

データ ブレークポイント

*このサンプルは Microsoft Game Development Kit (2020 年 6 月) と互換性があります。*

# 説明

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ために便利なハードウェアのデータの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。それらを使用するに当たっての主な欠点は、プロセッサ コアで使用可能なスロットが 4 つしかないことです。

Windows では、スレッドへのローカル ブレークポイントのみが許可されます。つまり、ブレークポイントは、対象のスレッドの実行中にのみ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るため、さまざまなコアで実行される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クリアする機能を提供するスタンドアロン パッケージ。
* SetHardwareBreakPointForThread
  + 対象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。
  + スロットが既に使用されている場合は上書きされます。
* ClearHardwareBreakPointForThread
  + 指定したスロットのブレークポイントをクリアします

# 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はカーネルでのみ利用可能であることです。デバッグ レジスタを設定する方法は、スレッド コンテキストを調整することです。これにより、スケジューラはコンテキスト切り替え中にデバッグ レジスタに内容を書き込みます。

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。このサンプルでは、コンテキスト切り替えを実行する一時ワーカー スレッドを作成します。

スローされた例外は、構造化例外処理 **(\_\_try**、 **\_\_except**) またはハンドルされない例外フィルターのいずれかを使用してキャッチできます。ただし、**MiniDumpWriteDump** で使用されているキャプチャ例外レコードは無視されます。ただし、例外が発生したスレッドで **MiniDumpWriteDump** が呼び出されている場合、問題のあるコードは、さらに上のコール スタックに残ります。ハンドルされない例外フィルターは、問題のあるスレッドのコンテキストで呼び出されます。

## 重要事項:

このシステムによってスローされる例外は、OS によって特別に処理されるシングル ステップ例外です。デバッガーがアタッチされている場合、デバッガーが最初にキャッチします。既定では、 Visual Studio はそれを無視し、コードをシングル ステップ実行しない限り、コントロールはタイトルの例外ハンドラーに渡されます。既定では、KD は例外の原因となっているコード行で中断します。ただし、デバッガーがアタッチされていない場合は、 **EnableKernelDebugging** が動作を定義します。有効になっている場合、タイトルはフリーズします。コンソールはデバッガーのアタッチを待機しています。**EnableKernelDebugging** が有効になっていない場合、タイトルの例外ハンドラーが呼び出されます。

DataBreak.cpp の **DataBreakThread** 関数は、デバッグ レジスタに必要な内容を記録します。

# 更新履歴

2020 年 8 月の初回リリース

# プライバシー ステートメント

サンプルをコンパイルして実行する場合、サンプルの使用状況を追跡するために、サンプル実行ファイルのファイル名が Microsoft に送信されます。このデータ コレクションからオプトアウトするには、Main.cpp の「Sample Usage Telemetry」というラベルの付いたコードのブロックを削除します。

全般的な Microsoft のプライバシー ポリシーの詳細については、「[Microsoft プライバシー ステートメント](https://privacy.microsoft.com/en-us/privacystatement/)」を参照してください。