In 16-bit Windows, some per-process data structures were really per-data segment

devblogs.microsoft.com/oldnewthing/20181220-00

December 20, 2018

Raymond Chen

In 16-bit Windows, one of the things you did at the start of your program (or more accurately, which the runtime did for you before calling your `WinMain`) was call the `LocalInit` function. This function initialized your local heap.

When a process started, it was given a data segment, which it could use for whatever purpose it liked. This data segment was the *instance*, and its handle was the *instance handle*, the unit of identification for the process. You could have two copies of the program running, and they would share code, but have different data segments (different instances, different instance handles), so they operated on different sets of memory.

This data segment was up to 64KB in size, and it held your program's stack, heap, and atom table. The operating system set the stack pointer at the end of your instance data segment, so the stack was already taken care of. Setting up the other two, though, are on you.

The `LocalInit` function takes your instance data segment and creates the necessary data structures to make it a heap. This is something the C runtime library startup code did for you. Once that's done, you can call `LocalAlloc` to allocate memory from it.

Once you initialized a data segment as a heap, it was also ready to serve as an atom table.

While it's normally the case that an application uses the local heap and atom table that comes in its instance data segment, it was technically legal to take any zero-initialized data segment and create a heap and atom table in it.

The `LocalInit` function took three parameters: The data segment handle, a pointer to the first byte of memory you want to use as the heap, and a pointer to the last byte of memory you want to use as the heap. So it explicitly lets you pick the data segment and which bytes in that segment you want to turn into a heap.

But the `LocalAlloc` function doesn't have a parameter for the data segment handle. Neither does `AddAtom` . They always operate on the current data segment, the one specified by the `ds` register. During execution of a program, the `ds` register holds that program's instance data segment, so that all references to global variables are relative to that data segment.

The trick is to load your custom data segment into the `ds` register before calling `Local-Alloc`, and then restoring the program's instance data segment after the call returns. Similarly for the other heap and atom functions.

Mind you, there was no way to express this in C, so you had to use either an assembly language wrapper or use inline assembly. That probably contributed to the fact that the trick was little-known and little-used.

But if you knew the trick, you could set up your own local heap and atom table anywhere you liked.

The dialog box manager in 16-bit Windows knew this trick, and if the dialog box contains any edit controls, then it sets up a new data segment to hold the data for those edit controls, so that the total text in the edit controls for the dialog box could go up to 64KB. (A single segment is used for all the edit controls in the dialog box.) There is a style `DS_ LOCALEDIT` which suppresses this behavior, in which case the memory for the edit controls come from the caller's data segment.

Raymond Chen

Follow