技术交底书

**基本信息表**

|  |  |
| --- | --- |
| 发明名称 | 支持跨内核态和用户态的操作系统源代码级调试方法 |
| 申报类型 | 发明专利 |
| 发明人 | 吴竞邦，张露元，陈志扬 |
| 技术联系人 | 吴竞邦 |
| 联系人电话及邮箱 | 18101260830，wujingbang@btbu.edu.cn |

# 一、背景技术

1、描述与本方案最接近的现有技术、现有专利或常用手段等（别人是怎么做的）

操作系统（OS）是驱动硬件运行的核心系统，其在硬件和用户之间起到了桥梁的作用。在我国大力发展自主中央处理器（CPU）与自主操作系统的大背景下，包括基于C语言以及基于RUST语言的操作系统内核代码调试一直是开发人员面临的技术难点。不同于通用应用程序代码调试，操作系统源代码运行状态多、工作逻辑复杂，使得传统的调试方法难以适应其开发调试工作。方便的源代码级调试工具，对监测程序运行状态和理解程序的逻辑十分重要，尤其是相对复杂的内核代码以及用户态、内核态的系统调用交互。

操作系统运行状态一般分为用户态和内核态，属于两个不同的特权级，拥有不同的执行权限。当应用程序需要执行一些特权操作或与操作系统交互时，它需要通过系统调用的方式从用户态切换到内核态。特权级的应用提供了应用程序与核心系统之间的隔离，可以有效防止应用程序对系统造成损害，并优化系统的资源利用。但另一方面，特权级的切换会清除调试状态信息，使得传统程序调试方法失效。除此以外，操作系统需要一个运行在一个指定的硬件环境中，所以调试方式和普通应用程序的调试有所不同。QEMU（Quick Emulator）是一个完全由软件实现的开源虚拟机，它可以提供操作系统运行所需的硬件条件，操作系统运行在QEMU模拟器中，操作系统调试器运行在宿主机中，使用这种方法可以让操作系统的开发与调试变得更加便捷。

调试器的主要功能是能够让使用者控制被调试软件的运行，并提供一系列工具实时查看被调试软件运行过程中的中间状态，为使用者定位程序问题以及修改代码提供帮助，是相关软件开发中不可缺少的一种工具。在GDB的基础上，有许多继承调试环境如VSCode、Visual Studio、Eclipse、XCode等，提供应用程序的代码开发和调试功能。

2、总结现有技术的缺点或不足（我为什么还要做）

上述这些调试器对操作系统源代码的调试支持不够完善，其中一个主要问题在于操作系统不同运行状态下程序符号表的切换导致断点等调试信息丢失，从而使得调试器失效。

首先，操作系统中包括用户态代码和内核态代码，对应不同的特权级，不同的特权级又对应不同的符号表。在操作系统运行的过程中，会频繁地进行特权级的切换，导致调试信息的丢失，而目前现有的调试器都无法进行跨特权级的断点调试。具体来说，特权级的切换主要涉及符号表的切换，符号表包含了编译后的代码中各种变量、函数、数据结构等的名称和地址信息，这些是调试代码所必需的内容，同时符号表也是编译后的代码与源代码之间的桥梁，使调试器能够将二进制代码中的地址映射回源代码的符号名，所以无论调试任何进程，调试器都需要先加载进程的符号表。而在操作系统中内核态程序和用户态程序的符号表是分开的，如果程序运行中进行了用户态和内核态的转换，符号表也要随之切换，符号表切换以后，用户设置的程序断点也会随之消失，比如在内核态设置用户态的断点以后，再进入用户态，用户态的断点将不会被触发。

其次，用户态中的多个用户进程分别对应不同的符号表，在CPU对用户进程进行调度切换运行状态时同样存在上述的调试信息失效的问题。如何解决操作系统中多符号表切换导致的调试信息失效是本专利解决的关键问题。

# 二、技术方案

1、本申请的关键点

*本申请主要的改进点是什么，并解释技术原理，即为什么能解决上面的技术缺陷。如果有多个改进点，请逐一列出。*

关键点包括

1. 跨特权级源代码断点设置

由于操作系统中内核态代码和用户态代码的符号表是分开，在操作系统切换特权级的时候，符号表也会切换，使得断点等调试信息失效，因此同时设置内核态、用户态的断点会产生冲突，无法跨越特权设置断点。为了解决这个问题，我们新增了一个断点组管理模块。断点组管理模块会先缓存设置的异常断点（例如当前操作系统处在内核态代码的运行中，操作人员在用户态代码中设置了断点，该断点即为异常断点），等到特权级切换到对应的状态时，再将缓存的断点信息进行激活，即在用户态运行时，缓存用户设置的内核态断点，等到程序运行到内核态时，再激活缓存的内核态断点；在内核态运行时，缓存用户设置的用户态断点，等运行到用户态时，再激活用户态断点。通过这种方法，操作人员可以在任意状态下对被调试操作系统的任意代码行设置断点，从而实现跨特权级的源代码调试。

1. 多个用户进程符号表切换

操作系统通常允许在用户态创建多个进程对应不同的工作，即用户进程，多个用户进程会抢占有限的中央处理器运行资源，即调度执行或用户进程切换。类似于用户态与内核态，每个用户进程都有自己的符号表，每次进行用户进程切换的时候都需要通过系统调用进入内核态，更新内存地址空间，然后返回用户态执行新的进程。我们设计了通过进程标识的获取对应不同的符号表，并在调试器检测到用户程序切换时对符号表进行切换，以此实现多用户进程的调试。

2、本申请的具体的实施例（我是怎么做的？）

*结合附图说明本申请的技术方案，必要时可加入表格、数据、效果图。电路结构的改进请提供电路图；机械结构的改进，请提供结构图；方法改进，请提供流程图；如果方便请提供可编辑的附图（包括CAD、VISIO等）。*

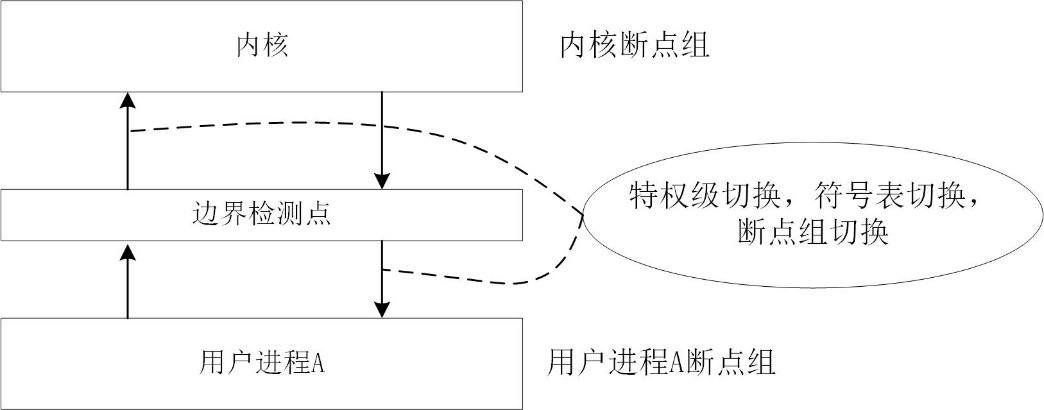
方案设计

1. 解决跨特权级源代码断点设置问题

设置断点组数据结构，用于保存因为特权级切换失效的断点调试信息：使用一个词典缓存了用户要求设置的所有断点（包括内核态和用户态）。词典中的每个元素都是一个键值对，其中键是程序运行所占内存地址空间的代号，值是该代号对应的断点组，包括用户态断点组和内核态断点组，分别保存了用户态和内核态对应内存地址空间内的所有断点。除此以外，为了满足调试需求，设置了一个当前有效断点组变量，即被调试操作系统当前执行的进程地址空间对应的断点组，只有当前有效断点组中的断点才会被激活，随后会被触发，不是当前有效断点组的断点只会被缓存到其他地址空间的断点组中，不会被触发。

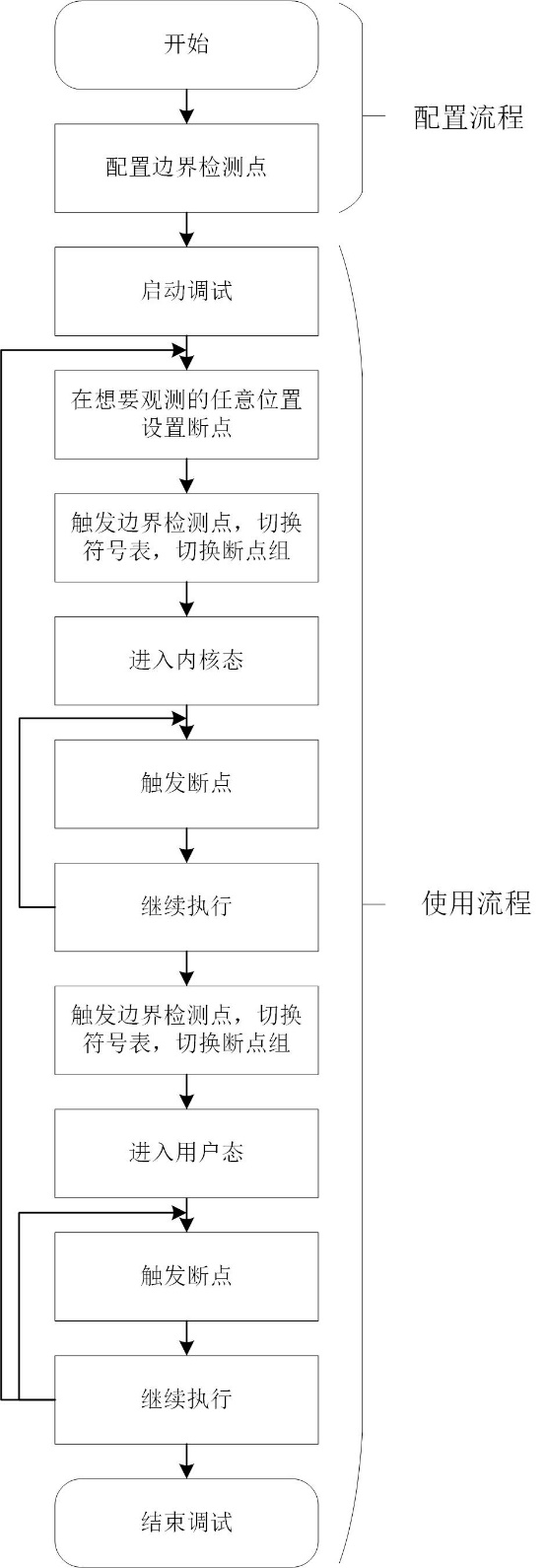
当用户在调试器中设置新断点时，调试器的断点组管理模块会先将这个断点的信息存储在对应的断点组中，然后判断这个断点所在的断点组是否为当前有效断点组。如果是，就立即激活这个断点。如果不是，那么这个断点暂时不会被激活，将会被保存到断点组中。在这种缓存机制下，用户态断点和内核态断点不会同时被激活，从而避免了内核态和用户态断点的冲突。

调试器检测到被调试操作系统的特权级切换：通过人工分析找到被调试操作系统的用户态和内核态切换的边界代码位置，并设置检测点。当操作系统运行到边界检测点时说明马上就会进行特权级的切换，此时调试器自动中断操作系统的运行，并进行对应符号表的切换以及断点组的切换，最后恢复操作系统的运行。如下图所示，当用户进程A想要通过系统调用进入内核态时，会触发边界检测点，这时调试器就会检测到被调试操作系统发生了特权级切换，接下来就会进行符号表切换、断点组切换，在切换过程中，调试器会删除原进程对应的地址空间中设置的断点，设置新进程地址空间中断点组的断点，从内核态回到用户态也是一样的流程。边界监测点不属于任何断点组，如果设置了边界监测点，就会立即被激活然后等待被触发。



特权级切换示意图

如下面流程图所展示的，操作者需要先找到边界监测点的位置，并进行配置，随后就可以启动调试。在调试过程中，被调试操作系统启动以后会先进入内核态，接着被调试操作系统继续运行，会暂停到边界监测点的位置，这时用户就可以在源代码的任意位置设置断点，但是只有内核态中的断点会被激活，随后被触发，最后会暂停到内核出口检测点的位置，调试器进行符号表和断点组的切换。接下来就会进入指定的用户进程，该用户进程中的所有断点都被激活，并暂停到用户进程中设置的第一个断点，在用户进程中，用户可以通过设置好的调试选项，在任意时刻进入内核态，在这个过程中，用户可以在任意时刻、任意代码行设置断点，观测各种变量信息，最终完成整个操作系统的调试。



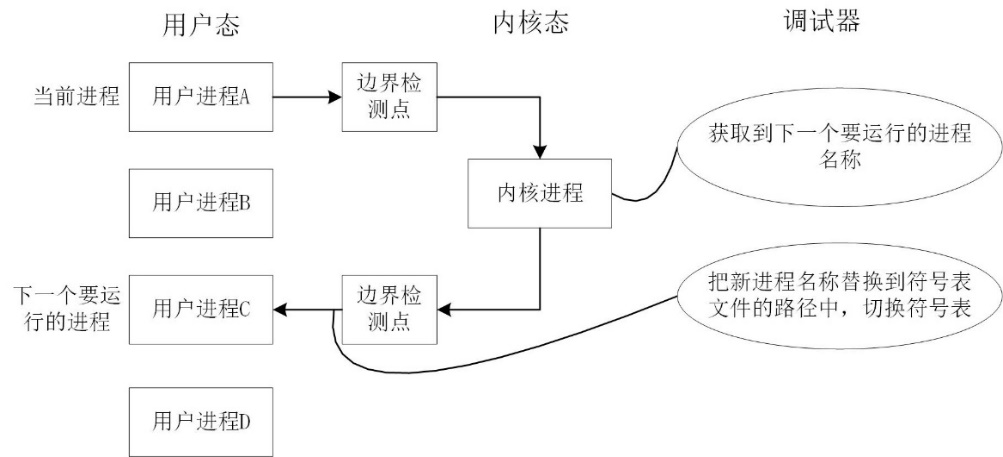
调试器配置、使用流程图

通过断点组对断点的缓存机制，用户可以在被调试操作系统任意特权级下的任意代码中设置断点，这些断点都会被保存，只有操作系统运行到某个进程的时候，相应的地址空间中的断点组的断点才会被激活，最终被触发，从而解决了跨特权级源代码断点设置的问题。

1. 解决多个用户进程符号表切换的问题

获取进程标识符：调试器是根据进程标识符，来进行符号表切换的，我们在调试器中使用一个变量来保存被调试进程的标识符。在操作系统中，切换用户进程都需要经过系统调用进入内核态，在内核态中可以获取到下一个运行的进程标识符。操作者需要在被调试操作系统的内核态源代码中，含有进程标识符变量的代码行设置断点，如sys\_exec函数中，通过给GDB发送获取变量信息的命令，并解析返回信息，来获取进程标识符。

多个用户进程切换符号表：在被调试操作系统刚刚运行起来的时候都是先从内核态开始运行的，然后会指定一个第一个要运行的用户态进程，如用户进程A，在调试器触发边界检测点的时候，调试器就会根据用户进程A的进程标识符进行符号表的切换，切换到指定的用户进程A的符号表，切换断点组，激活在用户进程A中设置的断点。接下来要进行进程切换的话，就需要从当前进程A进入内核态，当边界检测点被触发时，切换到内核的符号表，切换断点组，在内核态运行的时候，获取到下一个要执行的用户进程标识符，如用户进程C。在获取到用户进程C的标识符以后，赋值给保存进程标识符的变量，这样调试器就能够知道下一个要运行的进程是用户进程C。接下来从内核态进入用户态，在边界检测点进行符号表切换，此时要切换的符号表就是用户进程C的符号表，断点组切换，激活用户进程C中设置的断点，最终实现了多个用户进程的符号表切换，如下图所示。



多个用户进程的符号表切换示意图

调试器通过在内核态给GDB发送指定命令来获取到下一个要运行的进程标识符，获取到以后调试器就可以根据该标识符切换到正确的用户进程中，并对新进程进行调试。

# 三、技术效果

本申请的技术方案的技术优势（我做的有什么好处？）

本申请提供了一种支持跨内核态和用户态的操作系统源代码调试方法，在该方法中，我们设置了断点组管理模块，缓存用户设置的不在当前进程地址空间的断点，并在被调试操作系统运行到对应的地址空间的时候再激活断点，实现断点组的切换，解决了跨特权级的源代码断点设置冲突的问题；其次设置了边界检测点，使调试器能够识别被调试操作系统当前在哪个特权级中运行，并根据特权级切换不同的符号表，从而能够让使用者进行跨特权级的调试；最后为了支持多个用户进程的调试，我们在内核代码中获取下一个要运行进程的标识符，并告知调试器，调试器根据进程标识符来确定下一个要切换的符号表，获取到新进程的调试信息，对新进程进行调试，也能够支持多个用户进程的符号表切换，从而实现了对多个用户进程的调试。

# 四、替代方案

本方案的其他拓展方案或替代性解决方案