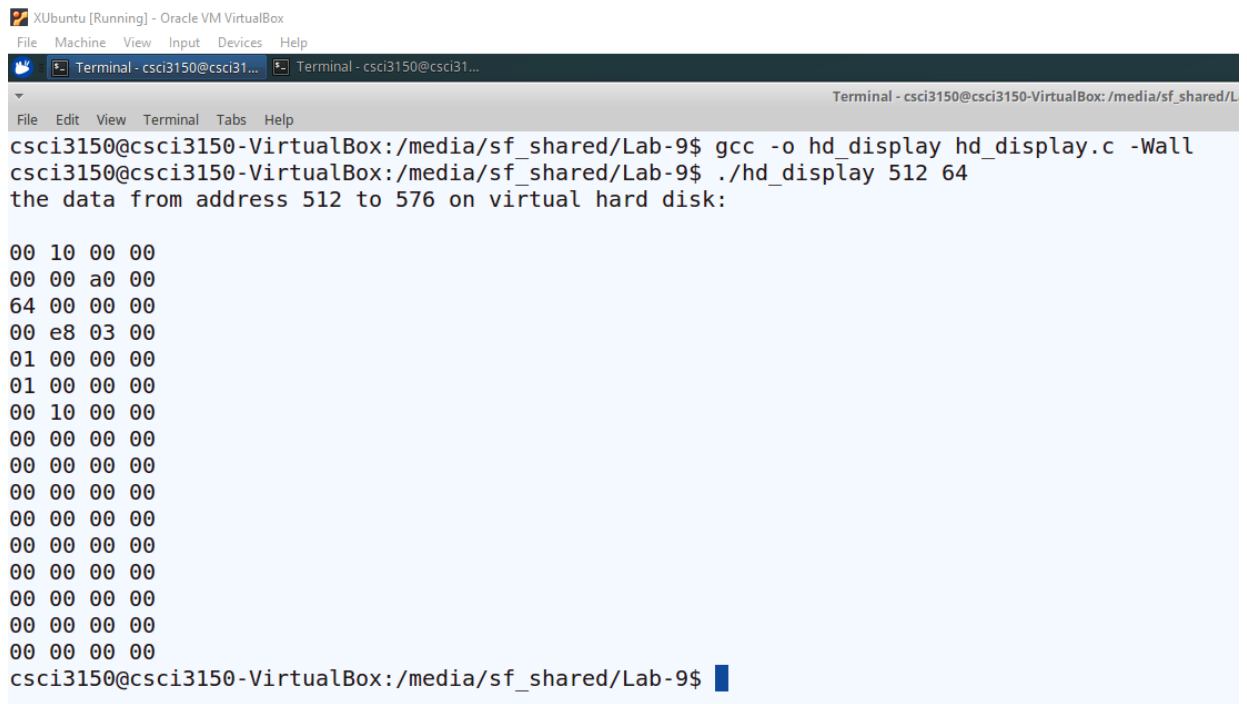
Then you can format HD by using mkfs_t as follows:

```
./mkfs_t HD
```

You can use **hd_display** to display the contents in the superblock (offset = 512), inode (offset = 4096) and data block (offset=10485760) regions in HD as follows:

```
gcc -o hd_display hd_display.c -Wall  
./hd_display 512 64  
./hd_display 4096 64  
./hd_display 10485760 64
```

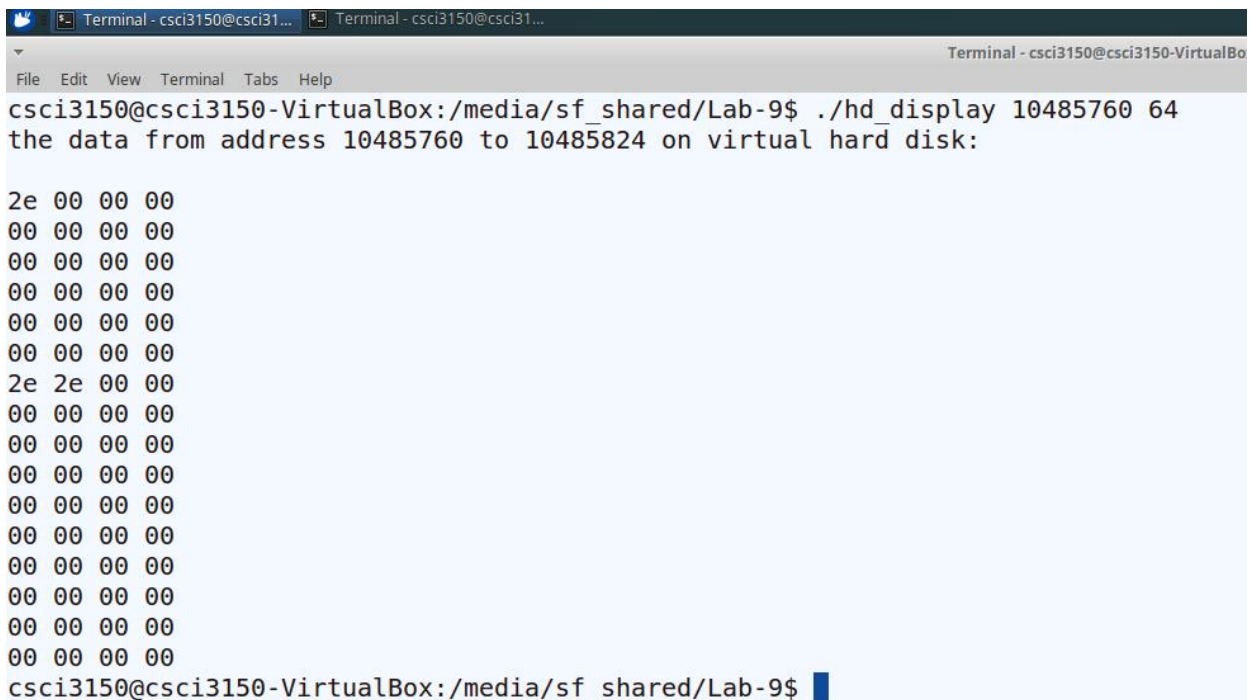
hd_display can display the content of HD starting from an offset and with the size provided. Some examples are shown below.



```
XUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal - csci3150@csci3150-VirtualBox: /media/sf_shared/Lab-9$ gcc -o hd_display hd_display.c -Wall
csci3150@csci3150-VirtualBox: /media/sf_shared/Lab-9$ ./hd_display 512 64
the data from address 512 to 576 on virtual hard disk:
00 10 00 00
00 00 a0 00
64 00 00 00
00 e8 03 00
01 00 00 00
01 00 00 00
00 10 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
csci3150@csci3150-VirtualBox: /media/sf_shared/Lab-9$
```

```
csci3150@csci3150-VirtualBox:/media/sf_shared/Lab-9$ ./hd_display 4096 64
the data from address 4096 to 4160 on virtual hard disk:
```

```
00 00 00 00
cc d3 f4 5b
00 00 00 00
30 00 00 00
01 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
02 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
csci3150@csci3150-VirtualBox:/media/sf_shared/Lab-9$
```



```
Terminal - csci3150@csci31... Terminal - csci3150@csci31...
File Edit View Terminal Tabs Help
csci3150@csci3150-VirtualBox:/media/sf_shared/Lab-9$ ./hd_display 10485760 64
the data from address 10485760 to 10485824 on virtual hard disk:

2e 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
2e 2e 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
csci3150@csci3150-VirtualBox:/media/sf_shared/Lab-9$
```

2.2 Implementation

In `mkfs_t.c`, there are mainly three functions:

- (1) **write_sb()**: write the parameters of SFS into the superblock region in HD;
- (2) **inode_rootdir()**: create inode 0 for root directory in the inode region in HD;
- (3) **init_rootdir()**: create the first two directory entries (for “.” and “..”) for the root directory in the data block region (data block 0).

Basically, in each function, we construct the corresponding data structure and write it back to the corresponding region in HD. Please read the code in `mkfs_c.c` for details.