What possible use are those extra bits in kernel handles? Part 1: Sentinels

devblogs.microsoft.com/oldnewthing/20080827-00

August 27, 2008

Raymond Chen

Kernel handles are always a multiple of four; the bottom two bits are available for applications to use. But why would an application need those bits anyway?

The short answer is *extending the handle namespace*. The long answer will take a few days to play out. (This series was written in response to Igor Levicki being unable to imagine "how this can save anything (in terms of performance)". Then again, who said that it had anything to do with performance? Actually, I'm surprised that my dear readers weren't familiar with the techniques described in this series. Perhaps I shouldn't have written this series and merely replied, "If you can't think of how this could be useful, then you are not my target audience." On the other hand, reader Aaargh! believes that whoever thought to make the bottom two bits of handles available to applications should receive an asswhooping.)

But we'll start with a warm-up. If you need some sentinel values for a `HANDLE`, you need to make sure your chosen sentinel value will never conflict with a valid `HANDLE` value. If you decide that your sentinel value is something like

```
// code in italics is wrong
#define DEBUGWINDOW_HANDLE ((HANDLE)0x1234)
```

then your program is going to start acting really strange if the kernel ever gave you handle value of 0x1234. Knowing that kernel handles are always multiples of four means that you can choose a value that *isn't* a multiple of four and use it as your sentinel value.

```
#define DEBUGWINDOW_HANDLE ((HANDLE)0x1233)
```

Since 0x1233 is not a multiple of four, you can rest assured that no actual kernel handle will have this value, and you can write your logging function like this:

```
void LogOutput(HANDLE hOutput, LPCVOID pv, DWORD cb)
{
 if (hOutput == NULL) {
   // logging disabled
 } else if (hOutput == DEBUGWINDOW_HANDLE) {
  AddToDebugWindow(pv, cb);
 } else {
  DWORD cbWritten;
  WriteFile(hOutput, pv, cb, NULL, &cbWritten);
 }
}
```

Since you can't `WriteFile` to a window handle, your logging function has to do something special if somebody decided that their output should go to the debug window. But if they chose to log to a normal kernel object (a file, the console, a serial port, whatever) then you can just write the data to that kernel object.

You've already seen this before; you just didn't realize it. The special values for `INVALID_HANDLE_VALUE` and kernel pseudo-handles such as `GetCurrentProcess` are not multiples of four for exactly this reason.

Now, sure, you could have defined your own `LogHandle` type and have all the logging go through that type instead of just logging to `HANDLE`s:

```
class LogHandle {
public:
  static LogHandle *GetDebugLogHandle();
  BOOL IsDebugWindow();
  HANDLE GetKernelHandle();
  static LogHandle *CreateFromKernelHandle(HANDLE KernelHandle);
  ~LogHandle() { }


private:
  LogHandle(BOOL IsDebugWindow, HANDLE KernelHandle);
  static LogHandle DebugWindow;


  BOOL IsLogToDebugWindow;
  HANDLE RegularKernelHandle;
};
```

Throughout, your program would use pointers to `LogHandle`s instead of actual handles, using functions like these to convert between them:

```
// Does not take ownership of the handle
LogHandle::LogHandle(BOOL IsDebugWindow, HANDLE KernelHandle)
    : IsLogToDebugWindow(IsDebugWindow)
    , RegularKernelHandle(KernelHandle)
{
}


LogHandle LogHandle::DebugWindow(TRUE, NULL);


LogHandle* LogHandle::GetDebugWindowLogHandle()
{
  return &DebugWindow;
}


BOOL LogHandle::IsDebugWindow()
{
  return IsLogToDebugWindow;
}


HANDLE LogHandle::GetKernelHandle()
{
  assert(!IsDebugWindow());
  return RegularKernelHandle;
}


LogHandle *LogHandle::CreateFromKernelHandle(HANDLE KernelHandle)
{
  return new LogHandle(FALSE, KernelHandle);
}
```

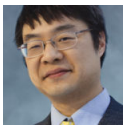
Or you could make everybody pass two parameters instead of one. For example, a class that went

```
class SomeObject {
public:
  SomeObject(int SomeParameter, BOOL SomeParameter,
             HANDLE LogHandle);
…
private:
  …
  HANDLE LogHandle; // log to this handle
};
```

would have to change to

```
class SomeObject {
public:
  SomeObject(int SomeParameter, BOOL SomeParameter,
             BOOL LogToDebugWindow, HANDLE LogHandle);
…
private:
  …
  BOOL LogToDebugWindow; // if TRUE, log to window
  HANDLE LogHandle; // if not logging to window, then log to here
};
```

Either way is an awful lot of work just to define a sentinel value. But still, at least you can avoid the need for a sentinel value by just passing more parameters. But sometimes that option isn't available. We'll look at that next time.



Raymond Chen

Follow