Diagnosing a hang: Everybody stuck in WinHttpGet-ProxyForUrl

devblogs.microsoft.com/oldnewthing/20200501-00

May 1, 2020

Raymond Chen

A customer reported that their program eventually ground to a halt with over 750 threads stuck in `WinHttpGetProxyForUrl`:

```
ntdll!ZwWaitForSingleObject+0x14
KERNELBASE!WaitForSingleObjectEx+0x8f
winhttp!OutProcGetProxyForUrl+0x160
winhttp!WinHttpGetProxyForUrl+0x349
contoso!submit_web_request+0x232
ntdll!TppWorkpExecuteCallback+0x35e
ntdll!TppWorkerThread+0x474
kernel32!BaseThreadInitThunk+0x14
ntdll!RtlUserThreadStart+0x21
```

(I’ve simplified the stack trace for expository purposes.)

What’s happening here is that you put some work on the thread pool, and that work called `WinHttpGetProxyForUrl`. This function is synchronous, but it makes HTTP network requests which are asynchronous. To bridge the gap, the `WinHttpGetProxyForUrl` function performs a synchronous wait for the asynchronous work to complete.

And my guess is that `WinHttpGetProxyForUrl` itself uses the thread pool to complete its asynchronous work.

What’s happening is that the program flooded the thread pool with `submit_web_request` work items. Those work items called `WinHttpGetProxyForUrl`, which queues its own work item and waits for it to complete. But those work items can’t run because the thread pool’s threads are all busy handling `submit_web_request` work items.

Eventually, the thread pool may realize that it’s not making progress and spin up a new thread to deal with the work that has been piling up. Maybe that thread will finish the work begun by `WinHttpGetProxyForUrl`, and that will allow one of the `submit_web_request` threads to continue. Once that thread is finished with the `Win-`

`HttpGetProxyForUrl` work item, it will go pull another work item from the queue, and odds are that it's going to get another `submit_ web_ request` work item, so now we're back where we started, except with one more stuck thread in the thread pool.

If the `submit_ web_ request` work items come in faster than `WinHttpGetProxyForUrl` can retire its own work items, the thread pool will fill up with threads blocked inside `submit_ web_ request`, and eventually the thread pool will reach its thread limit, and everything stops.

You're basically starving the thread pool by hijacking it with requests that themselves require the thread pool. All of the thread pool threads are stuck handling your requests, and none are left to do the work that your requests generated.

It's like you have a lot of heavy equipment that you want to move, so you hire every moving company in the city to move them. Company A shows up, and they say, "Hm, this is too big for us to move by ourselves. Let me call Company B, maybe they can help us." Company B says, "Sorry, I can't help you now. I just got an order to move a heavy piece of equipment." By starving out all of the available moving companies, you manage to prevent any of them from completing the job.

I suspect that this system is running in a network environment where WPAD is slow, which makes `WinHttpGetProxyForUrl` 's work item take longer to finish its job, and that makes it more likely that `submit_ web_ request` work items will arrive faster than `WinHttpGet-ProxyForUrl` work items can be retired.

Now that we've diagnosed the problem, what can we do to fix it?

One idea is to hire just one moving company and let them decide how many more moving companies they need. Put all your calls to `submit_ web_ request` on a single thread and retire them one at a time. This clogs up just one thread, leaving the others available to assist. On the other hand, this means that the requests cannot be handled in parallel.

A better fix is to change the way you use the thread pool so you don't keep a thread hostage for a long time.

I'm not an expert on WinHttp, but other people had some ideas on how to do this.

You can switch to `WinHttpGetProxyForUrlEx`, which returns immediately and calls you back when it has an answer. The `submit_ web_ request` function could call `WinHttp-GetProxyForUrlEx` and return immediately. This releases the thread pool thread to do other work—possibly even the work that `WinHttpGetProxyForUrlEx` needs to do in order to complete. When `WinHttpGetProxyForUrlEx` finishes its asynchronous work, it calls the callback, and the callback and do whatever work `submit_ web_ request` was planning on doing after getting the proxy information.

Basically, go asynchronous all the way. It’s not an unreasonable approach for this program, since the `submit_ web_ request` itself models an asynchronous request: It initiates the request and will call some caller-provided callback with the response from the servber. Since it’s already behaving asynchronously, you may as well make it even *more* asynchronous.

Another suggestion was to skip `WinHttpGetProxyForUrl` entirely and just pass the `WIN-HTTP_ ACCESS_ TYPE_ AUTOMATIC_ PROXY` flag to `WinHttpOpen`. This defers the proxy work to the `WinHttpOpen` function, and it can do that as part of its other asynchronous activities. This seems like a good idea because it gets you out of the proxy business entirely, and you still get the asynchronous behavior. It also gives you the satisfaction of fixing a bug by deleting code.

The customer confirmed that switching to the `WIN-HTTP_ ACCESS_ TYPE_ AUTOMATIC_ PROXY` flag fixed the problem.

Raymond Chen

Follow