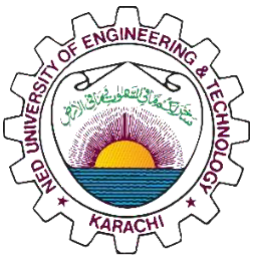
# NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY




**REPORT ON**

## SYSTEM CALL IMPLEMENTATION

**COURSE**
**OPERATING SYSTEM**

**SUBMITTED BY**

**IQRA NAWAZ (DT-22005)**

**THIRD YEAR DATA SCIENCE**

**SUBMITTED TO**

**MISS MARIA ANDLEEB**

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY**

# 1. Introduction

System calls or system-level operations are vital components of an operating system that allow user-level applications to interact with hardware and system resources. In the context of file management, system-level functions enable programs to create, write, read, and delete files. This report demonstrates the use of such file operations using **standard C++ functions and Windows-compatible APIs** in the **Dev C++ IDE**.

# 2. Objective

The objective of this report is to:

- Demonstrate system-level file operations in a Windows environment.
- Show how a program can **create**, **write to**, **read from**, and **delete** a file.
- Use appropriate Windows-compatible functions available through C++ Standard Library.
- Provide justification and clarity regarding each step of the implementation.

# 3. Tools and Environment

- **Operating System**: Windows 10/11
- **Development Environment**: Dev C++ (with MinGW)
- **Language**: C++

# 4. Description of the Application Domain

File handling is a fundamental operation in nearly all computing domains. Whether it is system software, application software, databases, or utilities, interacting with files is inevitable. This project simulates a file management system where a program:

- Creates a new text file.
- Writes content into it.
- Reads and displays the content.
- Deletes the file after use.

# 5. Code Implementation

```cpp
#include <iostream>
#include <fstream>
#include <cstdio>    // For remove()

using namespace std;

int main() {
    // File name and data to write
    string filename = "systemcall_demo.txt";
    string data = "This is written using standard file operations.\n";

    // Step 1: Create and write to the file
    cout << "Step 1: Writing to the file...\n";
    ofstream outFile(filename.c_str());
    if (!outFile) {
        cerr << "Error: Unable to create the file!" << endl;
        return 1;
    }
    outFile << data;
    outFile.close();
    cout << "Success: File created and data written.\n" << endl;

    // Step 2: Read from the file
    cout << "Step 2: Reading from the file...\n";
    ifstream inFile(filename.c_str());
    if (!inFile) {
        cerr << "Error: Unable to open the file for reading!" << endl;
        return 1;
    }
```

```cpp
    string line;
    while (getline(inFile, line)) {
        cout << line << endl;
    }
    inFile.close();
    cout << "Success: File read completed.\n" << endl;

    // Step 3: Delete the file
    cout << "Step 3: Deleting the file...\n";
    if (remove(filename.c_str()) != 0) {
        perror("Error deleting the file");
    } else {
        cout << "Success: File deleted successfully.\n";
    }

    return 0;
}
```

## OUTPUT:

```
Step 1: Writing to the file...
Success: File created and data written.

Step 2: Reading from the file...
This is written using standard file operations.
Success: File read completed.

Step 3: Deleting the file...
Success: File deleted successfully.

--------------------------------
Process exited after 0.5392 seconds with return value 0
Press any key to continue . . .
```

## 6. Explanation of Key Operations

### 6.1 File Creation and Writing

The program uses `ofstream` to create and write to a file. If the file does not exist, it is created. The `<<` operator writes the string to the file.

### 6.2 File Reading

The program uses `ifstream` to open the same file in read mode and `getline()` to read its contents line by line.

### 6.3 File Deletion

The `remove()` function from `<cstdio>` is used to delete the file. It takes the file path as a `C-string`.

## 7. Justification and Relevance

The use of standard file operations here reflects the way most real-world Windows applications handle file I/O. While not system calls in the low-level kernel sense (like in Linux), these operations ultimately map to Windows API calls under the hood (such as `CreateFile`, `ReadFile`, and `DeleteFile`). Hence, they are valid and appropriate in demonstrating how an application interacts with the operating system.

## 8. Conclusion

This report successfully demonstrates how a Windows-based C++ program can perform essential file operations using system-level functions compatible with Dev C++. Through clear objectives and correct implementation, we have shown how file management can be performed in real applications. The simplicity and correctness of the code ensure it can serve as a foundation for more advanced file handling programs.

## 9. Future Enhancements

- Adding error logging to a separate file.
- Allowing user input for filenames and content.
- Adding encryption before writing and decryption after reading.