Does Windows have a limit of 2000 threads per process?

devblogs.microsoft.com/oldnewthing/20050729-14

July 29, 2005

Raymond Chen

Often I see people asking why they can't create more than around 2000 threads in a process. The reason is not that there is any particular limit inherent in Windows. Rather, the programmer failed to take into account the amount of address space each thread uses.

A thread consists of some memory in kernel mode (kernel stacks and object management), some memory in user mode (the thread environment block, thread-local storage, that sort of thing), plus its stack. (Or stacks if you're on an Itanium system.)

Usually, the limiting factor is the stack size.

```
#include <stdio.h>
#include <windows.h>

DWORD CALLBACK ThreadProc(void*)
{
 Sleep(INFINITE);
 return 0;
}

int __cdecl main(int argc, const char* argv[])
{
int i;
 for (i = 0; i < 100000; i++) {
  DWORD id;
  HANDLE h = CreateThread(NULL, 0, ThreadProc, NULL, 0, &id);
  if (!h) break;
  CloseHandle(h);
 }
 printf("Created %d threads\n", i);
 return 0;
}
```

This program will typically print a value around 2000 for the number of threads.

Why does it give up at around 2000?

Because the default stack size assigned by the linker is 1MB, and 2000 stacks times 1MB per stack equals around 2GB, which is how much address space is available to user-mode programs.

You can try to squeeze more threads into your process by reducing your stack size, which can be done either by tweaking linker options or manually overriding the stack size passed to the `CreateThread` functions as described in MSDN.

```
HANDLE h = CreateThread(NULL, 4096, ThreadProc, NULL,
             STACK_SIZE_PARAM_IS_A_RESERVATION, &id);
```

With this change, I was able to squeak in around 13000 threads. While that's certainly better than 2000, it's short of the naive expectation of 500,000 threads. (A thread is using 4KB of stack in 2GB address space.) But you're forgetting the other overhead. Address space allocation granularity is 64KB, so each thread's stack occupies 64KB of address space even if only 4KB of it is used. Plus of course you don't have free reign over all 2GB of the address space; there are system DLLs and other things occupying it.

But the real question that is raised whenever somebody asks, "What's the maximum number of threads that a process can create?" is "Why are you creating so many threads that this even becomes an issue?"

The "one thread per client" model is well-known not to scale beyond a dozen clients or so. If you're going to be handling more than that many clients simultaneously, you should move to a model where instead of dedicating a thread to a client, you instead allocate an object. (Someday I'll muse on the duality between threads and objects.) Windows provides I/O completion ports and a thread pool to help you convert from a thread-based model to a work-item-based model.

Note that fibers do **not** help much here, because a fiber has a stack, and it is the address space required by the stack that is the limiting factor nearly all of the time.

Raymond Chen

Follow