What is the effect of memory-mapped file access on GetLastError()?

devblogs.microsoft.com/oldnewthing/20120209-00

February 9, 2012

Raymond Chen

A customer was using memory-mapped files and was looking for information as to whether access to the memory-mapped data modifies the value returned by `GetLastError`. A member of the kernel team replied, "No, memory-mapped I/O does not ever change the value returned by `GetLastError`."

That answer is simultaneously correct and wrong, a case of looking at the world through kernel-colored glasses.

While it's true that the kernel does not ever change the value returned by `GetLastError`, it's also the case that *you* might change it.

If you set up an exception handler, then your exception handler might perform operations that affect the last-error code, and those changes will be visible after the exception handler returns. (This applies to all exception handlers and filters, not just ones related to memory-mapped files.)

If you intend to return `EXCEPTION_CONTINUE_EXECUTION` because you handled the exception, then you probably should make sure to leave the last-error code the way you found it. Otherwise, the code that you interrupted and then resumed will have had its last-error code changed asynchronously. You just sabotaged it from above.

```
// Code in italics is wrong
LONG ExceptionFilter(LPEXCEPTION_POINTERS ExceptionPointers)
{
 if (IsAnExceptionICanRepair(ExceptionPointers)) {
   RepairException(ExceptionPointers);
   // fixed up error; continuing
   return EXCEPTION_CONTINUE_EXECUTION;
 }
 if (IsAnExceptionICanHandle(ExceptionPointers)) {
  // We cannot repair it, but we can handle it.
  return EXCEPTION_EXECUTE_HANDLER;
 }
 // Not our exception. Keep looking.
 return EXCEPTION_CONTINUE_SEARCH;
}
```

If the `IsAnExceptionICanRepair` function or `RepairException` function does anything that affects the last-error code, then when the exception filter is executed for a repairable exception, the last-error code is magically changed without the mainline code's knowledge. All the mainline code did was execute stuff normally, and somehow during a memory access or a floating point operation or some other seemingly-harmless action, the last-error code spontaneously changed!

If you are going to continue execution at the point the exception was raised, then you need to "put things back the way you found them" (except of course for the part where you repair the exception itself).

```
LONG ExceptionFilter(LPEXCEPTION_POINTERS ExceptionPointers)
{
 PreserveLastError preserveLastError;
 if (IsAnExceptionICanRepair(ExceptionPointers)) {
   RepairException(ExceptionPointers);
   // fixed up error; continuing
   return EXCEPTION_CONTINUE_EXECUTION;
 }
 if (IsAnExceptionICanHandle(ExceptionPointers)) {
  // We cannot repair it, but we can handle it.
  return EXCEPTION_EXECUTE_HANDLER;
 }
 // Not our exception. Keep looking.
 return EXCEPTION_CONTINUE_SEARCH;
}
```

Exercise: Why isn't it important to restore the last error code if you return `EXCEPTION_EXECUTE_HANDLER`?

Exercise: Is it important to restore the last error code if you return `EXCEPTION_CONTINUE_SEARCH`?

Raymond Chen

Follow