A very brief return to part 6 of Loading the Chinese/English dictionary

devblogs.microsoft.com/oldnewthing/20061006-04

October 6, 2006

Raymond Chen

Back in Part 6 of the first phase of the "Chinese/English dictionary" series (a series which I intend to get back to someday but somehow that day never arrives), I left an exercise related to the `alignment` member of the `HEADER` union.

Alignment is one of those issues that people who grew up with a forgiving processor architecture tend to ignore. In this case, the `WCHAR alignment` member ensures that the total size of the `HEADER` union is suitably chosen so that a `WCHAR` can appear immediately after it. Since we're going to put characters immediately after the `HEADER`, we'd better make sure those characters are aligned. If not, then processors that are alignment-sensitive will raise a `STATUS_DATATYPE_MISALIGNMENT` exception, and even processors that are alignment-forgiving will suffer performance penalties when accessing unaligned data.

There are many variations on the alignment trick, some of them more effective than others. A common variation is the one-element-array trick:

```
struct HEADER {
 HEADER* m_phdrPrev;
 SIZE_T  m_cb;
 WCHAR   m_rgwchData[1];
};
// you can also use "offsetof" if you included <stddef.h>
#define HEADER_SIZE FIELD_OFFSET(HEADER, m_rgwchData)
```

We would then use `HEADER_SIZE` instead of `sizeof(HEADER)`. This technique does make it explicit that an array of `WCHAR`s will come after the header, but it means that the code that wants to allocate a `HEADER` needs to be careful to use `HEADER_SIZE` instead of the more natural `sizeof(HEADER)`.

A common mistake is to use this incorrect definition for `HEADER_SIZE`:

```
#define HEADER_SIZE (sizeof(HEADER) - sizeof(WCHAR)) // wrong
```

This incorrect macro inadvertently commits the mistake it is trying to protect against! There might be (and indeed, will almost certainly be in this instance) structure padding after `m_rgwchData`, which this macro fails to take into account. On a 32-bit machine, there will likely be two bytes of padding after the `m_rgwchData` in order to bring the total structure size back to a value that permits another `HEADER` to appear directly after the previous one. In its excitement over dealing with internal padding, the above macro forgot to deal with trail padding!

It is the "array of `HEADER`s" that makes the original `union` trick work. Since the compiler has to be prepared for the possibility of allocating an array of `HEADER`s, it must provide padding at the end of the `HEADER` to ensure that the next `HEADER` begins at a suitably-aligned boundary. Yes, the `union` trick can result in "excess padding", since the type used for alignment may have less stringent alignment requirements than the other members of the aggregate, but better to have too much than too little.

Another minor point was brought up by commenter Dan McCarty: "Why is `MIN_CBCHUNK` set to 32,000 instead of 32K?" Notice that `MIN_CBCHUNK` is added to `sizeof(HEADER)` before it is rounded up. If the allocation granularity were 32768, then rounding up the sum to the nearest multiple would have taken us to 65536. Nothing wrong with that, but it means that our minimum chunk size is twice as big as the `#define` suggests. (Of course, since in practice the allocation granularity is 64KB, this distinction is only theoretical right now.)

Raymond Chen

Follow