(19)国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 120066944 A
(43)申请公布日 2025.05.30

(21)申请号 202510136247.4

(22)申请日 2025.02.07

(71)申请人 镁佳(北京)科技有限公司
地址 100102 北京市朝阳区创远路34号院6号楼11层1101室

(72)发明人 袁明明

(74)专利代理机构 北京三聚阳光知识产权代理有限公司 11250
专利代理师 张妙琪

(51)Int.Cl.
G06F 11/362(2025.01)
G06F 11/30(2006.01)
G06F 11/32(2006.01)

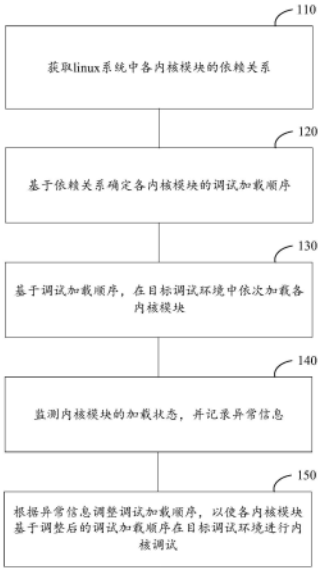
权利要求书2页 说明书14页 附图4页

(54)发明名称

内核调试方法及装置

(57)摘要

本发明实施例涉及计算机技术领域,公开了内核调试方法及装置,该方法包括:获取linux系统中各内核模块的依赖关系;基于依赖关系确定各内核模块的调试加载顺序;基于调试加载顺序,在目标调试环境中依次加载各内核模块;监测内核模块的加载状态,并记录异常信息;根据异常信息调整调试加载顺序,以使各内核模块基于调整后的调试加载顺序在目标调试环境进行内核调试。应用本发明的技术方案,能够提高对linux系统中各内核模块的调试效率及效果。



CN 120066944 A

1.一种内核调试方法,其特征在于,应用于linux系统,所述方法包括:

获取所述linux系统中各内核模块的依赖关系;

基于所述依赖关系确定各所述内核模块的调试加载顺序;

基于所述调试加载顺序,在目标调试环境中依次加载各所述内核模块;

监测所述内核模块的加载状态,并记录异常信息;

根据所述异常信息调整所述调试加载顺序,以使各所述内核模块基于调整后的调试加载顺序在所述目标调试环境进行内核调试。

2.根据权利要求1所述的方法,其特征在于,所述获取所述linux系统中各内核模块的依赖关系,包括:

获取所述内核模块的配置文件,并基于所述配置文件得到各所述内核模块的模块名称;

基于所述模块名称读取对应的元数据;

分析所述元数据中的依赖字段,得到依赖模块名称;

基于所述依赖模块名称,获取所述linux系统中各内核模块的依赖关系。

3.根据权利要求2所述的方法,其特征在于,所述基于所述配置文件得到各所述内核模块的模块名称,包括:

读取所述配置文件中各所述内核模块的定义块;

基于所述定义块中的名称标签提取所述内核模块的模块名称。

4.根据权利要求2所述的方法,其特征在于,所述基于所述配置文件得到各所述内核模块的模块名称,还包括:

获取所述模块名称中的重复名称,并将名称重复的所述内核模块标记为冲突模块;

统计所述冲突模块的冲突位置及数量,并基于所述冲突位置及数量生成对应的错误报告及配置检查提醒信息,以提醒用户基于所述错误报告进行所述配置文件的检查。

5.根据权利要求4所述的方法,其特征在于,所述获取所述模块名称中的重复名称,并将名称重复的所述内核模块标记为冲突模块,包括:

基于所述模块名称构建名称集合;

通过哈希表记录所述名称集合中各所述模块名称出现的次数;

若任意一所述模块名称的出现次数大于1,则将所述模块名称对应的所述内核模块标记为冲突模块。

6.根据权利要求4所述的方法,其特征在于,所述统计所述冲突模块的冲突位置及数量,并基于所述冲突位置及数量生成对应的错误报告及配置检查提醒信息,以提醒用户基于所述错误报告进行所述配置文件的检查,包括:

将所述冲突模块的名称和出现位置汇总成冲突列表;

基于所述冲突列表生成对应的错误内容;

获取所述错误内容的字符数;

若所述字符数大于字符阈值,则基于所述错误内容的生成错误摘要;

基于所述错误摘要生成对应的错误报告及配置检查提醒信息,以提醒用户基于所述错误报告文件进行所述配置文件的检查。

7.根据权利要求2所述的方法,其特征在于,所述基于所述依赖关系确定各所述内核模

块的调试加载顺序,包括:

基于所述依赖关系构建依赖图;

通过拓扑排序算法对所述依赖图进行排序,得到依赖排序结果;

基于所述依赖排序结果确定各所述内核模块的调试加载顺序。

8.根据权利要求7所述的方法,其特征在于,所述基于所述依赖关系构建依赖图,包括:

创建有向无环图;

基于各所述内核模块的模块名称在所述有向无环图中添加依赖节点;

基于所述依赖关系在所述依赖节点之间添加有向边;

基于所述依赖节点及所述有向边构建所述依赖图。

9.根据权利要求8所述的方法,其特征在于,所述基于所述依赖关系构建依赖图,还包括:

若所述有向边的数量大于所述依赖节点的节点量平方根,则基于目标验证策略对所述依赖关系进行再次验证;

基于验证后的所述依赖关系,对所述依赖节点及所述有向边进行调整,得到调整依赖节点及调整有向边;

基于所述调整依赖节点及所述调整有向边,构建所述依赖图。

10.一种内核调试装置,其特征在于,应用于linux系统,所述装置包括:

关系获取模块,用于获取所述linux系统中各内核模块的依赖关系;

顺序确定模块,用于基于所述依赖关系确定各所述内核模块的调试加载顺序;

内核加载模块,用于基于所述调试加载顺序,在目标调试环境中依次加载各所述内核模块;

异常记录模块,用于监测所述内核模块的加载状态,并记录异常信息;

内核调试模块,用于根据所述异常信息调整所述调试加载顺序,以使各所述内核模块基于调整后的调试加载顺序在所述目标调试环境进行内核调试。

内核调试方法及装置

技术领域

[0001] 本发明实施例涉及计算机技术领域,具体涉及内核调试方法及装置。

背景技术

[0002] Linux作为一种广泛使用的开源操作系统,其内核调试对于系统的稳定性和性能优化至关重要。然而,Linux内核的复杂性使得调试工作变得相当具有挑战性。相关技术中对Linux内核的调试,通常采用定制的Linux内核调试工具,但这些工具在调试环境中,通常会由于内核模块的加载顺序不当,从而导致系统挂起或无法正常运行,从而影响调试的效率和效果。

发明内容

[0003] 鉴于上述问题,本发明实施例提供了一种内核调试方法及装置。

[0004] 根据本发明实施例的一个方面,提供了一种内核调试方法,应用于linux系统,该方法包括:获取linux系统中各内核模块的依赖关系;基于依赖关系确定各内核模块的调试加载顺序;基于调试加载顺序,在目标调试环境中依次加载各内核模块;监测内核模块的加载状态,并记录异常信息;根据异常信息调整调试加载顺序,以使各内核模块基于调整后的调试加载顺序在目标调试环境进行内核调试。通过上述过程,可以提高对linux系统中各内核模块的调试效率及效果。

[0005] 在一种可选的实施方式中,获取linux系统中各内核模块的依赖关系,包括:

[0006] 获取内核模块的配置文件,并基于配置文件得到各内核模块的模块名称;

[0007] 基于模块名称读取对应的元数据;

[0008] 分析元数据中的依赖字段,得到依赖模块名称;

[0009] 基于依赖模块名称,获取linux系统中各内核模块的依赖关系。

[0010] 在一种可选的实施方式中,基于配置文件得到各内核模块的模块名称,包括:

[0011] 读取配置文件中各内核模块的定义块;

[0012] 基于定义块中的名称标签提取内核模块的模块名称。

[0013] 在一种可选的实施方式中,基于配置文件得到各内核模块的模块名称,还包括:

[0014] 获取模块名称中的重复名称,并将名称重复的内核模块标记为冲突模块;

[0015] 统计冲突模块的冲突位置及数量,并基于冲突位置及数量生成对应的错误报告及配置检查提醒信息,以提醒用户基于错误报告进行配置文件的检查。

[0016] 在一种可选的实施方式中,获取模块名称中的重复名称,并将名称重复的内核模块标记为冲突模块,包括:

[0017] 基于模块名称构建名称集合;

[0018] 通过哈希表记录名称集合中各模块名称出现的次数;

[0019] 若任意一模块名称的出现次数大于1,则将模块名称对应的内核模块标记为冲突模块。

[0020] 在一种可选的实施方式中,统计冲突模块的冲突位置及数量,并基于冲突位置及数量生成对应的错误报告及配置检查提醒信息,以提醒用户基于错误报告进行配置文件的检查,包括:

[0021] 将冲突模块的名称和出现位置汇总成冲突列表;

[0022] 基于冲突列表生成对应的错误内容;

[0023] 获取错误内容的字符数;

[0024] 若字符数大于字符阈值,则基于错误内容的生成错误摘要;

[0025] 基于错误摘要生成对应的错误报告及配置检查提醒信息,以提醒用户基于错误报告文件进行配置文件的检查。

[0026] 在一种可选的实施方式中,基于依赖关系确定各内核模块的调试加载顺序,包括:

[0027] 基于依赖关系构建依赖图;

[0028] 通过拓扑排序算法对依赖图进行排序,得到依赖排序结果;

[0029] 基于依赖排序结果确定各内核模块的调试加载顺序。

[0030] 在一种可选的实施方式中,基于依赖关系构建依赖图,包括:

[0031] 创建有向无环图;

[0032] 基于各内核模块的模块名称在有向无环图中添加依赖节点;

[0033] 基于依赖关系在依赖节点之间添加有向边;

[0034] 基于依赖节点及有向边构建依赖图。

[0035] 在一种可选的实施方式中,基于依赖关系构建依赖图,还包括:

[0036] 若有向边的数量大于依赖节点的节点量平方根,则基于目标验证策略对依赖关系进行再次验证;

[0037] 基于验证后的依赖关系,对依赖节点及有向边进行调整,得到调整依赖节点及调整有向边;

[0038] 基于调整依赖节点及调整有向边,构建依赖图。

[0039] 根据本发明实施例的另一方面,提供了一种内核调试装置,应用于linux系统,包括:关系获取模块,用于获取linux系统中各内核模块的依赖关系;顺序确定模块,用于基于依赖关系确定各内核模块的调试加载顺序;内核加载模块,用于基于调试加载顺序,在目标调试环境中依次加载各内核模块;异常记录模块,用于监测内核模块的加载状态,并记录异常信息;内核调试模块,用于根据异常信息调整调试加载顺序,以使各内核模块基于调整后的调试加载顺序在目标调试环境进行内核调试。通过上述模块,可以提高对linux系统中各内核模块的调试效率及效果。

[0040] 根据本发明实施例的另一方面,提供了一种计算机设备,包括:处理器、存储器、通信接口和通信总线,处理器、存储器和通信接口通过通信总线完成相互间的通信;存储器用于存放至少一可执行指令,可执行指令使处理器执行前述一种内核调试方法的操作。

[0041] 根据本发明实施例的又一方面,提供了一种计算机可读存储介质,存储介质中存储有至少一可执行指令,可执行指令使计算机设备/装置执行前述一种内核调试方法的操作。

[0042] 根据本发明实施例的又一方面,提供了一种计算机程序产品,包括计算机指令,计算机指令用于使计算机执行上述第一方面或其对应的任一实施方式的内核调试方法的操

作。

[0043] 上述说明仅是本发明实施例技术方案的概述,为了能够更清楚了解本发明实施例的技术手段,而可依照说明书的内容予以实施,并且为了让本发明实施例的上述和其它目的、特征和优点能够更明显易懂,以下特举本发明的具体实施方式。

附图说明

[0044] 附图仅用于示出实施方式,而并不认为是对本发明的限制。而且在整个附图中,用相同的参考符号表示相同的部件。在附图中:

[0045] 图1示出了本发明提供的一种内核调试方法的流程示意图;

[0046] 图2示出了本发明提供的一种内核调试方法的另一流程示意图;

[0047] 图3示出了本发明提供的一种内核调试方法的又一流程示意图;

[0048] 图4示出了本发明提供的一种内核调试装置的结构示意图;

[0049] 图5示出了本发明提供的一种计算机设备的结构示意图。

具体实施方式

[0050] 为使本发明实施例的目的、技术方案和优点更加清楚,下面将结合本发明实施例中的附图,对本发明实施例中的技术方案进行清楚、完整地描述,显然,所描述的实施例是本发明一部分实施例,而不是全部的实施例。基于本发明中的实施例,本领域技术人员在没有做出创造性劳动前提下所获得的所有其他实施例,都属于本发明保护的范围。

[0051] 图1示出了本发明一种内核调试方法的第一实施例的流程图,应用于linux系统。如图1所示,该方法包括以下步骤:

[0052] 步骤110,获取linux系统中各内核模块的依赖关系。

[0053] 具体的,linux系统中各内核模块的依赖关系,是通过对linux系统的内核源码进行静态分析来实现的。具体而言,首先解析linux系统的内核源码,提取出各个内核模块之间的依赖关系信息,并构建相应的依赖关系图。然后,在开发目标软件时,根据待开发功能及对应的目标主线程,在依赖关系图中找到与目标主线程相关的内核模块,进而确定需要调试的内核模块及其依赖关系,从而准确地获取linux系统中各内核模块的依赖关系,为后续的内核调试工作提供有力的支持。

[0054] 进一步地,在解析linux系统的内核源码时,可以采用现有的源码解析工具或自行编写解析脚本,对内核源码中的头文件、源文件以及模块间的调用关系等进行深度分析,以确保提取出的依赖关系信息的准确性和完整性。同时,为了直观展示各内核模块之间的依赖关系,构建的依赖关系图可以采用树状图、网状图等多种表现形式,便于开发人员理解和分析。

[0055] 步骤120,基于依赖关系确定各内核模块的调试加载顺序。

[0056] 具体的,基于依赖关系确定各内核模块的调试加载顺序时,需综合考虑内核模块间的依赖层级和调用关系。首先,对于无依赖关系的独立内核模块,可以直接进行调试加载。其次,对于存在依赖关系的内核模块,需按照依赖层级从低到高的顺序依次进行调试加载,确保在加载某个内核模块前,其依赖的所有内核模块均已被正确加载和初始化。此外,还需根据内核模块间的循环依赖情况,通过调整模块设计或加载策略来避免循环依赖导致

的加载失败问题。通过合理的调试加载顺序,可以确保各内核模块在目标软件中的正确运行和相互协作。

[0057] 在一种可选的实施方式中,还可以基于预设的内核模块加载规则对内核模块进行排序。即在内核启动之前,根据一定的逻辑和规则来确定各个模块的加载优先级。这些规则可以基于依赖关系、初始化时间或者安全性等多方面因素制定。通过这种预排序,能够确保关键模块优先加载,减少加载冲突和失败的可能性。例如,假设有一个内核模块依赖于另一个基础模块,那么在预设规则中,基础模块的加载优先级将会更高,确保依赖关系得到满足。

[0058] 此外,内核模块加载规则的制定还需考虑系统的稳定性和性能。对于一些对系统稳定性影响较大的模块,可以设定其具有较高的加载优先级,以确保系统能够快速进入稳定状态。同时,对于初始化时间较长或者资源占用较大的模块,可以合理安排其加载时机,避免在启动时造成系统资源紧张或者响应延迟。在安全性方面,可以将涉及安全验证或者防护功能的模块设置为高优先级加载,以有效抵御外部攻击或者恶意软件的入侵。通过综合考虑多种因素,制定科学合理的内核模块加载规则,可以进一步提升系统的整体性能和安全性。

[0059] 步骤130,基于调试加载顺序,在目标调试环境中依次加载各内核模块。

[0060] 具体的,基于调试加载顺序,在目标调试环境中依次加载各内核模块的过程中,可以实时监控各内核模块的加载状态,以确保每个内核模块都能够正确加载并运行。同时,还可以记录加载过程中的相关信息,如加载时间、加载结果等,以便后续分析和优化。如果在加载过程中遇到任何问题,如模块依赖关系不满足、加载失败等,则立即停止加载过程,并给出相应的错误提示,以便开发人员及时定位并解决问题。此外,开发人员根据实际需求调整加载顺序和加载参数,以满足不同场景下的调试需求,从而确保内核模块在目标调试环境中按照预定的顺序正确加载,为后续的系统调试和测试工作打下坚实的基础。

[0061] 在一种可选的实施方式中,在完成内核模块排序后,将排序结果应用到实际的加载过程中时,可以由内核调度器管理,确保按照既定顺序加载每个模块。在这该过程中,目标调试环境提供了必要的支持工具,如日志记录、断点调试等,帮助开发者更好地观察和控制内核模块的加载行为。例如,开发团队在虚拟机上搭建了一个目标调试环境,使用了KVM(基于内核的虚拟机)和QEMU(快速仿真),并在该目标调试环境中执行内核模块的加载过程,从而便于观察和调试内核启动的每一个细节。

[0062] 步骤140,监测内核模块的加载状态,并记录异常信息。

[0063] 如上,通过对内核模块加载过程的加载状态持续监控,可以在出现问题时立即捕获并记录相关信息。这样不仅有助于及时发现问题,还可以为后续的问题分析提供重要的数据支持。

[0064] 在一种可选的实施方式中,上述异常信息可以包括加载失败的内核模块名称、错误代码、发生时间和上下文环境等。例如,通过内核日志系统dmesg记录内核模块每次加载的结果,并通过klogd进程定期检查日志中的错误消息,确保任何异常都能被迅速发现和处理。

[0065] 此外,为了提高系统的稳定性,还可以在检测到某个内核模块加载失败时,还可以尝试重新加载该模块,或者根据错误代码和上下文环境,采取其他适当的恢复措施。例如,

如果是因为依赖关系未满足导致的加载失败,可以尝试先加载缺失的依赖模块,然后再重新加载目标模块,从而可以最大程度地减少因内核模块加载失败而对整个系统造成的影响。

[0066] 步骤150,根据异常信息调整调试加载顺序,以使各内核模块基于调整后的调试加载顺序在目标调试环境进行内核调试。

[0067] 具体的,在调整调试加载顺序时,可以对异常信息进行分析,确定哪些内核模块的加载顺序可能导致冲突或失败,并据此调整加载顺序。例如,如果某个内核模块依赖于其他模块提供的服务或资源,则系统会确保这些依赖模块在目标模块之前加载。如果异常信息指示某个内核模块因为依赖关系问题而加载失败,那么调整后的调试加载顺序可以优先加载该模块所依赖的其他模块,以确保依赖关系得到满足。另外,如果某个模块在多次尝试加载后仍然失败,系统可以将其标记为“不可加载”,并在后续的调试过程中跳过该模块,以避免不必要的等待和错误。

[0068] 通过上述方式,可以有效避免加载顺序不当导致的加载失败,从而提高内核调试的效率和成功率。同时,还可以通过记录每次调整后的加载顺序和调试结果,为后续的内核开发和调试提供参考和依据。

[0069] 在一种可选的实施方式中,当检测到内核模块加载出现异常时,还可以触发调整机制,即重新评估模块的依赖关系和其他影响因素,进而对加载顺序进行优化调整,以在不停止整个系统的情况下解决问题,提高了系统的稳定性和鲁棒性。例如,假设在加载某个网络驱动模块时发生了挂起,经分析发现该模块存在初始化资源争用的问题,此时可以通过调整配置文件或使用专门的调试工具,将该模块的加载顺序向后推迟,让其他不涉及资源争用的关键模块先加载,从而恢复系统的正常运行。这样不仅能解决当前的挂起问题,还能预防未来可能出现的类似情况。

[0070] 本发明实施例的内核调试方法,通过合理排序、逐步加载、实时监测和动态调整,有效解决了内核模块加载过程中可能出现的各种问题,特别是因加载顺序不当导致的系统挂起问题,大幅提升了内核调试的效率和成功率。

[0071] 图2示出了本发明内核调试方法的另一个实施例的流程图,应用于linux系统。如图2所示,该方法包括以下步骤:

[0072] 步骤210,获取linux系统中各内核模块的依赖关系。

[0073] 具体的,上述步骤210包括:

[0074] 步骤2101,获取内核模块的配置文件,并基于配置文件得到各内核模块的模块名称。

[0075] 在一种可选的实施方式中,在基于配置文件得到各内核模块的模块名称时,可以读取配置文件中各内核模块的定义块;基于定义块中的名称标签提取内核模块的模块名称。即,通过从内核模块的配置文件中读取内核模块信息,以提取出每个内核模块的模块名称,从而确定后续需要处理的具体内核模块。

[0076] 具体的,可以通过逐行或按段读取配置文件的方式,将配置文件中的每个模块定义块分离出来。基于定义块中的名称标签提取模块名称。这一过程从读取到的模块定义块中查找并提取表示模块名称的关键字段。字段通常被称为名称标签。比如,在配置文件中,如果有一个条目如下:alias sound-slot-0snd-card-0,则此步骤需要能够识别出这里的

名称标签为snd-card-0。

[0077] 在一种可选的实施方式中,还可以获取模块名称中的重复名称,并将名称重复的内核模块标记为冲突模块;统计冲突模块的冲突位置及数量,并基于冲突位置及数量生成对应的错误报告及配置检查提醒信息,以提醒用户基于错误报告进行配置文件的检查。

[0078] 具体的,当提取了所有内核模块的模块名称后,需要对它们进行比较,检查是否多个模块被赋予了相同的名称。在发现有相同名称的情况下,这些模块即被视为冲突模块,并进行特别标记。如果有两个或更多的定义条目都使用了同样的名称标签,则将这些条目标识为冲突。

[0079] 如果冲突模块数量C大于0,则记录冲突模块并提示用户检查配置文件。此处C指的是已检测到的所有冲突模块的总数。冲突模块总数的取值范围理论上为大于或者等于零,但最理想的状态是没有冲突发生(C=0)。此规则设定的意义在于,通过检测是否有C>0的情况来判断配置文件中是否存在潜在问题,及时提醒用户注意,以保证系统的稳定性和可预测性。当发现C>0时,程序不仅会在日志中记录下所有的冲突模块信息,还会给出明确的警告信息,指引用户对相应的配置项进行审查修正,从而有助于避免由于配置错误导致的加载失败或其他未知行为。

[0080] 在一种可选的实施方式中,在获取模块名称中的重复名称,并将名称重复的内核模块标记为冲突模块时,可以基于模块名称构建名称集合;通过哈希表记录名称集合中各模块名称出现的次数;若任意一模块名称的出现次数大于1,则将模块名称对应的内核模块标记为冲突模块。

[0081] 具体的,首先遍历所有提取到的模块名称,构建名称集合,旨在从系统的各个内核模块中获取所有模块名称,确保没有遗漏,并将这些模块名称放入一个集合中,得到名称集合。名称集合的用途是为了方便后续处理和查找操作。例如,从Linux内核中提取到的模块名称可能包括模块A、模块B、模块C,并将它们放入一个集合{模块A,模块B,模块C}中。

[0082] 接着,使用哈希表记录每个名称出现的次数时,哈希表是一种高效的存储和查询数据结构,能够快速统计每个模块名称(名称)在集合中出现的次数。具体实现时,可以在遍历模块名称集合的同时,将每个名称及其出现次数存储到哈希表中。例如,如果模块名称集合为{模块A,模块B,模块A,模块C},则哈希表记录结果为:{模块A:2,模块B:1,模块C:1}。如果某个名称出现次数F大于1,则将其标记为冲突模块。这里的参数F表示模块名称在哈希表中记录的出现次数,其取值范围为非负整数。当F>1时,表示该模块名称在至少出现两次,应被标记为冲突模块。例如,在上一步中的哈希表{模块A:2,模块B:1,模块C:1}中,模块A的出现次数为2,因此将其标记为冲突模块。

[0083] 随后,输出所有冲突模块的列表,并生成错误报告,旨在将所有标记为冲突的模块名称收集起来,形成一个列表,并生成详细的错误报告,以便开发人员进一步处理这些问题。假设确定模块A为冲突模块,那么最终输出的冲突模块列表为[模块A],并且生成相应的错误报告,报告中详细说明哪些模块名称存在重复问题,帮助开发者迅速定位和解决问题。

[0084] 综上,整个过程中,通过遍历模块名称集合、使用哈希表记录出现次数、标记冲突模块和生成错误报告,确保系统中模块名称的唯一性,提高系统稳定性和可靠性。F>1是基于模块名称在正常情况下应该唯一的假设,如果同一名称出现多次,则意味着存在潜在冲突,需特别关注和处理。

[0085] 在一种可选的实施方式中,在统计冲突模块的冲突位置及数量,并基于冲突位置及数量生成对应的错误报告及配置检查提醒信息,以提醒用户基于错误报告进行配置文件的检查时,可以将冲突模块的名称和出现位置汇总成冲突列表;基于冲突列表生成对应的错误内容;获取错误内容的字符数;若字符数大于字符阈值,则基于错误内容生成错误摘要;基于错误摘要生成对应的错误报告及配置检查提醒信息,以提醒用户基于错误报告文件进行配置文件的检查。

[0086] 具体的,将冲突模块名称和出现位置汇总成列表时,首先系统遍历当前运行环境中的所有模块,检查这些模块之间的兼容性,一旦发现某个或某些模块之间存在兼容性问题,则记录这些模块的名称及其在内存或文件系统中的位置,目的是为后续处理提供详实的数据基础。基于列表生成详细的错误报告时,可以利用上一步产生的数据,创建一个包含冲突模块详细信息的文本,包括但不限于模块版本、冲突原因及可能的解决方案。这样可以帮助系统管理员或开发者快速定位问题根源,并采取措施修复问题。

[0087] 接着,将上述生成的错误报告保存至用户或配置文件预先定义的位置,如特定目录下的文件,以便相关人员随时查阅。确保数据存储安全可靠的同时,也方便后期的问题追踪和历史对比。如果报告长度L超过设定的最大长度M,则截取前M个字符并加上省略号。此处的L表示实际生成的错误报告的总字符数,M是预设的最大允许长度值,一般推荐设置为2000至5000字符之间,具体最优值可以根据实际应用需求灵活调整。此操作的目的是防止错误报告文件过大,导致存储空间紧张或传输困难,同时保证报告的关键内容得以保留。

[0088] 例如,假设在调试某Linux系统的内核时,检测到有两个网络驱动程序模块由于实现细节的不同而无法同时加载。根据这个列表,系统生成了详细的错误报告,解释了两个驱动程序为何无法共存,并给出了重新配置或卸载其中一个模块的建议。报告最终被保存到了系统的/var/log目录下,考虑到安全性和访问效率,文件大小被控制在不超过3000字符。此外,L可能达到甚至超过3000,若M被设置为3000,则超出的部分将以省略号表示,确保文件不会超出限定的最大尺寸。该设置有效平衡了信息完整性和资源管理的需求。

[0089] 步骤2102,基于模块名称读取对应的元数据。

[0090] 其中,上述元数据通常包含了模块的各种属性和信息。分析元数据中的依赖字段,记录依赖模块名称。这是指从元数据文件中提取出指定模块的依赖信息。根据记录的依赖模块名称,递归地解析每个依赖模块的信息,直到所有依赖模块都被解析。这里D表示每次解析时当前模块的直接依赖模块数量。通常D的初始值为该模块直接依赖的数量,当递归解析完毕时,D应为0,表示没有更多的依赖需要解析。

[0091] 通过上述步骤,可以全面且详细地获取内核模块及其所有依赖关系的信息,有助于在调试和维护内核过程中准确定位问题和优化性能。

[0092] 步骤2103,分析元数据中的依赖字段,得到依赖模块名称。

[0093] 步骤2104,基于依赖模块名称,获取linux系统中各内核模块的依赖关系。

[0094] 步骤220,基于依赖关系确定各内核模块的调试加载顺序。

[0095] 详细请参见图1所示实施例的步骤120,在此不再赘述。

[0096] 步骤230,基于调试加载顺序,在目标调试环境中依次加载各内核模块。

[0097] 详细请参见图1所示实施例的步骤130,在此不再赘述。

[0098] 步骤240,监测内核模块的加载状态,并记录异常信息。

[0099] 详细请参见图1所示实施例的步骤140,在此不再赘述。

[0100] 步骤250,根据异常信息调整调试加载顺序,以使各内核模块基于调整后的调试加载顺序在目标调试环境进行内核调试。

[0101] 详细请参见图1所示实施例的步骤150,在此不再赘述。

[0102] 综上,本发明实施例的内核调试方法,通过获取linux系统中各内核模块的依赖关系;基于依赖关系确定各内核模块的调试加载顺序;基于调试加载顺序,在目标调试环境中依次加载各内核模块;监测内核模块的加载状态,并记录异常信息;根据异常信息调整调试加载顺序,以使各内核模块基于调整后的调试加载顺序在目标调试环境进行内核调试,从而提高了对linux系统中各内核模块的调试效率及效果。

[0103] 图3示出了本发明内核调试方法的另一个实施例的流程图,应用于linux系统。如图3所示,该方法包括以下步骤:

[0104] 步骤310,获取linux系统中各内核模块的依赖关系。

[0105] 详细请参见图2所示实施例的步骤210,在此不再赘述。

[0106] 步骤320,基于依赖关系确定各内核模块的调试加载顺序。

[0107] 具体的,上述步骤320包括:

[0108] 步骤3201,基于依赖关系构建依赖图。

[0109] 步骤3202,通过拓扑排序算法对依赖图进行排序,得到依赖排序结果。

[0110] 步骤3203,基于依赖排序结果确定各内核模块的调试加载顺序。

[0111] 在一种可选的实施方式中,基于预设的内核模块加载规则对内核模块进行排序时,首先,获取内核模块的依赖关系信息。这意味着通过分析内核模块的配置文件或其他元数据,确定每个模块所需依赖的其他模块。这些信息通常包含在模块的配置文件或编译过程中生成的依赖列表中。例如,通过读取文件,可以获取到各模块之间的依赖关系。接着,根据依赖关系信息构建依赖图,以将上述获取的依赖关系信息转化为图形表示,即每个模块为图中的一个节点,模块之间的依赖关系则表示为边。例如,如果模块A依赖于模块B,那么在依赖图中,A到B会有一条有向边。这种图结构使得后续的排序算法能够有效执行。然后,使用拓扑排序算法对依赖图进行排序。拓扑排序算法适用于有向无环图DAG,它可以将节点按照一定的顺序排列,使每个节点的所有前置节点都排在其前面。具体地,常见的拓扑排序算法如Kahn算法或深度优先搜索DFS均可以应用于本场景。例如,在一个实施例中,可以使用Kahn算法来处理依赖图,从而获得一个符合依赖关系的有序模块列表。最后,依据排序结果调整内核模块的加载顺序,确保依赖关系正确的加载模块。最后一步是在拓扑排序完成后,根据排序结果重新安排内核模块的加载顺序。这样做可以确保所有模块按照正确的依赖关系被依次加载,避免因加载顺序不当导致的问题。例如,假设拓扑排序后的模块加载顺序为B->C->A,那么在实际加载时,模块B会首先被加载,然后是模块C,最后是模块A,这样就能保证所有的依赖关系都能正确处理。

[0112] 可以理解的,通过上述步骤不仅能够实现对内核模块加载顺序的有效地管理,优化内核调试过程,还有助于提升系统的稳定性和可靠性,进而简化开发和维护人员的工作负担。

[0113] 在一种可选的实施方式中,基于依赖关系构建依赖图时,可以先创建有向无环图;基于各内核模块的模块名称在有向无环图中添加依赖节点;基于依赖关系在依赖节点之间

添加有向边;基于依赖节点及有向边构建依赖图。

[0114] 在一种可选的实施方式中,基于依赖关系构建依赖图时,若有向边的数量大于依赖节点的节点量平方根,则基于目标验证策略对依赖关系进行再次验证;基于验证后的依赖关系,对依赖节点及有向边进行调整,得到调整依赖节点及调整有向边;基于调整依赖节点及调整有向边,构建依赖图。

[0115] 在创建有向无环图(DAG)时,首先建立一数据结构,该数据结构用于表示一系列节点及其之间的关系,而这些关系是单向的且不存在任何循环路径。在这个数据结构中,节点代表具体的内核模块或函数,而有向边则表示从一个模块到另一个模块的依赖关系。例如,在调试Linux内核时,可以创建一个空的DAG,用来记录各模块间如何相互调用或依赖。

[0116] 在基于依赖关系信息在图中添加节点时,可以根据内核模块或函数的实际依赖情况,将相应的模块或函数添加为DAG中的节点。具体地,假设正在分析的内核部分包含四个关键模块,那么这一步就是将这四个模块作为独立的节点加入到空DAG中,以便于后续建立它们之间的连接关系。

[0117] 在根据依赖关系在节点之间添加有向边时,可以依据实际的依赖情况,使用有向边连接图中的相应节点。有向边的方向表明了调用或者依赖的方向。例如,在处理网络堆栈问题时,若发现TCP模块需要依赖IP模块来工作,则应在TCP模块节点指向IP模块节点的方向上绘制一条有向边,表明这种依赖关系的存在。

[0118] 进一步地,如果有向边数量E大于节点数量N的平方根,则进行额外的依赖关系验证,以确保图的正确性,其中E表示有向边的数量,N表示节点的数量。这里的公式sqrt{N}<E主要作用是在图变得较为复杂(节点之间的连接异常多)时,启动额外的依赖关系验证机制以确保不会因为误报而影响最终调试结果的质量。在这个不等式中,N可以从1增加到几千,甚至更多;而E需要是非负整数。该条件下的最优设置是没有绝对标准,因为它取决于具体的系统大小和复杂度,但总体目标是为了减少可能存在的错误配置。当满足这个条件时,意味着系统中存在过多复杂的连接或潜在的循环依赖,需要进一步核查依赖关系的有效性和必要性,避免因逻辑错误导致调试困难或系统性能下降。举例来讲,当检测到一个大型Linux内核系统中的有向边数量远超过预期值时,通过触发额外验证过程成功发现了某些冗余或异常的依赖项,进而简化了整个内核结构,提高了调试效率和稳定性。

[0119] 为了确保所构建的依赖图的准确性,需要进行额外的依赖关系验证,以检查图中是否存在潜在的错误或遗漏的依赖关系时,可以通过对比内核源代码或文档中的依赖关系描述,以验证图中每个节点及其连接是否准确无误。如果发现有任何不一致之处,需要返回前面的步骤进行修正,直到确保所有依赖关系都被正确表示在图中。这种验证过程不仅提高了依赖图的准确性,也为后续的调试工作提供了更为可靠的基础。

[0120] 在验证过程中,可以采用多种技术手段,如静态代码分析、动态测试等,以全面检查依赖关系的正确性。静态代码分析可以通过扫描内核源代码,自动识别并提取出模块之间的依赖关系,然后与依赖图进行对比。动态测试则是在系统运行时,通过监控模块加载和卸载的过程,观察实际发生的依赖关系是否与图中所示一致。这些验证手段的结合使用,可以大大提高依赖图构建的准确性和可靠性。同时,为了确保验证过程的有效性和高效性,我们还需要不断优化和调整验证策略,以适应不同规模和复杂度的内核系统。

[0121] 此外,为了确保依赖图的构建既高效又准确,在构建过程中还采用了分阶段的验

证策略。在初始阶段,即依赖关系信息的收集阶段,就进行基础的数据完整性和一致性检查,避免将错误或矛盾的信息引入后续的图构建步骤。随着依赖图的逐步构建,每增加一个节点或一条边,都会立即触发局部验证逻辑,确保新加入的元素不会破坏图的整体结构,比如检查是否形成了环路,或者是否违反了已定义的依赖规则。

[0122] 在依赖图完全构建完成后,除了上述基于有向边数量和节点数量比例的额外验证外,还会进行一次全面的图遍历检查,以最终确认图的正确性。这一步骤不仅包括对节点和边的存在性、连接性的验证,还可以深入检查依赖关系的语义正确性,比如确保没有循环依赖、没有缺失的依赖等。通过这些多层次的验证措施,可以最大程度地保证依赖图的准确性和可靠性,为后续的内核调试工作提供坚实的基础。

[0123] 在一种可选的实施方式中,在创建有向无环图(DAG)时,可以先初始化图结构;遍历所有依赖关系,将每个模块作为图的节点;使用边连接具有依赖关系的节点;如果存在循环依赖,即节点间形成环路R,则记录环路节点并抛出异常,其中R表示环路节点。

[0124] 初始化图结构是指建立一个初始的图对象,为后续的操作准备数据结构。这个图对象包含一个空的节点集合和一个空的边集合,以便在后续步骤中逐步填充。这一步确保了一个清晰且有序的基础环境,使得图能够被有效构建和管理。

[0125] 遍历所有依赖关系,将每个模块作为图的节点。这一过程需要遍历系统中的所有模块,识别出每个模块的唯一标识,并将其作为图中的一个节点添加到图结构中。这里的模块可以是系统的组件、功能模块或者内核模块等。此步骤的关键是准确地提取和记录每个模块的信息,确保图的完整性。

[0126] 使用边连接具有依赖关系的节点。在这一过程中,根据之前遍历到的模块之间的依赖关系,为具有直接或间接依赖关系的节点之间添加边。例如,假设模块A依赖于模块B,而模块B依赖于模块C,则会在这三个模块之间添加相应的边:A→B和B→C。这种依赖关系的表达形式确保了图能够正确表示系统中的依赖结构。

[0127] 如果存在循环依赖,即节点间形成环路R,则记录环路节点并抛出异常。这里的关键是检测图中是否存在环路。环路的存在会导致系统无法正常运行,因为它会无限循环依赖,最终导致资源耗尽或死锁。通过拓扑排序算法或其他图遍历方法可以检测环路。若发现环路,则记录环路中的所有节点,并抛出一个异常来中断进一步的图构建操作。这一步确保了所创建的图是无环的,符合有向无环图(DAG)的定义。

[0128] 在一个实施例中,假设有一种用于调试Linux内核的系统,该系统包含多个模块,每个模块都有自己的功能和依赖关系。接着,根据模块之间的依赖关系添加边。

[0129] 对于参数R,它表示形成环路的一组节点,是一个集合。范围从至少包含两个节点的最简单环路(如A→B→A)到多个节点构成的复杂环路。最优值是没有环路,即R为空集,这样可以保证图的正常构建和使用。设置R这个参数的目的是能够快速且准确地识别和处理图中的循环依赖问题,确保最终生成的图是一个有效的有向无环图(DAG)。

[0130] 接下来,在获取初始化图结构时,可以先创建空的节点集合和边集合。这一步确保了初始状态下没有数据污染,为空节点和边提供了一个纯净的起点。定义节点类和边类。节点类和边类通常包含了必要的属性和方法,使得图中的各组成部分能够正确表示其功能。例如,在用于调试Linux内核的系统中,节点可能代表不同的内核模块,而边则可能表示模块之间的调用关系或依赖关系。接着,使用哈希表管理节点及其属性。通过使用哈希表,能

够高效地查找和管理大量节点,确保快速访问节点属性和相关信息。在这一实施例中,节点的属性可以包括模块名、路径、版本信息等。

[0131] 如果哈希表的大小S大于模块总数T,则进行优化处理以减少存储空间,其中S表示哈希表的大小,T表示模块总数。这里的S是一个自然数,反映了哈希表中的条目数量;T也是一个自然数,表示系统的模块数量。当S>T时,说明哈希表存在多余的空间,可能导致资源浪费。在这种情况下,可以通过删除不必要的哈希表条目或压缩哈希表来优化存储。具体地,可以在遍历所有节点后,检查哈希表中是否存在多余的空位,并对其进行清理。此外,还可以考虑使用动态调整大小的哈希表,根据实际情况调整哈希表的大小,从而优化存储。

[0132] 例如,假设有一个包含10个模块的Linux内核调试系统,初始创建了一个能容纳100个条目的哈希表。经过初始化图结构并添加所有节点后,发现实际使用的节点只有10个,此时S=100,而T=10,显然存在大量的空余条目。为了减少存储空间,可以删除多余的哈希表条目,或者将哈希表重新调整为更合适的大小,如调整为只容纳10个条目的大小。通过这样的优化,可以有效提高系统的性能和效率。

[0133] 接下来,在使用哈希表管理节点及其属性的过程中,由于哈希表是一种高效的管理工具,可以提高数据检索速度,特别是在处理大量节点及其属性的情况下。具体步骤如下:

[0134] 首先,遍历所有节点,将其添加到哈希表中。这一步确保了系统中所有的节点都能被有效地管理和检索。例如,假设系统中有多个内核模块节点,每个节点都包含模块的名称、版本和状态信息。遍历这些节点后,它们将被逐个插入到哈希表中。

[0135] 其次,为每个节点分配唯一标识符。这个标识符用于区分不同节点,保证每个节点在哈希表中都有唯一的对应位置。例如,在一个具体的实施例中,唯一标识符可以通过结合节点的名称和类型来生成。具体地,如果节点的名称是netfilter,类型是module,唯一标识符可能是netfilter_module。

[0136] 再者,记录节点的属性,如名称、类型等。这一步确保了每个节点的信息都被准确记录,便于后续查询和操作。例如,节点netfilter可能具有属性:名称netfilter,类型module,状态active。

[0137] 最后,如果哈希表中的冲突率H大于预设的阈值Q,则采用更大的哈希表容量重新初始化哈希表,以提高查找效率。这里,H表示哈希表的冲突率,即哈希表中因散列地址相同而引起的冲突次数的比例;Q表示预设的冲突率阈值,通常设定在0.1到0.3之间,具体最优值取决于实际应用场景的性能需求。例如,假设当前哈希表冲突率H=0.35,超过了预设的阈值Q=0.3,那么将触发重新初始化,新哈希表的容量可以是原来的两倍,例如从1024增加到2048。重新初始化哈希表后,所有节点重新计算散列值并插入新的哈希表中,从而降低冲突率,提高系统的性能和稳定性。

[0138] 通过上述步骤,可以在调试Linux内核时高效地管理和查询节点及其属性,确保系统的运行效率和可靠性。

[0139] 步骤330,基于调试加载顺序,在目标调试环境中依次加载各内核模块。

[0140] 详细请参见图1所示实施例的步骤130,在此不再赘述。

[0141] 步骤340,监测内核模块的加载状态,并记录异常信息。

[0142] 详细请参见图1所示实施例的步骤140,在此不再赘述。

[0143] 步骤350,根据异常信息调整调试加载顺序,以使各内核模块基于调整后的调试加载顺序在目标调试环境进行内核调试。

[0144] 详细请参见图1所示实施例的步骤150,在此不再赘述。

[0145] 本发明的内核调试方法,通过预先设定的内核模块加载规则对所有待加载的内核模块进行有序排列;接着,在目标调试环境中按照排序好的顺序逐个加载这些内核模块;在此过程中,系统会对每一个内核模块的加载情况进行实时监控,以便及时捕捉任何可能发生的异常情况,并对这些异常情况进行详细记录;最后,依据所记录的异常信息,进一步优化和调整内核模块的加载顺序,以此避免或解决由不当加载顺序引起的系统挂起等问题。

[0146] 该方法旨在通过系统的、有步骤的方式,解决由于内核模块加载顺序不当而导致的系统不稳定乃至崩溃的情况。首先,通过对内核模块加载顺序进行合理规划,确保了依赖关系明确的模块能够按照正确的顺序被加载,从而减少了因为依赖缺失导致的失败概率。然后,在实际加载过程中引入实时监测机制,不仅可以立即发现并记录潜在的加载失败,而且能为后续的问题诊断提供宝贵的第一手资料。最关键的是,基于这些实时记录下来的异常信息。同时,通过反馈机制来动态地调整加载顺序,不仅提高了调试效率,还能显著减少系统挂起的现象,从而保证Linux内核调试过程的平滑与高效。

[0147] 图4示出了本发明一种内核调试装置的实施例的结构示意图,应用于linux系统。如图4所示,该装置包括:

[0148] 关系获取模块410,用于获取linux系统中各内核模块的依赖关系;

[0149] 顺序确定模块420,用于基于依赖关系确定各内核模块的调试加载顺序;

[0150] 内核加载模块430,用于基于调试加载顺序,在目标调试环境中依次加载各内核模块;

[0151] 异常记录模块440,用于监测内核模块的加载状态,并记录异常信息;

[0152] 内核调试模块450,用于根据异常信息调整调试加载顺序,以使各内核模块基于调整后的调试加载顺序在目标调试环境进行内核调试。

[0153] 在一种可选的实施方式中,关系获取模块410包括:

[0154] 配置文件获取子模块,用于获取内核模块的配置文件,并基于配置文件得到各内核模块的模块名称;

[0155] 元数据读取子模块,用于基于模块名称读取对应的元数据;

[0156] 元数据分析子模块,用于分析元数据中的依赖字段,得到依赖模块名称;

[0157] 依赖关系获取子模块,用于基于依赖模块名称,获取linux系统中各内核模块的依赖关系。

[0158] 在一种可选的实施方式中,关系获取模块410还包括:

[0159] 定义块读取子模块,用于读取配置文件中各内核模块的定义块;

[0160] 模块名称提取子模块,用于基于定义块中的名称标签提取内核模块的模块名称。

[0161] 在一种可选的实施方式中,关系获取模块410还包括:

[0162] 冲突模块标记子模块,用于获取模块名称中的重复名称,并将名称重复的内核模块标记为冲突模块;

[0163] 冲突信息统计子模块,用于统计冲突模块的冲突位置及数量,并基于冲突位置及数量生成对应的错误报告及配置检查提醒信息,以提醒用户基于错误报告进行配置文件的

检查。

[0164] 在一种可选的实施方式中,冲突模块标记子模块,具体用于基于模块名称构建名称集合;通过哈希表记录名称集合中各模块名称出现的次数;若任意一模块名称的出现次数大于1,则将模块名称对应的内核模块标记为冲突模块。

[0165] 在一种可选的实施方式中,冲突信息统计子模块,具体用于将冲突模块的名称和出现位置汇总成冲突列表;基于冲突列表生成对应的错误内容;获取错误内容的字符数;若字符数大于字符阈值,则基于错误内容的生成错误摘要;基于错误摘要生成对应的错误报告及配置检查提醒信息,以提醒用户基于错误报告文件进行配置文件的检查。

[0166] 在一种可选的实施方式中,顺序确定模块420包括:

[0167] 依赖图构建子模块,用于基于依赖关系构建依赖图;

[0168] 依赖图排序子模块,用于通过拓扑排序算法对依赖图进行排序,得到依赖排序结果;

[0169] 加载顺序确定子模块,用于基于依赖排序结果确定各内核模块的调试加载顺序。

[0170] 在一种可选的实施方式中,依赖图构建子模块,具体用于创建有向无环图;基于各内核模块的模块名称在有向无环图中添加依赖节点;基于依赖关系在依赖节点之间添加有向边;基于依赖节点及有向边构建依赖图。

[0171] 在一种可选的实施方式中,依赖图构建子模块,具体用于若有向边的数量大于依赖节点的节点量平方根,则基于目标验证策略对依赖关系进行再次验证;基于验证后的依赖关系,对依赖节点及有向边进行调整,得到调整依赖节点及调整有向边;基于调整依赖节点及调整有向边,构建依赖图。

[0172] 上述各个模块和单元的更进一步的功能描述与上述对应方法实施例相同,在此不再赘述。

[0173] 通过上述装置及其组成部分,本发明实施例提供的技术方案具有如下优点:

[0174] 请参阅图5,图5是本发明可选实施例提供的一种计算机设备的结构示意图,如图5所示,该计算机设备包括:一个或多个处理器510、存储器520,以及用于连接各部件的接口,包括高速接口和低速接口。各个部件利用不同的总线互相通信连接,并且可以被安装在公共主板上或者根据需要以其它方式安装。处理器可以对在计算机设备内执行的指令进行处理,包括存储在存储器中或者存储器上以在外部输入/输出装置(诸如,耦合至接口的显示设备)上显示GUI的图形信息的指令。在一些可选的实施方式中,若需要,可以将多个处理器和/或多条总线与多个存储器和多个存储器一起使用。同样,可以连接多个计算机设备,各个设备提供部分必要的操作(例如,作为服务器阵列、一组刀片式服务器、或者多处理器系统)。图5中以一个处理器510为例。

[0175] 处理器510可以是中央处理器,网络处理器或其组合。其中,处理器510还可以进一步包括硬件芯片。上述硬件芯片可以是专用集成电路,可编程逻辑器件或其组合。上述可编程逻辑器件可以是复杂可编程逻辑器件,现场可编程逻辑门阵列,通用阵列逻辑或其任意组合。

[0176] 其中,存储器520存储有可由至少一个处理器510执行的指令,以使至少一个处理器510执行实现上述实施例示出的方法。

[0177] 存储器520可以包括存储程序区和存储数据区,其中,存储程序区可存储操作系

统、至少一个功能所需要的应用程序;存储数据区可存储根据一种小程序落地页的展现的计算机设备的使用所创建的数据等。此外,存储器520可以包括高速随机存取存储器,还可以包括非瞬时存储器,例如至少一个磁盘存储器件、闪存器件、或其他非瞬时固态存储器件。在一些可选的实施方式中,存储器520可选包括相对于处理器510远程设置的存储器,这些远程存储器可以通过网络连接至该计算机设备。上述网络的实例包括但不限于互联网、企业内部网、服务器集群、移动通信网及其组合。

[0178] 存储器520可以包括易失性存储器,例如,随机存取存储器;存储器也可以包括非易失性存储器,例如,快闪存储器,硬盘或固态硬盘;存储器520还可以包括上述种类的存储器的组合。

[0179] 该计算机设备还包括通信接口530,用于该计算机设备与其他设备或通信网络通信。

[0180] 本发明实施例还提供了一种计算机可读存储介质,存储介质存储有至少一可执行指令,该可执行指令在计算机设备/内核调试装置上运行时,使得计算机设备/内核调试装置执行上述任意方法实施例中的内核调试方法。

[0181] 本发明实施例还提供了一种计算机程序产品,包括计算机指令,计算机指令用于使计算机执行上述第一方面或其对应的任一实施方式的内核调试方法。

[0182] 在此提供的算法或显示不与任何特定计算机、虚拟系统或者其它设备固有相关。此外,本发明实施例也不针对任何特定编程语言。

[0183] 在此处所提供的说明书中,说明了大量具体细节。然而能够理解,本发明的实施例可以在没有这些具体细节的情况下实践。类似地,为了精简本发明并帮助理解各个发明方面中的一个或多个,在上面对本发明的示例性实施例的描述中,本发明实施例的各个特征有时被一起分组到单个实施例、图、或者对其的描述中。其中,遵循具体实施方式的权利要求书由此明确地并入该具体实施方式,其中每个权利要求本身都作为本发明的单独实施例。

[0184] 本领域技术人员可以理解,可以对实施例中的设备中的模块进行自适应性地改变并且把它们设置在与该实施例不同的一个或多个设备中。可以把实施例中的模块或单元或组件组合成一个模块或单元或组件,以及此外可以把它们分成多个子模块或子单元或子组件。除了这样的特征和/或过程或者单元中的至少一些是相互排斥之外。

[0185] 应该注意的是上述实施例对本发明进行说明而不是对本发明进行限制,并且本领域技术人员在不脱离所附权利要求的范围的情况下可设计出替换实施例。在权利要求中,不应将位于括号之间的任何参考符号构造成对权利要求的限制。单词“包含”不排除存在未列在权利要求中的元件或步骤。位于元件之前的单词“一”或“一个”不排除存在多个这样的元件。本发明可以借助于包括有若干不同元件的硬件以及借助于适当编程的计算机来实现。在列举了若干装置的单元权利要求中,这些装置中的若干个可以是通过同一个硬件项来具体体现。单词第一、第二、以及第三等的使用不表示任何顺序。可将这些单词解释为名称。上述实施例中的步骤,除有特殊说明外,不应理解为对执行顺序的限定。

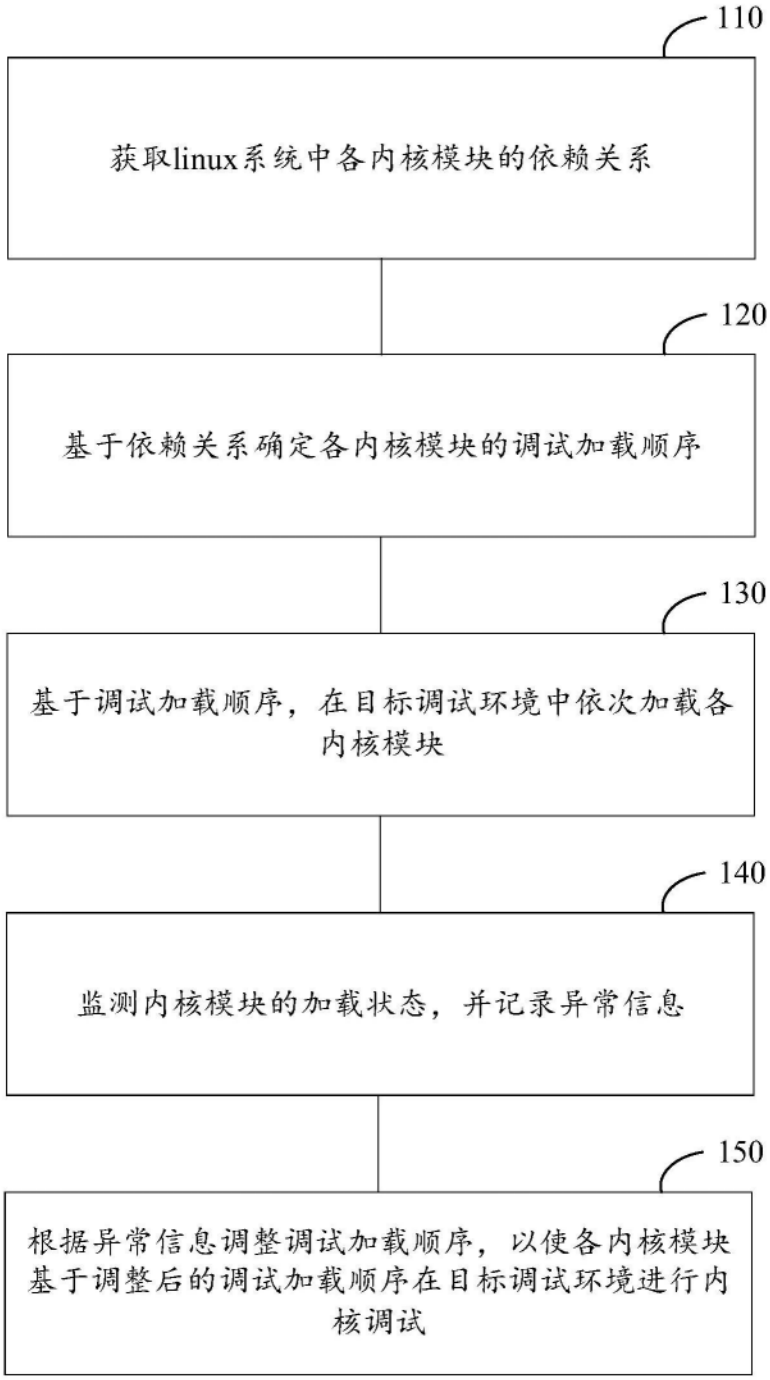

110
获取linux系统中各内核模块的依赖关系
120
基于依赖关系确定各内核模块的调试加载顺序
130
基于调试加载顺序，在目标调试环境中依次加载各内核模块
140
监测内核模块的加载状态，并记录异常信息
150
根据异常信息调整调试加载顺序，以使各内核模块基于调整后的调试加载顺序在目标调试环境进行内核调试

图1

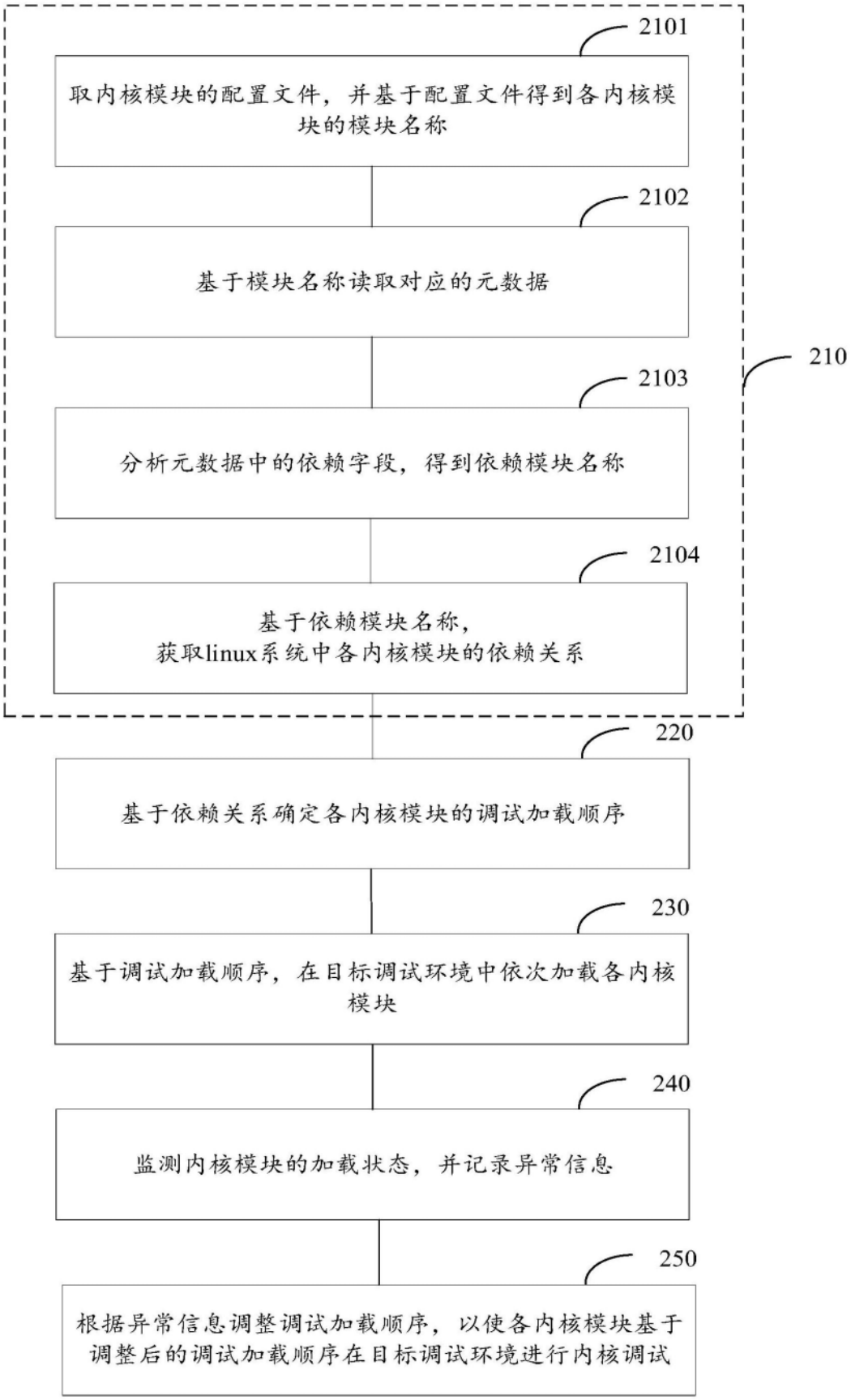

2101
取内核模块的配置文件，并基于配置文件得到各内核模块的模块名称
2102
基于模块名称读取对应的元数据
2103
分析元数据中的依赖字段，得到依赖模块名称
2104
基于依赖模块名称，
获取linux系统中各内核模块的依赖关系
210
220
基于依赖关系确定各内核模块的调试加载顺序
230
基于调试加载顺序，在目标调试环境中依次加载各内核模块
240
监测内核模块的加载状态，并记录异常信息
250
根据异常信息调整调试加载顺序，以使各内核模块基于调整后的调试加载顺序在目标调试环境进行内核调试

图2

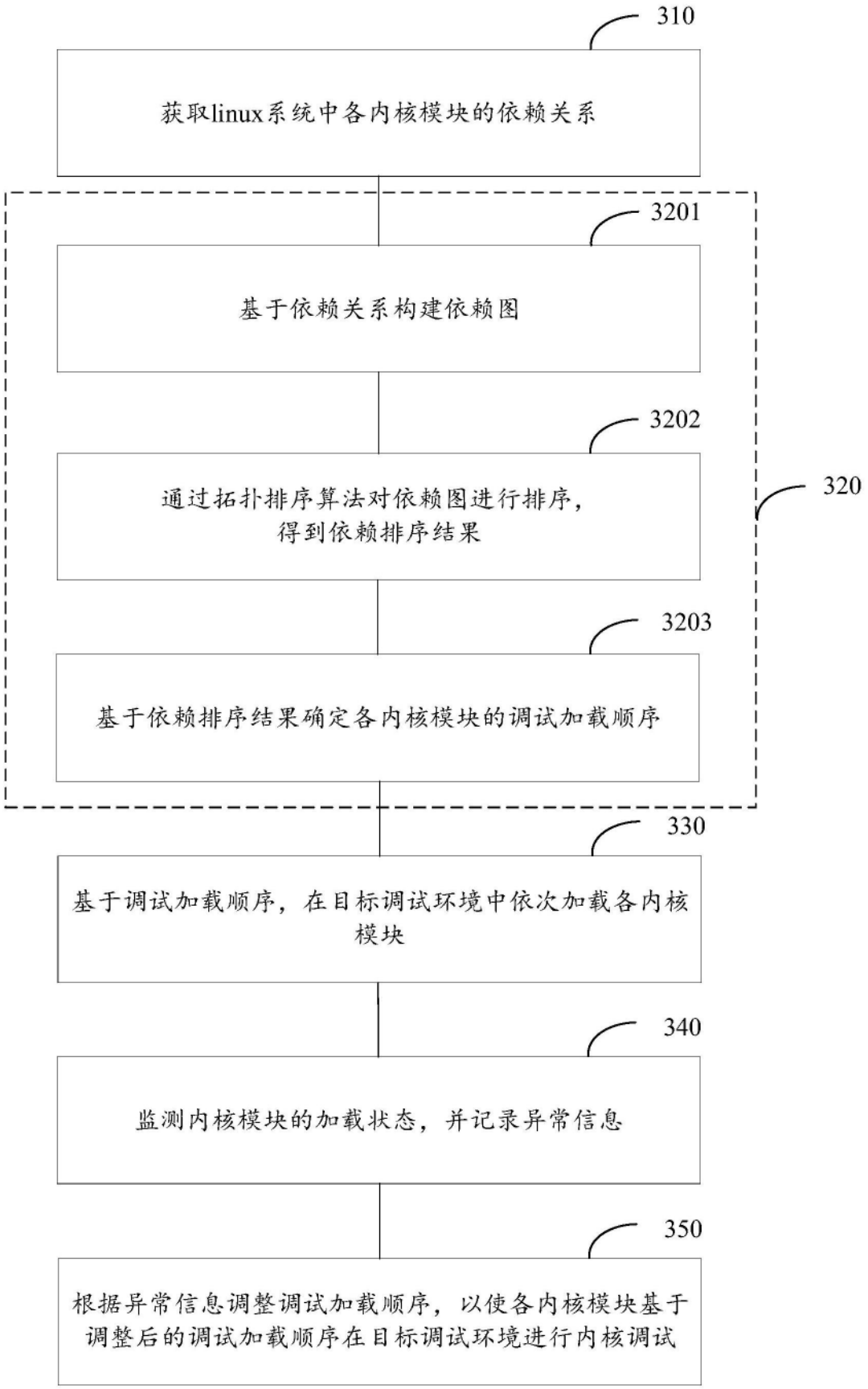

310
获取linux系统中各内核模块的依赖关系
320
3201
基于依赖关系构建依赖图
3202
通过拓扑排序算法对依赖图进行排序，
得到依赖排序结果
3203
基于依赖排序结果确定各内核模块的调试加载顺序
330
基于调试加载顺序，在目标调试环境中依次加载各内核模块
340
监测内核模块的加载状态，并记录异常信息
350
根据异常信息调整调试加载顺序，以使各内核模块基于调整后的调试加载顺序在目标调试环境进行内核调试

图3

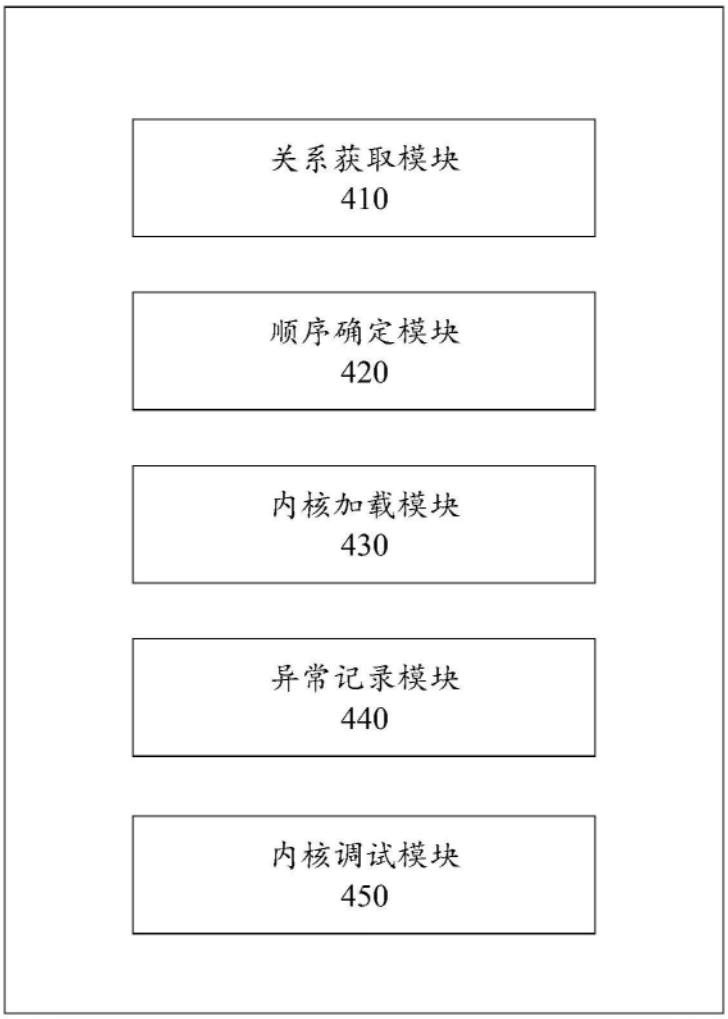

关系获取模块
410
顺序确定模块
420
内核加载模块
430
异常记录模块
440
内核调试模块
450

图4

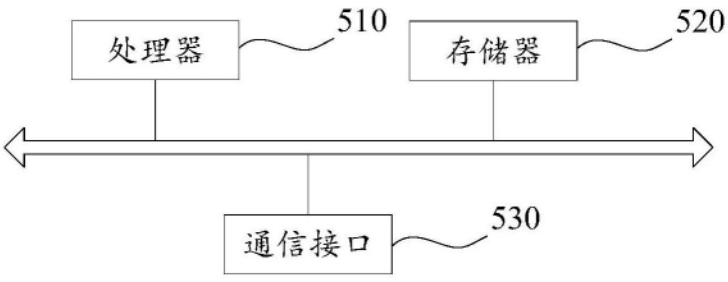

处理器
510
存储器
520
通信接口
530

图5