Project: ISO JTC1/SC22/WG21: Programming Language C++
Doc No: WG21 **P1164R1**
Date: 2019-02-22
Reply to: Nicolai Josuttis (nico@josuttis.de)
Audience: LWG
Prev. Version: P1164R0

Make create_directory() intuitive (so that it is always an error, if there is already a file that is no directory)

https://wg21.link/lwg2935 was resolved so that it is not an error if calling create_directory() or create_directories() fails, because at the passed path there is already a file that is not a directory.

This is highly counter-intuitive and a source of trouble.

In addition note that it is an error, if any filename other than the trailing filename exists and is not a directory. Thus, for

create_directory("a/b/c")

or

create_directories("a/b/c")

we get

- Error if a exists and is not a directory
- Error if b exists and it not a directory
- No error if c exists and it not a directory

So, we should add back the postcondition that after a successful call is_directory(p) is true.

Benefits of the proposed fix

If we make the change proposed here, we get a lot of benefits:

- Intuitive and self-explanatory behavior

- An easy way to handle the problem already having a non-directory directly at the location where the problem occurs instead f getting strange errors or side-effects later on
  - Programmers can implement

```
create_directory(p)
```
…   // perform a lot of stuff assuming the call went well
```
ofstream f(p / "data.txt");
if (!f) {
```
    … // handle problems with opening the file
```
}
```

without the problem that they see a confusing error message at a later stage, which they would have to avoid with code like:

```
create_directory(p)
```
…   // perform a lot of stuff assuming the call went well
```
ofstream f(p / "data.txt");
if (!f) {
  if (!is_directory(p)) {
    throw ????   // handle error if create_directory() failed
  }
  … // handle problems with opening the file
}
```

or:

```
if (!create_directory(p) && !is_directory(p)) {
  throw ????   // error handling if this is a consequence of a failed previous call
```

… // perform a lot of stuff assuming the call went well

```
ofstream f(p / "data.txt");
if (!f) {
```

… // handle problems with opening the file

```
}
```

- Consistent error handling if different filenames in the path exist as non-directory

Counter Arguments

There were arguments to resolve the issues at done, which we would like to comment:

- The postcondition to have a directory there anyway might be broken due to a data race with other calls dealing with the same path.
  - In general, this is true, but this can't be an argument to accept non-intuitive behavior even if no data race occurs. Otherwise we can argue that any postcondition of any call doesn't make any sense at all.
  - If according to the context or situation a data race does not happen, the call should do the expected right thing. This applies to environments where no data race is possible as well as situations where data races are usually avoided (people are used to have different directories for different tasks/persons/applications to avoid problems like this).

- The underlying operating system does not support this case directly so that another OS call is necessary.
  - Note that in the good case if the call succeeds and created a new directory no additional OS call is necessary. Only if the OS call to create the directory failed, an additional check might be necessary to find out whether there is already something else. That is, internally the call has to perform in addition something like:

        if (*createDirectoryFailed* && !is_directory(p)) {
          … // corresponding error handling
        }

Which error condition should be used?

There are two options:

1. Make this implementation-defined
2. Use errc::not_a_directory (errno ENOTDIR)

Discuss: We propose option 1.

This should be a defect against C++17.

Proposed Wording

 (All against N4800)

Make the following edits to 28.11.14.6 [fs.op.create_directories]:

```
bool create_directories(const path& p);
bool create_directories(const path& p, error_code& ec) noexcept;
```

> -1- *Effects:* Calls `create_directory()` for each element of `p` that does not exist.

> -2- *Returns:* `true` if a new directory was created for the directory `p` resolves to, otherwise `false`. ~~The signature with argument `ec` returns `false` if an error occurs.~~

> -3- *Throws:* As specified in 28.11.6 [fs.err.report].

> -4- *Complexity:* $\mathcal{O}(n)$ where *n* is the number of elements of `p`.

Make the following edits to 28.11.14.7 [fs.op.create_directory]:

```
bool create_directory(const path& p);
bool create_directory(const path& p, error_code& ec) noexcept;
```

> -1- *Effects:* Creates the directory `p` resolves to, as if by POSIX `mkdir()` with a second argument of `static_cast<int>(perms::all)`. ~~Creation failure because `p` already exists is not an error.~~ If `mkdir()` fails because p resolves to an existing directory, no error is reported. Otherwise on failure an error is reported.

> -2- *Returns:* `true` if a new directory was created, otherwise `false`. ~~The signature with argument `ec` returns `false` if an error occurs.~~

> -3- *Throws:* As specified in 28.11.6 [fs.err.report].

```
bool create_directory(const path& p, const path& existing_p);
bool create_directory(const path& p, const path& existing_p,
                      error_code& ec) noexcept;
```
> *-4- Effects:* Create the directory p resolves to, with attributes copied from directory existing_p. The set of attributes copied is operating system dependent. ~~Creation failure because p resolves to an existing directory shall not be treated as an error.~~ If `mkdir()` fails because p already resolves to an existing directory, no error is reported. Otherwise on failure an error is reported.
> [*Note:* For POSIX-based operating systems, the attributes are those copied by native API stat(existing_p.c_str(), &attributes_stat) followed by mkdir(p.c_str(), attributes_-stat.st_mode). For Windows-based operating systems, the attributes are those copied by native API CreateDirectoryExW(existing_p.c_str(), p.c_str(), 0). —*end note*]

> *-5- Returns:* `true` if a new directory was created with attributes copied from directory `existing_p`, otherwise `false`.

> *-6- Throws:* As specified in 28.11.6 [fs.err.report].