Ruby master - Bug #18923
Dir.glob Ernno::ENAMETOOLONG - Caused by outdated logic in open_dir_handle (win32.c)
07/17/2022 08:33 PM - test35965@gmail.com (Alexander Riccio)

Status: Open
Priority: Normal
Assignee:
Target version:

<table>
<thead>
<tr>
<th>ruby -v:</th>
<th>Backport:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ruby 3.1.1p18 (2022-02-18 revision 53f5fc4236) [x64-mingw-ucrt]</td>
<td>2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN</td>
</tr>
</tbody>
</table>

Description
This bug - as do most of my bug reports - started out while dealing with something productive and completely unrelated :)

In short: open_dir_handle in win32.c handles long paths incorrectly.

At best, this will cause programs that use Dir.glob to crash on windows when paths in the tree exceed the length allocated by ruby. I believe this is wrong in master too.

I'm a bit confused by error handling for GetFinalPathNameByHandleW. It looks correct, but its a confusing enough API that I'm going to mention it further down just in case.

In more detail:

As the developer of altWinDirStat, I'm way more familiar than I'd like with the labyrinth of weirdness that is handling paths on Windows. Long file paths have been supported through UNC paths and similar weirdness for as long as I've been around, maybe since the beginning of NT.

Many probably know all of this already, but will recap off the top of my head for context:

"Long" means up to the UTF-16 limit of 32k characters or so (and yes, IIRC, official documentation does not give the exact number of 32,768 because they say something about substitutions and such), which in practice is half the max value UNICODE_STRING struct can store in the USHORT Length member. Windows 10 has recently removed the MAX_PATH restriction from many dated Windows APIs that used to require kinda-UNC path prefixing ("\?"), which makes things easier for users, but also broke a lot of poorly written software that handled buffers incorrectly.

In ruby, any program that relies on Dir.glob will crash with a valid final path that exceeds MAX_PATH, since GetFinalPathNameByHandleW will return a length longer than FINAL_PATH_MAX.

One option to fix this is to make the stack buffer large enough to hold any possible string. I don't think this is a good idea. But it would be the smallest change.

A different option is to add this error explicitly to the Dir.glob docs, and make every single program work around this. I don't think that's a good fix either.

Another option to fix this that I do not necessarily recommend is to do the classic windows thing and call GetFinalPathNameByHandleW once with a zero-sized buffer and then use the return value of GetFinalPathNameByHandleW to allocate a sufficiently large buffer on the heap to hold the buffer, then call it again. Much slower, but definitely works well. This could cause a lot of heap churn if you have a lot of files (e.g. anybody who has a node_modules somewhere in their monorepo). Apparently people are already paying attention to filepath-sized allocations in ruby.

I'm very very OCD about how I use heap when I'm writing native code, it's a huge pain, but OCD means OCD and slightly better performance :). If you want to do it, the other option I see is first try the API with a MAX_PATH sized stack buffer, and if that works, excellent! Lightning fast code, no heap. If the stack buffer is too small, then I'll allocate a heap buffer big enough to hold the size of the string requested by whatever that win32 api has suggested I use in the return code. Excellent performance, but you need to make sure you're not introducing new bugs when you implement the code twice now, and not to mix up the heap/stack buffers.

With any fix, I suggest checking the last error on failure and reporting that instead. That may be better than just using the length of the input string. If the error is any of the errors listed in the documentation, it probably doesn't make sense to use the length of the input string anyways... in that case there's something wrong?

It might be a good idea for someone with the time to go through and update all the MAX_PATH-adjacent code to support long paths. That will be a huge endeavor, but worth it.

08/22/2022
You can reproduce this with one line, calling `Dir.glob([[]])` on any valid path that's longer than 260 characters.

I'm also noticing that Ruby is still using `lstrlenW` and `lstrcat` in `win32.c`... this is a terrible idea! `lstrcat` catches access violations/segfaults and then just continues program execution. I've seen this happen in other software and lost data to it. It's a pervasive enough problem on windows that I once thought of writing an EMET-style shim to redirect those system calls to normal `stdlib` functions that crash on access violations. You don't need to switch to `strsafe` func or `secure c lib` func, it's an easy enough drop-in fix to switch to standard functions. Nothing will break unless something was already broken. I'll open a simple bug for that.

The "maybe an issue" is that the `GetFinalPathNameByHandleW` appears even more confusing than most old win32 path-handling functions. See the docs:

If the function succeeds, the return value is the length of the string received by `lpszFilePath`, in TCHARs. This value does not include the size of the terminating null character.

...then a note for the ANSI version on vista and following windows versions, with subtly contradictory behavior, and then the failure behavior:

If the function fails because `lpszFilePath` is too small to hold the string plus the terminating null character, the return value is the required buffer size, in TCHARs. This value includes the size of the terminating null character.

If the function fails for any other reason, the return value is zero. To get extended error information, call `GetLastError`.

...and then a table that probably talks about the possible values `GetLastError` will return, though annoyingly, doesn't say that explicitly.

I've been off-by-one when dealing with APIs like this enough times to not trust myself or any human around it. The SAL in the headers isn't as good as it is for some similarly-worded APIs, so it's not useful here.