

数据断点

*此示例兼容于 Microsoft 游戏开发工具包（2020 年 6 月）*

# 说明

此示例演示了如何创建可用于检测不同类型的内存访问的硬件数据断点。它们完全由处理器内核处理，这意味着它们不会影响执行速度。它们的主要缺点是处理器内核上只有 4 个可用槽。

Windows 仅允许线程上有本地断点。这意味着只有在执行有问题的线程时，断点才会处于活动状态。由于硬件断点绑定到线程上，因此线程在各个内核上执行时，它们会跟随该线程。如有需要，可以使用相同的断点设置每个线程。

# 目录

## DataBreak.cpp/h

* 提供设置和清除硬件数据断点功能的独立包。
* SetHardwareBreakPointForThread
  + 根据有问题的槽和地址设置断点。
  + 如果已使用该槽，则会覆盖该槽。
* ClearHardwareBreakPointForThread
  + 清除指定槽上的断点

# 使用说明

若要设置硬件断点，需要调整调试寄存器的内容。主要问题是只能通过内核访问这些寄存器。设置调试寄存器的技巧是调整线程上下文。这将导致计划员在上下文切换期间将内容写入调试寄存器。

仅当线程挂起时，才能更改线程的上下文。因此，如果目标线程是当前线程，则需要另一个线程来执行该操作。此示例将创建一个临时工作线程来执行上下文切换。

可以通过结构化异常处理 **(\_\_try**, **\_\_except**) 或未经处理的异常筛选器来捕获引发的异常。但是，如果捕获异常记录与 **MiniDumpWriteDump** 一起使用，则系统会忽略该记录。但是，如果在触发异常的线程上调用 **MiniDumpWriteDump**，则有问题的代码仍将位于调用堆栈中，只是位置会上移。系统会在有问题线程的上下文中调用未处理的异常筛选器。

## 重要提示：

此系统引发的异常是操作系统特别处理的单步异常。如果连接了调试程序，该程序会首先捕获该异常。默认情况下，Visual Studio 会忽略它，且控件会将异常处理程序传递给相应游戏，除非你在单步执行代码。默认情况下，内核调试程序将在引发异常的代码处中断。如果未连接调试程序，则 **EnableKernelDebugging** 会定义相应行为。如果该函数已启用，则游戏将被冻结。主机会等待连接调试程序。如果未启用 **EnableKernelDebugging**，则会调用游戏异常处理程序。

DataBreak.cpp 中的 **DataBreakThread** 函数会记录调试寄存器所需的内容。

# 更新历史记录

初始版本 - 2020 年 8 月

# 隐私声明

在编译和运行示例时，示例可执行文件的文件名将发送给 Microsoft，以帮助跟踪示例使用情况。若要选择退出此数据收集，你可以删除 Main.cpp 中标记为“示例使用遥测”的代码块。

有关 Microsoft 的一般隐私策略的详细信息，请参阅 [Microsoft 隐私声明](https://privacy.microsoft.com/en-us/privacystatement/)。