

Re: Data loss appending data to file

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2006-01/msg00026.htm>

- *From:* Yannick Fortin <yannick.fortin@xxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 03 Jan 2006 11:13:53 -0500
-

I found it.

This is the piece of code where the problem is. To put you in context, the app reads a text file one character at the time in binary mode and "emulates" the behavior of various line printers before storing the data of a printer page in a string array. This allows us to emulate, for example, the "bold" of a line printer where the same line would be repeated twice with a carriage return but no line feed at the end of the first of the two lines.

The app is written in Delphi, where buffered binary file access is not available in the standard libraries file APIs. So we have written our own using Windows APIs. Here is the culprit code (translated in C++):

```
void TInputFile::ResetFlags()
{
    if(FOpened)
        SetFilePointer(mu_fileHandle, 0, NULL, FILE_BEGIN);

    // snip: reset all internal variables...
}

void TInputFile::Close()
{
    if(!FOpened)
        return;

    Win32Check(CloseHandle(mu_fileHandle), "Error closing handle");
    ResetFlags();
    FOpened = False; // *** this line is much younger than the rest
end;
```

Re: Data loss appending data to file

The code is verbatim; as you can see, the return value of `SetFilePointer()` is not checked.

You also have to know that the file handle returned by `CreateFile` is a small integer (e.g. 1924). This indicates that it's not a pointer to a structure in memory but an entry in a system table, or something like that. But the point to note is: they get reused.

Armed with all this knowledge, we can now see what happens. Thread 1 closes the handle to its file. The context switch occurs before `ResetFlags()` is called. Thread 2 opens a file; the handle it receives happens to be the same as what Thread 1 had. Context switch back to Thread 1 which now executes the `SetFilePointer()` with the handle that now belong to Thread 2.

I found the error by replacing all calls to the Windows API by library file access. An exception was promptly raised in single-threaded mode when it tried to seek in a closed file. If the `SetFilePointer` had its return value checked, we would have found it immediately. It's the only API in the entire lib that's not checked for success. Damn.

The fix: move `"FOpened = false;"` up one line.

Thank you all for your help in this matter. I wished the problem was a little more... glorious, but hey, at least now it works.

Yannick

.