The default number of threads in an I/O completion port is the number of processors, but is that logical or physical processors?

devblogs.microsoft.com/oldnewthing/20181123-00

November 23, 2018

Raymond Chen

The `CreateIoCompletionPort` function lets you specify how many concurrent threads can be processing work from the completion port. But if you pass a value of 0 for `NumberOf-ConcurrentThreads`, "the system allows as many concurrently running threads as there are processors in the system."

Are these physical processors or logical processors?

They are logical processors.

One way to figure this out is that the goal of the I/O completion port is to keep CPU usage at exactly 100%. If the I/O completion port consumed only as many threads as physical processors, then you wouldn't quite get to 100% CPU utilization, because there would be extra capacity on the unused logical processors.

Another way to figure this out is to use your understanding of history. I/O completion ports were created long before hyperthreading was invented, so this code treated all logical processors as full-fledged processors.

And a third way to figure it out is to test it.

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

#define THREADS 10

LONG ActiveThreads = 0;

DWORD CALLBACK IoThread(void* Port)
{
 DWORD Bytes;
 ULONG_PTR Key;
 OVERLAPPED* Overlapped;
 while (GetQueuedCompletionStatus(Port, &Bytes,
              &Key, &Overlapped, 1000)) {
  TCHAR msg[64];
  auto count = InterlockedIncrement(&ActiveThreads);
  StringCchPrintf(msg, ARRAYSIZE(msg), TEXT(">%d\r\n"), count);
  OutputDebugString(msg);

  DWORD Tick = GetTickCount();
  while (GetTickCount() - Tick < 1000) { }

  count = InterlockedDecrement(&ActiveThreads);
  StringCchPrintf(msg, ARRAYSIZE(msg), TEXT("<%d\r\n"), count);
  OutputDebugString(msg);
 }
 return 0;
}

int __cdecl main(int, char**)
{
 HANDLE Port = CreateIoCompletionPort(INVALID_HANDLE_VALUE,
                nullptr, 0, 0);

 HANDLE ThreadHandles[THREADS];
 int i;
 for (i = 0; i < THREADS; i++) {
  DWORD Id;
  ThreadHandles[i] = CreateThread(0, 0, IoThread, Port, 0, &Id);
 }

 for (i = 0; i < THREADS * 2; i++) {
  PostQueuedCompletionStatus(Port, 0, 0, nullptr);
 }

 for (i = 0; i < THREADS; i++) {
  WaitForSingleObject(ThreadHandles[i], INFINITE);
 }

 return 0;
}
```

Pick a value for `THREADS` that is greater than the number of logical processors.

We start by creating an I/O completion port and a bunch of threads whose job it is to complete work posted to the port. we then post a lot of work to the port and wait for the threads to drain the work.

Each thread grabs a work item, then increments a counter that lets us know how many threads are actively processing work. The thread then goes into a tight spin loop for one second. It has to do this rather than `Sleep` because the thread needs to be actively doing work for it to be counted against the I/O completion port's concurrency limit.

After wasting some time, the thread decrements the count of active threads, and then goes back to looking for more work.

Along the way, we print the number of active threads.

Run this program, and you'll see that it retires work in chunks, and the number of threads in each chunk is the number of logical processors.

So there, confirmed by experimentation.

Raymond Chen

Follow