A question about avoiding page faults the first time newly-allocated memory is accessed

devblogs.microsoft.com/oldnewthing/20170512-00

May 12, 2017

Raymond Chen

A customer had a question about memory allocation.

> When allocating memory with `malloc` and `new`, the memory is not loaded into the physical memory immediately. Instead, the memory is placed in RAM only when the application writes to the memory address. The process of moving pages into physical memory incurs page faults.
>
> In order to avoid page faults, we use `VirtualLock` to lock all allocated memory into physical memory immediately after allocating it. According to MSDN, "Locking pages into memory may degrade the performance of the system by reducing the available RAM and forcing the system to swap out other critical pages to the paging file."
>
> Assume that we have plenty of free RAM on the machine (say, 64GB). Is there any option to configure Windows to load the memory into RAM an allocation time and to swap it into the disk only when it runs out of memory?

The customer says that they have "plenty of free RAM", which I interpret to mean "so much RAM that there will never be any paging." But then the customer says, "when it runs out of memory", which contradicts the previous statement that they have "plenty of RAM."

So let's just ignore the "plenty of RAM" part of the statement. It is confusing and doesn't add to the discussion.

Assuming there is free memory available, the initial page fault will grab a page of free memory and zero it out. This is no slower than grabbing a page of free memory and zeroing it out at allocation time. All you did was change the time the work takes place; the total amount of work remains the same.

In other words, you will gain no performance increase by doing this. Just allocate the memory normally and don't do anything special to "force" it into memory. Forcing it into memory up front means that some other memory may need to be paged out in order to satisfy your immediate demand, and then when that other page is needed, it will need to be paged back in. The system as a whole will run slower than if you waited until the memory was accessed.

And if the page that you prematurely wrote to needs to get evicted before you get around to using it for real, then the system will discard it (assuming it's still all zeroes), and you're back where you started.

But what is the customer's real problem that that makes them think that preloading zero pages is going to help? I had a few theories.

"We think that page faults are bad and are doing everything we can to get rid of them."

The customer is not distinguishing between hard faults and soft faults. Hard faults are bad. Soft faults not so much. The page faults you are eliminating here are soft faults.

"We have carefully profiled our program and have determined that touching at allocation will reduce performance fluctuations during processing, even if it comes at a cost in overall performance. We prefer predictability over throughput."

This is a valid concern.

"We saw the function `VirtualLock` and said, 'Hey, that sounds cool! How do I get a piece of that action?'"

This is a case of finding a hammer and looking for nails.

The customer liaison replied that the customer agreed that the total time is the same, but they want to do it at allocation time in order to avoid taking page faults during processing.

If it is indeed the case that the customer is sophisticated enough to be able to measure the additional cost in terms of elapsed time (as opposed to simply counting page faults because *page faults are bad mmkay?*), then sure, go ahead and touch all the pages at allocation time. There is no built-in setting to do this automatically, though. You'll just have to do it yourself.

It's possible that the customer is sophisticated enough to measure this, but the way they worded the question suggests that they simply didn't understand how the operating system manages resources. Sometimes a very advanced question is hard to distinguish from a very naïve question.

Raymond Chen

Follow