How do I prevent a child process from displaying the Windows Error Reporting dialog?

devblogs.microsoft.com/oldnewthing/20160204-00

February 4, 2016

Raymond Chen

A customer wanted to know if there was a way to disable Windows Error Reporting for a child process. Their scenario was that they had a main program which runs a child program that is expected to crash (because it's a unit test). Normally, this would result in the Windows Error Report dialog appearing, offering to send a crash report to Microsoft. This dialog is unwanted, since it makes it difficult to run the unit test as part of an automated script, plus there's no point sending crash reports for a unit test that is expected to crash.

One person suggested using the `WerAddExcludedApplication` function to add the program to the list of programs to be excluded from error reporting, and then call the `WerRemove-ExcludedApplication` function when the test completes. There are some problems with this approach:

- The name of your unit test application may happen to match that of a legitimate application, and the `WerAddExcludedApplication` function takes only the file name, not a full path. You can try to arrange for your unit test name to be sufficiently unique that this sort of collision is unlikely.
- If the test harness crashes, then `WerRemoveExcludedApplication` will never get called, and then the unit test ends up excluded permanently.
- Suppose two copies of the unit test are running. Given that you're doing automated testing, there's a decent chance that this will happen; for example, the automated system may be running unit tests as part of a gated check-in system, and multiple check-ins could be undergoing validation at once. In that case, both test harnesses will call `WerAddExcludedApplication` when they start up, and the first one to finish will call `WerRemoveExcludedApplication`. This rips the registration out from under the second instance, which is still running, and the crash dialog will appear.

Basically, we are applying a global solution to a local problem.

Better to apply a local solution: Since the error mode is inherited by child processes, you can have the test harness call `SetErrorMode` with the `SEM_NOGPFAULTERRORBOX` flag to disable the error reporting dialog for itself and its children.

Here's a Little Program to illustrate. Remember that Little Programs do little to no error checking. In this case, the Little Program also leaks handles!

```
#include <windows.h>
#include <stdio.h>

int main(int argc, char**argv)
{
    printf("Error mode is %d\n", GetErrorMode());
    if (argc == 1) {
        TCHAR buf[256];
        GetModuleFileName(nullptr, buf, 256);
        STARTUPINFO si = { sizeof(si) };
        PROCESS_INFORMATION pi;
        SetErrorMode(SEM_NOGPFAULTERRORBOX);
        printf("Spwaning child, error mode is %d\n", GetErrorMode());
        CreateProcess(buf, TEXT("a b c d"), nullptr, nullptr, FALSE,
            0, nullptr, nullptr, &si, &pi);
        WaitForSingleObject(pi.hProcess, INFINITE);
        printf("Done\n");
    } else {
        DebugBreak();
    }
    return 0;
}
```

If you run this program, it will start by reporting that its error mode is zero, and then it changes to 2 (which is the value of the `SEM_NOGPFAULTERRORBOX` flag), and then the child process will report that its error mode is 2 (inherited), and then it will crash by invoking a nonexistent debugger. Since error dialogs are disabled, the child process will exit silently.

Raymond Chen

Follow