Loading the dictionary, part 5: Avoiding string copying

devblogs.microsoft.com/oldnewthing/20050518-42

May 18, 2005

Raymond Chen

Looking at the profile for our program so far, 35% of the CPU time is spent copying strings around. Let's see if we can improve that.

The best way to speed up copying strings is not to copy them in the first place. Using a `wstring` in our `DictionaryEntry` structure forces the `vector` class to copy the string data, when all we really need to copy is the pointer and size information. The actual characters can stay put. C++ has a copy constructor but not a "move" constructor.

Let's use plain string pointers rather than `wstring` objects. The "copy constructor" for a string pointer is just to copy the pointer—exactly what we want here.

```
struct DictionaryEntry
{
 bool Parse(const WCHAR* begin, const WCHAR* end);
 void Destruct() {
  delete[] m_pszTrad;
  delete[] m_pszSimp;
  delete[] m_pszPinyin;
  delete[] m_pszEnglish;
 }
 LPWSTR m_pszTrad;
 LPWSTR m_pszSimp;
 LPWSTR m_pszPinyin;
 LPWSTR m_pszEnglish;
};
```

The `DictionaryEntry` is no longer a structure of `wstring`s but rather is just a structure of `LPWSTR`s, which copy much faster. The cost for this, however, is having to free all the strings manually in the dictionary destructor (which we will see below).

Since we aren't using `wstring`s any more, we need to allocate the memory for the strings and copy them the old fashioned way.

```
LPWSTR AllocString(const WCHAR* begin, const WCHAR* end)
{
 int cch = end - begin + 1;
 LPWSTR psz = new WCHAR[cch];
 lstrcpynW(psz, begin, cch);
 return psz;
}
bool DictionaryEntry::Parse(
       const WCHAR* begin, const WCHAR* end)
{
 const WCHAR* pch = std::find(begin, end, L' ');
 if (pch >= end) return false;
 m_pszTrad = AllocString(begin, pch);
 begin = std::find(pch, end, L'[') + 1;
 if (begin >= end) return false;
 pch = std::find(begin, end, L']');
 if (pch >= end) return false;
 m_pszPinyin = AllocString(begin, pch);
 begin = std::find(pch, end, L'/') + 1;
 if (begin >= end) return false;
 for (pch = end; *--pch != L'/'; ) { }
 if (begin >= pch) return false;
 m_pszEnglish = AllocString(begin, pch);
 return true;
}
```

There isn't a `std::rfind` function, so I coded up a backwards-search-for-slash loop inline. **Exercise**: Why don't I have to check that `pch` hasn't underflowed beyond the beginning of the string?

```
class Dictionary
{
public:
 Dictionary();
 ~Dictionary();
 int Length() { return v.size(); }
 const DictionaryEntry& Item(int i) { return v[i]; }
private:
 vector<DictionaryEntry> v;
};
Dictionary::Dictionary()
{
 ...
   if (cchResult){
    ~~// wstring line(buf, cchResult);~~
    DictionaryEntry de;
    if (de.Parse(buf, buf + cchResult)) {
     ...
}
Dictionary::~Dictionary()
{
 for (vector<DictionaryEntry>::iterator i = v.begin();
      i != v.end(); i++) {
  i->Destruct();
 }
}
```

The last bits of the change are to get rid of the temporary `wstring` we parsed from, since we don't need it any more, and to free all the strings in the `Dictionary` 's destructor.

This new program clocks in at 120ms (or 290ms if you include the time it takes to destroy the dictionary). Again, we're twice as fast as the previous version, but still haven't quite crossed the 100ms barrier. And the time to destroy the dictionary isn't starting to be a more significant factor. But I have another trick up my sleeve.

[Raymond is currently on vacation; this message was pre-recorded.]

Raymond Chen

Follow