Feedback on practical use of C++17 std::filesystem::recursive_directory_iterator

Noël Tchidjo Moyo
noel.tchidjomoyo@murex.com
@NoelTchidjo

Motivation

Problem

With the constant increase of complex applications, which have several years of life, it becomes necessary for the tools processing the files of these applications to be able to quickly list the types of files that interest them.

The object type std :: recursive_directory_iterator introduced in the C++17 standard, allows to list the entries of a directory. Its use is usually coupled with a test to check if the extension of the file corresponds to that which interests us. This use can in some complex projects suffer of performance issue. This is the reason why we have investigated two API change proposals that allow to pass directly a filter as parameter of std :: recursive_directory_iterator constructor.

The change proposals are:

1. The addition of a regular expression as parameter to recursive_directory_iterator constructor
2. The addition of a user-provided lambda as parameter to recursive_directory_iterator constructor.

We have modified GCC libstdc++<filesystem> API and CLANG libc++<filesystem> API in order to achieve this. We then present the resulting code expressiveness and time performance. The tests have been performed by running a directory traverser program over several large open source file sets. Code and patches are located at https://github.com/bonpiedlaroute/cppcon2018

General Solution

```
for(auto& entry:fs::recursive_directory_iterator(folder))
{
   auto ext = entry.path().extension().c_str();
   if( strcmp(ext, ".c") == 0 || strcmp(ext, ".h") == 0
     || strcmp(ext, ".cpp") == 0
     || strcmp(ext, ".hpp") == 0 )
   {
     do_some_work(entry.path().filename().c_str());
   }
}
```

Problem with the General Solution

Huge time spent to list files of interest on projects with large file sets

- We already know the type of file of interest
- What will be the resulting code expressiveness and time performance if we pass directly a filter to std::recursive_directory_iterator ?

Proposed Solution

recursive_directory_iterator with std::regex parameter

```
explicit recursive_directory_iterator_r(const fs::path& __p,
   const std::regex& reg,  const fs::pattern_options& po)
```

** pattern_options parameter allows to specify the type of entry on which we should apply the regex (file, directory etc.). It also indicates the type of entry to be returned by the iterator.*

usage with std::regex parameter

```
for(auto& entry: fs::recursive_directory_iterator_r(folder,
   std::regex(".*\\.h|.*\\.c|.*\\.cpp|.*\\.hpp"),
   fs::pattern_options::file_only))
{
  do_some_work(entry.path().filename().c_str());
}
```

recursive_directory_iterator with lambda parameter

```
explicit recursive_directory_iterator_l(const fs::path& __p,
const std::function<bool(const char* )> lambda,
const fs::pattern_options& po)
```

usage with lambda parameter

```
for(auto& entry : fs::recursive_directory_iterator_l(folder,
[](const char* filename)
{
   const char* ext = get_filename_ext(filename);
   return strcmp(ext, ".c") == 0
                  || strcmp(ext, ".h") == 0
                  || strcmp(ext, ".cpp") == 0
                  || strcmp(ext, ".hpp") == 0;
 },
 fs::pattern_options::file_only ))
 {
   do_some_work(entry.path().filename().c_str());
 }
```

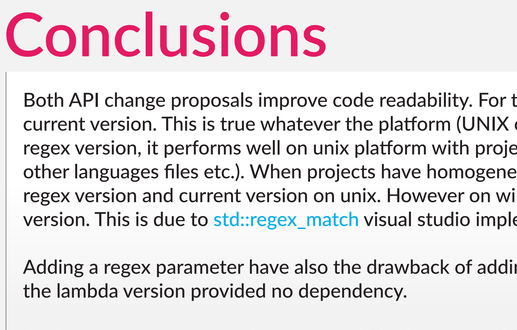
gcc7-3 and clang5-0 patches for implementation of the api change

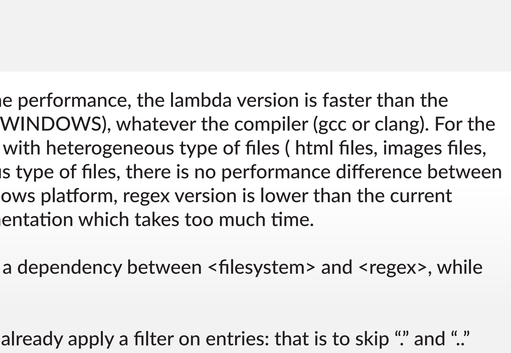
https://github.com/bonpiedlaroute/cppcon2018/blob/master/unix/recursive_directory_iterator_gcc7-3.patch

https://github.com/bonpiedlaroute/cppcon2018/blob/master/windows/recursive_directory_iterator_clang-5.0.0.patch

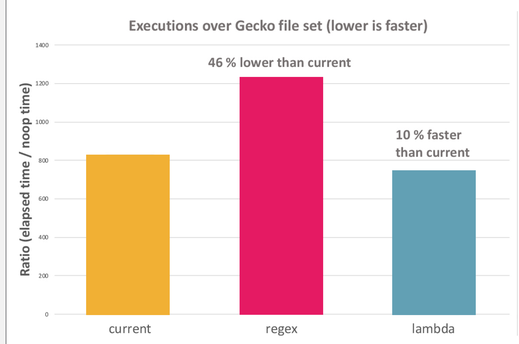
Results

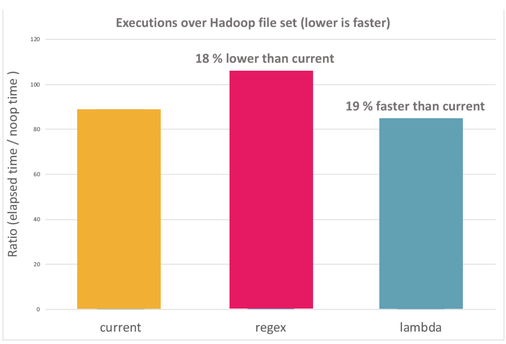
time performance with gcc7-3 libstdc++<filesystem> implementation (UNIX)





time performance with clang5-0 libc++<filesystem> implementation (WINDOWS)





Conclusions

Both API change proposals improve code readability. For time performance, the lambda version is faster than the current version. This is true whatever the platform (UNIX or WINDOWS), whatever the compiler (gcc or clang). For the regex version, it performs well on unix platform with project with heterogeneous type of files (html files, images files, other languages files etc.). When projects have homogeneous type of files, there is no performance difference between regex version and current version on unix. However on windows platform, regex version is lower than the current version. This is due to std::regex_match visual studio implementation which takes too much time.

Adding a regex parameter have also the drawback of adding a dependency between <filesystem> and <regex>, while the lambda version provided no dependency.

Implementations on compilers are not difficult, as compilers already apply a filter on entries: that is to skip “.” and “..” entries.