

[19] 中华人民共和国国家知识产权局

[51] Int. Cl.



# [12] 发明专利申请公布说明书

G06F 9/48 (2006.01)  
G06F 9/455 (2006.01)  
G06F 9/54 (2006.01)

[21] 申请号 200710300437.7

[43] 公开日 2008年7月2日

[11] 公开号 CN 101211277A

[22] 申请日 2007.12.27

[21] 申请号 200710300437.7

[30] 优先权

[32] 2006.12.27 [33] JP [31] 352948/2006

[71] 申请人 株式会社东芝

地址 日本东京都

共同申请人 东芝解决方案株式会社

[72] 发明人 水野聪

[74] 专利代理机构 北京市中咨律师事务所  
代理人 杨晓光 李 峥

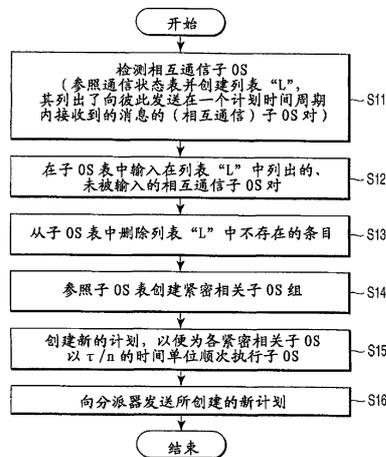
权利要求书 3 页 说明书 21 页 附图 12 页

## [54] 发明名称

选择子 OS 的一个执行计划的方法与使用该方法的虚拟机监视器

## [57] 摘要

本发明涉及选择子 OS 的一个执行计划的方法以及使用该方法的虚拟机监视器。在虚拟机系统中，多个子 OS 在由虚拟机监视器 (VMM) (31) 提供的虚拟机执行环境下分时执行。VMM (31) 指定经由由 VMM (31) 提供的通信接口彼此相互通信的子 OS 组 (S11 - S14)。接着，VMM (31) 将包含在所述多个子 OS 且包含在所指定的子 OS 组中的各子 OS 的执行计划的时间片设置得较短 (S15)。



1. 一种在虚拟机系统中切换多个子 OS 的执行计划的方法, 在该系统中, 所述子 OS 在由虚拟机监视器提供的虚拟机执行环境下分时执行, 所述子 OS 中的任意两个在经由由所述虚拟机监视器提供的通信接口彼此相互通信的同时执行处理, 所述方法的特征在于包含:

通过所述虚拟机监视器指定经由所述通信接口彼此相互通信的子 OS 组; 以及

通过所述虚拟机监视器将包含在所指定的子 OS 组中的各个子 OS 的执行计划的时间片设置得较短。

2. 根据权利要求 1 的方法, 其特征在于: 在所述多个子 OS 中, 其相互通信在一个计划时间周期内完成的子 OS 对的集合被指定为所指定的子 OS 组。

3. 根据权利要求 1 的方法, 其特征在于: 当存在与各个子 OS 对中的一个子 OS 进行相互通信并在一个计划时间周期内完成相互通信的另一子 OS 时, 判断为所述另一子 OS 属于所指定的子 OS 组。

4. 根据权利要求 1 的方法, 其特征在于还包含:

通过所述虚拟机监视器控制所述指定与所述设置的重复, 所述重复以预定的计划时间周期执行;

每当所述子 OS 组被指定时, 通过所述虚拟机监视器计算第一平均通信时间, 所述第一平均通信时间为所指定的子 OS 组的子 OS 之间的相互通信需要的当前平均通信时间; 以及

通过所述虚拟机监视器将所述第一平均通信时间与第二平均通信时间进行比较, 所述第二平均通信时间在时间片被设置为比本次长一个等级时计算得出,

其中, 当所述第一平均通信时间短于所述第二平均通信时间时将所述时间片设置为比本次短一个等级, 当所述第一平均通信时间不短于所述第二平均通信时间时, 将所述时间片设置为比本次长一个等级。

5. 一种虚拟机监视器，其建立虚拟机执行环境，在所述虚拟机执行环境下，多个子 OS 分时执行，所述虚拟机监视器的特征在于包含：

通信接口，其被配置为通过所述子 OS 的请求在所述多个子 OS 的任意两个之间进行相互通信；

通信状态表，其保持通过所述通信接口在所述子 OS 之间执行的相互通信的状态；

计划器，其被配置为根据所述通信状态表指定使用所述通信接口彼此相互通信的子 OS 组，并创建新的计划，在该计划中，包含在所指定的子 OS 组中的各子 OS 的执行计划的时间片被设置得短；以及

分派器，其被配置为使得包含在所指定的子 OS 组中的子 OS 按照所述新计划被执行。

6. 根据权利要求 5 的虚拟机监视器，其特征在于：所述计划器将其相互通信在一个计划时间周期内完成的子 OS 对的集合指定为所述子 OS 组，所述子 OS 对的集合包含在所述多个子 OS 中。

7. 根据权利要求 6 的虚拟机监视器，其特征在于：当存在与各个子 OS 对中的一个子 OS 进行相互通信并在所述一个计划周期内完成相互通信的另一子 OS 时，所述计划器判断为所述另一子 OS 属于所指定的子 OS 组。

8. 根据权利要求 6 的虚拟机监视器，其特征在于：

对于经由所述通信接口彼此相互通信的各子 OS 对，所述通信状态表保持通信状态信息，该信息包括在比所述一个计划时间周期短的时间内接收的消息的数量；且

所述计划器在预定的计划时间周期参照所述通信状态表，并将与表示所述消息的数量不为零的通信状态信息对应的所有子 OS 对指定为包含在所述子 OS 组中的子 OS 对。

9. 根据权利要求 5 的虚拟机监视器，其特征在于：

保持在所述通信状态表中的通信状态信息包括平均发送时间值以及消息的数量，所述平均发送时间为从消息发送到其接收需要的平均时间；

所述计划器包含平均通信时间计算单元与比较单元，所述平均通信时间计算单元被配置为每当所述子 OS 组被指定时根据包含在所指定的子 OS 组中的所有子 OS 对、基于保持在所述通信状态表中的通信状态信息中的平均发送时间来计算第一平均通信时间，该时间为所指定的子 OS 组的所有子 OS 之间相互通信需要的当前平均通信时间，所述比较单元被配置为将所述第一平均通信时间与当时间片被设置得比本次长一个等级时计算得到的第二平均通信时间进行比较；且

当所述第一平均通信时间短于所述第二平均通信时间时，所述计划器将所述时间片设置得比本次短一个等级，当所述第一平均通信时间不短于所述第二平均通信时间时，将所述时间片设置得比本次长一个等级。

## 选择子 OS 的一个执行计划的方法与使用该方法的虚拟机监视器

### 技术领域

本发明涉及提供虚拟机执行环境的虚拟机系统。具体而言，本发明涉及基于子 OS 之间的通信状态选择子 OS（子操作系统）的一个执行计划（execution schedule）的方法以及使用该方法的虚拟机监视器，其中，子 OS 是在虚拟机执行环境下执行的。

### 背景技术

近来，如日本特开 No.2006-039763 所公开，已经在例如个人计算机等计算机系统中进行了使用虚拟机（VM）技术的研究和开发。实现虚拟机系统（VM 系统）的商业应用程序已经得到广泛使用。主框架已经包括了使用由硬件（HW）构成的虚拟机支持单元（VM 支持单元）的 VM 系统。

通常，个人计算机等计算机系统包含包括处理器（真实处理器）、多种输入/输出（I/O）单元（真实 I/O 单元）和存储器（真实存储器）在内的 HW（真实 HW）。在这种计算机系统中，操作系统（OS）被执行，且多种应用程序（应用）在 OS 上运行。虚拟机应用（VM 应用）作为应用之一为人们所知。

VM 应用包括虚拟机监视器（VMM）。VMM 通过运行 VM 应用实现。VMM 也被称为虚拟机管理器。VMM 管理 VM 系统，并构成虚拟硬件单元（虚拟 HW 单元）。虚拟 HW 单元包含虚拟硬件（虚拟 HW），例如虚拟处理器、虚拟 I/O 单元、虚拟存储器单元（虚拟存储器）等等。VMM 在逻辑上对虚拟 HW 进行仿真，以便实现虚拟机执行环境或用于虚拟机系统的执行环境（虚拟机系统环境）。环境与适合作为子 OS 的 OS 一起装载，且 OS（子 OS）被操作。

VMM 将子 OS 的代码（执行代码）展开到作为虚拟 HW 的存储器单

元中，并以虚拟处理器仿真的形式对执行代码进行执行。当 VMM 仿真虚拟 I/O 单元时，来自子 OS 的输入/输出请求（I/O 请求）被处理。

如上所述，VMM 可在通用计算机系统中建立虚拟机执行环境（虚拟机系统环境），以便在其中执行多个子 OS。如上所述，VMM 被实现为在 OS（主 OS）上运行的一个应用。

VMM 在计算机系统的真实 HW 上实现也是已知的。这种 VMM 被称为 VMM 单元，并在本质上作为 OS。VMM 单元对真实处理器、真实 I/O 单元等真实 HW 进行虚拟化，并作为功能向各个子 OS 提供虚拟化 HW。VMM 单元在其上建立虚拟机执行环境。VMM 单元对虚拟处理器、虚拟 I/O 单元等虚拟 HW 进行仿真，或根据时间和区域向虚拟 HW 分配真实处理器、真实 I/O 单元、真实存储器单元等真实 HW（HW 资源）。VMM 单元因此建立起虚拟机执行环境。子 OS 被装载到虚拟机执行环境之中，并由虚拟处理器执行。

虚拟机执行环境中子 OS 在其下被执行的环境被称为子 OS 执行环境。通常，在虚拟机系统中执行多个子 OS。用于在子 OS 之间进行通信的通信接口对于每个子 OS 在其中被执行的执行环境来说是有用的。VMM 因此提供这种通信接口。

通信接口的典型功能如下：

- (1) 子 OS 之间的中断（interrupt）功能
- (2) 子 OS 之间共享存储器功能
- (3) 子 OS 之间的消息传送功能

功能（1）为用于从一个子 OS 向另一个指定子 OS 发送中断的机制。接收子 OS 具有用于通过例如中断因数信息来检测哪一子 OS 发送该中断的装置。

功能（2）为用于在特定子 OS 之间共享存储器空间的机制。发送子 OS 将数据写到存储器空间（共享存储器空间），接着，接收子 OS 读取该数据。数据传送因此能够执行。为了数据传送的同步，只要使用上述功能（1）的机制。

为了实现功能(3),发送子 OS 必须指定有待发送的数据(发送数据)和目标子 OS(接收子 OS),并请求 VMM 传送数据。VMM 将指定数据复制到目标子 OS 的接收缓冲器,接着,设置到目标子 OS 的接收中断。

如果功能(1)与(2)一起使用或单独使用功能(3),可实现基本的子 OS 通信。

在现有技术中的虚拟机系统中,如果上述通信接口被用于初级机制,可在由 VMM 提供的虚拟机执行环境中所执行的子 OS 之间实现例如传输控制协议/网间协议(TCP/IP)等通信协议。因此,可以想到,多个子 OS 通过上述机制彼此进行通信,并相互协作地提供服务。例如,直接连接到外部通信通道(外部网络)的子 OS 作为防火墙(FW),另一子 OS 经由虚拟网络连接到此子 OS。

然而,当子 OS 在彼此进行通信的同时执行处理时,存在这样的情况:通信在效率上没有改善。这种情况将在下文中阐释。首先假设四个子 OS #A、#B、#C、#D 在由 VMM 构建的虚拟机执行环境下运行,且四 OS 中的子 OS #A 与#C 在相互通信的同时执行处理。具体而言,假设子 OS #A 中的进程#a 与子 OS #C 中的进程#c 在相互之间发送/接收数据的同时执行处理。

VMM 支持用于在子 OS 之间进行通信的上述三个功能(1)到(3)。子 OS #A 与#C 使用这些功能彼此通信。这里假设进程#a 与#c 请求它们各自的子 OS 经由 TCP/IP 接口发送/接收数据。在这种情况下,各个子 OS 接收和处理请求,并经由由 VMM 提供的通信接口向另一子 OS 发送消息。通信接口例如为在子 OS 之间传送消息的功能。

假设在子 OS #A 与#C 之间重复下面四个处理。

(1) 子 OS #A(进程#a)向子 OS #C 发送三个消息(网络数据包)(处理 a1)。

(2) 子 OS #C(进程#c)接收来自子 OS #A 的三个消息并对它们进行处理(处理 c1)。

(3) 子 OS #C(进程#c)由所处理的消息产生新的消息并将之发送到

子 OS #A (处理 c2)。

(4) 子 OS #A (进程#a) 接收来自子 OS #C 的新消息并对之进行处理 (处理 a2)。

上述四个处理重复时执行的操作顺序如图 4A 所示。在图 4A 中, 处理 a1 与 a2 的组合用处理 a 表示, 处理 c1 与 c2 的组合用处理 c 表示。在图 4A 中, 水平轴表示过去的的时间。

在图 4A 中, “ $\tau$ ” 表示由 VMM 向各子 OS 分配处理器 (CPU) 的时间单位, 并通常称为时间片 (time quantum)。为简明起见, 假设在图 4A 的实例中没有一个子 OS 在时间片  $\tau$  中途被切换。具体而言, 分别在从  $t_{n+1}$ 、 $t_{n+2}$ 、 $t_{n+3}$ 、 $t_{n+4}$  开始的时间片  $\tau$  中将处理器分配给子 OS #A、#B、#C、#D。类似地, 分别在从  $t_{n+5}$ 、 $t_{n+6}$ 、 $t_{n+7}$ 、 $t_{n+8}$  开始的时间片  $\tau$  中将处理器分配给子 OS #A、#B、#C、#D。然而, 实际上, 通过中断等事件, 子 OS 在时间片  $\tau$  中途被切换。

在图 4A 中, 向下的箭头表示发送消息, 向上的箭头表示接收消息。箭头的数量对应于消息的数量。附着到各向上箭头 (接收消息) 头部的包含 “ $\tau$ ” 的字符串意味着其对应的消息在该字符串表示的时间内被发送。包含 “ $\tau$ ” 的字符串下面被括住的符号意味着其对应的消息由该符号表示的子 OS 发送。

在图 4A 的实例中, 如上所述, 处理器在时刻  $t_{n+1}$  被分派给子 OS #A, 以便启动子 OS #A。接着, 子 OS #A 的进程#a 开始执行处理 a1。因此, 三个消息被发送到子 OS #C。此后 (时刻  $t_x$  后), 直到至少一个消息从子 OS #C 返回之前, 进程#a 不能执行处理 (处理 a2)。即使消息从子 OS #C 返回, 如果子 OS #A 不可运行, 直到子 OS #A 变得可运行之前, 进程#a 不能执行处理 (处理 a2)。然而, 如果处理 a1 与 a2 的处理时间总和小于时间片  $\tau$ , 可在时间片  $\tau$  的剩余时间内执行子 OS #A 的进程#a 以外的进程。

VMM 将由子 OS #A 的进程#a 发送的消息发送到子 OS #C。然而, 此时, 子 OS #C 尚未启动运行。在图 4A 的实例中, 在时刻  $t_{n+3}$  后, 子 OS #C 能启动运行。因此, 在处理 a1 中被发送到子 OS #C 的消息不是马上由子

OS #C 接收，而是在时刻  $t_n+3$  时。

如果处理  $c_1$  与  $c_2$  所需的时间短于时间片  $\tau$ ，子 OS #C 的进程#c 执行处理  $c_1$  与  $c_2$ ，并向子 OS #A 发送一个消息。此后（时刻  $t_y$  之后），子 OS #C 的进程#c 的处理不能继续，但子 OS #C 中的另一进程被执行。当从  $t_n+1$  开始过去  $4\tau$  时，子 OS #A 能在时刻  $t_n+5$  时重新运行，以便接收来自子 OS #C 的消息并对之进行处理（处理  $a_2$ ）。因此，在  $4\tau$  的时间周期内执行一系列的处理  $a_1$ 、 $c_1$ 、 $c_2$ 、 $a_2$ 。从图 4A 显然可以看出，这种情况对于进程#a、#c 来说效率非常低下。

因此，可以想到，优先将处理器分派到受到未决（pending）中断的子 OS。如上所述，子 OS 之间的通信作为发送目标中的中断实现。使用这种分派技术，当通信消息被发送时，处理器被分派到目标子 OS。然而，每当消息被发送时，子 OS 被切换。因此，切换子 OS 的额外成本将大大增加。在上面介绍的如果多个消息未被发送则进程#c 不执行下一处理的情况下，系统性能可能下降，如将在下面介绍的那样。

如果控制仅在进程#a 的第一个消息发送中被传递，进程#c 不能完成处理  $c_1$ ，因为另外两个消息未被发送。换句话说，进程#c 不得不在子 OS #A 被重新计划为向进程#c 发送另外的（两个）消息之前一直等待。作为由于中断而切换子 OS 的结果，系统性能可能根据环境而劣化。

可以想到，作为另一种技术，将时间片  $\tau$  设置为较小的值。该值越小，执行每个子 OS 的机会越多。在这种情况下，消息被发送，接着，消息被发送到的目标子 OS 在短时间内变得可运行。因此，消息接收处理得以早地执行。然而，如果时间片  $\tau$  减小，用于切换子 OS 的成本增加，整个系统的效率降低。

通常，时间片  $\tau$  被设置为最优值，使得目标系统的响应时间令人满意，且用于选择子 OS 的成本低（该成本对系统没有不良影响）。为了获得令人满意的响应时间，时间片  $\tau$  应被设置为小的值，并且，为了得到低的成本，其应被设置为大的值。因此，如果最初被设置为最优值的时间片  $\tau$  被减小，存在整个系统的效率下降的担心。

## 发明内容

本发明的一个目标在于当子 OS 执行处理时改进在子 OS 之间执行的通信的效率。

根据本发明一实施例，提供了一种在虚拟机系统中切换多个子 OS 的执行计划的方法，其中，子 OS 在由虚拟机监视器提供的虚拟机执行环境下分时执行，在虚拟机系统中，任何两个子 OS 在经由由虚拟机监视器提供的通信接口彼此相互通信的同时执行处理。该方法包含：通过虚拟机监视器指定经由通信接口彼此相互通信的子 OS 组；由虚拟机监视器将包含在所指定的子 OS 组中的各子 OS 的执行计划的时间片设置得较短。

## 附图说明

附图并入说明书并构成说明书的一部分，其示出了本发明的实施例，并与上面给出的一般介绍以及下面给出的对实施例的详细介绍一起用于阐释本发明的原理。

图 1 为一框图，其示出了实现根据本发明一实施例的虚拟机系统的计算机系统；

图 2 为一框图，其示出了图 1 所示的机器系统中的虚拟机监视器（VMM）的构造；

图 3 为一框图，其示出了根据该实施例的虚拟机系统的状态实例，其中，两个子 OS 在彼此通信的同时执行处理；

图 4A 与 4B 为时序图，其各自示出了图 3 所示系统状态中重新计划之前和之后的处理序列；

图 5 为图 4A 所示处理序列中的通信状态表；

图 6 为一流程图，其示出了用于根据本发明该实施例的虚拟机系统中的时间片控制处理的过程；

图 7 为子 OS 表的实例；

图 8A 与 8B 为时序图，其各自示出了当三个子 OS 在彼此通信的同时

执行处理时在重新计划之前和之后的处理序列;

图 9 为图 8A 所示处理序列中的通信状态表;

图 10 为子 OS 表的实例,其用于基于图 9 所示通信状态表执行的时间片控制处理;

图 11 为图 8B 所示处理序列中的通信状态表;

图 12 为一框图,其示出了根据本发明该实施例的变型的虚拟机监视器 (VMM) 的构造;

图 13 为 n 值定义表的实例;

图 14A 与 14B 为流程图,其示出了在根据本发明该实施例的该变型中用于时间片控制处理的过程; 以及

图 15 为平均通信时间表的实例。

### 具体实施方式

现在将参照附图介绍本发明一实施例。图 1 为一框图,其示出了实现根据本发明一实施例的虚拟机系统 (VM 系统) 的计算机系统 1 的构造。计算机系统 1 包含构成真实机器的硬件 (HW) 10。人们知道, HW 10 包含真实处理器 11、I/O 单元 (输入/输出单元) 12、存储器 (未示出)。被称为主 OS 的 OS 20 在 HW 10 上运行。在 OS (主 OS) 20 上执行包含虚拟机应用 (VM 应用) 30 在内的多种应用。

VM 应用 30 包含虚拟机监视器 (VMM) 31。在执行 VM 应用 30 时,实现 VMM 31。VMM 31 管理 VM 系统并建立 VM 系统的环境 (虚拟机执行环境)。在图 1 中,构建四个虚拟机执行环境 (VM 执行环境) 32A、32B、32C、32D。

VM 执行环境 32i (i=A、B、C、D) 包含虚拟 HW 环境 (虚拟 HW 单元) 33i 和子 OS 执行环境 34i。虚拟 HW 环境 33i 包含虚拟 HW, 例如虚拟处理器 (VP) 331i、虚拟 I/O 单元 332i、虚拟存储器单元 (未示出) 等等。VMM 31 在逻辑上仿真虚拟 HW, 以便实现上述 VM 执行环境 32i。VMM 31 将运行在 VM 执行环境 32i 中的子 OS 执行环境 34i 中的 OS 作为

子 OS 35i (#i) 装载到子 OS 执行环境 34i 中。被装载到子 OS 执行环境 34i 中的子 OS 35i (#i) 在子 OS 执行环境 34i 中被执行。

VMM 31 在包含在虚拟 HW 环境 33i 中的虚拟存储器单元中展开子 OS 35i (#i) 的代码 (执行代码), 并以虚拟处理器 331i 的仿真的形式对执行代码进行执行。因此, VMM 31 继续进行子 OS 35i 的运行。来自子 OS 35i 的 I/O 请求在 VMM 31 仿真虚拟 I/O 单元 332i 时被处理。

图 2 为一框图, 其示出了图 1 所示 VMM 31 的构造。VMM 31 包含计划器 311、分派器 312、通信服务单元 313、通信状态表 314 以及子 OS 表 315。计划器 311 确定子 OS 35i (#i) 的计划。多种类型可被考虑为由计划器 311 实施的计划策略。假设在本实施例中计划器 311 实现简单的循环计划。基本上, 计划器 311 指示分派器 312 以被称为时间片 ( $\tau$ ) 的时间单位执行子 OS 35i (#i)。然而, 在本实施例中, 计划器 311 也指示分派器根据情况改变时间片的值。

分派器 312 实际上根据计划器 311 的指示将处理器 (真实处理器 11) 分派到子 OS 35i (#1)。当计划器 311 所指示时间片的时间过去时, 分派器 312 将处理器 (真实处理器 11) 从子 OS 35i (#1) 移除并将处理器分配给另一子 OS。重复进行这种处理。

通信服务单元 313 作为用于处理子 OS 之间的通信请求 (其从子 OS 35i (#i) 被发送到 VMM 31) 的通信接口。通信请求以管理程序调用 (系统调用) 的形式被发布到 VMM 31。在这种调用中, 目标子 OS 和消息数据被指定。通信服务单元 313 将被指定的消息发送到被指定的子 OS, 并设置接收中断。当处理器被分派给目标子 OS 时, 将接收中断通知到目标子 OS。目标子 OS 可将由通信服务单元 313 发送的消息数据作为接收数据进行参照。

通信状态表 314 用于保持关于由通信服务单元 313 所处理消息的发送的信息。具体而言, 通信状态表 314 用于保持关于消息发送的信息, 其中, 子 OS 35i 被定义为发送源, 子 OS 35j ( $j \neq i$ ) 被定义为发送目标。子 OS 35A、35B、35C、35D、35i、35j 将在下文中分别被表示为子 OS #A、#B、#C、

#D、#i 和 #j。子 OS 表 315 用于将彼此发送消息的子 OS 对保持为相互通信子 OS 对，其中，该消息在一个计划时间周期（下文中介绍）内被接收。

图 3 示出了由图 1 所示计算机系统 1 实现的 VM 系统（虚拟机系统）的状态实例。在该实例中，子 OS #A、#B、#C、#D 由 VMM 31 所构建的 VM 执行环境 32A、32B、32C、32D（见图 1）运行。在图 3 所示的系统状态中，子 OS #A 中的进程#a 和子 OS #C 中的进程#c 在彼此相互通信的同时执行处理。具体而言，四个处理（a1、c1、c2、a2）在子 OS #A 中的进程#a 和子 OS #C 中的进程#c 之间重复进行。

图 4A 与 4B 各自示出了当重复四个处理时在重新计划之前和之后的处理序列。在图 4A 与 4B 中，向下的箭头表示发送消息，向上的箭头表示接收消息。箭头的数量对应于消息的数量。附着到各向上箭头（接收消息）头部的含有“ $\tau$ ”的字符串意味着其对应的消息在由该字符串表示的时间内（例如  $2\tau$  对应于  $2\tau$  的时间段）被发送。字符串下方被括住的符号意味着其对应的消息从该符号表示的子 OS 发送（如果被括住的符号为 A，消息从子 OS #A 发送）。

图 5 示出了图 2 的通信状态表 314 的实例，其中，被计划的子 OS 为子 OS #A、#B、#C、#D。对于作为发送目标的子 OS #i（ $i=A、B、C、D$ ）和作为发送源的子 OS #j（ $j=A、B、C、D; j \neq i$ ）的所有组合，表 314 具有条目 #ij，其保持关于消息发送的信息（下文被称为通信状态信息）。

通信状态表 314 的条目 #ij 中所保持的通信状态信息包括消息数量和平均发送时间。消息数量是在一个计划时间周期中从子 OS #i 发布到子 OS #j 的消息的数量，或在短于一个计划时间周期的时间段中从子 OS #i 发送且由子 OS #j 接收的消息的数量。平均发送时间是直到子 OS #j 接收来自子 OS #i 的消息为止所需要的平均时间，或为从子 OS #i 到子 OS #j 的消息发送所需要的平均时间。在图 5 的实例中，平均发送时间被括住。

在本实施例中，假设一个计划时间周期表示为：

$\tau \times$  当前被计划的子 OS 的数量。当如本实施例中那样在时间片  $\tau$  的时间中执行简单的循环计划时，一个计划时间周期等于从将处理器分派到各

子 OS 到其下一个分派所需要的时间。在图 4 中, 例如, 从  $t_{n+1}$  到  $t_{n+5}$  的时间段对应于一个计划时间周期, 即  $4\tau$ 。

通信状态信息由通信服务单元 313 创建, 其中, 子 OS #i 为发送源, 子 OS #j 为发送目标。通信状态表 314 的条目 #ij 的信息被更新为所创建的通信状态信息。计划器 311 可在每个计划时间周期结束时参照在表 314 中更新的通信状态信息。

图 5 的通信状态表 314 示出了由图 4A 所示的处理序列表示的消息发送的状态 (通信状态)。例如, 作为关于在最近一个计划时间周期中在子 OS #A 与子 OS #C 之间的消息发送的通信状态信息, “3 ( $2\tau$ )” 被保持在表 314 的条目中, 该条目对应于成对的发送源子 OS #A 和发送目标子 OS #C。值 “3” 表示在短于所述一个计划时间周期的时间内由子 OS #A 发送且由子 OS #C 接收的消息的数量。被括住的号码 “ $2\tau$ ” 表示从子 OS #A 到子 OS #C 的消息发送所需要的平均发送时间。

在本实施例中, 每个计划时间周期, 计划器 311 执行下面的用于控制时间片的处理 (时间片控制处理)。计划器 311 参照通信状态表 314 以便基于子 OS 之间的通信状态检索彼此密切 (closely) 相互通信的子 OS 对。如果计划器 311 检测到目标子 OS 对, 通过降低将处理器分配到各子 OS 的粒度 (degree) 来防止消息发送的延迟。每个计划时间周期, 计划器 311 执行时间片控制处理。时间片控制处理可每两个或两个以上的计划时间周期执行一次。

将参照图 6 所示流程图介绍由计划器 311 执行时间片控制处理的过程。首先, 计划器 311 检测彼此密切相互通信的所有子 OS 对 (相互通信子 OS) (步骤 S11)。彼此密切相互通信是从各子 OS 发送消息、使得该消息能在一个计划时间周期中被接收到。具体而言, 计划器 311 参照图 5 所示的通信状态表 314 以检索所有这样的子 OS 对 #i 与 #j: 其发送消息, 使得这些子 OS 能在一个计划时间周期内接收到消息。具体而言, 子 OS #i、#j 包括: 子 OS #i, 其发送消息, 使得子 OS #j 能在一个计划时间周期内接收到该消息; 子 OS #j, 其发送消息, 使得子 OS #i 能在与所述一个计划时间

周期的同一时间周期内接收到该消息。

在本实施例中，参照对于通信状态表 314 所示各子 OS 组合的消息数量，消息数量不为零（非零）的子 OS 组合（子 OS 对）被检测为目标子 OS 组合（相互通信子 OS 对）。创建所有检测得到的子 OS 对的列表（列表“L”）。在图 5 所示的通信状态表 314 中，仅包含作为元素的子 OS 对 #A 与 #C 的列表“L”被创建。

计划器 311 将所创建的列表“L”与子 OS 表 315 进行比较。如果列表“L”包含没有输入表 315 的子 OS 对，计划器 311 将该子 OS 对输入到表 315 中（步骤 S12）。

如果存在不包括在列表“L”中但被输入到子 OS 表 315 中的子 OS 对，或者，如果子 OS 表 35 包括列表“L”中未显示的子 OS 对被输入的条目，将该条目（条目信息）从子 OS 表 315 中删除（步骤 S13）。步骤 S13 中删除的条目（条目信息）为这样的条目（条目信息）：该条目中，以前彼此相互通信且目前不彼此通信的子 OS 对被输入。

图 7 示出了在步骤 S12 和 S13 被执行后获得的子 OS 表 315 的实例。在该实例中，子 OS 对 #A 与 #C 作为相互通信子 OS 对（#i 与 #j）被输入到子 OS 表 315 的第一条目中。

然后，参照子 OS 表 315，计划器 311 创建紧密相关子 OS 组（步骤 S14）。紧密相关子 OS 组是在子 OS 表 315 中输入的子 OS 对的集合（组）。该集合（组）不仅包括子 OS 对，还包括与该子 OS 对中的一个相互通信的另一子 OS。因此，子 OS 表 315 的每个条目不仅包括用于输入子 OS 对的列，还包括用于输入组——在前一列中输入的子 OS 属于该组——的 ID（组 ID）的列。

在图 7 所示的子 OS 表 315 中，仅子 OS #A、#C 组成紧密相关子 OS 组。然而，如果通信状态表 314 是图 9 所示的，子 OS #A、#C、#D 组成紧密相关子 OS 组。紧密相关子 OS 组是彼此频繁发送/接收消息的子 OS 的集合。

在步骤 S14 中，计划器 311 向每个紧密相关子 OS 组提供作为组 ID 的

其唯一 ID (例如号码), 并将 ID 输入到子 OS 表的组 ID 列中。计划器 311 执行步骤 S14, 接着, 前进到步骤 S15。在步骤 S15 中, 计划器 311 将属于每个紧密相关子 OS 组的子 OS 起初以时间片  $\tau$  的单位被计划的时间周期分割为  $\tau/n$  的时间周期, 以便对子 OS 重新进行计划, 其中,  $n$  被称为分割常数。在本实施例中, 假设分割常数“ $n$ ”为预定恒定值, 即二。然而, 可如下面介绍的那样动态改变分割常数“ $n$ ”。在步骤 S15 中, 计划器 311 创建 (确定) 用于以  $\tau/n$  的时间单位顺次执行被重新计划的子 OS 的新计划 (表示新计划的计划表)。

接着, 计划器 311 向分派器 312 发送所创建的新计划 (计划表) 的信息, 并指示分派器 312 基于新计划进行重新计划 (步骤 S16)。根据新计划, 分派器 312 实际上进行重新计划, 以便向子 OS #A、#C 分派处理器。

当子 OS #A、#C 如同上面介绍的本实施例中那样彼此发送/接收消息时, 作为时间片控制处理的结果, 这些子 OS 变为属于同一紧密相关子 OS 组。因此, 重新计划处理器以  $\tau/2$  的时间单位向子 OS #A、#C 的分派。作为重新计划的结果, 属于紧密相关子 OS 组的子 OS #A、#C 如图 4B 所示地执行。至于不属于紧密相关子 OS 组的子 OS #B、#D, 仍然计划处理器以  $\tau$  的时间单位每计划时间周期 ( $=4\tau$ ) 到这些子 OS 的分派。在本实施例中, 注意, 对于属于紧密相关子 OS 组的子 OS 的重新计划不影响对于其他子 OS 的计划。

在图 4B 所示的实例中, 从子 OS #A 到子 OS #C 的消息发送的平均发送时间以及从子 OS #C 到子 OS #A 的消息发送的平均发送时间分别为大约  $0.5\tau$  和  $1.5\tau$ 。上面介绍的四个处理 a1、c1、c2、a2 的序列的周期 (平均处理周期) 从图 4A 所示的  $4\tau$  减小到图 4B 所示的  $2\tau$ 。

在本实施例中, 当子 OS 在彼此相互通信的同时彼此关联地执行处理时, 仅用于执行该子 OS 的计划的计划的时间片被缩短。缩短时间片可防止性能由于惯常引起的通信延迟而降低, 而对其他子 OS 没有影响。特别地, VM 系统的子 OS 之间的通信通常基于存储器进行, 其减小了额外成本。存在这种通信在延迟时间 (等待时间) 上与真实网络系统的通信相比减小得多

的可能。因此，在本实施例中，VM系统的子OS之间高速通信的性能能够更为增强，因此，整个VM系统的性能能够更为增强。

在图4A与4B的实例中，不考虑进程#a、#c以外的任何将由子OS #A、#C执行的进程。实际上，不保证总是产生图4B所示的结果。然而，在图4A所示的实例中，无论子OS #A、#C具有什么负载，没有多于图4A所示的生产量(throughout)。在本实施例中，根据图4B所示的子OS的负载，处理可高效率地执行。

通过本实施例的重新计划改进处理效率的另一实例将关于子OS #A(其进程#a)、子OS #C(其进程#c)、子OS #D(其进程#d)在彼此相互通信的同时执行处理的情况进行介绍。图8A与8B各自示出了当三个子OS #A、#C、#D在彼此相互通信的同时执行处理时在重新计划之前和之后执行的处理序列。假设在这种情况下分割常数“n”为二。

在图8A与8B的实例中，每个计划时间周期执行下面六个处理：

(1) 子OS #A(进程#a)向子OS #C发送三个消息(网络数据包)中的两个，并向子OS #D发送另一个(处理a'1)；

(2) 子OS #C(进程#c)从子OS #A接收两个消息并对它们进行处理(处理c'1)；

(3) 子OS #C(进程#c)由被处理的消息创建新的消息，并将新消息发送到子OS #A(处理c'2)；

(4) 子OS #D(进程#d)从子OS #A接收消息并对之进行处理(处理d'1)；

(5) 子OS #D(进程#d)由所处理的消息创建新的消息，并将新消息发送到子OS #A(处理d'2)；以及

(6) 子OS #A(进程#a)从子OS #C和#D接收消息并对之进行处理(处理a'2)。

处理(发送处理)a'1和c'2需要以此顺序(a'1→c'2)进行处理，处理(发送处理)a'1和d'2需要以此顺序(a'1→d'2)进行处理。

图9示出了在图8A所示的处理序列中的通信状态表314的实例。图

10 示出了基于图 9 所示通信状态表 314 的状态通过执行时间片控制处理所获得的子 OS 表 315 的实例。图 8B 示出了当处理器根据基于图 10 所示的子 OS 表 315 所创建的新计划被分派到子 OS #A、#C、#D 时所获得的处理序列（即在重新计划后获得的处理序列）。

在图 10 所示的子 OS 表 315 的实例中，紧密相关子 OS 组包含子 OS #A、#C、#D。重新计划处理器以  $\tau/2$  的时间单位到子 OS #A、#C、D 的分配。这种重新计划降低了处理器到三个子 OS #A、#C、#D 的分配的粒度。上面的六个处理（a'1, c'1, c'2, d'1, d'2, a'2）的处理顺序由图 8A 所示的状态被改进为图 8B 所示的状态。对于不属于紧密相关子 OS 组的子 OS #B，仍然计划处理器以时间单位  $\tau$  每个计划时间周期（ $=4\tau$ ）对子 OS #B 的分配。

图 8A 中的平均处理周期为  $4\tau$ ，图 8B 中的为  $2\tau$ 。由此看出，图 8A 中的平均处理周期从  $4\tau$  减小到  $2\tau$ 。图 11 示出了图 8B 的状态中的通信状态表 314 的实例。

#### [变型]

在上面的实施例中，假设分割常数“n”为预定恒定值（ $n=2$ ）。分割常数“n”为用于确定处理器到子 OS 的分配的时间单位的值（参数）。在上面的实施例中，处理器以  $\tau \times (1/n)$  的时间单位或以  $\tau/n$  的时间单位被分配。然而，消息通信所需要的时间受到分配时间  $\tau/n$  的影响。因此，分割常数“n”必须适当设置。

现在将介绍对上述实施例的变型，其中，包括适当确定分割常数“n”的值的处理（n 值确定处理）在内的时间片控制处理由计划器 311 执行。图 12 为一框图，其示出了根据这种变型的虚拟机监视器（VMM）31 的构造。为了方便，在图 12 中，与图 2 中相同的元件用同样的参考标号表示。图 12 所示的 VMM 31 与图 2 中的 VMM31 的不同之处在于其包含 n 值定义表 316 和平均通信时间表 317。与图 2 所示的计划器 311 不同，图 12 所示 VMM 31 的计划器 311 包含平均通信时间计算单元 311a 与比较单元 311b。单元 311a 与 311b 以及表 316 与 317 将在后文中介绍。

下面介绍包括  $n$  值确定处理在内的时间片控制处理的概要。

(a) 计划器 311 对于紧密相关 OS 组使用不同的分割常数“ $n$ ”，并在必要时改变“ $\tau$ ”。如果紧密相关子 OS 组  $G_i$  的分割常数“ $n$ ”被表示为  $n_i$ ，计划器 311 对属于紧密相关子 OS 组  $G_i$  的子 OS 重新计划，使得用于子 OS 的进程分配的时间周期变为  $\tau/n_i$ 。

(b) 假设分割常数“ $n$ ”被预定。在此变型中，分割常数“ $n$ ”被设置为  $d_0$ 、 $d_1$ 、 $d_2$ 、 $d_3$  四个值中的任意一个。 $d_0$ 、 $d_1$ 、 $d_2$ 、 $d_3$  的具体值由  $n$  值定义表 316 定义。

图 13 示出了  $n$  值定义表 316 的实例。在图 13 所示的表中， $d_0$ 、 $d_1$ 、 $d_2$ 、 $d_3$  分别对应于 1、2、3、4，它们是分割常数“ $n$ ”的值（ $n$  值）。如果  $n$  值定义表 316 的内容被改变，或者，如果  $d_0$ 、 $d_1$ 、 $d_2$ 、 $d_3$  的定义被改变，可定义分割常数“ $n$ ”的不同值。

假设在该变型中  $d_0$ 、 $d_1$ 、 $d_2$ 、 $d_3$  表示的值以  $d_0$ 、 $d_1$ 、 $d_2$ 、 $d_3$  的顺序增大（或者，时间片被进一步分割）。另外，假设  $d_0$ （其为分割常数“ $n$ ”的最小值）为一。另外，假设该变型中定义的最大分割常数“ $n$ ”用  $d_{\max}$  表示。在该变型中， $d_{\max}$  为  $d_3$ 。

(c) 在包括应用于本变型的  $n$  值确定处理在内的时间片控制处理中，直到时间片控制处理被调用了  $A$  次，在从被  $d_i$  定义的  $n_i$  的值被确定时开始的时间段中，事实上不执行任何确定处理。在该变型中，直到时间片控制处理被调用了  $A$  次，从其在  $n$  值确定处理中被确定时开始，不改变  $n_i$  的值。尽管处理量由于  $n_i$  的值的频繁改变而增加，能防止由于处理量增加引起的额外成本的增加。

在该变型中，如同上面的实施例那样，计划器 311 每个计划时间周期执行时间片控制处理。具体而言，计划器 311 基于子 OS 之间的通信状态检索通信状态表 314 以得到彼此密切相互通信的子 OS 对（其彼此发送消息，使得它们能在一个计划时间周期内接收到消息）。如果子 OS 对被检测到，计划器 311 通过降低处理器到子 OS 的分配粒度来防止消息发送中的延迟。然而，计划器 311 对于每个紧密相关 OS 组动态改变分割常数

“n”的值，并控制对于各组的处理器分配粒度。时间片控制处理可每两个或两个以上的计划时间周期执行一次。

下面将参照图 14A 与 14B 所示的流程图介绍用于包括 n 值确定处理在内的时间片控制处理的过程。首先，计划器 311 在其重复时间片控制处理之前预先将变量 WAIT\_CNT 初始化为零。变量 WAIT\_CNT 用于控制时间片控制处理被执行的时刻。当时间片控制处理被调用时，计划器 311 参照变量 WAIT\_CNT 以确定变量 WAIT\_CNT 是否为零（步骤 S21）。如果变量 WAIT\_CNT 不为零，计划器 311 将变量 WAIT\_CNT 减小 1（步骤 S22），并完成时间片控制处理。

当变量 WAIT\_CNT 为零时（步骤 S21），计划器 311 如下所述地控制处理器到各紧密相关 OS 的分配粒度。首先，计划器 311 执行与上面的实施例的步骤 S11 到 S14 对应的步骤 S23 到 S26。换句话说，计划器 311 检测相互通信子 OS 对并创建相互通信子 OS 对的列表“L”（步骤 S32）。在表 315 中，计划器 311 也输入新检测到且不包含在子 OS 表 315 中的相互通信子 OS 对（步骤 S24）。于是，计划器 311 从子 OS 表 315 中删除不包含在列表“L”中的子 OS 对在其中被输入的条目（步骤 S25），并创建紧密相关子 OS 组（步骤 S26）。

计划器 311 执行步骤 S23 到 S26，接着，进行到步骤 S27。在步骤 S27 中，计划器 311 将对于新检测到的紧密相关子 OS 组的条目（条目信息）添加到平均通信时间表 317。在步骤 S27 中，计划器 311 也从平均通信时间表 317 中删除对于已被检测到且被输入到表 317 中的紧密相关子 OS 组的以及对于此次未检测到的紧密相关子 OS 组的条目（条目信息）。

图 15 示出了平均通信时间表 317 的实例。平均通信时间表 317 包括对于它们各自的紧密相关子 OS 组的条目。表 317 的条目用于输入对于由紧密相关子 OS 组的  $d_0$  到  $d_3$  定义的分割常数“n”的值的平均通信时间的测量值。平均通信时间将在下文中介绍。

在步骤 S27 中，当对于新检测到的紧密相关子 OS 组的条目（条目信息）被添加到平均通信时间表 317 中时，条目信息已被初始化。这里，用

所添加的条目表示的、由  $d_0$  到  $d_3$  定义的分割常数“n”的值的平均通信时间的信息被设置为具有无效值的未定义状态（该状态用符号“-”表示）。

当计划器 311 执行步骤 S27 时，其如下执行 n 值确定处理（步骤 S28 到 S36）。首先，计划器 311 判断对于在子 OS 表 315 中输入的所有紧密相关子 OS 组是否执行了 n 值确定处理（步骤 S28）。如果存在未执行 n 值确定处理的紧密相关子 OS 组（未处理的紧密相关子 OS 组），计划器 311 在子 OS 表 315 中从未处理的紧密相关子 OS 组中选择一个作为紧密相关子 OS 组  $G_i$ （步骤 S29）。

计划器 311 更新对应于所选择的紧密相关子 OS 组  $G_i$  的平均通信时间表 317 的条目信息，或者，其更新紧密相关子 OS 组  $G_i$  的平均通信时间（步骤 S30）。平均通信时间是由计划器 311 中的平均通信时间计算单元 311a 计算得到的。平均通信时间与发送目标的子 OS 在一个计划时间周期内接收的消息有关，该消息被包括在从属于紧密相关子 OS 组  $G_i$  的子 OS 发送的消息中，同时，使用由  $d_k$  ( $k=0, 1, 2, 3$ ) 定义的分割常数“n”的值。换句话说，平均通信时间是从由发送目标的子 OS 在一个计划时间周期内接收的消息被发送以及直到它们被接收的平均时间周期（通信时间周期）。通过平均通信时间计算单元 311a，从平均通信时间的计算主体排除超出一个计划时间周期接收的消息。

为了计算紧密相关子 OS 组  $G_i$  的平均通信时间，平均通信时间计算单元 311a 将在上一个计划周期内在属于组  $G_i$  的子 OS 之间传送的消息的总通信时间周期（ $\sum$  消息通信时间）除以消息总数（N）。具体而言，通过下面的公式计算组  $G_i$  的平均通信时间：

$$(\sum \text{消息通信时间}) / N \quad (1)$$

公式（1）中的消息是在一个计划时间周期内由发送目标的子 OS 接收的消息，其被包括在由属于紧密相关子 OS 组  $G_i$  的所有子 OS 发送的消息中。

如上所述，在该变型中，计算平均通信时间的目标时间周期为上一个计划时间周期。然而，目标时间周期可被定义为多个计划时间周期。平均

通信时间周期基于所述多个计划时间周期的相应的平均通信时间周期而获得。

下面将参照图 8A 所示的时序图介绍由平均通信时间计算单元 311a 计算平均通信时间的方法的实例。参照图 8A，子 OS #A、#C、#D 组成一个紧密相关子 OS 组。这里假设紧密相关子 OS 组的组 ID 为 1，组为  $G_i$ ，即“ $G_i=G_1 (i=1)$ ”。由于时间片为  $\tau$ ，“ $n=1$ ”或“ $d_k=d_0 (k=0)$ ”。通信状态表 314 的内容如图 9 所示。在图 8A 中，消息总数 ( $N$ ) 为五。

如下所述，紧密相关子 OS 组  $G_i$  的平均通信时间由平均通信时间计算单元 311a 进行计算：

$$\begin{aligned} & (\sum \text{消息通信时间}) / N = (2 \times 2\tau + 1 \times 3\tau + 1 \times 2\tau + 1 \times \tau) / 5 \\ & = 2\tau \end{aligned}$$

在步骤 S30 中，计划器 311 将由平均通信时间计算单元 311a 计算得到的平均通信时间定义为  $\tau$ 。在本实例中， $T=2\tau$ 。在步骤 S30 中，计划器 311 在当前应用列“ $d_k=d_0 (k=0)$ ”中设置计算得到的平均通信时间  $T (=2\tau)$ ，其被包括在与紧密相关子 OS 组  $G_i (G_1)$  对应的平均通信时间表 317 的条目中包括的列  $d_0$  到  $d_3$  中。换句话说，计划器 311 将列“ $d_k=d_0 (k=0)$ ”的旧的设置信息更新到平均通信时间  $T (=2\tau)$ 。更新后的平均通信时间  $T$  与紧密相关子 OS 组  $G_i (G_1)$  以及“ $d_k=d_0 (k=0)$ ”有关。更新的平均通信时间  $T$  被表示为  $T_{i,k}$ ，其中， $T_{i,k}=T_{1,0} (i=1, k=0) = 2\tau$ 。

当紧密相关子 OS 组  $G_i$  的当前分割常数“ $n$ ”的值（即“ $n_i$ ”的值）为  $d_k$  时，计划器 311 中的比较单元 311b 将该值与最小值（1）进行比较，并判断  $d_k$  是否为最小值（1），即  $d_0 (k=0)$ （步骤 S31）。如果紧密相关子 OS 组  $G_i$  的当前分割常数“ $n$ ”（= $n_i$ ）的值  $d_k$  不是最小值（1），比较单元 311b 将此次计算得到的平均通信时间  $T_{i,k}$ （在  $d_k$  时）与值“0”进行比较，并判断平均通信时间  $T_{i,k}$  是否为零（步骤 S32）。如果平均通信时间  $T_{i,k}$ （或当前平均通信时间  $T_{i,k}$ ）不为零，比较单元 311b 将平均通信时间  $T_{i,k}$  与  $d_{k-1}$  时的平均通信时间  $T_{i,k-1}$  进行比较，并判断  $T_{i,k}$  是否小于  $T_{i,k-1}$ （步骤 S33）。平均通信时间  $T_{i,k-1}$  在刚好在当前应用的列  $d_k$  之前的  $d_{k-1}$

的列中被设置，其被包括在与紧密相关子 OS 组  $G_i$  (组 ID= $i$ ) 对应的平均通信时间表 317 的条目中所包括的列  $d_0$  到  $d_3$  中。换句话说， $T_{i,k-1}$  为以比前一次长一个等级的时间片设置的最后的平均通信时间。

如果  $T_{i,k}$  小于  $T_{i,k-1}$ ，计划器 311 将新的分割常数“ $n$ ” ( $=n_i$ ) 的值设置到  $d_{k+1}$ ，其比当前值  $d_k$  长一个等级 (步骤 S34)。换句话说，计划器 311 使得 CPU 分配时间单元比当前时间单位短一个等级。如果  $d_{k+1}$  未被定义或者当前值  $d_k$  为  $n$  值定义表 316 定义的“ $n$ ”的最大值  $d_{max}$ ，计划器 311 不能设置长于最大值  $d_{max}$  的值。在这种情况下，计划器 311 将新的分割常数“ $n$ ” ( $=n_i$ ) 的值定义为当前值  $d_k$ ，或  $d_{max}$ 。计划器 311 执行步骤 S34，并返回到步骤 S28。

如果紧密相关子 OS 组  $G_i$  的当前分割常数“ $n$ ” ( $=n_i$ ) 的值  $d_k$  为最小值 (1)，或者如果当前值  $d_k$  为  $d_0$  ( $k=0$ ) (步骤 S31)，计划器 311 将分割常数“ $n$ ” ( $=n_i$ ) 的值设置到值  $d_{k+1}$ ，其长于当前值  $d_k$  一个等级，除非平均通信时间  $T_{i,k}$  ( $k=0$ ) 为零 (步骤 S34)。由于在此变型中  $k$  等于零 ( $k=0$ )，分割常数“ $n$ ” ( $=n_i$ ) 的值  $d_0$  从  $d_0 (=1)$  被改变为  $d_1 (=2)$ 。

于是，假设紧密相关子 OS 组  $G_i$  的当前分割常数“ $n$ ” ( $=n_i$ ) 的值  $d_k$  不为最小值 (1)。另外，假设此次获得的平均通信时间  $T_{i,k}$  (在  $d_k$  时) 不为零，但平均通信时间  $T_{i,k}$  等于或长于  $d_{k-1}$  时的平均通信时间  $T_{i,k-1}$  (步骤 S31 到 S33)。在这种情况下，计划器 311 将新的分割常数“ $n$ ” ( $=n_i$ ) 的值定义为值  $d_{k-1}$ ，其比当前值  $d_k$  小一个等级 (步骤 S36)。换句话说，计划器 311 使 CPU 分配时间单位比当前时间单位长一个等级。计划器 311 执行步骤 S36 并接着返回到步骤 S28。

在图 14A 与 14B 所示的流程图中，平均通信时间  $T_{i,k}$  为零的情况也被考虑 (步骤 S32 和 S35)。然而，在步骤 S23 和 S24 中， $T_{i,k}$  为零的紧密相关子 OS 组未被输入到子 OS 表 315 中。在本变型中，不存在平均通信时间  $T_{i,k}$  为零的紧密相关子 OS 组。图 14A 与 14B 所示的流程图包括当  $T_{i,k}$  为零时执行的处理以便描述共用过程。如果紧密相关子 OS 组  $G_i$  的当前分割常数“ $n$ ” ( $=n_i$ ) 的值  $d_k$  不为最小值 (1) 但此次 ( $d_k$ ) 获得的平均通

信时间  $T_{i,k}$  为零 (步骤 S31 与 S32), 计划器 311 将新分割常数 “n” ( $=n_i$ ) 的值定义为值  $d_{k-1}$ , 其比当前值  $d_k$  小一个等级 (步骤 S36)。如果紧密相关子 OS 组  $G_i$  的当前分割常数 “n” ( $=n_i$ ) 的值  $d_k$  为最小值 (1) 且此次获得的平均通信时间  $T_{i,k}$  为零 (步骤 S31 与 S35), 计划器 311 不执行分割常数 “n” ( $=n_i$ ) 的处理, 而是返回到步骤 S28。

对于所有的紧密相关子 OS 组, 计划器 311 执行上面的处理 (n 值确定处理), 其开始于步骤 S28。如果计划器 311 对于所有紧密相关子 OS 组完成了 n 值确定处理 (步骤 S28), 其将初始值 A 设置到变量 WAIT\_CNT (步骤 S37)。

计划器 311 将在 n 值确定处理中为紧密相关子 OS 组  $G_i$  的每一个确定 (设置) 的  $n_i$  的值应用到组  $G_i$ , 以创建用于以  $\tau/n_i$  的时间单位顺次执行属于组  $G_i$  的子 OS 的新计划 (步骤 S38)。计划器 311 将新计划的信息发送到分派器 312, 并指示分派器 312 基于新计划进行重新计划 (步骤 S39)。此后, 在图 14A 与 14B 的流程图所示的处理被调用 A 次时, 变量 WAIT\_CNT 被简单地逐个减小 (步骤 S21 和 S22)。因此, 属于紧密相关子 OS 组  $G_i$  的子 OS 通过  $n_i$  的同样的值进行计划。

在图 8A 的实例中, 作为 n 值确定处理 (步骤 S28 到 S36) 的结果, 包含子 OS #A、#C、#D 的紧密相关子 OS 组  $G_i$  ( $G_i=G_1$ ) 的  $n_1$  ( $n_i=n_1$ ) 的值从  $d_0$  变为  $d_1$ 。换句话说,  $n_1$  的值从 1 变为 2。结果, 紧密相关子 OS 组  $G_1$  的新的时间分配单位从  $\tau$  变为  $\tau/2$ 。因此, 子 OS #A、#C、#D 以新的时间分配单位重新计划, 如图 8B 所示。在图 8B 的实例中, 消息总数 N 为 10。

如上所述, 图 9 示出了在图 8B 所示状态中的通信状态表 314 的实例。如果平均通信时间计算单元 311a 基于通信状态表 314 为紧密相关子 OS 组  $G_1$  计算平均通信时间, 平均通信时间如下计算:

$$\begin{aligned} & (\sum \text{消息通信时间}) / N \\ &= (4 \times \tau/2 + 1 \times 2\tau + 1 \times \tau + 1 \times \tau + 1 \times 2\tau + 2 \times \tau/2) / 10 \\ &= 0.9\tau \end{aligned}$$

由上面的计算可见, 平均通信时间从  $2\tau$  减小到  $0.9\tau$ 。当根据图 14A 与 14B 所示流程图的处理被接下来执行时, “ $0.9\tau$ ” 被记录在图 15 所示平均通信时间表 317 的紧密相关子 OS 组 G1 所对应的条目的列 d1 中。于是,  $n_i$  的值从  $d_1 (=2)$  变为  $d_2 (=3)$ , 分配时间单位被进一步细分。计划器 311 减小  $n_i$  的值, 在该时间段中, 可获得效果 (即  $d_k$  中的新平均通信时间  $T_{i,k}$  短于  $d_{k-1}$  处的平均通信时间  $T_{i,k-1}$  的时间段)。如果没有获得效果, 计划器 311 将  $n_i$  的值从  $d_k$  返回到其上一值  $d_{k-1}$ 。结果,  $n_i$  的最优值被设置到紧密相关子 OS 组的每一个。

上面没有说明, 在步骤 S27 中从平均通信时间表 317 中删除的紧密相关子 OS 组的每个子 OS 以时间单位  $\tau$  重新计划, 同时, 分割常数 “n” 的值被返回到  $d_0 (=1)$ 。

在上面的实施例及其变型中, 假设 VMM 31 被实现为在 OS 20 上运行的一个应用 (VM 应用 30)。然而, 本发明可应用于 VMM (虚拟机监视器) 在计算机系统的真实 HW 上作为 VMM 单元实现的情况。

本领域技术人员将会容易地想到其他的优点和变型。因此, 本发明在其更宽广的实施形态上不限于这里示出和介绍的具体细节和典型实施例。因此, 在不脱离所附权利要求书及其等价内容所限定的一般发明构思的精神或范围的情况下, 可进行多种修改。

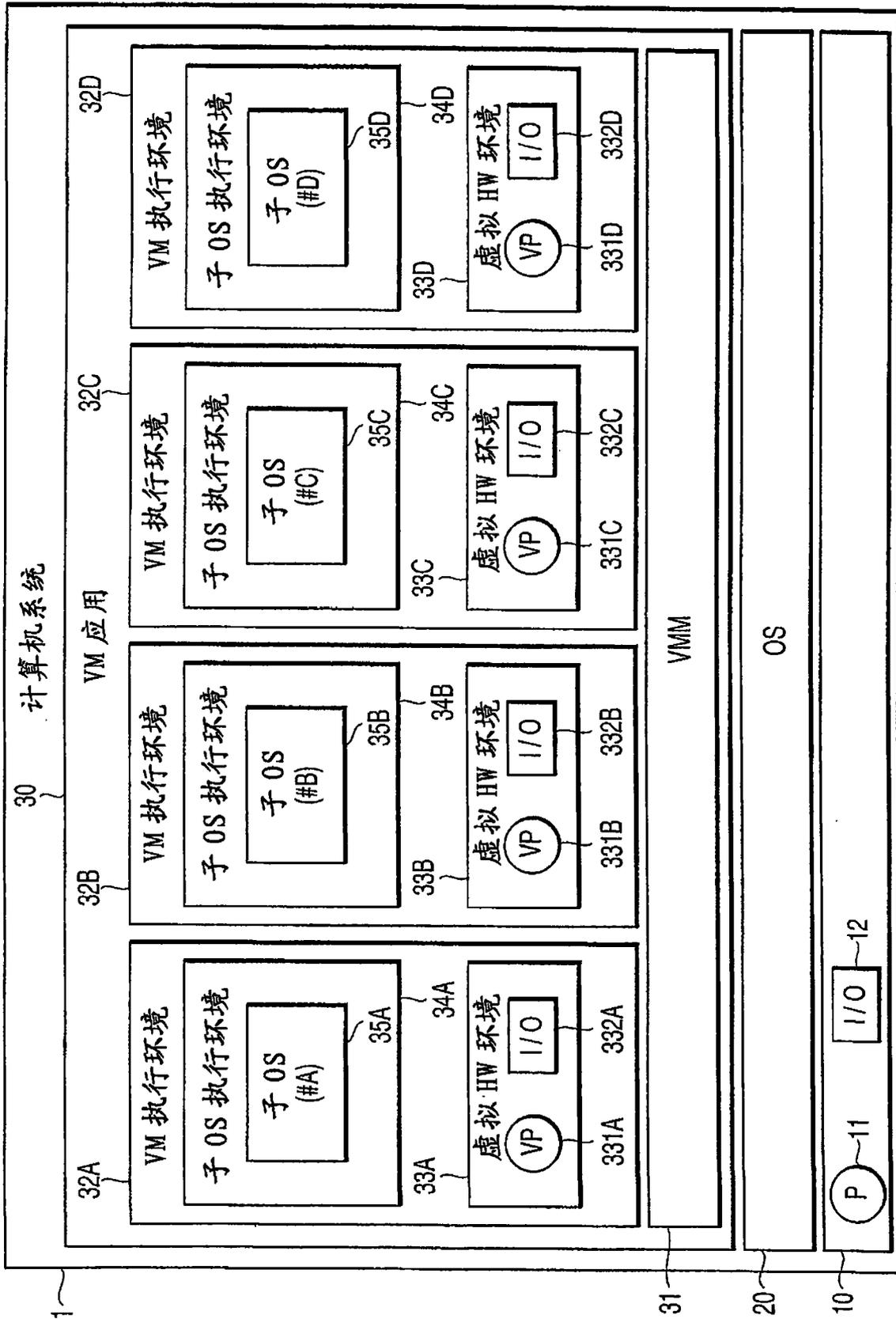


图 1

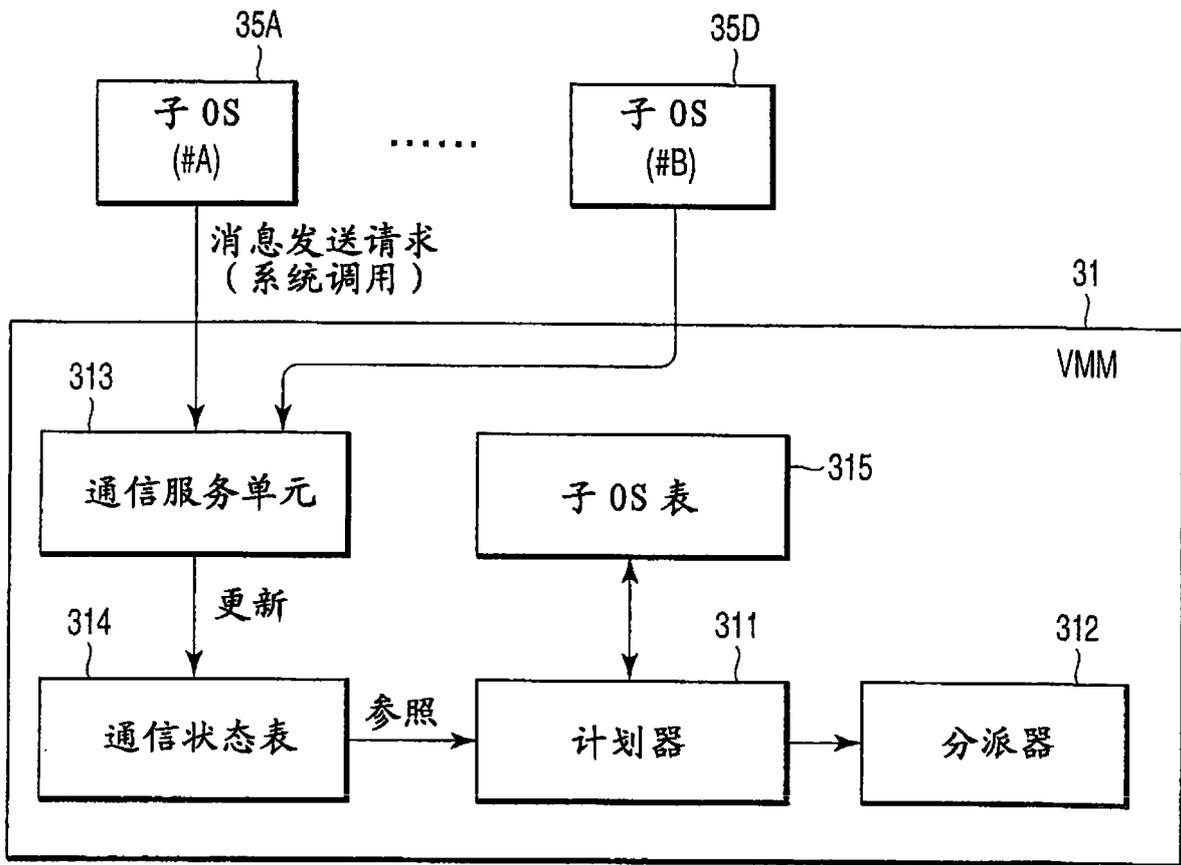


图 2

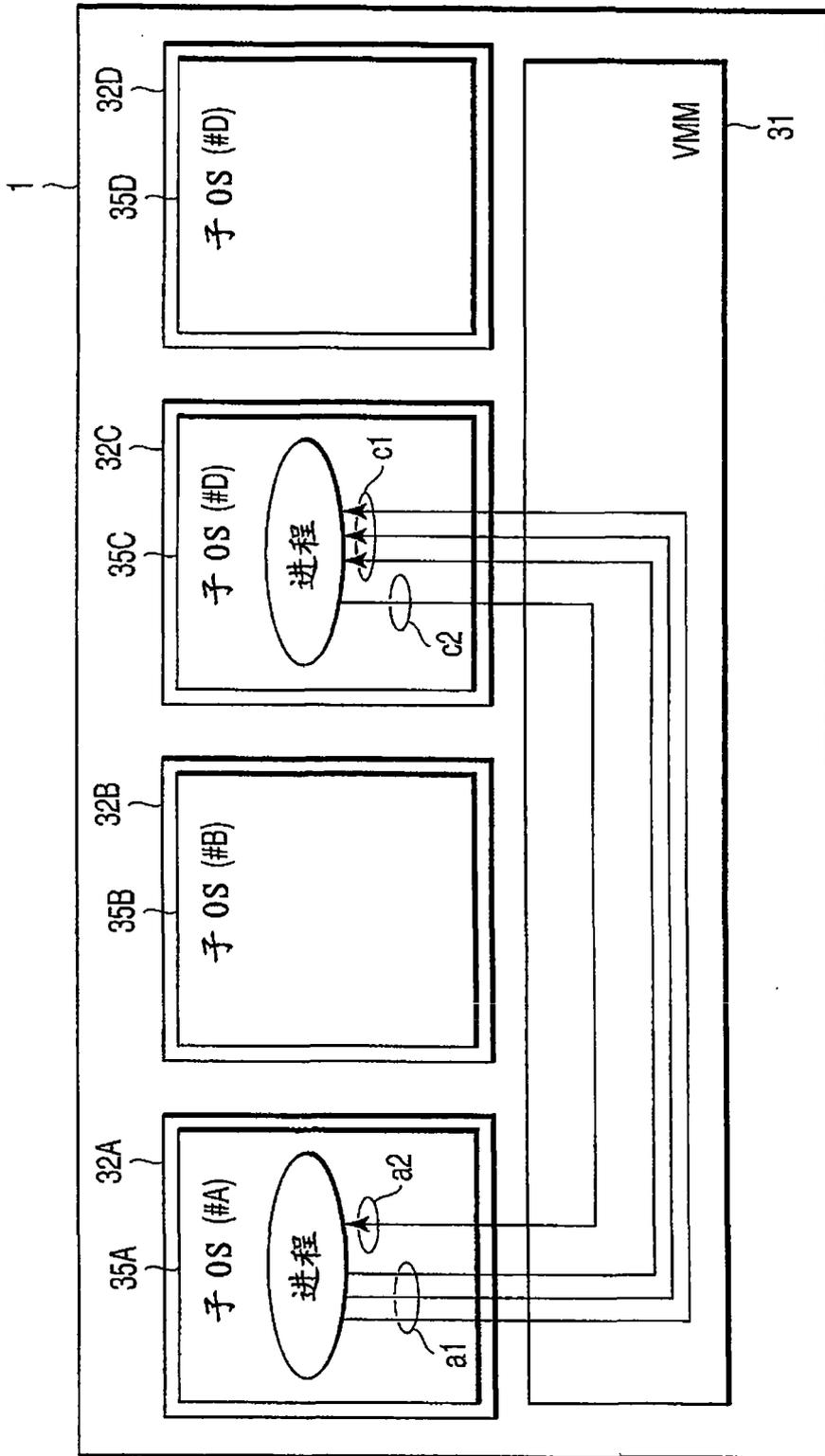


图 3

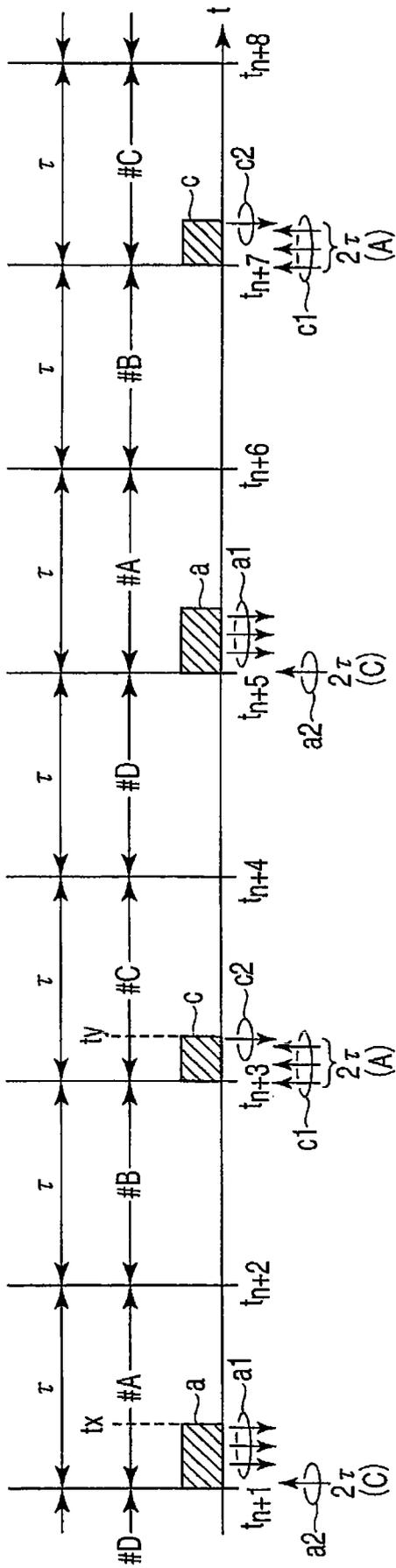


图 4A

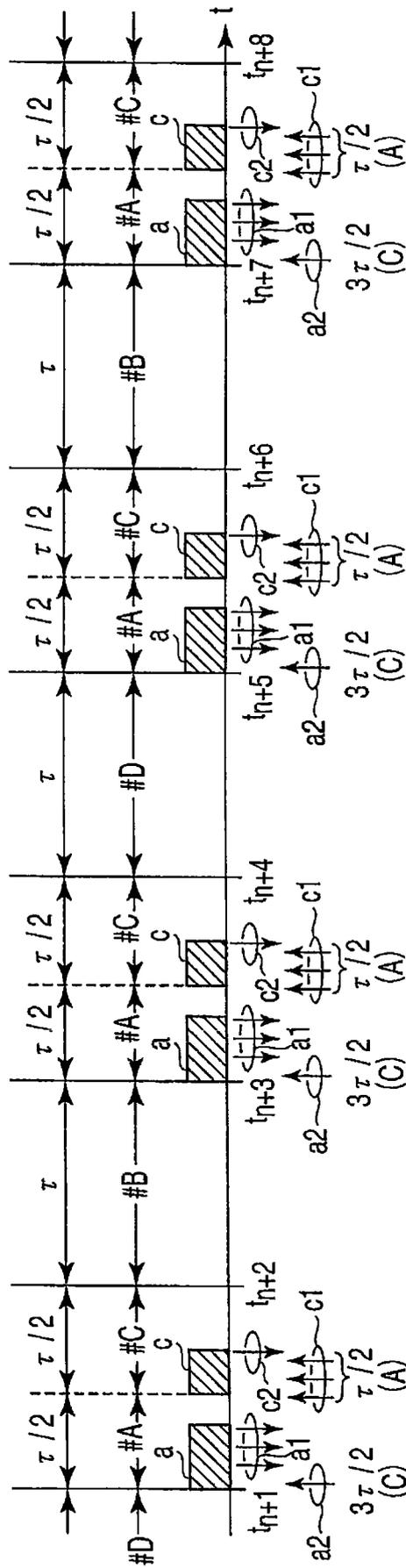


图 4B

314  
}

发送源 \ 发送目标	子 OS #A	子 OS #B	子 OS #C	子 OS #D
子 OS #A		0 ( )	3 (2 $\tau$ )	0 ( )
子 OS #B	0 ( )		0 ( )	0 ( )
子 OS #C	1 (2 $\tau$ )	0 ( )		0 ( )
子 OS #D	0 ( )	0 ( )	0 ( )	

消息数  
(平均发送时间)

图 5

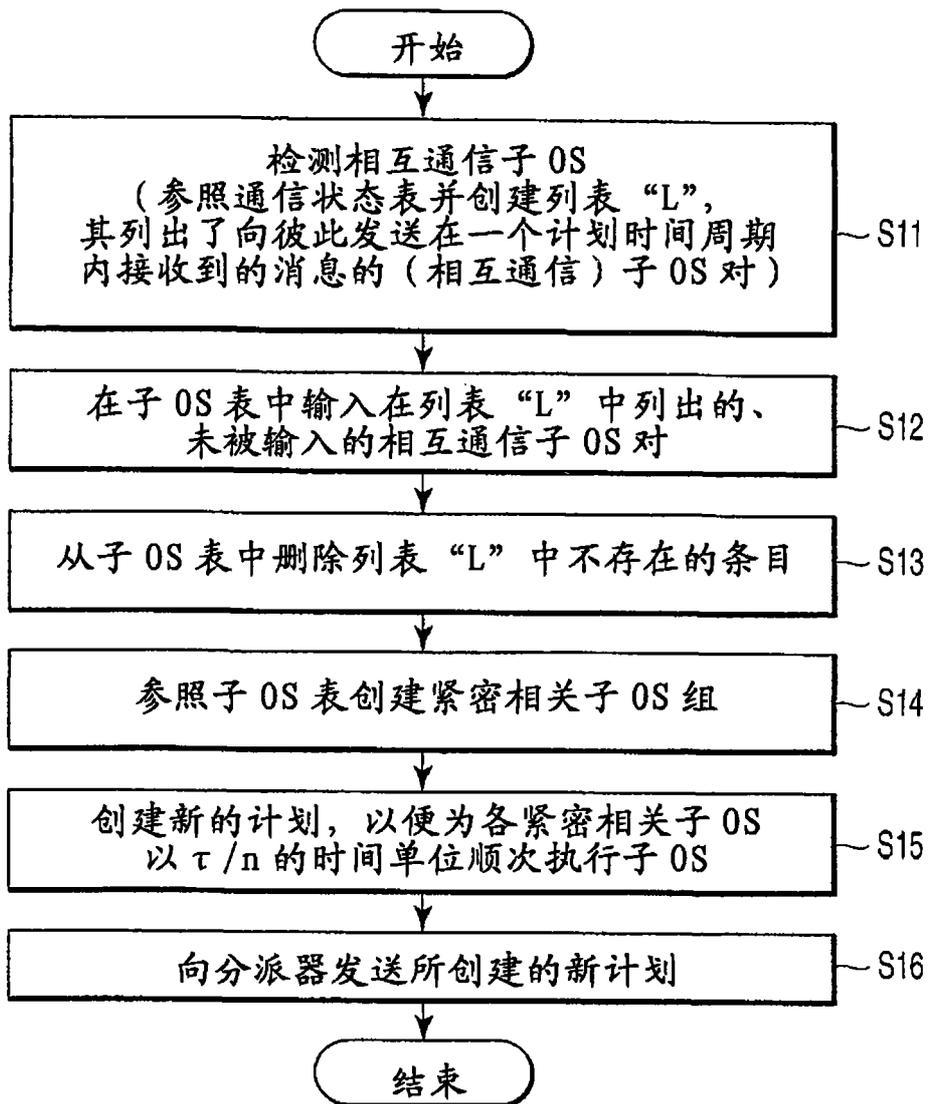


图 6

315

No.	子 OS # i	子 OS # j	组 ID
1	子 OS # A	子 OS # C	1
2	—	—	—
3	—	—	—
4	—	—	—

图 7

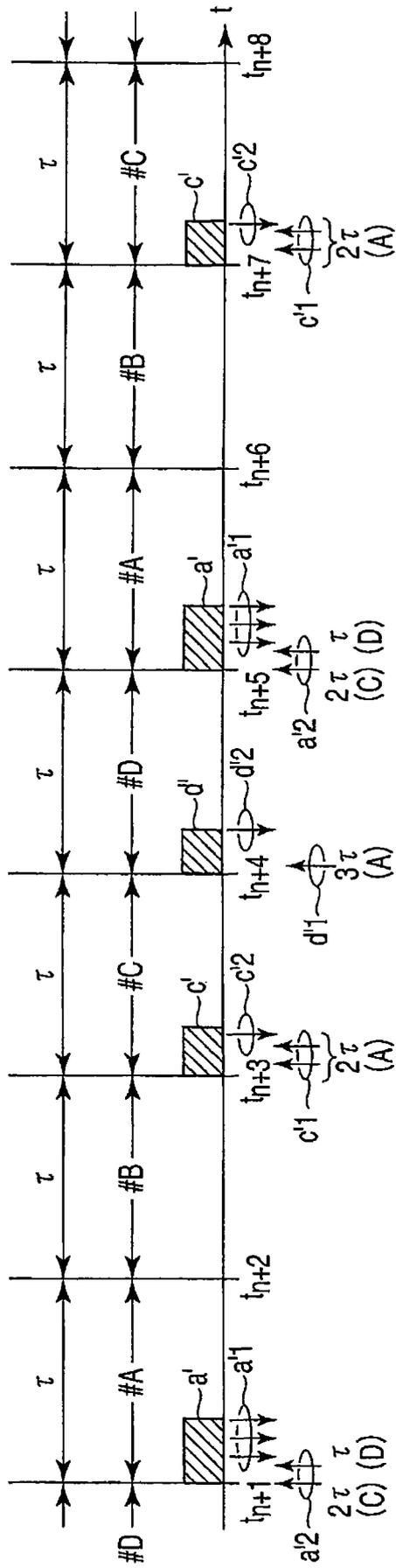


图 8A

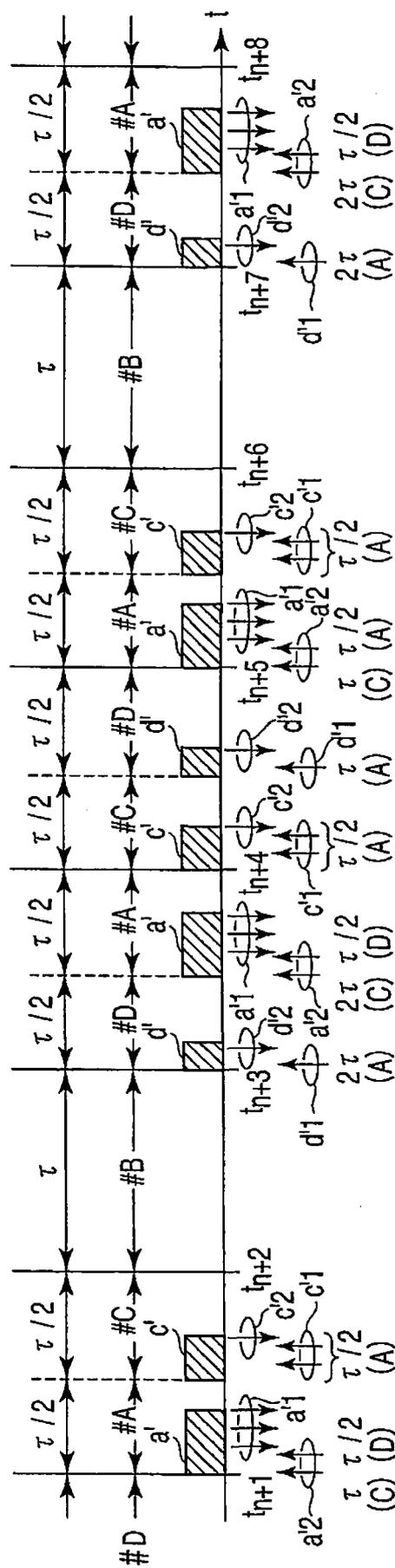


图 8B

314

发送目标 发送源	子 OS #A	子 OS #B	子 OS #C	子 OS #D
子 OS #A		0 ( )	2 (2τ)	1 (3τ)
子 OS #B	0 ( )		0 ( )	0 ( )
子 OS #C	1 (2τ)	0 ( )		0 ( )
子 OS #D	1 (1τ)	0 ( )	0 ( )	

消息数  
(平均发送时间)

图 9

315

No.	子 OS # i	子 OS # j	组 ID
1	子 OS # A	子 OS # C	1
2	子 OS # A	子 OS # D	1
3	—	—	—
4	—	—	—

图 10

314

发送目标 发送源	子 OS #A	子 OS #B	子 OS #C	子 OS #D
子 OS #A		0 ( )	4 ( $\tau/2$ )	1 ( $2\tau$ ) 1 ( $\tau$ )
子 OS #B	0 ( )		0 ( )	0 ( )
子 OS #C	1 ( $\tau$ ) 1 ( $2\tau$ )	0 ( )		0 ( )
子 OS #D	1 ( $\tau/2$ )	0 ( )	0 ( )	

消息数  
(平均发送时间)

图 11

317

组 ID	$d_0$	$d_1$	$d_2$	$d_3$
1	$2\tau$	$0.9\tau$	—	—
2	...	...	...	...
...	...	...	...	...

[—]: 未定义: 无效值

图 15

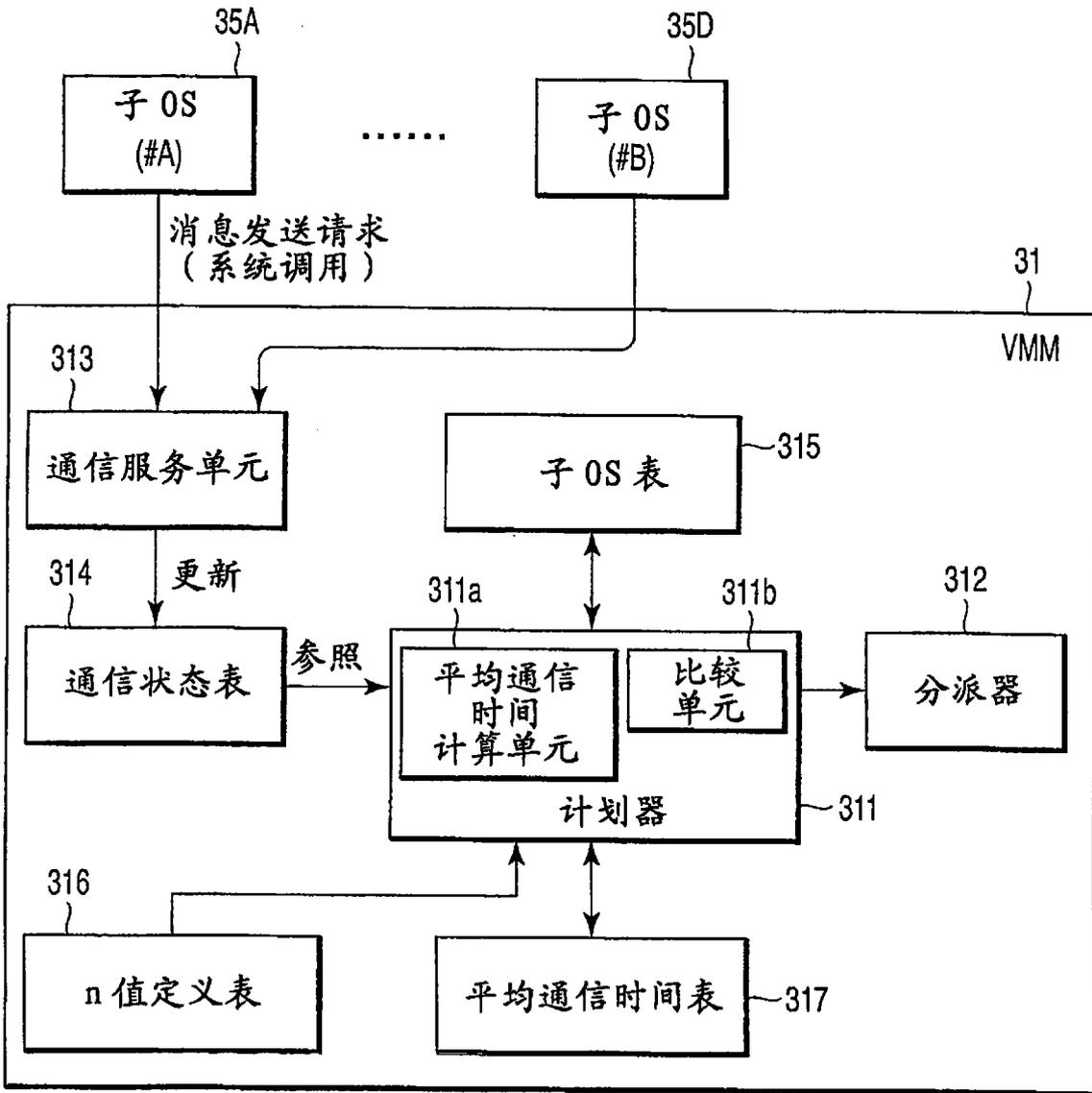


图 12

316

d0	1
d1	2
d2	3
d3	4

图 13

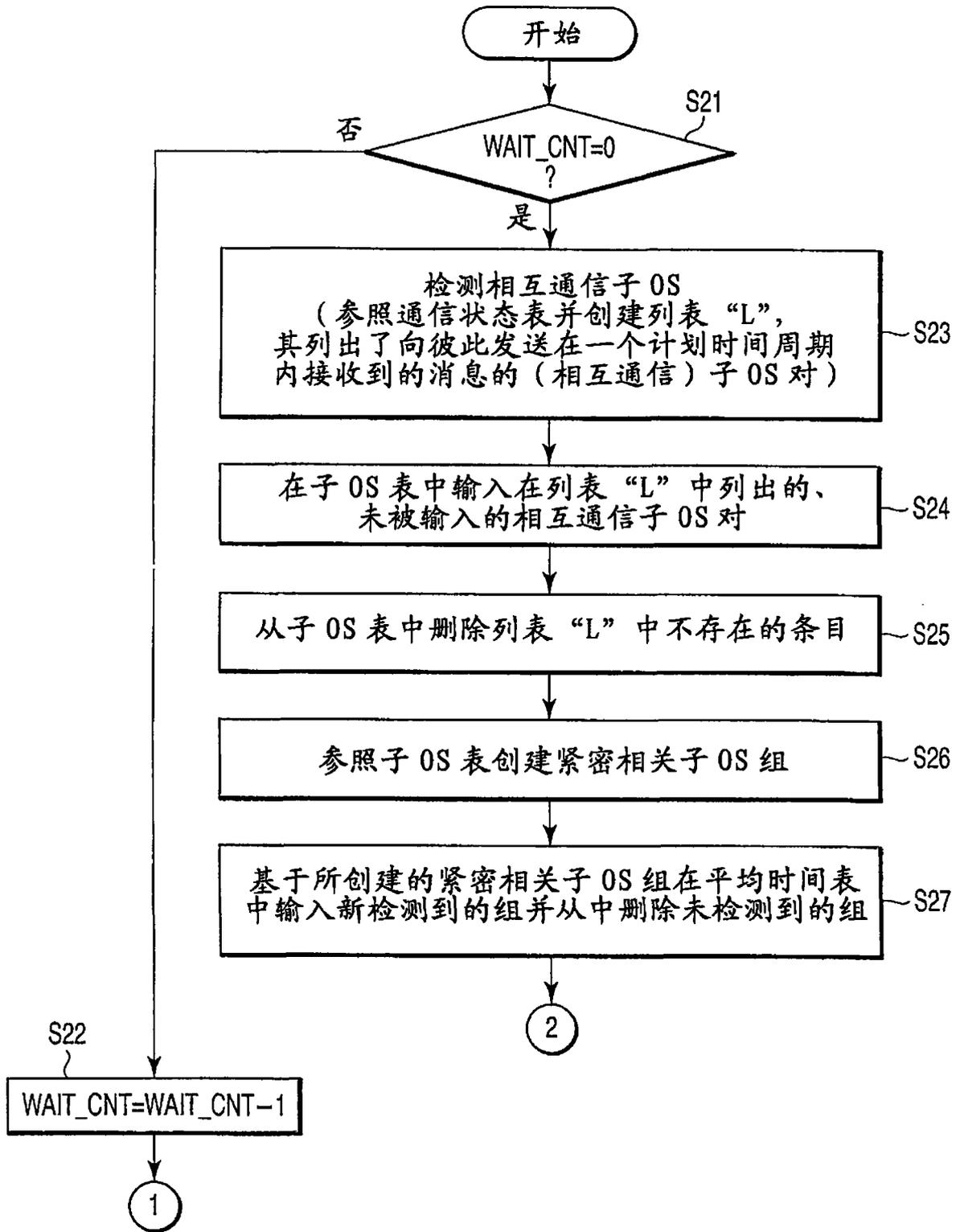


图 14A

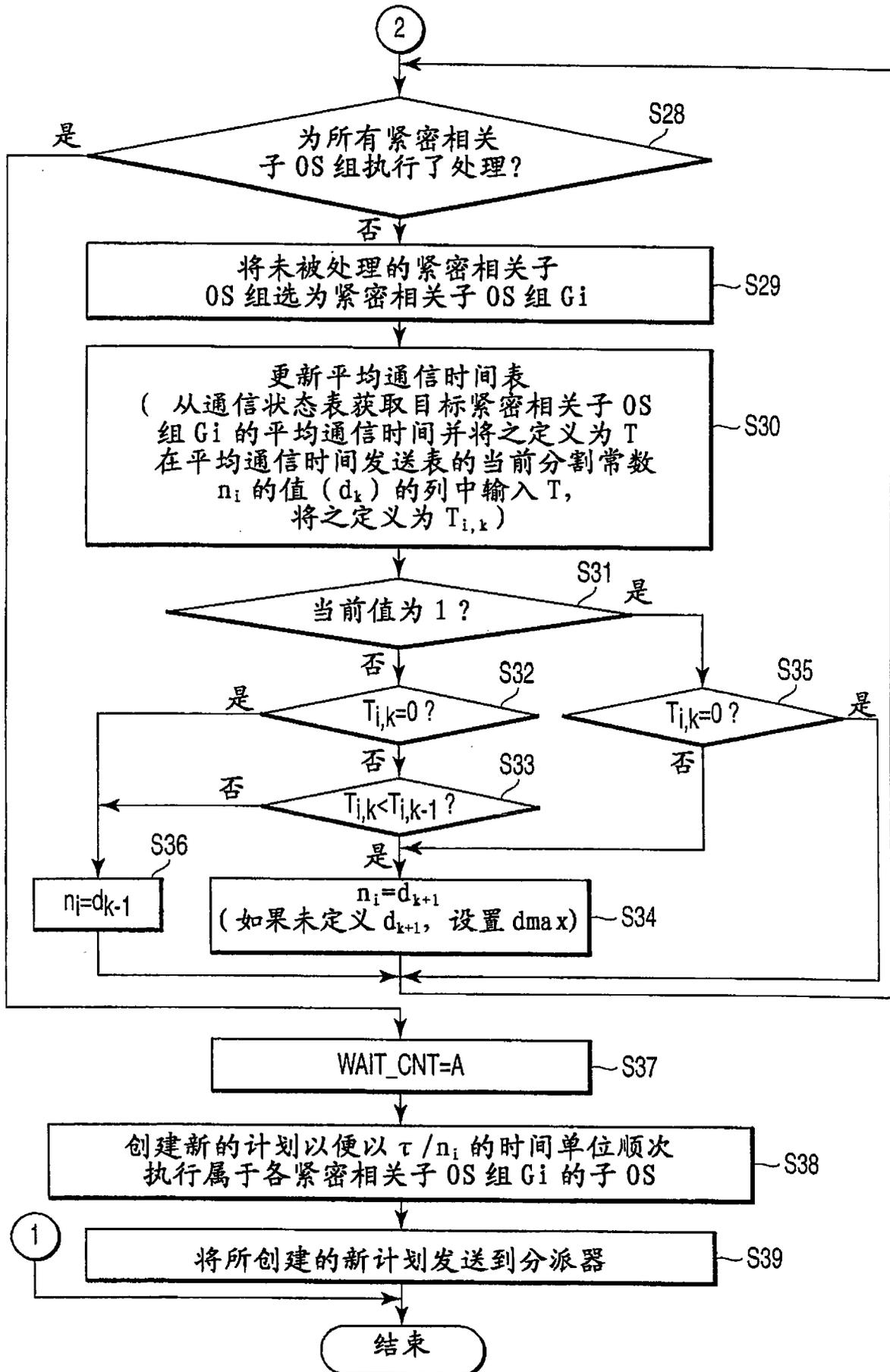


图 14B