Changing the conditions under which ReadFile produces fewer bytes than requested

devblogs.microsoft.com/oldnewthing/20151104-00

November 4, 2015

Raymond Chen

In response to an article on hierarchical storage management, Karellen suggests that the problem could be ameliorated by having the hierarchical storage manager keep the first 4KB of the file online, thereby allowing programs that sniff the start of the file for metadata to continue operating without triggering a recall. "The way that file read operations tend to work (fread, read, and ReadFile), if an application opens a file and requests a large read, just returning the first 4KB is a valid response."

Premature short reads may technically be a valid response, but it won't be the correct response.

When your program reads from a file, do you retry partial reads? Be honest.

Suppose you want to read a 32-bit value from a file. You probably write this.

```
uint32_t value;

DWORD bytesRead;
if (ReadFile(file, &value, sizeof(value),
             &bytesRead, nullptr) &&
             bytesRead == sizeof(value)) {
  // Got the value - use it...
}
```

You probably don't write this:

```
uint32_t value;
BYTE *nextRead = reinterpret_cast<BYTE*>&value;
DWORD bytesRemaining = sizeof(value);
while (bytesRemaining) {
  DWORD bytesRead;
  if (!ReadFile(file, &value, bytesRemaining,
                &bytesRead, nullptr)) return false;
  if (bytesRead == 0) break; // avoid infinite loop
  bytesRemaining -= bytesRead;
  nextRead += bytesRead;
}

if (bytesRemaining == 0) {
 // Got the value - use it...
}
```

Most programs assume that a short read from a disk file indicates that the end of the file has been reached, or some other error has occurred. Consider, for example, this file parser:

```
struct CONTOSOFILEHEADER
{
  uint32_t magic;
  uint32_t version;
};

bool IsContosoFile(HANDLE file)
{
 CONTOSOFILEHEADER header;
 DWORD bytesRead;
 if (!ReadFile(file, &header, sizeof(header),
               &bytesRead, nullptr)) {
   // Couldn't read the file - assume not a Contoso file.
   return false;
 }

 if (bytesRead != sizeof(header)) {
   // File doesn't hold a header - not a Contoso file.
   return false;
 }

 if (header.magic != CONTOSO_MAGIC) {
  // Does not start with magic number - not a Contoso file.
  return false;
 }

 if (header.version != CONTOSO_VERSION_1 &&
     header.version != CONTOSO_VERSION_2) {
  // Unsupported version - not a Contoso file.
  return false;
 }

 // Passed basic tests.
 return true;
}
```

The problem is even worse if you use `fread`, because `fread` does not provide information on how to resume a partial read. It reports only the total number of items read in full; you get no information about how much progress was made in the items that were read only in part.

```
 // Read 10 32-bit integers.
 uint32_t flags[10];
 auto itemsRead = fread(flags, sizeof(uint32_t), 10, fp);
 if (itemsRead < 10) {
   if (!feof(fp) && !ferror(fp)) {
     // At this point, we have a short read.
     // We are now screwed.
   }
 }
```

Since nobody is actually prepared for a short read to occur on disk files anywhere other than the end of the file, you shouldn't introduce a new failure mode that nobody can handle.

Because they won't handle it.

And recall that the original question was in the context of displaying a file in a folder. Even if you know that Hierarchical Storage Management is not involved, you still have to deal with the cost of opening the file at all. If the folder is on a remote server where each I/O operation has 500ms of latency, then enumerating the contents of a directory with 1000 files will take over eight minutes. I suspect the user will have lost patience by then.

Raymond Chen

Follow