
Deep Study Guide: Real-Time File Blocker in C++ (Full Mastery)

---

1 C++ STL: unordered_set

Purpose: Store blocked file hashes for **fast lookup**.

Key Concepts:

- **Hash table** structure → O(1) average lookup
- insert(), contains(), erase()
- Differences from std::set:
  - set → ordered, tree-based, O(log n) lookup
  - unordered_set → unordered, hash-based, O(1) average

Mini-example:

```
#include <unordered_set>
#include <string>
#include <iostream>

int main() {
   std::unordered_set<std::string> blocklist;
   blocklist.insert("abc123");
   if (blocklist.contains("abc123"))
      std::cout << "Blocked!\n";
}
```

References:

- C++ reference – unordered_set

---

2 C++ STL: std::thread & detach()

Purpose: Run the **periodic scan in the background** without blocking the main thread.

Key Concepts:

- std::thread → launches a new thread
- Lambda capture [folderPath, blocklist] → safely pass variables
- detach() → thread runs independently

- join() → main thread waits for completion

Mini-example:

```
#include <thread>
#include <chrono>
#include <iostream>

int main() {
   std::thread([](){
      while(true) {
         std::cout << "Background scan...\n";
         std::this_thread::sleep_for(std::chrono::seconds(2));
      }
   }).detach();

   std::cout << "Main thread continues...\n";
   std::this_thread::sleep_for(std::chrono::seconds(10));
}
```

References:

- C++ reference – thread
- Study: race conditions, thread safety

---

3 C++17 Filesystem: std::filesystem

Purpose: Access files and directories easily.

Key Classes & Functions:

- fs::path → represents file/folder path
- fs::directory_iterator → loop over files in a folder
- fs::recursive_directory_iterator → loop over all files/subfolders
- .string() → standard string
- .wstring() → Windows wide string (for APIs)
- fs::is_regular_file() → check if path is a normal file

Mini-example:

```
#include <filesystem>
#include <iostream>
namespace fs = std::filesystem;

int main() {
   fs::path folder = R"(C:\files)";
   for (const auto& entry : fs::directory_iterator(folder)) {
      if (fs::is_regular_file(entry))
         std::cout << entry.path().filename() << "\n";
   }
}
```

References:

- C++ reference – filesystem

---

4 Windows API: Real-Time Monitoring

Purpose: Detect new/modified files **immediately**.

Key Functions:

CreateFile()

- Opens a handle to a directory or file
- Parameters:

```
HANDLE hDir = CreateFile(
   folderPath.wstring().c_str(),
   FILE_LIST_DIRECTORY,
   FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE,
   nullptr,
   OPEN_EXISTING,
   FILE_FLAG_BACKUP_SEMANTICS | FILE_FLAG_OVERLAPPED,
   nullptr
);
```

Explanation of Flags:

- FILE_LIST_DIRECTORY → directory handle for monitoring

- FILE_SHARE_* → allows other processes to access files while monitoring
- OPEN_EXISTING → open only existing directory
- FILE_FLAG_BACKUP_SEMANTICS → required for directories
- FILE_FLAG_OVERLAPPED → asynchronous I/O

ReadDirectoryChangesW()

- Monitors **directory changes**: create, delete, modify, rename
- Uses HANDLE hDir from CreateFile
- DWORD bytesReturned → output bytes
- Must call CloseHandle(hDir) when done

References:

- ReadDirectoryChangesW documentation

HANDLE and DWORD

- HANDLE → generic Windows object handle (file, thread, etc.)
- DWORD → 32-bit unsigned integer used in WinAPI
- CloseHandle() → close handle safely

---

5 SHA-256 Hashing (OpenSSL)

Purpose: Identify files by **content**, not name.

Steps in Your Code:

1. Open file in binary mode
2. Read chunks → update hash context
3. Finalize → get SHA-256 digest
4. Convert bytes → hex string

Mini-example:

```
#include <openssl/evp.h>

#include <fstream>

#include <sstream>

#include <iomanip>
```

```
std::string sha256_file(const std::filesystem::path& filePath) {
    std::ifstream file(filePath, std::ios::binary);
    EVP_MD_CTX* ctx = EVP_MD_CTX_new();
    const EVP_MD* md = EVP_sha256();
    EVP_DigestInit_ex(ctx, md, nullptr);

    char buffer[4096];
    while (file.read(buffer, sizeof(buffer)) || file.gcount() > 0)
        EVP_DigestUpdate(ctx, buffer, file.gcount());

    unsigned char hash[EVP_MAX_MD_SIZE];
    unsigned int hashLen;
    EVP_DigestFinal_ex(ctx, hash, &hashLen);
    EVP_MD_CTX_free(ctx);

    std::ostringstream oss;
    for (unsigned int i = 0; i < hashLen; i++)
        oss << std::hex << std::setw(2) << std::setfill('0') << (int)hash[i];
    return oss.str();
}
```

References:

- OpenSSL EVP documentation

---

6 Putting It Together

Your project combines:

- **C++ STL** → hash tables, threads
- **C++17 filesystem** → file/folder access
- **OpenSSL** → SHA-256 hashes
- **Windows API** → real-time monitoring
- **Multithreading** → background scanning

Learning Strategy:

1. Write **small test programs** for each concept separately.
2. Print everything: hashes, file names, events, handles
3. Combine gradually: initial scan → periodic scan → real-time monitor → hash checking → deletion/quarantine

---

✅ **Extra Resources**

1. C++ STL & Threads:
    - C++ Reference
    - "The C++ Standard Library" by Nicolai Josuttis
2. C++17 Filesystem:
    - C++ Reference – filesystem
3. Windows API:
    - MS Docs – ReadDirectoryChangesW
    - MS Docs – CreateFile
4. OpenSSL / Hashing:
    - OpenSSL EVP
    - Practical OpenSSL examples

If you want, I can also **write a fully annotated version of your project code** with **line-by-line explanations**, so this becomes a **complete master document** where every function, thread, flag, and Windows API call is explained in detail.

Do you want me to do that next?