Loading the dictionary, part 3: Breaking the text into lines

devblogs.microsoft.com/oldnewthing/20050513-26

May 13, 2005

Raymond Chen

Even after moving the character conversion out of the `getline` function, profiling reveals that `getline` is still taking nearly 50% of our CPU. The fastest code is code that isn't there, so let's get rid of `getline` altogether. Oh wait, we still need to break the file into lines. But maybe we can break the file into lines faster than `getline` did.

```
#include <algorithm>
class MappedTextFile
{
public:
 MappedTextFile(LPCTSTR pszFile);
 ~MappedTextFile();
 const CHAR *Buffer() { return m_p; }
 DWORD Length() const { return m_cb; }
private:
 PCHAR   m_p;
 DWORD   m_cb;
 HANDLE  m_hf;
 HANDLE  m_hfm;
};
MappedTextFile::MappedTextFile(LPCTSTR pszFile)
    : m_hfm(NULL), m_p(NULL), m_cb(0)
{
 m_hf = CreateFile(pszFile, GENERIC_READ, FILE_SHARE_READ,
                   NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
 if (m_hf != INVALID_HANDLE_VALUE) {
  DWORD cb = GetFileSize(m_hf, NULL);
  m_hfm = CreateFileMapping(m_hf, NULL, PAGE_READONLY, 0, 0, NULL);
  if (m_hfm != NULL) {
   m_p = reinterpret_cast<PCHAR>
                (MapViewOfFile(m_hfm, FILE_MAP_READ, 0, 0, cb));
   if (m_p) {
    m_cb = cb;
   }
  }
 }
}
MappedTextFile::~MappedTextFile()
{
 if (m_p) UnmapViewOfFile(m_p);
 if (m_hfm) CloseHandle(m_hfm);
 if (m_hf != INVALID_HANDLE_VALUE) CloseHandle(m_hf);
}
```

This very simple class babysits a read-only memory-mapped file. (Yes, there is a bit of oddness with files greater than 4GB, but let's ignore that for now, since it's a distraction from our main point.)

Now that the file is memory-mapped, we can just scan it directly.

```
Dictionary::Dictionary()
{
 MappedTextFile mtf(TEXT("cedict.b5"));
 typedef std::codecvt<wchar_t, char, mbstate_t> widecvt;
 std::locale l(".950");
 const widecvt& cvt = _USE(l, widecvt); // use_facet<widecvt>(l);
 const CHAR* pchBuf = mtf.Buffer();
 const CHAR* pchEnd = pchBuf + mtf.Length();
 while (pchBuf < pchEnd) {
  const CHAR* pchEOL = std::find(pchBuf, pchEnd, '\n');
  if (*pchBuf != '#') {
   size_t cchBuf = pchEOL - pchBuf;
   wchar_t* buf = new wchar_t[cchBuf];
   mbstate_t state = 0;
   char* nextsrc;
   wchar_t* nextto;
   if (cvt.in(state, pchBuf, pchEOL, nextsrc,
                  buf, buf + cchBuf, nextto) == widecvt::ok) {
    wstring line(buf, nextto - buf);
    DictionaryEntry de;
    if (de.Parse(line)) {
     v.push_back(de);
    }
   }
   delete[] buf;
  }
  pchBuf = pchEOL + 1;
 }
}
```

We simply scan the memory-mapped file for a `'\n'` character, which tells us where the line ends. This tells us the location and length of the line, which we use to convert it to Unicode and continue our parsing.

Exercise:Why don't we have to worry about the carriage return that comes before the linefeed?

Exercise:Why don't we have to worry about possibly reading past the end of the file when we check `*pchBuf != '#'`?

With this change, the program now loads the dictionary in 480ms (or 550ms if you include the time it takes to destroy the dictionary). That's over twice as fast as the previous version.

But it's still not fast enough. A half-second delay between hitting `Enter` and getting the visual feedback is still unsatisfying. We can do better.

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

Raymond Chen

Follow