



(12) 发明专利

(10) 授权公告号 CN 103109276 B

(45) 授权公告日 2016. 01. 20

(21) 申请号 201080067546. 9

(22) 申请日 2010. 09. 07

(85) PCT国际申请进入国家阶段日
2012. 12. 19

(86) PCT国际申请的申请数据
PCT/KR2010/006068 2010. 09. 07

(87) PCT国际申请的公布数据
W02012/033237 KO 2012. 03. 15

(73) 专利权人 现代自动车株式会社
地址 韩国首尔
专利权人 起亚自动车株式会社
梨花女子大学校产学协力团

(72) 发明人 崔炳珠 徐珠瑛 梁承完 吴定锡

(74) 专利代理机构 北京尚诚知识产权代理有限公司 11322

代理人 龙淳

(51) Int. Cl.
G06F 11/36(2006. 01)
G06F 11/30(2006. 01)

(56) 对比文件
CN 102959519 A, 2013. 03. 06, 权利要求

1-20, 说明书第 1 页第 6 段至第 2 页第 12 段, 第 2 页第 17 段至第 7 页第 83 段.

CN 101398780 A, 2009. 04. 01, 权利要求第 1-7 项, 说明书第 2 页第 2 段至第 3 页第 3 段, 第 4 页第 3-13 段, 第 5 页倒数第二段至第 11 页最后一段, 图 1-6.

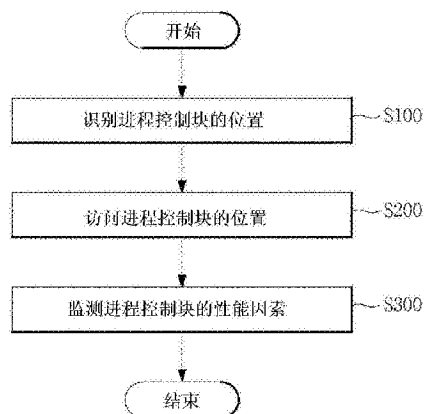
KR 20090081749 A, 2009. 07. 29, 全文.
KR 20080079343 A, 2008. 09. 01, 全文.
KR 20090001897 A, 2009. 01. 09, 全文.
KR 20030041612 A, 2003. 05. 27, 全文.

审查员 崔岩

权利要求书1页 说明书6页 附图3页

(54) 发明名称
系统测试方法

(57) 摘要
本发明公开了一种测试系统和方法, 其配置成识别进程控制块的位置, 访问进程控制块的位置, 以及监测与进程控制块有关的性能因素。



1. 一种测试系统性能的方法,包括:

通过处理器根据关于进程的进程控制块的基址识别所述进程控制块的位置,所述基址在特定的存储器空间中进行管理或者在特定的固定地址进行管理,并且被计算为栈指针;

通过所述处理器在使用虚拟驱动程序访问内核存储器以便内核的地址空间能够共享时,访问所述进程控制块的位置;以及

通过所述处理器在循环所述进程控制块的进程和线程列表时,监测与所述进程控制块有关的性能因素。

2. 根据权利要求 1 所述的方法,其中所述进程控制块设置在所述内核存储器中。

3. 根据权利要求 1 所述的方法,其中监测所述进程控制块的性能因素的处理包括在循环所述进程控制块的进程和线程列表时以预定的时间间隔测量所述进程控制块的性能因素。

4. 根据权利要求 2 所述的方法,其中监测所述进程控制块的性能因素的处理包括测量所述进程控制块的进程和线程的性能因素。

5. 根据权利要求 4 所述的方法,其中所述进程控制块的性能因素包括处理器使用率、存储器使用率和缺页中的一个或多个。

6. 根据权利要求 5 所述的方法,其中所述进程控制块的进程的性能因素包括标识符(ID)、进程状态、进程优先级、堆使用率、进程运行时间、使用时间和内核时间中的一个或多个。

7. 根据权利要求 5 所述的方法,其中所述进程控制块的线程的性能因素包括 ID、运行状态、基本优先级、当前优先级、使用时间、内核时间和调用栈中的一个或多个。

8. 根据权利要求 1 所述的方法,其中所述进程控制块存储与进程 ID、寄存器上下文、进程地址、存储器使用率、共享函数列表、进程资源、进程优先级和进程状态有关的一条或多条信息。

9. 根据权利要求 1 所述的方法,其中监测所述进程控制块的性能因素的处理包括通过分析代码插入内核中来监测所述进程控制块的性能因素。

10. 根据权利要求 1 所述的方法,其中监测所述进程控制块的性能因素的处理包括使用模拟程序监测所述进程控制块的性能因素。

11. 根据权利要求 1 所述的方法,其中监测所述进程控制块的性能因素的处理包括使用包括寄存器的硬件监测所述进程控制块的性能因素。

12. 一种测试系统性能的系统,包括:

主机系统,其包括:

配置成存储一个或多个数据结构的存储器;和

处理器,所述处理器配置成根据关于进程的进程控制块的基址识别所述进程控制块的位置,所述基址在特定的存储器空间中进行管理或者在特定的固定地址进行管理,并且被计算为栈指针,所述处理器还配置成在使用虚拟驱动程序访问内核存储器以便内核的地址空间能够共享时,访问所述进程控制块的位置,以及在循环所述进程控制块的进程和线程列表时,监测与所述进程控制块有关的性能因素。

13. 根据权利要求 12 所述的系统,其中所述进程控制块设置在所述内核存储器中。

系统测试方法

技术领域

[0001] 本发明涉及测试系统和方法。

背景技术

[0002] 近来,电子控制技术即使在如汽车业的机械行业也得到越来越多的关注。电子控制技术的核心是嵌入式软件,并且软件的质量对整个产品的质量有很大影响。

[0003] 过去已开发出促进质量的软件测试技术和测试工具。然而,传统软件测试技术和测试工具应用于嵌入式软件的测试,而不考虑嵌入式软件具体独特的特性。

[0004] 首先,嵌入式软件是为特定硬件和功能而定制和制作的。大多数嵌入式软件被设计成对于例如存储器的可用系统资源不足并运行的限制性条件进行优化。因此,与在主机环境中运行的传统软件相比,在目标环境中运行的嵌入式软件在系统实际运行时具有非常有限的资源。

[0005] 其次,嵌入式软件主要由例如电子信号或通信协议的各种类型的外部输入运行,而不是由例如菜单选择的用户命令运行。因此存在的问题在于,主要响应于用户命令运行的传统软件测试技术和测试工具应用于嵌入式软件的测试而未考虑这些特性。

[0006] 再次,在传统软件测试技术中,仅在测试期间存在持续的高处理器使用率时,这被考虑为指示性能瓶颈的症状。因此,即使组件的性能已跌至基本性能水平之下,也不认为组件使用率是异常症状。因此,存在系统性能不能精确测试的问题。

[0007] 发明内容

[0008] [技术问题]

[0009] 本发明的实施例涉及与测试系统和方法有关的技术,其配置成在不影响系统运行环境的情况下,使用系统运行所必须的最少量的资源采集用于识别性能瓶颈、瓶颈的原因和瓶颈发生位置的数据。

[0010] [技术方案]

[0011] 根据本发明的一个示例性实施例,本发明提供了一种测试系统和方法,包括识别进程控制块的位置,访问进程控制块的位置,以及监测进程控制块的性能因素。

[0012] [有利效果]

[0013] 有利地,本文说明的测试系统和方法可配置成在不影响系统运行环境的情况下,使用系统运行所必须的最少量的资源。

[0014] 此外,本发明的示例性实施例还可基于来自系统外部的输入测试系统,并且确定处理器的使用率是否已跌至基本水平之下,该情况表示已形成瓶颈。

附图说明

[0015] 图1是示出根据本发明的实施例的系统测试方法的流程图;

[0016] 图2示出存储在计算机可读介质上的、配置成实现根据本发明的示例性实施例的测试系统和方法的程序指令;并且

[0017] 图 3 是示出用于执行根据本发明的示例性实施例的测试系统和方法的系统的示例性构造的图。

具体实施方式

[0018] 本发明的系统测试方法包括识别进程控制块位置、访问进程控制块位置和监测进程控制块的性能因素。

[0019] [发明方式]

[0020] 以下将参照附图更加充分地说明本发明的示例性实施例。

[0021] 在根据本发明的系统测试方法中,进程控制块是指配置成管理与系统中由操作系统(OS)运行的进程的执行的有关的信息的数据结构。

[0022] 例如,进程控制块可表示实时管理与系统运行进程的近期执行有关的信息的内核的数据结构。

[0023] 根据本发明的系统测试方法通过进程控制块与存储器相关的函数表挂钩(hook),以便检测系统的存储器中发生的缺陷。

[0024] 具体地,根据本发明的示例性实施例的系统测试方法可破解(hack)例如缺页率和处理器使用率的系统执行信息,并基于系统执行信息分析系统性能,以便分析进程控制块的数据中的性能瓶颈和瓶颈原因。

[0025] 进程控制块的破解能够使与传统测试过程相关的系统性能的降低最小化,因为其仅专注于采集进程控制块的性能分析所必须的数据。

[0026] 此外,本发明能够使系统性能的降低最小化,同时满足系统运行环境中性能测试的要求。

[0027] 例如,根据本发明的系统测试方法可在以下条件下执行。

[0028] [系统测试条件]

[0029] 1. 测试范围:整个系统中的所有硬件和软件组件运行时的系统性能测试。

[0030] 2. 测试方法:以非重新编译、非重新链接和非重新执行的方式执行的运行时测试,以便保证系统执行方法。

[0031] 3. 测试目标:加载到系统上的二进制码的测试,其中不改变未包括如调试信息的附加代码的测试目标软件的原始源代码/二进制码。

[0032] 4. 性能数据:采集性能数据,其中不仅考虑处理器,而且考虑存储器、I/O 设备和网络资源,以便确定性能瓶颈的原因。

[0033] 5. 追踪瓶颈位置:采集如函数的能够实现源代码级分析的数据,以便确定性能瓶颈的位置。

[0034] 6. 硬件独立性:在与无附加硬件或不依赖附加硬件进行测试的系统运行环境相同的硬件条件下的测试。

[0035] 7. 软件独立性:在与不使用测试内核(instrumented kernel)或虚拟机进行测试的系统运行环境相同的硬件条件下的测试。

[0036] 8. 性能延迟率:使因测试导致的系统性能延迟率最小化,以便保证实时运行。

[0037] 9. 代码添加率:使因测试导致的代码添加率最小化,以便在有限的资源内运行。

[0038] 然而应该注意的是,条件可根据系统运行环境而变化,并且说明性实施例可应用

于各种系统运行环境,而不限于上述系统运行环境。

[0039] 进程控制块是包括与系统中运行的进程的执行有关的信息的内核数据结构。进程控制块还包括用于识别如可用存储器大小的性能瓶颈和性能因素的处理器使用率统计。内核保持处理器使用率和性能因素的最新值。

[0040] 根据本发明的示例性实施例的测试系统和方法可用于破解数值的代理开发的性能分析。以下说明分析性能瓶颈的原因所必须的性能因素和与性能因素相关的进程控制块的结构。

[0041] 本发明的说明性实施例中的性能是指在给定系统的限制条件下系统或组件执行功能的程度。性能测试是关于系统是否满足特定性能要求的评估。

[0042] 通过经性能测试分析性能瓶颈及其原因并解决其中的性能瓶颈,能够改进系统性能。

[0043] 系统的性能瓶颈是因争夺如存储器、I/O 设备和网络带宽等有限资源而导致系统性能恶化的现象。系统性能瓶颈可由各种来源或原因导致,例如资源短缺、争夺共享资源、资源的单独占有、资源的错误配置和资源的错误运行。

[0044] 例如,当可用存储器不足时,可发生由存储器减少导致的性能瓶颈。能够识别存储器不足症状的核心存储器性能因素可包括缺页和存储器使用率。

[0045] 根据本发明的示例性实施例的测试系统和方法可识别多种性能瓶颈。

[0046] 首先,根据本发明的示例性实施例的测试系统和方法可基于缺页确定已发生存储器瓶颈。

[0047] 例如,当缺页计数高于正常值时,这可对应于存储器瓶颈。缺页是当程序试图访问存在于其地址空间但目前不存在于系统存储器中的数据或代码时发生的现象。当缺页发生时,OS 取出存储器中的数据或代码,并使程序能够继续运行,如同缺页未发生一样。OS 对缺页的异常处理延迟了应用程序的处理时间并影响整个系统性能。

[0048] 此外,根据本发明的示例性实施例的测试系统和方法可通过存储器使用率识别性能瓶颈。系统存储器包括物理存储器使用率和虚拟存储器使用率,并且还可包括用于各进程的堆存储器(heap memory)使用率。根据本发明的示例性实施例的系统测试方法基于存储器使用率之和确定已发生性能瓶颈。

[0049] 此外,根据本发明的示例性实施例的测试系统和方法可基于处理器使用率(或 CPU 使用率)识别性能瓶颈。

[0050] 例如,根据本发明的测试系统和方法可在处理器使用率高于正常值并有可用存储器时确定 CPU 存在瓶颈。相反,当处理器使用率高于正常值且存储器用尽时,性能问题可能由存储器瓶颈而非 CPU 瓶颈导致。

[0051] 根据本发明的示例性实施例的测试系统和方法还可基于进程使用率识别性能瓶颈。进程使用率是指在系统的执行时间中从整个 CPU 使用率中排除空闲时间后的时间。

[0052] 此外,根据本发明的示例性实施例的测试系统和方法可基于用户时间确定性能瓶颈。用户时间是指执行停留在用户空间内的时间。也就是,执行应用程序所用的时间。

[0053] 另外,根据本发明的示例性实施例的测试系统和方法可基于内核时间确定性能瓶颈。内核时间是指执行停留在内核空间内的时间。也就是,内核处理服务所用的时间。

[0054] 进程控制块是管理 OS 内核以便运行时控制进程的数据结构。一般地,例如进程

ID、寄存器上下文、进程的地址空间、进程的存储器使用率、共享函数列表、进程共享的资源、进程的优先级和状态等的执行信息存储在进程控制块中。

[0055] 图 1 是示出根据本发明的实施例的测试系统和方法的流程图。

[0056] 图 2 示出存储在例如磁盘或任何其它存储设备的非短暂性计算机可读介质上的、用于实现根据本发明的示例性实施例的测试系统和方法的程序指令。

[0057] 参照图 1, 根据本发明的示例性实施例的测试系统和方法包括: 在步骤 S100 通过配置成执行该方法的处理器识别进程控制块的位置, 在步骤 S200 通过处理器访问进程控制块, 以及在步骤 S300 通过处理器监测进程控制块的性能因素。

[0058] 在步骤 S100 识别进程控制块的位置的处理中, 当进程产生或进程消失时, 因为进程控制块产生或进程控制块消失, 所以不确定进程控制块存储的位置。

[0059] 然而, 与占用处理器(例如 CPU)的当前进程的进程控制块有关的信息可以是已知的, 因为关于当前进程的进程控制块的基址在特定的存储器空间中进行管理或者在特定的固定地址进行管理。

[0060] 例如, 在图 2 的程序指令中, 如在第 7 行, 关于当前进程的进程控制块的基址可被计算为栈指针, 并且如在第 31 行, 所有进程的进程控制块可被访问。

[0061] 在步骤 S200 访问进程控制块的位置的处理中, 进程控制块被置于内核的存储器空间中。

[0062] 在这种情况下, 考虑到安全, 可不允许访问内核的存储器空间。如果访问内核的存储器空间如上所述被阻止, 那么伪码可以虚拟驱动程序的形式实现, 以便内核的地址空间能够共享, 如图 2 的程序代码的下面几行中那样。因此, 置于内核的存储器空间中的进程控制块可以被访问。

[0063] 在步骤 S300 监测程序控制块的性能因素的处理中, 如在图 2 的程序代码的第 61 行和第 65 行, 可使用定时器中断并可以预定的时间间隔(例如, 1 秒或 100 毫秒)测量性能数据。

[0064] 此外, 如在图 2 的程序代码的第 31 行和第 33 行, 根据本发明的示例性实施例的测试系统和方法可在循环进程控制块的进程和线程列表时测量性能因素, 以便不仅分析系统单元中的系统性能, 而且分析进程或线程单元中的系统性能。

[0065] 在这种情况下, 可为各个线程存储调用栈信息, 以便精确地追踪系统性能瓶颈发生的位置。为每次监测测量的性能因素如下。

[0066] 进程控制块的性能因素可包括处理器使用率、存储器使用率和缺页中的一个或多个。

[0067] 进程控制块的进程性能因素可包括标识符(ID)、进程状态、进程优先级、堆使用率、进程运行时间、使用时间和内核时间中的一个或多个。

[0068] 进程控制块的线程性能因素可包括 ID、运行状态、基本优先级、当前优先级、使用时间、内核时间和调用栈信息中的一个或多个。

[0069] 图 3 示出用于执行根据本发明的测试系统和方法的系统的构造。

[0070] 参照图 3, 用于执行根据本发明的测试系统和方法的系统包括代理单元 120 和测试管理单元 210。

[0071] 代理单元 120 包括在目标系统 100 即测试目标中, 并配置成测量性能数据。代理

单元 120 执行图 1 中所示的算法。

[0072] 测试管理单元 210 包括在例如可运行如处理器的个人计算机(PC)的主机系统 200 中。测试管理单元 210 分析由代理单元 120 采集的性能数据,检测性能瓶颈,并基于分析的性能数据分析测试覆盖率。

[0073] 具体地,代理单元 120 安装在设置于运行环境中的目标系统 100 上,并且当目标系统响应于用户的测试开始或结束命令而运行时,其用于周期性地测量与系统性能有关的数据。

[0074] 例如,代理单元 120 可包括 PerfAgent.dll 和 PerfROBO.exe。

[0075] PerfAgent.dll 是实现图 1 算法的虚拟内核设备驱动程序。虚拟内核设备驱动程序为了性能测试而破解关于内核的进程控制块的信息。

[0076] PerfROBO.exe 用作测试服务器,用于控制是否响应于用户的测试开始或结束命令激活虚拟内核设备驱动程序。这两种驱动程序均可由处理器执行。

[0077] 当虚拟内核设备驱动程序(例如 PerfAgent.dll)响应于测试开始命令被激活时,测试服务器通过计时器设定而执行系统性能监测模块,并且当测试应用户请求结束时,测试服务器通过结束设定的计时器而终止测试。

[0078] 同时,测试管理单元 210 可在指定的存储介质中存储所采集的性能数据。测试管理单元 210 用于分析主机系统 200 的日志文件并基于分析的日志文件检测在运行时发生的性能瓶颈。

[0079] 例如,如果测试管理单元 210 以二进制码形式将性能数据存储在存储介质中,那么测试管理单元 210 可使用二进制执行代码和采集的测试目标的分析数据(profiling data)作为输入,并在输出测试覆盖范围和性能信息时基于调用栈信息显示关于问题功能位置的信息。

[0080] 在根据本发明的示例性实施例的测试系统和方法中,可使用各种监测系统性能的方法。

[0081] 首先,可使用将分析代码插入内核中的方法。在这种方法中,将分析代码静态或动态地插入内核中。

[0082] 在将分析代码动态地插入内核中的方法中,分析代码可在运行时插入内核代码中。在这种情况下,可不引导系统而插入用于采集性能分析所需数据的代码。

[0083] 在将分析代码静态地插入内核中的方法中,使用先前产生的内核。

[0084] 例如,用于监测重要系统性能因素的分析代码可嵌入内核中。在性能监测的情况下,通过先前插入内核中的分析代码接收性能数据。在这种情况下,当监测性能时,可测量多种系统性能因素,包括处理器使用率和存储器使用率。

[0085] 此外,使用模拟器的方法可用作性能监测方法。该方法有助于在开发的早期阶段检查系统性能。

[0086] 此外,硬件可用作性能监测方法。硬件性能因素是指一组特别制作的寄存器。如果使用硬件性能因素,那么可使用比基于软件的性能因素更低的系统开销监测与 CPU、缓存和存储器相关的性能。在这种方法中,源代码或二进制码无需修改。

[0087] 虽然已结合目前被认为实用的示例性实施例说明了本发明,但是应该理解的是,本发明并不局限于所公开的实施例,而是相反,意在涵盖包括在所附权利要求的精神和范

围内的各种改型和等同配置

[0088] 工业适用性

[0089] 本发明的示例性实施例配置成仅使用必须的资源测试系统,而不影响系统的运行环境。

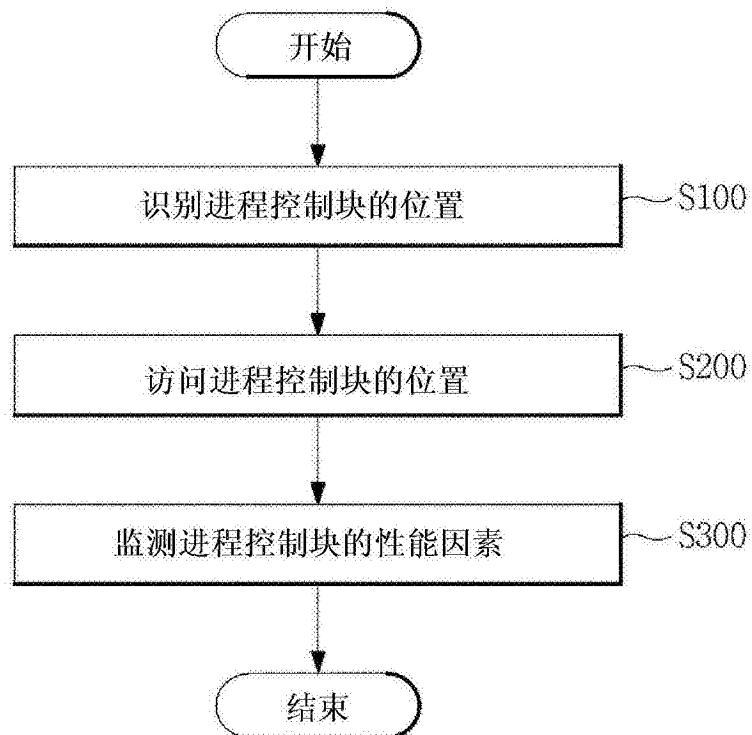


图 1

```

01 void * pPCB; // PCB information
02 int uTimerId; // Timer ID for performance monitoring
03 int uTimeOut = 1000; // Time-out value in milliseconds
04 DWORD GetKernelStackPoint(); // API for getting value of kernel stack point
05 int SetTimer(uElapse, lpTimerFunc); // API for creating a timer with the specified time-out value
06 void KillTimer(uTimerId); // API for destroying the specified timer
07 void SaveLog(data, ...); // API for saving data into log-file

08 void PERFAGENT_GetPCB(void)
09 { // Accessing the PCB memory space
10     pPCB = GetKernelStackPoint() & 0xFFFFE000; // e.g., in case of WinCE platform
11 }

12 void PERFAGENT_MemoryUsage()
13 { // Measuring memory usage
14     DWORD total = pPCB->NumPages * pPCB->SysPageSize;
15     DWORD unused = pPCB->FreePhyPage * pPCB->SysPageSize;
16     DWORD usage = total - unused;
17     Save(usage);
18 }

19 void PERFAGENT_HeapUsage(pProc)
20 { // Measuring heap usage
21     DWORD usage ← calculate heap usage using pProc->memPointer;
22     Save(usage);
23 }

24 void PERFAGENT_CallStack(pThread)
25 { // Measuring thread's call stack
26     void * site ← dump callstack using pThread->pestkTop;
27     Save(site);
28 }

29 void PERFAGENT_PCBHooking()
30 { // PCB backing
31     PPROCESS pProc = pPCB->procList.pFwd;
32     while (pProc) {
33         PTHREAD pThread = pProc->thrdList.pFwd;
34         while (pThread) {
35             // save thread status
36             Save(pThread->dwId);
37             Save(pThread->bBPrio, pThread->bCPrio);
38             Save(pThread->bRunState, pThread->bWaitState);
39             Save(pThread->dwUTime);
40             Save(pThread->dwKTime);
41             PERFAGENT_CallStack(pThread);
42             pThread = pThread->pFwd; // next thread
43         }
44         // save process status
45         Save(pProc->dwId);
46         Save(pProc->bPrio);
47         Save(pProc->bState);
48         Save(pProc->dwProcTime);
49         PERFAGENT_HeapUsage(pProc);
50         pProc = pProc->pFwd; // next process
51     }
52     // save system status
53     Save(pPCB->dwProcTime);
54     Save(pPCB->nPageFault);
55     PERFAGENT_MemoryUsage();
56 }

57 // Performance Agent
58 void PERFAGENT_Start(void)
59 {
60     PERFAGENT_GetPCB();
61     uTimerId = SetTimer(uTimeOut, PERFAGENT_PCBHooking);
62 }

63 void PERFAGENT_Stop(void)
64 {
65     KillTimer(uTimerId);
66 }

```

图 2

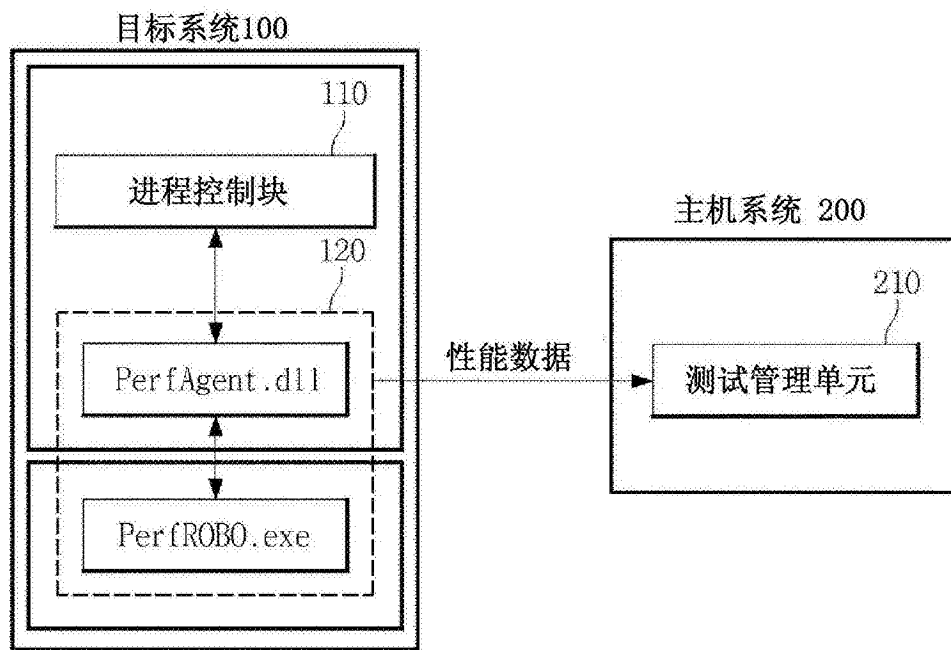


图 3