OPERATING SYSTEM LAB PROJECT

Certainly! This code is a simple file management system implemented in C++ for a Windows environment. It provides a graphical user interface (GUI) to create, delete, read, write, and view the content of files within a simulated file system. Let's break it down step by step:

Includes and Namespace

```
cpp
Copy code
#include <iostream>
#include <vector>
#include <map>
#include <string>
#include <windows.h>

using namespace std;
```

- **Includes**: The code includes standard C++ libraries for input-output (`iostream`), dynamic arrays (`vector`), associative arrays (`map`), and string manipulations (`string`). It also includes `windows.h` for Windows API functions.
- **Namespace**: It uses the `std` namespace to avoid prefixing standard library components with `std::`.

Constants

```
cpp
Copy code
const int BLOCK_SIZE = 1024;
const int NUM_BLOCKS = 1000;
```

- **BLOCK_SIZE**: Defines the size of a block (1024 bytes).
- **NUM_BLOCKS**: Defines the number of blocks in the simulated file system (1000 blocks).

Structures

```
cpp
Copy code
struct File {
    string name;
```

```
    int size;
    vector<int> blocks;
};

struct Directory {
    map<string, File> files;
};
```

- **File**: Represents a file with a name, size, and a list of block indices where the file's data is stored.
- **Directory**: Represents a directory containing files, mapping file names to `File` structures.

FileSystem Class

```
cpp
Copy code
class FileSystem {
private:
    vector<bool> allocated_blocks;
    Directory root_directory;

public:
    FileSystem() {
        allocated_blocks.resize(NUM_BLOCKS, false);
    }
```

- **allocated_blocks**: A vector of booleans indicating whether each block is allocated (true) or free (false).
- **root_directory**: A `Directory` object representing the root directory of the file system.

Constructor

```
cpp
Copy code
FileSystem() {
    allocated_blocks.resize(NUM_BLOCKS, false);
}
```

- Initializes the `allocated_blocks` vector with `NUM_BLOCKS` elements, all set to `false` (indicating all blocks are free).

File Management Methods

```
cpp
Copy code
void createFile(const string& filename, int size) {
    // Code for creating a file
}

void deleteFile(const string& filename) {
    // Code for deleting a file
```

```
}

void readFile(const string& filename) {
    // Code for reading a file
}

void writeFile(const string& filename, const string& data) {
    // Code for writing to a file
}

void viewFileContent(const string& filename) {
    // Code for viewing file content
}
```

- **createFile**: Creates a new file with the specified name and size.
- **deleteFile**: Deletes a file with the specified name.
- **readFile**: Simulates reading a file by displaying which blocks would be read.
- **writeFile**: Simulates writing data to a file by displaying which blocks would be written to.
- **viewFileContent**: Simulates viewing the content of a file by displaying the blocks and placeholder content.

Private Helper Method

```
cpp
Copy code
int findFreeBlock() {
    for (int i = 0; i < NUM_BLOCKS; ++i) {
        if (!allocated_blocks[i]) {
            return i;
        }
    }
    return -1;
}
```

- **findFreeBlock**: Searches for a free block and returns its index. If no free blocks are found, it returns -1.

Window Procedure

```
cpp
Copy code
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam) {
    // Code to handle window messages
}
```

- **WndProc**: The window procedure handles messages sent to the window, such as creating controls, button clicks, and window destruction.

Entry Point

```
cpp
Copy code
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow) {
    // Code to create and show the window
}
```

- **WinMain**: The entry point for a Windows application. It registers the window class, creates the window, and enters the message loop.

Detailed Window Procedure Implementation

```
cpp
Copy code
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam) {
    static FileSystem fs;
    static HWND hFileName, hFileSize, hFileData;
    switch (message) {
    case WM_CREATE: {
        // Code to create controls (Static text, Edit boxes, and Buttons)
        break;
    }
    case WM_COMMAND: {
        int wmId = LOWORD(wParam);
        WCHAR fileName[256];
        GetWindowTextW(hFileName, fileName, 256);
        string filename(fileName, fileName + wcslen(fileName));

        switch (wmId) {
        case 1: {
            // Code for creating a file
            break;
        }
        case 2: {
            // Code for deleting a file
            break;
        }
        case 3: {
            // Code for reading a file
            break;
        }
        case 4: {
            // Code for writing to a file
            break;
        }
        case 5: {
            // Code for viewing file content
            break;
        }
        }
        break;
    }
    case WM_DESTROY: {
        PostQuitMessage(0);
```

```
        break;
    }
    default:
        return DefWindowProcW(hwnd, message, wParam, lParam);
    }
    return 0;
}
```

- **WM_CREATE**: Handles the creation of window controls.
- **WM_COMMAND**: Handles button click events to perform file operations based on the `wmId`.
- **WM_DESTROY**: Posts a quit message to exit the application.
- **Default**: Default message handling.

Registering and Creating the Window

```
cpp
Copy code
WNDCLASSW wc = {0};
wc.lpszClassName = L"MyWindowClass";
wc.lpfnWndProc = WndProc;
wc.hInstance = hInstance;
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
RegisterClassW(&wc);

HWND hwnd = CreateWindowW(L"MyWindowClass", L"File Management System",
WS_OVERLAPPEDWINDOW | WS_VISIBLE, 100, 100, 400, 300, NULL, NULL, hInstance,
NULL);

MSG msg;
while (GetMessageW(&msg, NULL, 0, 0)) {
    TranslateMessage(&msg);
    DispatchMessageW(&msg);
}
```

- **WNDCLASSW**: Defines the window class.
- **RegisterClassW**: Registers the window class.
- **CreateWindowW**: Creates the main window.
- **Message Loop**: Processes messages for the main window until `WM_QUIT` is received.

This code provides a basic simulation of a file management system using Windows GUI components. Each operation is handled through message processing in `WndProc`, and results are displayed using message boxes.