Manipulação de arquivos e diretórios

Dr. Rodrigo Mologni Gonçalves dos Santos

Conteúdo

Apresentação da biblioteca

A biblioteca *Filesystem*

- Fornece recursos para **executar operações em sistemas de arquivos e seus componentes**, como caminhos, arquivos e diretórios.
- Está disponível a partir do **C++17**.
- Compatibilidade com Windows* e Unix.

Criação de arquivos e diretórios

Criar diretório

```
bool create_directory( const std::filesystem::path& p );

bool create_directory( const std::filesystem::path& p,
                       std::error_code& ec ) noexcept;

bool create_directory( const std::filesystem::path& p,
                       const std::filesystem::path& existing_p );

bool create_directory( const std::filesystem::path& p,
                       const std::filesystem::path& existing_p,
                       std::error_code& ec ) noexcept;

bool create_directories( const std::filesystem::path& p );

bool create_directories( const std::filesystem::path& p,
                         std::error_code& ec );
```

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

int main()
{
    fs::current_path(fs::temp_directory_path());
    fs::create_directories("sandbox/1/2/a");
    fs::create_directory("sandbox/1/2/b");
    fs::permissions("sandbox/1/2/b", fs::perms::others_all, fs::perm_options::remove);
    fs::create_directory("sandbox/1/2/c", "sandbox/1/2/b");
    std::system("ls -l sandbox/1/2");
    std::system("tree sandbox");
    fs::remove_all("sandbox");
}
```

```
drwxr-xr-x 2 user group 4096 Apr 15 09:33 a
drwxr-x--- 2 user group 4096 Apr 15 09:33 b
drwxr-x--- 2 user group 4096 Apr 15 09:33 c
sandbox
└── 1
    └── 2
        ├── a
        ├── b
        └── c
```

Copiar arquivo ou diretório

```
void copy( const std::filesystem::path& from,
           const std::filesystem::path& to );

void copy( const std::filesystem::path& from,
           const std::filesystem::path& to,
           std::error_code& ec );

void copy( const std::filesystem::path& from,
           const std::filesystem::path& to,
           std::filesystem::copy_options options );

void copy( const std::filesystem::path& from,
           const std::filesystem::path& to,
           std::filesystem::copy_options options,
           std::error_code& ec );
```

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

int main()
{
    fs::create_directories("sandbox/dir/subdir");
    std::ofstream("sandbox/file1.txt").put('a');
    fs::copy("sandbox/file1.txt", "sandbox/file2.txt"); // copy file
    fs::copy("sandbox/dir", "sandbox/dir2"); // copy directory (non-recursive)
    const auto copyOptions = fs::copy_options::update_existing
                           | fs::copy_options::recursive
                           | fs::copy_options::directories_only;
    fs::copy("sandbox", "sandbox_copy", copyOptions);
    static_cast<void>(std::system("tree"));
    fs::remove_all("sandbox");
    fs::remove_all("sandbox_copy");
}
```

```
.
├── sandbox
│   ├── dir
│   │   └── subdir
│   ├── dir2
│   ├── file1.txt
│   └── file2.txt
└── sandbox_copy
    ├── dir
    │   └── subdir
    └── dir2

8 directories, 2 files
```

Copiar arquivo

```
bool copy_file( const std::filesystem::path& from,
                const std::filesystem::path& to );
bool copy_file( const std::filesystem::path& from,
                const std::filesystem::path& to,
                std::error_code& ec );
bool copy_file( const std::filesystem::path& from,
                const std::filesystem::path& to,
                std::filesystem::copy_options options );
bool copy_file( const std::filesystem::path& from,
                const std::filesystem::path& to,
                std::filesystem::copy_options options,
                std::error_code& ec );
```

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

int main()
{
    fs::create_directory("sandbox");
    std::ofstream("sandbox/file1.txt").put('a');

    fs::copy_file("sandbox/file1.txt", "sandbox/file2.txt");

    // now there are two files in sandbox:
    std::cout << "file1.txt holds: "
              << std::ifstream("sandbox/file1.txt").rdbuf() << '\n';
    std::cout << "file2.txt holds: "
              << std::ifstream("sandbox/file2.txt").rdbuf() << '\n';

    // fail to copy directory
    fs::create_directory("sandbox/abc");
```

```
    try
    {
        fs::copy_file("sandbox/abc", "sandbox/def");
    }
    catch (fs::filesystem_error& e)
    {
        std::cout << "Could not copy sandbox/abc: " << e.what() << '\n';
    }
    fs::remove_all("sandbox");
}
```

```
file1.txt holds: a
file2.txt holds: a
Could not copy sandbox/abc: copy_file: Is a directory: "sandbox/abc", "sandbox/def"
```

Modificação de arquivos e diretórios

Renomear arquivo ou diretório

```
void rename( const std::filesystem::path& old_p,
             const std::filesystem::path& new_p );
void rename( const std::filesystem::path& old_p,
             const std::filesystem::path& new_p,
             std::error_code& ec ) noexcept;
```

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

int main()
{
    std::filesystem::path p = std::filesystem::current_path() / "sandbox";
    std::filesystem::create_directories(p / "from");
    std::ofstream{ p / "from/file1.txt" }.put('a');
    std::filesystem::create_directory(p / "to");

//  fs::rename(p / "from/file1.txt", p / "to/"); // error: "to" is a directory
    fs::rename(p / "from/file1.txt", p / "to/file2.txt"); // OK
//  fs::rename(p / "from", p / "to"); // error: "to" is not empty
    fs::rename(p / "from", p / "to/subdir"); // OK

    std::filesystem::remove_all(p);
}
```

Alterar permissão de acesso de arquivo ou diretório

```
void permissions( const std::filesystem::path& p,
                  std::filesystem::perms prms,
                  std::filesystem::perm_options opts = perm_options::replace );
void permissions( const std::filesystem::path& p,
                  std::filesystem::perms prms,
                  std::error_code& ec ) noexcept;
void permissions( const std::filesystem::path& p,
                  std::filesystem::perms prms,
                  std::filesystem::perm_options opts,
                  std::error_code& ec );
```

```
#include <filesystem>
#include <fstream>
#include <iostream>

void demo_perms(std::filesystem::perms p)
{
    using std::filesystem::perms;
    auto show = [=](char op, perms perm)
    {
        std::cout << (perms::none == (perm & p) ? '-' : op);
    };
    show('r', perms::owner_read);
    show('w', perms::owner_write);
    show('x', perms::owner_exec);
    show('r', perms::group_read);
    show('w', perms::group_write);
    show('x', perms::group_exec);
    show('r', perms::others_read);
    show('w', perms::others_write);
    show('x', perms::others_exec);
```

```
    std::cout << '\n';
}

int main()
{
    std::ofstream("test.txt"); // create file

    std::cout << "Created file with permissions: ";
    demo_perms(std::filesystem::status("test.txt").permissions());

    std::filesystem::permissions(
        "test.txt",
        std::filesystem::perms::owner_all | std::filesystem::perms::group_all,
        std::filesystem::perm_options::add
    );

    std::cout << "After adding u+rwx and g+rwx: ";
    demo_perms(std::filesystem::status("test.txt").permissions());

    std::filesystem::remove("test.txt");
```

```
}
```

```
Created file with permissions: rw-r--r--
After adding u+rwx and g+wrx:  rwxrwxr--
```

Obter ou definir diretório de trabalho atual

```
std::filesystem::path current_path();
std::filesystem::path current_path( std::error_code& ec );
void current_path( const std::filesystem::path& p );
void current_path( const std::filesystem::path& p,
                   std::error_code& ec ) noexcept;
```

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

int main()
{
    std::cout << "Current path is " << fs::current_path() << '\n';
    fs::current_path(fs::temp_directory_path());
    std::cout << "Current path is " << fs::current_path() << '\n';
}
```

```
Current path is "D:/local/ConsoleApplication1"
Current path is "E:/Temp"
```

Verificação de arquivos e diretórios

Verificar se arquivo ou diretório existem

```
bool exists( std::filesystem::file_status s ) noexcept;
bool exists( const std::filesystem::path& p );
bool exists( const std::filesystem::path& p,
             std::error_code& ec ) noexcept;
```

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

void demo_exists(const fs::path& p, fs::file_status s = fs::file_status{})
{
    std::cout << p;
    if(fs::status_known(s) ? fs::exists(s) : fs::exists(p))
        std::cout << " exists\n";
    else
        std::cout << " does not exist\n";
}

int main()
{
    const fs::path sandbox{"sandbox"};
    fs::create_directory(sandbox);
    std::ofstream{sandbox/"file"}; // create regular file
```

```
    fs::create_symlink("non-existing", sandbox/"symlink");

    demo_exists(sandbox);

    for (const auto& entry : fs::directory_iterator(sandbox))
        demo_exists(entry, entry.status()); // use cached status from directory entry

    fs::remove_all(sandbox);
}
```

```
"sandbox" exists
"sandbox/symlink" does not exist
"sandbox/file" exists
```

Verificar se é um arquivo

```
bool is_regular_file( std::filesystem::file_status s ) noexcept;
bool is_regular_file( const std::filesystem::path& p );
bool is_regular_file( const std::filesystem::path& p,
                      std::error_code& ec ) noexcept;
```

Verificar se é um diretório

```
bool is_directory( std::filesystem::file_status s ) noexcept;
bool is_directory( const std::filesystem::path& p );
bool is_directory( const std::filesystem::path& p,
                   std::error_code& ec ) noexcept;
```

```
#include <cstdio>
#include <cstring>
#include <filesystem>
#include <fstream>
#include <iostream>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/un.h>
#include <unistd.h>

namespace fs = std::filesystem;

void demo_status(const fs::path& p, fs::file_status s)
{
    std::cout << p;
    // alternative: switch(s.type()) { case fs::file_type::regular: ...}
    if (fs::is_regular_file(s))
        std::cout << " is a regular file\n";
    if (fs::is_directory(s))
        std::cout << " is a directory\n";
```

```
    if (fs::is_block_file(s))
        std::cout << " is a block device\n";
    if (fs::is_character_file(s))
        std::cout << " is a character device\n";
    if (fs::is_fifo(s))
        std::cout << " is a named IPC pipe\n";
    if (fs::is_socket(s))
        std::cout << " is a named IPC socket\n";
    if (fs::is_symlink(s))
        std::cout << " is a symlink\n";
    if (!fs::exists(s))
        std::cout << " does not exist\n";
}

int main()
{
    // create files of different kinds
    fs::create_directory("sandbox");
    fs::create_directory("sandbox/dir");
    std::ofstream{"sandbox/file"}; // create regular file
```

```
    fs::create_symlink("file", "sandbox/symlink");

    mkfifo("sandbox/pipe", 0644);
    sockaddr_un addr;
    addr.sun_family = AF_UNIX;
    std::strcpy(addr.sun_path, "sandbox/sock");
    int fd = socket(PF_UNIX, SOCK_STREAM, 0);
    bind(fd, reinterpret_cast<sockaddr*>(&addr), sizeof addr);

    // demo different status accessors
    for (auto it{fs::directory_iterator("sandbox")}; it != fs::directory_iterator(); ++it)
        demo_status(*it, it->symlink_status()); // use cached status from directory entry
        demo_status("/dev/null", fs::status("/dev/null")); // direct calls to status
        demo_status("/dev/sda", fs::status("/dev/sda"));
        demo_status("sandbox/no", fs::status("/sandbox/no"));

    // cleanup (prefer std::unique_ptr-based custom deleters)
    close(fd);
    fs::remove_all("sandbox");
}
```

```
"sandbox/file" is a regular file
"sandbox/dir" is a directory
"sandbox/pipe" is a named IPC pipe
"sandbox/sock" is a named IPC socket
"sandbox/symlink" is a symlink
"/dev/null" is a character device
"/dev/sda" is a block device
"sandbox/no" does not exist
```

Verificar se é um arquivo ou diretório vazio

```
bool is_empty( const std::filesystem::path& p );
bool is_empty( const std::filesystem::path& p,
               std::error_code& ec );
```

```cpp
#include <cstdio>
#include <filesystem>
#include <fstream>
#include <iostream>

int main()
{
    namespace fs = std::filesystem;

    const fs::path tmp_dir{fs::temp_directory_path()};
    std::cout << std::boolalpha
              << "Temp dir: " << tmp_dir << '\n'
              << "is_empty(): " << fs::is_empty(tmp_dir) << '\n';

    const fs::path tmp_name{tmp_dir / std::tmpnam(nullptr)};
    std::cout << "Temp file: " << tmp_name << '\n';

    std::ofstream file{tmp_name.string()};
    std::cout << "is_empty(): " << fs::is_empty(tmp_name) << '\n';
    file << "cppreference.com";
```

```
    file.flush();
    std::cout << "is_empty(): " << fs::is_empty(tmp_name) << '\n'
              << "file_size(): " << fs::file_size(tmp_name) << '\n';
    file.close();
    fs::remove(tmp_name);
}
```

```
Temp dir: "/tmp"
is_empty(): false
Temp file: "/tmp/fileCqd9DM"
is_empty(): true
is_empty(): false
file_size(): 16
```

Exclusão de arquivos ou diretórios

Excluir arquivo ou diretório

```
bool remove( const std::filesystem::path& p );
bool remove( const std::filesystem::path& p,
             std::error_code& ec ) noexcept;
std::uintmax_t remove_all( const std::filesystem::path& p );
std::uintmax_t remove_all( const std::filesystem::path& p,
                           std::error_code& ec );
```

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

int main()
{
    fs::path tmp{std::filesystem::temp_directory_path()};
    std::filesystem::create_directories(tmp / "abcdef/example");
    std::uintmax_t n{fs::remove_all(tmp / "abcdef")};
    std::cout << "Deleted " << n << " files or directories\n";
}
```

```
Deleted 2 files or directories
```

Recursos adicionais

Obter conteúdo de um diretório

```
directory_entry
directory_iterator
recursive_directory_iterator
```

```
#include <algorithm>
#include <filesystem>
#include <fstream>
#include <iostream>

int main()
{
    const std::filesystem::path sandbox{"sandbox"};
    std::filesystem::create_directories(sandbox/"dir1"/"dir2");
    std::ofstream{sandbox/"file1.txt"};
    std::ofstream{sandbox/"file2.txt"};

    std::cout << "directory_iterator:\n";
    // directory_iterator can be iterated using a range-for loop
    for (auto const& dir_entry : std::filesystem::directory_iterator{sandbox})
        std::cout << dir_entry.path() << '\n';

    std::cout << "\ndirectory_iterator as a range:\n";
    // directory_iterator behaves as a range in other ways, too
    std::ranges::for_each(
```

```
        std::filesystem::directory_iterator{sandbox},
        [](const auto& dir_entry) { std::cout << dir_entry << '\n'; });

    std::cout << "\nrecursive_directory_iterator:\n";
    for (auto const& dir_entry : std::filesystem::recursive_directory_iterator{sandbox})
        std::cout << dir_entry << '\n';

    // delete the sandbox dir and all contents within it, including subdirs
    std::filesystem::remove_all(sandbox);
}
```

```
directory_iterator:
"sandbox/file2.txt"
"sandbox/file1.txt"
"sandbox/dir1"

directory_iterator as a range:
"sandbox/file2.txt"
"sandbox/file1.txt"
"sandbox/dir1"

recursive_directory_iterator:
"sandbox/file2.txt"
"sandbox/file1.txt"
"sandbox/dir1"
"sandbox/dir1/dir2"
```

Tratamento de exceções

`filesystem_error`

```
#include <filesystem>
#include <iostream>
#include <system_error>

int main()
{
    const std::filesystem::path from{"/none1/a"}, to{"/none2/b"};

    try
    {
        std::filesystem::copy_file(from, to); // throws: files do not exist
    }
    catch (std::filesystem::filesystem_error const& ex)
    {
        std::cout << "what(): " << ex.what() << '\n'
                  << "path1(): " << ex.path1() << '\n'
                  << "path2(): " << ex.path2() << '\n'
                  << "code().value(): " << ex.code().value() << '\n'
                  << "code().message(): " << ex.code().message() << '\n'
                  << "code().category(): " << ex.code().category().name() << '\n';
```

```
    }

    // All functions have non-throwing equivalents
    std::error_code ec;
    std::filesystem::copy_file(from, to, ec); // does not throw
    std::cout << "\nNon-throwing form sets error_code: " << ec.message() << '\n';
}
```

```
what():  filesystem error: cannot copy file: No such file or directory [/none1/a] [/none2/b]
path1(): "/none1/a"
path2(): "/none2/b"
code().value():    2
code().message():  No such file or directory
code().category(): generic

Non-throwing form sets error_code: No such file or directory
```

Referências

C++ Reference: Filesystem Library

https://en.cppreference.com/w/cpp/filesystem