データブレークポイント

*このサンプルは、2015 年 11 月の Xbox One XDK 以降と互換性があります。*

# 概要

このサンプルでは、さまざまな種類のメモリアクセスを検出するのに役立つ、ハードウェアデータ ブレークポイントを作成する方法を示します。これらはプロセッサコアによって完全に処理されます。つまり、実行速度には影響しません。それらを使用することの主な欠点は、プロセッサコアで利用可能な4つのスロットしかないということです。

Windows は、スレッドへのローカルブレークポイントのみを許可します。つまり、問題のスレッドが実行されている間だけブレークポイントがアクティブになります。ハードウェアブレークポイントはスレッドにバインドされているため、さまざまなコアで実行されるときにスレッドに従います。必要に応じて、各スレッドに同じブレークポイントを設定できます。

# 目次

## DataBreak.cpp/h

* ハードウェアデータブレークポイントを設定および解除する機能を提供するスタンドアロンパッケージ。
* SetHardwareBreakPointForThread
  + 問題のスロットとアドレスを指定してブレークポイントを設定します。
  + スロットがすでに使用されている場合は上書きされます。
* ClearHardwareBreakPointForThread
  + 指定されたスロットのブレークポイントを解除します

# 実装上の注意

ハードウェアブレークポイントを設定するには、デバッグレジスタの内容を調整する必要があります。主な問題は、これらのレジスタへのアクセスがカーネルにしか利用できないということです。デバッグレジスタを設定するためのトリックは、スレッドコンテキストを調整することです。これにより、コンテキストの切り替え中にスケジューラがデバッグレジスタに内容を書き込むようになります。

スレッドのコンテキストは、スレッドが中断されている場合にのみ変更できます。このため、ターゲットスレッドが現在のスレッドである場合は、操作を実行するために別のスレッドが必要です。サンプルは、コンテキスト切り替えを実行するための一時的なワーカースレッドを作成します。

スローされた例外は、どちらかの構造化例外処理 **(\_\_try**, **\_\_except**)でもしくは、未処理の例外フィルタを介してキャッチすることができます。ただし、キャプチャ例外レコードは、**MiniDumpWriteDump**と共に使用されている場合は無視されます。 しかし、 **MiniDumpWriteDump** は問題のあるコードがコールスタックに残っているという例外を発生させたスレッドで呼び出されています。未処理の例外フィルタは、問題のあるスレッドのコンテキストで呼び出されます。

## 重要な注記：

このシステムによってスローされる例外は、OSによって特別に扱われるシングルステップ例外です。デバッガーが添付されている場合、デバッガーは最初にキャッチします。デフォルトでは、Visual Studio はそれを無視し、コードをシングルステップしない限り、制御はタイトル例外ハンドラに渡されます。デフォルトでは、KD は例外を引き起こしたコードの行で改行します。デバッガが接続されていない場合は、 **EnableKernelDebugging** が動作を定義します。有効になっていると、ERA はフリーズします。コンソールはデバッガが接続されるのを待っています。もし **EnableKernelDebugging** が有効になっていない場合、タイトル例外ハンドラが呼び出されます。

DataBreak.cpp における **DataBreakThread** 機能は、デバッグレジスタに必要な内容を文書化しています。

# 更新履歴

初回リリース 2016 年 10 月