Filesystem functions in C++

Jussi Auvinen

Frankfurt Institute for Advanced Studies

C++ User Group Meeting
July 30, 2014

Examples of file handling in scripts

Bash:

```
directory="$PWD/testdir"
filename="testfile.txt"
if [ -e $directory ]; then
  cd $directory
  if [ -e $filename ]; then
    echo "data" >> $filename
  else
    echo "header" > $filename
    echo "data" >> $filename
  fi
else
  mkdir $directory
  cd $directory
  echo "header" > $filename
  echo "data" >> $filename
fi
```

Perl:

```
use strict;
use Cwd;
my $workdir = getcwd;
my $directory="$workdir/testdir";
my $filename="testfile.txt";

if ( -d $directory ) {
    chdir $directory;
    if ( -f $filename ) {
        open(DATAFILE, ">>$filename");
        print DATAFILE "data\n";
        close(DATAFILE);
    } else {
        open(DATAFILE, ">$filename");
        print DATAFILE "header\n";
        print DATAFILE "data\n";
        close(DATAFILE);
    }
} else {
    mkdir $directory;
    chdir $directory;
    open(DATAFILE, ">$filename");
    print DATAFILE "header\n";
    print DATAFILE "data\n";
    close(DATAFILE);
}
```

Python:

```
import os
directory = os.getcwd() + "/testdir/"
filename = "testfile.txt"

if os.path.isdir(directory):
  os.chdir(directory)
  if os.path.isfile(directory + filename):
    f = open(filename, 'a')
    f.write("data\n")
    f.close()
  else:
    f = open(filename, 'w')
    f.write("header\n")
    f.write("data\n")
    f.close()
else:
  os.mkdir(directory)
  os.chdir(directory)
  f = open(filename, 'w')
  f.write("header\n")
  f.write("data\n")
  f.close()
```

File handling in C++

No standard for filesystem operations in C++! Problem: Portability.

Some "Realities" copied from Boost Filesystem Library Design:

`http://www.boost.org/doc/libs/1_55_0/libs/filesystem/doc/design.htm`

- Some operating systems have a single directory tree root, others have multiple roots.
- Some file systems provide both a long and short form of filenames.
- Some file systems have different syntax for file paths and directory paths.
- Some file systems have different rules for valid file names and valid directory names.

Current options: libc, Boost, Qt Project
Incoming: File System TS

POSIX (Portable Operating System Interface)

From "The Open Group Base Specifications Issue 7, IEEE Std 1003.1, 2013 Edition":

"POSIX.1-2008 defines a standard operating system interface and environment, including a command interpreter (or "shell"), and common utility programs to support applications portability at the source code level.

`http://pubs.opengroup.org/onlinepubs/9699919799/`

- File access permissions, file hierarchy, filenames, timestamps
- File format notation (`printf`, `scanf`)
- Character set
- Locale (character case conversion, ordering, date format...)
- Environment variables
- Regular expressions (string selection)
- Directory structure and devices (/, `/tmp`, `/dev`, ...)
- General terminal interface

libc (assuming POSIX compliance)

```
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main(void) {
  char cwd[1024], dircopy[1024];
  getcwd(cwd, sizeof(cwd));
  const char *directory = strncat(cwd, "/testdir/", 10);
  strncpy(dircopy, directory, sizeof(dircopy));
  const char *filepath = strncat(dircopy, "testfile.txt", 13);
  struct stat s;
  stat(directory, &s);
  if (S_ISDIR(s.st_mode)) {
    FILE *fp;
    if (access(filepath, F_OK) == 0) {
      fp = fopen(filepath, "a");
      fprintf(fp, "data\n");
      fclose(fp);
    } else {
      fp = fopen(filepath, "w");
      fprintf(fp, "header\ndata\n");
      fclose(fp);
    }
  } else {
    mkdir(directory, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
    FILE *fp;
    fp = fopen(filepath, "w");
    fprintf(fp, "header\ndata\n");
    fclose(fp);
  }
  return 0;
}
```

`char *getcwd(char *, size_t)` – get working directory

`int stat(const char *, struct stat *)` – info about path

`struct stat` – contains file info fields, including `st_mode`

`S_ISDIR(st_mode)` – POSIX macro to check if directory

`int access(const char *, int)` – check if file can be accessed

`int mkdir(const char *, mode_t)` – create directory with permissions given in `mode_t`

Boost.Filesystem

- POSIX-compliant
- A modern C++ interface, highly compatible with the C++ standard library
- "Reasonably" portable across operating systems
- Error handling and reporting via C++ exceptions (the default) or error codes.

Boost.Filesystem

```
#include <boost/filesystem.hpp>
#include <boost/filesystem/fstream.hpp>

namespace bf = boost::filesystem;

int main() {
  bf::path directory = bf::absolute("testdir");
  bf::path filepath = directory / "testfile.txt";

  if (bf::is_directory(directory)) {
    if (bf::exists(filepath)) {
      bf::ofstream file(filepath, std::ios::app);
      file << "data\n";
      file.close();
    } else {
      bf::ofstream file(filepath);
      file << "header\n";
      file << "data\n";
      file.close();
    }
  } else {
    bf::create_directory(directory);
    bf::ofstream file(filepath);
    file << "header\n" << "data\n";
    file.close();
  }
  return 0;
}
```

`Class path` – represents a path, and contains a pathname. The path does not necessarily exist in external storage, and the pathname is not necessarily valid for the current operating system or for a particular file system.

`path absolute(const path&, const path& base=current_path())` – absolute path. If root directory is not included, append to current working directory

`bool is_directory(const path&)` – check if path is directory

`bool exists(const path&)` – check if path has known status

`ofstream` – same as `std::ofstream` but uses `basic_path` instead of `char *`

`bool create_directory(const path&)` – attempt to create the directory path resolves to, as if by `mkdir()` with a second argument of `S_IRWXU|S_IRWXG|S_IRWXO`

Proposed standard: Filesystem TS (Technical Specification)

`http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4099.html`

- Based on Boost.Filesystem
- C++14
- `std::experimental::filesystem::v1::`

Qt Project

`http://qt-project.org/`

- Cross-platform application and UI framework
- C++ or QML (a CSS / JavaScript like language)

Qt Filesystem

```
#include <QDir>
#include <QFile>
#include <QString>
#include <QTextStream>

int main() {
  QString directory = "testdir/";
  QString filename = directory + "testfile.txt";
  QDir dir(directory);
  QFile file(filename);
  if (dir.exists()) {
    if (file.exists()) {
      file.open(QIODevice::Append | QIODevice::Text);
      QTextStream f(&file);
      f << "data\n";
      file.close();
    } else {
      file.open(QIODevice::WriteOnly | QIODevice::Text);
      QTextStream f(&file);
      f << "header\n" << "data\n";
      file.close();
    }
  } else {
    QDir().mkdir(directory);
    file.open(QIODevice::WriteOnly | QIODevice::Text);
    QTextStream f(&file);
    f << "header\n" << "data\n";
    file.close();
  }
}
```

`Class QString` – Unicode character string object

`Class QDir` – manipulate path names, access information regarding paths and files, and manipulate the underlying file system. Path can be either relative or absolute.

- `bool QDir::mkdir(const QString &) const` – create sub-directory

`Class QFile` – interface for reading from and writing to files

- `bool QFile::open(OpenMode)` – open the file using `OpenMode`

`Class QTextStream` – interface for reading and writing text

`Class QIODevice` – base interface class of all I/O devices.

- `enum QIODevice::OpenModeFlag`
- `flags QIODevice::OpenMode`