Filesystem

Filesystem Management

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Objectives

- Filesystem management in Windows includes
  - File and directory processing
    - DeleteFile, MoveFile (MoveFileEx), CreateDirectory, RemoveDirectory, SetCurrentDirectory, GetCurrentDirectory
  - File and directory visit
    - FindFirstFile, FindNextFile, FindClose

Delete a File

```
BOOL DeleteFile (
  LPCTSTR lpFileName
);
```

- Delete a file by specifying a file name
  - It does not work with an open file
  - It does not work with a directory
- Return value
  - TRUE, if and only if the delete operation succeeds
  - FALSE, otherwise

Rename or Move a File

```
BOOL MoveFile (
  LPCTSTR lpExisting,
  LPCTSTR lpNew
);
```

```
BOOL MoveFileEx (
  LPCTSTR lpExisting,
  LPCTSTR lpNew,
  DWORD dwFlags
);
```

...Ex = Extended Version
More powerful; not supported by earlier Windows versions

- Rename or move a file or a directory
- Return value
  - A non-zero value (TRUE), if success
  - A zero value (FALSE), if failure

Rename or Move a File

- Parameters
  - lpExisting
    - The name of the existing file or directory
  - lpNew
    - The name of the new file or directory
    - In general
      - Wildcards are not allowed in files/dirs names
      - Directory must be on the same drive
      - If lpNew is NULL the file/directory is deleted

```
BOOL MoveFileEx (
  LPCTSTR lpExisting,
  LPCTSTR lpNew,
  DWORD dwFlags
);
```

Rename or Move a File

- Use **MoveFileEx** to
  - Overwrite existing files
  - Move files on different drivers
    - Implemented on a copy and then a delete operation

MoveFile fails if
The new file already exists
Source and target are on different file system or drivers

```
BOOL MoveFileEx (
  LPCTSTR lpExisting,
  LPCTSTR lpNew,
  DWORD dwFlags
);
```

Rename or Move a File

- dwFlags

Value	Action / Meaning
MOVEFILE_REPLACE_EXISTING	Replace an existing file
MOVEFILE_WRITE_THROUGH	Do not return until the copied file is flushed through to the disk
MOVEFILE_COPY_ALLOWED	Need to be used to copy on a new volume. When copying to a different volume, copy then delete old file.
MOVEFILE_DELAY_UNTIL_REBOOT	Administration: copy when restart the OS

```
BOOL MoveFileEx (
  LPCTSTR lpExisting,
  LPCTSTR lpNew,
  DWORD dwFlags
);
```

Make directory

```
BOOL CreateDirectory (
  LPCTSTR lpPath,
  LPSECURITY_ATTRIBUTES lpsa
);
```

- Create a new (empty) directory
- Parameters
  - lpPath
    - Points to a null-terminated string with the directory name to be created
  - lpsa
    - Security attributes
    - Often equal to NULL

Make directory

```
BOOL RemoveDirectory (
  LPCTSTR lpPath
);
```

- Remove a directory
  - Directory must be empty
- Parameter
  - lpPath
    - Points to a null-terminated string with the directory name to be removed

Set Working Directory

```
BOOL SetCurrentDirectory (
  LPCTSTR lpCurDir
);
```

- Each **process** has a current or working directory
  - There is a current directory for each drive
  - Programs can set and get the current directory
- Warning
  - The current directory is **global** to a process and is shared by all threads in a process

Concurrent threads **cannot** set different directories.
They have to use absolute paths.

Set Working Directory

- Return value
  - A non-zero value (TRUE), if success
  - A zero value (FALSE), if failure
- Parameter
  - lpCurDir
    - The path to the new current directory
    - It can be a relative or an absolute path
    - Examples
      - SetCurrentDirectory (_T("C:"));
      - SetCurrentDirectory (_T("C:\user\tmp\"));

```
BOOL SetCurrentDirectory (
  LPCTSTR lpCurDir
);
```

Get Working Directory

```
DWORD GetCurrentDirectory (
  DWORD cchCurDir,
  LPTSTR lpCurDir
);
```

- It gets the **full** current pathname, and it returns it into the specified buffer
- Return
  - The string length of the returned pathname
  - The required buffer size if the buffer is not large enough
    - This includes the space for the null string terminator
  - Zero if the function fails

Get Working Directory

- Parameters
  - cchCurDir
    - Character length of the buffer for the directory name (cch denote "Count in characters")
  - lpCurDir
    - Points to the buffer to receive the full (absolute) pathname string

Windows uses this technique whenever the result's length is not known

Alert: Be sure the buffer is really as long as you say it is
Potential buffer overflow

```
DWORD GetCurrentDirectory (
  DWORD cchCurDir,
  LPTSTR lpCurDir
);
```

Example

Display and change the path of the working directory

```
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <io.h>
#define N 100
int _tmain (int argc, LPTSTR argv []) {
  TCHAR name[N];
  DWORD len;
  len = GetCurrentDirectory (N, name);
  if (len==0 || len>N)
    _tfprintf (_T("GetCurrentDir failed.\n“));
  else
    _tprintf (_T("GetCurrentDir: %s\n"), name);
```

Receives a directory name on the command line

Gets **old** current directory name and displays it

Example

Sets new current directory name

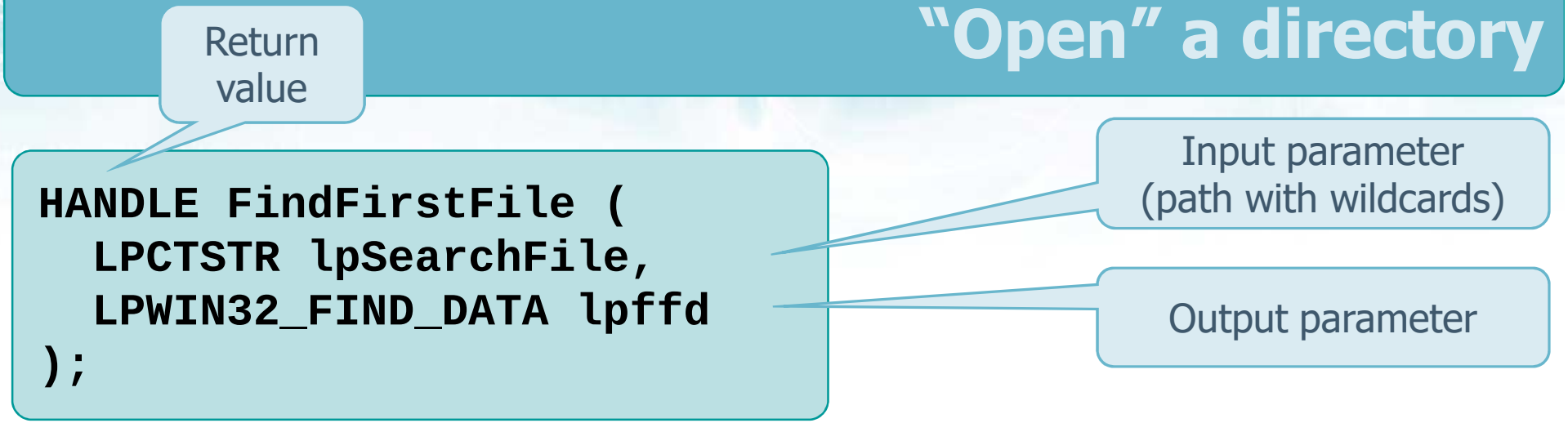
```
if (!SetCurrentDirectory (argv[1]))
   _tfprintf (_T("SetCurrentDir failed.\n"));
  else
   _tfprintf (_T("SetCurrentDir: Moved to %s\n“),
     argv[1]);

  len = GetCurrentDirectory (N, name);
  if (len==0 || len>N)
    _tfprintf (_T("GetCurrentDir failed.\n“));
  else
    _tfprintf (_T("GetCurrentDir: %s\n"), name);

  return (0);
}
```

Gets **new** current directory name and displays it

Scan a Directory

- To read a directory content it is possible to use a logic similar to the one adopted to read a file
- Following file operations, it is required to
  - Open the directory, i.e., generate a search handle satisfying specific requirements
    - Function **FindFirstFile**
  - Read the directory content one entry at a time, untill all entries have been read
    - Function **FindNextFile**
  - End the reading operation, i.e., close the directory
    - Function **FindClose**

"Open" a directory



- A file search requires a **search handle**
- The function **FindFirstFile**
  - Examines all entries of one directory and subdirectories looking for a name match with **lpSearchFile**
  - After that, it
    - Return the pointer to a **structure** describing the first object satisfying **lpSearchFile** and
    - Returns a search **handle**

"Open" a directory

- Parameter values
  - lpSearchFile
    - Points to a directory or a pathname
    - Wildcards can be used ('*' and '?')
  - lpffd
    - Points to a **WIN32_FIND_DATA** structure
    - The structure contains information on the first entry satisfying **lpSearchFile**

Search for a specific file (e.g, " name.ext") or a set (e.g., "name.*")

Use "32" even on a 64-bit system

```
HANDLE FindFirstFile (
  LPCTSTR lpSearchFile,
  LPWIN32_FIND_DATA lpffd
);
```

"Open" a directory

The _WIN32_FIND_DATA Structure

```
typedef struct _WIN32_FIND_DATA {
  DWORD dwFileAttributes;
  FILETIME ftCreationTime;
  FILETIME ftLastAccessTime;
  FILETIME ftLastWriteTime;
  DWORD nFileSizeHigh;
  DWORD nFileSizeLow;
  DWORD dwReserved0;
  DWORD dwReserved1;
  TCHAR cFileName[MAX_PATH];
  TCHAR cAlternateFileName[14];
} WIN32_FIND_DATA;
```

See Createfile attributes

64-bit integers

Reserved for future use

File name

MS-DOS (8+3)-bits file name

"Open" a directory

- Return value
  - In case of success, a "search handle"
    - The handle can be used to obtain further information on the next entry satisfying **lpSearchFile** in **lpSearchFile**
      - It is used in all subsequent operations of that **lpSearchFile**
  - In case of failure
    - The constant value INVALID_HANDLE_VALUE

See **FindFirstFileEx** for more options (e.g., case sensitivity)

```
HANDLE FindFirstFile (
  LPCTSTR lpSearchFile,
  LPWIN32_FIND_DATA lpffd
);
```

File Searching

```
BOOL FindNextFile (
  HANDLE hFindFile,
  LPWIN32_FIND_DATA lpffd
);
```

Obtained with **FindFirstFile**

- Once the handle (**hFindFile**) given by FindFirstFile is available, FindNextFile may obtain the data info for the subsequent entry
  - This information is stored into a new WIN32_FIND_DATA object (referenced by **lpffd**)
- Return value
  - TRUE, when the search can go on
  - FALSE, when no more files satisfy the search pattern

File Searching

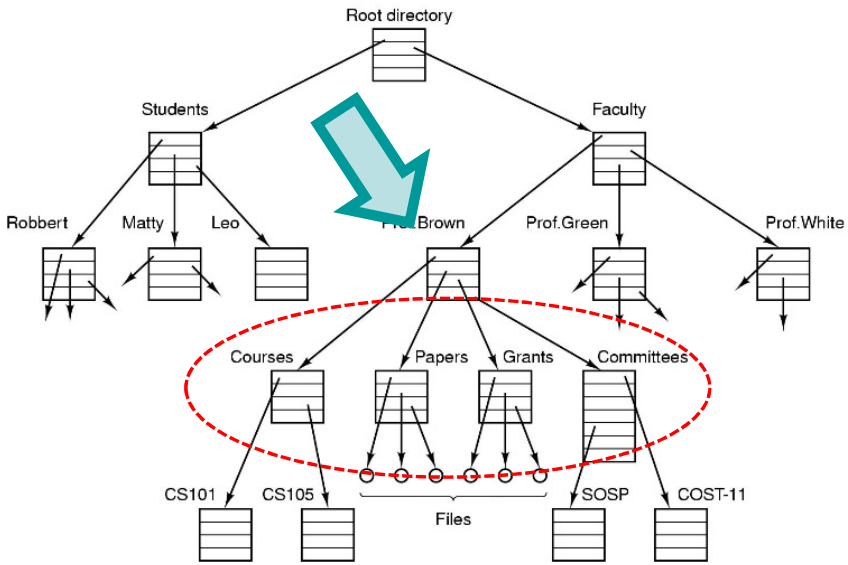
```
BOOL FindClose (
  HANDLE hFindFile
);
```

- When the search is complete **FindClose** closes the search handle
- Note the exception
  - The directory HANDLE (albeit being a HANDLE object) is **not** closed with CloseHandle

Example: Visit (flat)

Visit (read the content) of a (flat) directory

- Write a Win32/64 application which is able to
  - Receive a string as a parameter
    - The string indicates a **relative** (or an **absolute**) path to a file system directory tree
  - Visit the entire directory content
    - Do not recur into sub-directories
    - Display entry names and specify for each entry if it is a file or a directory



Example: Visit (flat)

```
#define UNICODE
#define _UNICODE
#define _CRT_SECURE_NO_WARNINGS

#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <io.h>

#define TYPE_FILE 1
#define TYPE_DIR  2
#define TYPE_DOT  3

static void TraverseDirectory (LPTSTR);
DWORD FileType (LPWIN32_FIND_DATA);
```

Example: Visit (flat)

```
int _tmain (int argc, LPTSTR argv []) {

  TraverseDirectory (argv[1]);

  return 0;
}

static void TraverseDirectory (LPTSTR PathName) {
  HANDLE SearchHandle;
  WIN32_FIND_DATA FindData;
  DWORD FType;
  TCHAR currPath[MAX_PATH+1];
```

argv[1] is the directory path

Visits and prints-out the directory content

Example: Visit (flat)

```
  SetCurrentDirectory (PathName);
  SearchHandle = FindFirstFile (_T("*"), &FindData);
  do {
    FType = FileType (&FindData);
    if (FType == TYPE_FILE)
      _tprintf(_T("FILE: %s\n"),FindData.cFileName);
    if (FType == TYPE_DIR) {
      _tprintf(_T("DIR : %s\n"),FindData.cFileName);
    }
  } while (FindNextFile (SearchHandle, &FindData));
  FindClose (SearchHandle);

  return;
}
```

Set desired directory as current one

We want to print-out **all** entries

Printing info

Go-on until there is a new entry

Example: Visit (flat)

Use **FileType** to discover file type

```
static DWORD FileType(LPWIN32_FIND_DATA pFileData) {
  BOOL IsDir;
  DWORD FType;
  FType = TYPE_FILE;
  IsDir = (pFileData->dwFileAttributes &
    FILE_ATTRIBUTE_DIRECTORY) != 0;
  if (IsDir)
    if (lstrcmp(pFileData->cFileName,_T("."))==0
      || lstrcmp(pFileData->cFileName,_T(".."))==0)
      FType = TYPE_DOT;
    else FType = TYPE_DIR;
  return FType;
}
```

IsDir has to be TRUE for directories

Pay attention to "corner" cases

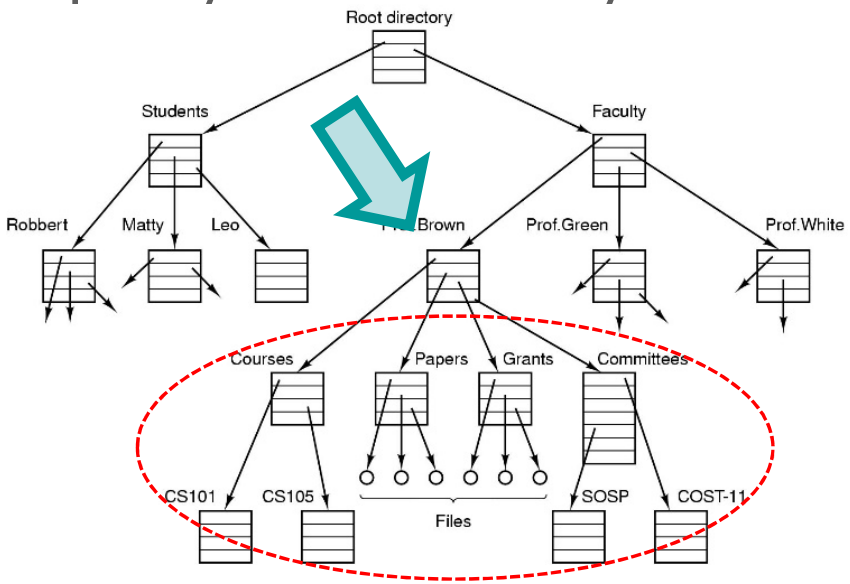
Example: Visit (recursive)

Visit (read the content) of a directory tree

- Write a Win32/64 application which is able to
  - Receive a string as a parameter
    - The string indicates a **relative** (or an **absolute**) path to a file system directory tree
  - Visit the entire directory content
    - **Recur** into sub-directories
    - Print-out entry names and specify for each entry if it is a file or a directory

Extend TraverseDirectory to recur on sub-directories

Recur on dirs iff they are **not** "." or ".."
Pay attention to **relative** vs **absolute** paths



Example: Visit (recursive)

Program abstract (difference only)

```
static void
TraverseDirectoryRecursive (
  LPTSTR PathName,
  DWORD level
  )
{
  HANDLE SearchHandle;
  WIN32_FIND_DATA FindData;
  DWORD FType, i;

  SetCurrentDirectory (PathName);
  SearchHandle = FindFirstFile (_T("*"), &FindData);
```

New parameter

Note: Single thread program
Current directory can be set for the entire process

A relative (or an absolute path)

Example: Visit (recursive)

```
do {
  FType = FileType (&FindData);
  if (FType == TYPE_FILE) {
    for (i=0; i<level; i++)
      _tprintf (_T ("  "));
    _tprintf (_T ("level=%d FILE: %s\n"), level,
      FindData.cFileName);
  }
  if (FType == TYPE_DIR) {
    for (i=0; i<level; i++)
      _tprintf (_T ("  "));
    _tprintf (_T ("level=%d DIR : %s\n"), level,
      FindData.cFileName);
```

Printing file names with indentation

Printìng dir names with indentation

Do **not** recur on TYPE_DOT entries

Example: Visit (recursive)

Run recursion
(entering new sub-directory)

To use absolute paths, concatenate
current one with cFileName

```
      TraverseDirectoryRecursive (FindData.cFileName,
        level+1);
```

Backtrack: Move up one level on the directory tree
Not required when using absolute paths to set current directories

```
      SetCurrentDirectory (_T (".."));
    }
  } while (FindNextFile (SearchHandle, &FindData));

  FindClose (SearchHandle);

  return;
}
```

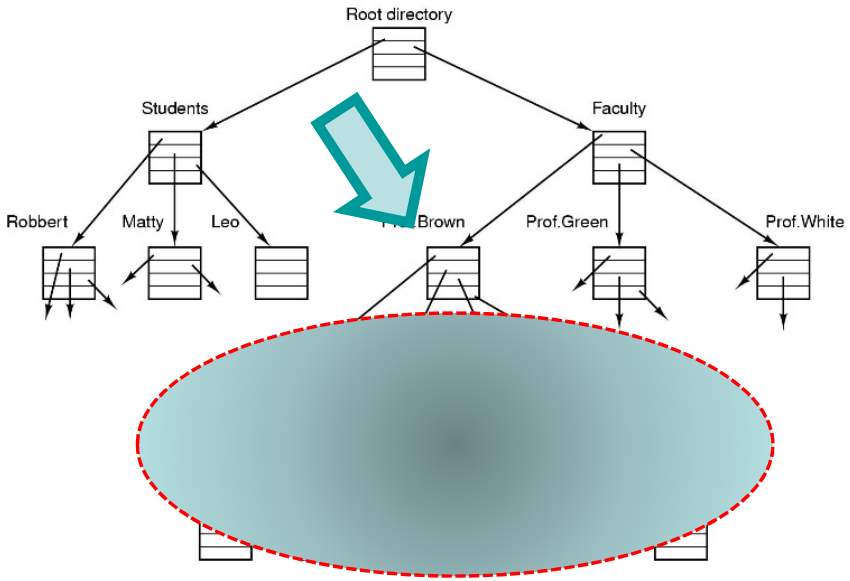
Example: Delete

Delete a directory tree

- Write a Win32/64 application which is able to
  - Receive a string as a parameter
    - The string indicates a relative path to a file system directory tree
  - Delete the entire directory content (recurring into sub-directories)

Extend TraverseDirectoryRecursive to delete files and dirs

Delete files on the way down
Delete directories on the way back
(dirs must be empty)



Example: Delete

Program abstract (difference only)

Same considerations for relative and absolute paths

```
int _tmain (int argc, LPTSTR argv []) {
  . . .
  DeleteDirectoryRecursive (argv[1], 1);
  SetCurrentDirectory (_T(".."));

  if (RemoveDirectory (argv[1])) {
    _tprintf (_T ("level=0 DEL DIR: %s\n"),
      argv[1]);
  } else {
    _tprintf (_T ("level=0 DEL DIR (%s) FAILED!\n"),
      argv[1]);
  }
  Sleep (5000);
  return 0;
}
```

When back …
don’t forget to delete main directory

Example: Delete

```
static void DeleteDirectoryRecursive (
  LPTSTR PathName, DWORD level) {
  HANDLE SearchHandle;
  WIN32_FIND_DATA FindData;
  DWORD FType, i;
  SetCurrentDirectory (PathName);
  SearchHandle = FindFirstFile (_T("*"), &FindData);
  do {
    FType = FileType (&FindData);
    if (FType == TYPE_FILE) {
      if (DeleteFile (FindData.cFileName)) {
          _tprintf (_T ("l=%d DelFile=%s\n"),
            level, FindData.cFileName);
       } else {
          _tprintf (_T ("l=%d DelFile=%s FAILED!\n"),
            level, FindData.cFileName);
       }
    }
```

Delete instead of printing

Example: Delete

```
    if (FType == TYPE_DIR) {
      DeleteDirectoryRecursive (FindData.cFileName,
       level+1);
      SetCurrentDirectory (_T(".."));
      for (i=0; i<level; i++)
        _tprintf (_T ("  "));
        if (RemoveDirectory (FindData.cFileName)) {
          _tprintf (_T ("l=%d DelDir: %s\n"),
            level, FindData.cFileName);
        } else
          _tprintf (_T ("l=%d DelDir (%s) FAILED!\n"),
           level, FindData.cFileName);
        }
    }
  } while (FindNextFile (SearchHandle, &FindData));
  FindClose (SearchHandle);
  return;
}
```

Recur First ...

... move back ...

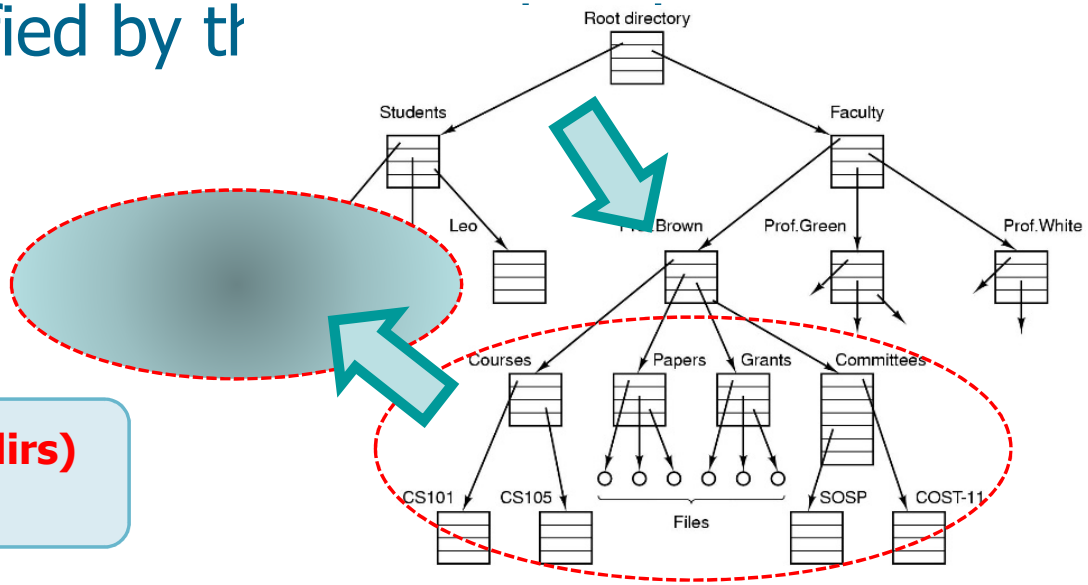
... then delete

Example: Copy

Copy a directory tree

- Write a Win32/64 application which is able to
  - Receive two strings parameters
    - The first string indicates a **relative** path to an existing file system directory tree
    - The second string indicates a **relative** path to an empty file system directory tree
  - Copy the directory tree specified in the first path into the one specified by th

Extend TraverseDirectoryRecursive to copy files and create dirs



Need to have two paths (one for each dirs)
Pay attention not to get lost into dirs

Example: Copy

Program abstract (difference only)

```
#define L 500

int _tmain (int argc, LPTSTR argv []) {
  TCHAR tmpPath[L], DestPathName[L];

  GetCurrentDirectory (L, tmpPath);
  _stprintf (DestPathName, _T("%s\\%s"),
     tmpPath, argv[2]);
  TraverseAndCreate (argv[1], DestPathName);

  Sleep (5000);

  return 0;
}
```

Source path → argv[1]
Destination path → argv[2]

Build absolute path for destination (working +relative)

Source path, i.e., relative path

Destination path, i.e., absolute path

Example: Copy

```
static void TraverseAndCreate (
  LPTSTR SourcePathName, LPTSTR DestPathName
  )
{
  HANDLE SearchHandle;
  WIN32_FIND_DATA FindData;
  DWORD FType, l;
  TCHAR NewPath[L];

  _tprintf (
     _T ("--> Create Dir : %s\n"),
     DestPathName);

  CreateDirectory (DestPathName, NULL);
  SetCurrentDirectory (SourcePathName);

  SearchHandle = FindFirstFile (_T("*"), &FindData);
```

Create new directory (using absolute path)

lpsa = NULL

Set current directory, i.e., move onto new source directory

Example: Copy

```
do {
 FType = FileType (&FindData);

 l = _tcslen(DestPathName);
 if (DestPathName[l-1] == '\\') {
   _stprintf (NewPath, _T("%s%s"),
     DestPathName, FindData.cFileName);
 } else {
   _stprintf (NewPath, _T("%s\\%s"),
     DestPathName, FindData.cFileName);
 }
```

Create new absolute destination path (string concatenation)

Do no use '\' as a separator more than once

Use '\' as a proper separator

Example: Copy

```
    if (FType == TYPE_FILE) {
      CopyFile (FindData.cFileName, NewPath, FALSE);
    }

    if (FType == TYPE_DIR) {
      TraverseAndCreate (FindData.cFileName,
        NewPath);
      SetCurrentDirectory (_T (".."));
    }

  } while (FindNextFile (SearchHandle, &FindData));

  FindClose (SearchHandle);

  return;
}
```

Copy file

Recur

Move back