

CSCE 313-200

Introduction to Computer Systems

Spring 2023

File System IV

Dmitri Loguinov

Texas A&M University

April 4, 2023

Chapter 12: Roadmap

12.1 Overview

12.2 File organization

12.3 Directories

12.4 Sharing

12.5 Record blocking

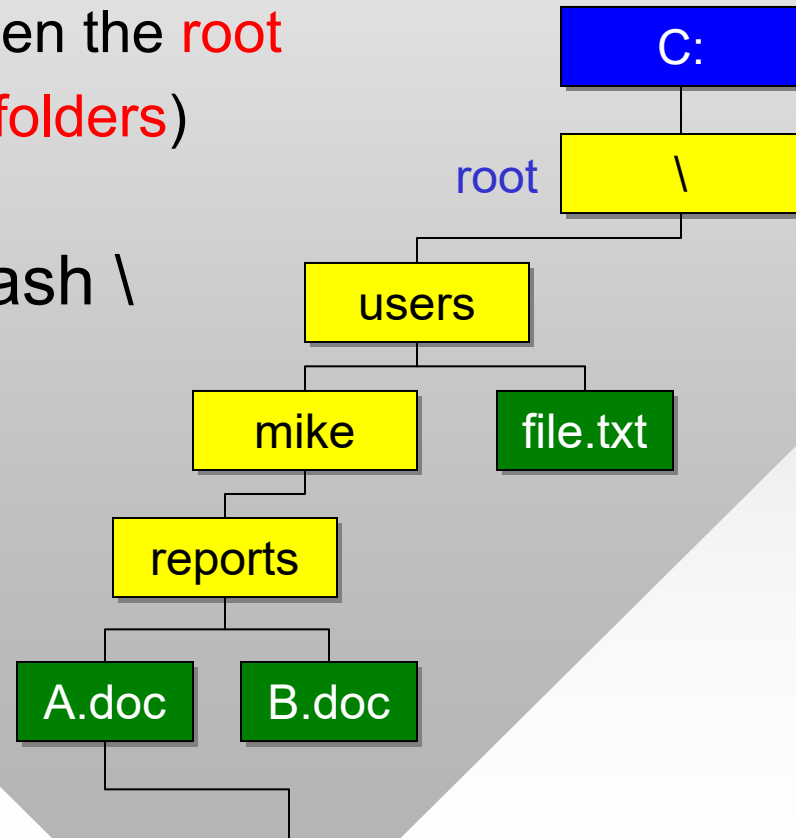
12.6 Secondary storage

12.7 File security

12.8-12.10 Unix, Linux, Windows

Directories

- Files on disk are organized into a hierarchical structure called **directory tree**
 - Starts from the optional drive, then the **root**
 - Non-leaf nodes are **directories (folders)** and leaves are usually files
- Windows: separator is backslash \
 - Unix: forward slash /
- Concatenation of directories from the root to the node is the **path** of that node
- Windows files/folders are not case-sensitive, Unix are



path = C:\users\mike\reports\A.doc

Directories

- Each process executes in some **current working directory** (CWD) where all files without a full path are read/written
 - fopen (“report.txt”, ...) executes in CWD
 - C-style _getcwd(), Windows API GetCurrentDirectory()
- How to find out where the exe file was started from?
 - Parse argv[0], which always contains the full path
- Manipulating directories
 - _chdir, _mkdir, _rmdir
 - SetCurrentDirectory, CreateDirectory, RemoveDirectory
- **Absolute** paths start from the root, **relative** from current directory

More APIs

- Directory may be watched for changes
 - Files modified, written, renamed, size changed, etc.
 - Including subdirectories
- Addition useful things:
 - CopyFile, MoveFile, DeleteFile, EncryptFile, FindFirstFile, GetFileSize, GetTempFileName
- **Flushing write buffers:** fflush, FlushFileBuffers
- Reading zip files: LZOpenFile / LZRead
- A list of file management APIs:
 - [http://msdn.microsoft.com/en-us/library/aa364232\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa364232(v=VS.85).aspx)

```
HANDLE WINAPI FindFirstChangeNotification (  
    __in  LPCTSTR lpPathName,  
    __in  BOOL bWatchSubtree,  
    __in  DWORD dwNotifyFilter );
```

Chapter 12: Roadmap

12.1 Overview

12.2 File organization

12.3 Directories

12.4 Sharing

12.5 Record blocking

12.6 Secondary storage

12.7 File security

12.8-12.10 Unix, Linux, Windows

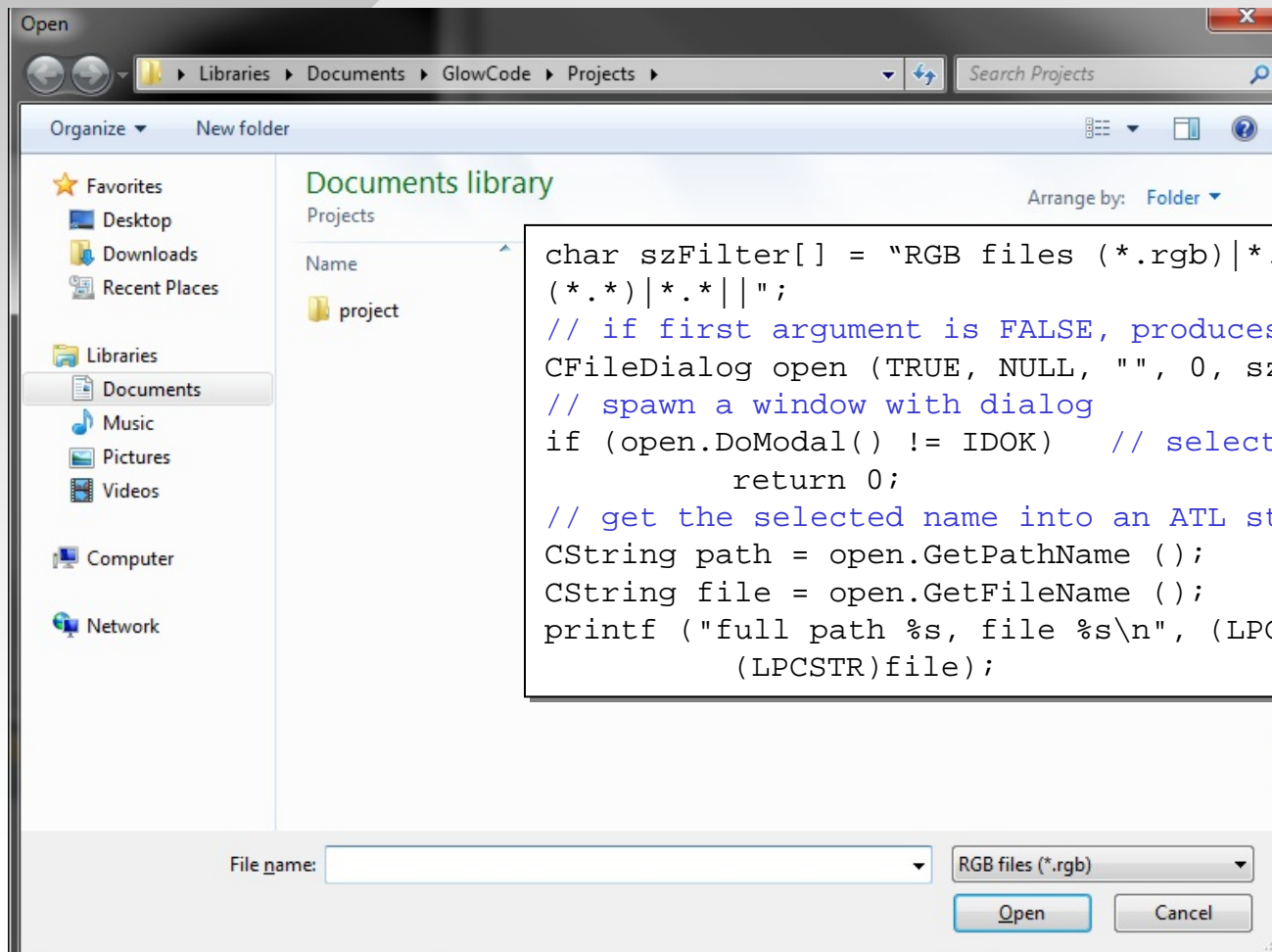
File Sharing

- Each file has one **owner** and a set of **permissions**
 - These specify what groups/users have what type of access to the file; real headache to manage from a Windows program
- **Sharing mode** determines concurrent access to file
 - In CreateFile, `FILE_SHARE_WRITE | FILE_SHARE_READ` allows to read files that are being written to by another process (assuming it also opens the file with the same parameters)
- **Attributes** are bit values of flags associated with file
 - E.g., archive, compressed, device, directory, encrypted, hidden, normal, read-only, system file
 - Can be manipulated using `GetFileAttributes`, `SetFileAttributes`
- If file needs to be locked temporarily, use `LockFile` instead of prohibiting sharing altogether

More on Files

```
#define _AFXDLL
#include <afxdlgs.h>
#include <Mmsystem.h>    // if using timeGetTime()
#include <Windows.h>     // cannot precede afxdlgs.h
// Project properties -> C/C++ -> Code Generation ->
// Runtime Library = Multi-threaded DLL (/MD)
```

- ATL/MFC dialog to prompt users to choose a file



Chapter 12: Roadmap

12.1 Overview

12.2 File organization

12.3 Directories

12.4 Sharing

12.5 Record blocking

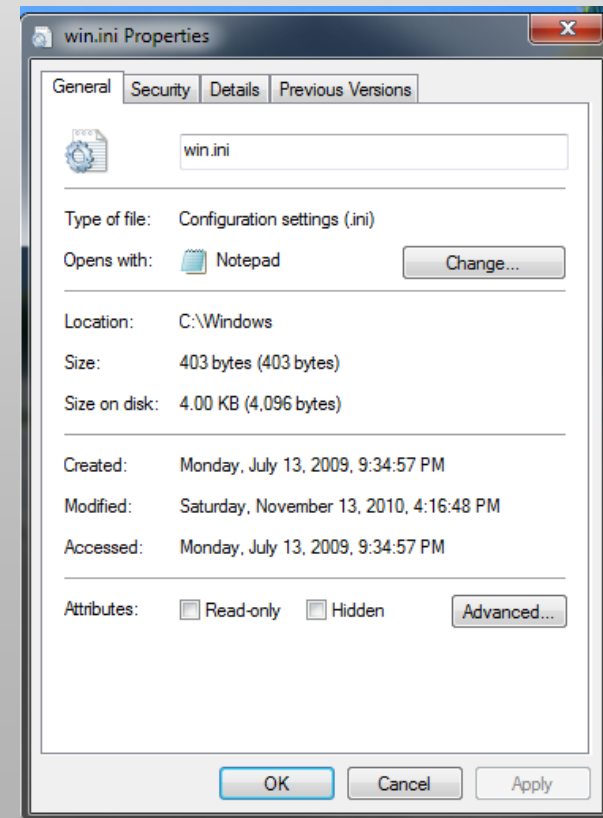
12.6 Secondary storage

12.7 File security

12.8-12.10 Unix, Linux, Windows

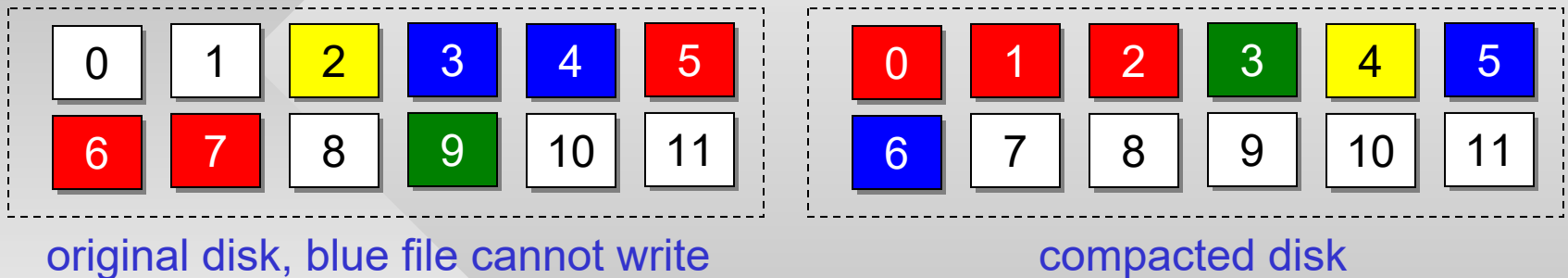
File Allocation

- Disk is split into groups of sectors called **clusters**
 - Cluster size is a multiple of sector size, usually up to 64 KB
- File space allocated in terms of clusters
 - **Internal fragmentation** refers to wasted space inside each cluster
- How to allocate clusters to files?
- **1) Pre-allocation**
 - Size declared ahead of time, cannot be expanded later
- **2) Dynamic allocation**
 - As more clusters are needed, they are provided by the OS



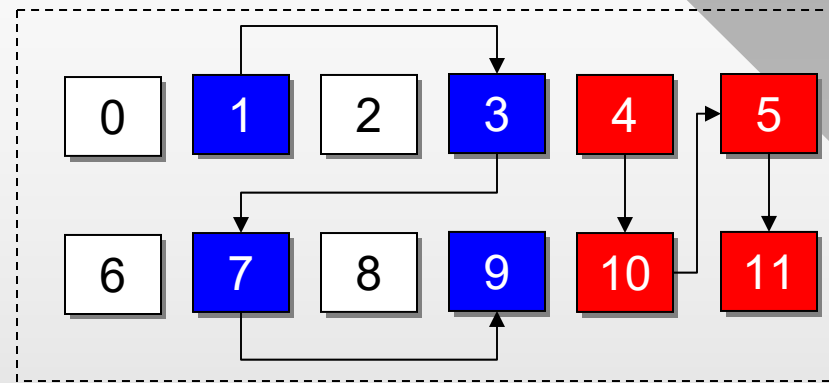
File Allocation

- 2.1) Contiguous allocation
 - Files are given only adjacent sets of blocks on disk



- First problem is **external fragmentation** (i.e., empty space between files not large enough for new files)
- Second problem when not enough contiguous space
 - Either request is denied, or the disk must undergo compaction
- Slow and inefficient, not used in practice

File Allocation

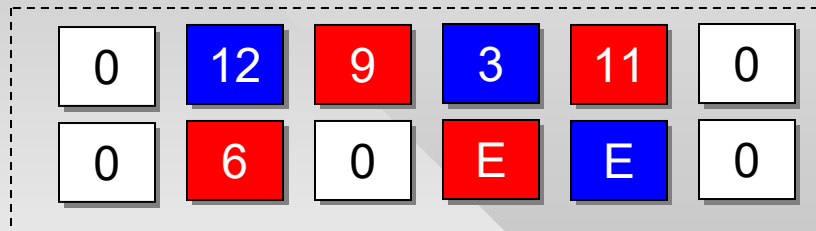


- 2.2) Chained allocation
 - Clusters in files are organized into linked lists by storing a pointer to next cluster
 - Must read the current cluster to find the next one
- **File fragmentation** into non-sequential disk blocks
 - Affects access speed as it may require extensive seeking
- Also, impossible to predict locality in cluster access
 - Difficult to realize that blocks 4-5 and 10-11 are sequential
 - Thus, the red file is read in 4 seeks instead of 2
- Not widely used in practice by itself

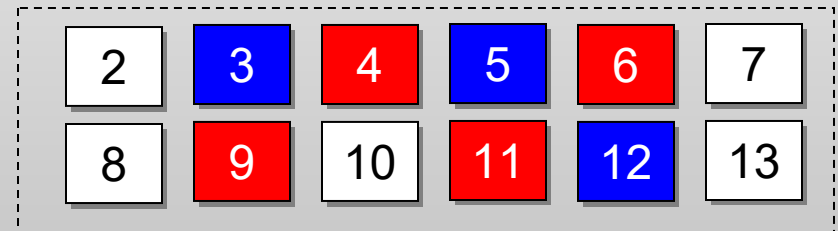
File Allocation

- 2.3) Indexed allocation

- Special File Allocation Table (FAT) specifies **next cluster**
- 0 = empty, E = EOC (end of cluster chain), 1 = reserved



FAT



disk

- Blue file stored in clusters 5→3→12, red file in 4→9→6→11
- Primary example MS-DOS FAT12 / FAT16 / FAT32
 - Each directory holds entries with files/subdirectories
 - These include the name, attributes, creation/modification time, size, and the **first cluster** in FAT

File Allocation

- Many systems are hybrids of 2.2-2.3
 - UFS (Unix File System) with multi-level Inode tables
 - NTFS with B+ trees, ext3 / ext4 in Linux with H-trees
- Size limits
 - FAT16 = 2 GB and FAT32 = 16 TB (MBR limit 2 TB)
 - Most other modern systems scale to enormous numbers (e.g., NTFS 2^{80} bytes, UFS 2^{73} , ext4 2^{60} , exFAT 2^{76} for flash)
- How to manage free space?
 - Full index for the state of each cluster (e.g., FAT)
 - Bitmaps of free/occupied clusters (e.g., NTFS in Windows, HPFS in OS/2, exFAT / FAT64 in Windows CE, ext4 in Linux)
 - Free blocks chained on disk (pretty slow)
 - Separate queue/stack of free block IDs stored on disk

More Terminology

- When FAT16/32 wasn't able to use the entire disk
 - It was split into **partitions**, each with own drive letter C:, D:, etc.
- Now partitions fall under a more general term **volume**
 - Volume combines one or more partitions grouped into a logical drive (e.g., RAID-0/1 or spanned)
 - To obtain cluster size, # of free clusters, and total volume size, use GetDiskFreeSpace
- **Links** allow files to be referenced under different paths
 - **Hard link** points to the first cluster of file (see CreateHardLink)
 - **Soft/symbolic link** stores a text path to the file, may exist without the target file (see CreateSymbolicLink)
- **Shortcuts** are special files understood only by Windows Explorer, unrelated to links