(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 116028209 B
(45)授权公告日 2023.10.20

(21)申请号 202210741197.9
(22)申请日 2022.06.28
(65)同一申请的已公布的文献号
申请公布号 CN 116028209 A
(43)申请公布日 2023.04.28
(66)本国优先权数据
202210530859.8 2022.05.16 CN
(73)专利权人 荣耀终端有限公司
地址 518040 广东省深圳市福田区香蜜湖街道东海社区红荔西路8089号深业中城6号楼A单元3401
(72)发明人 程慧文
(74)专利代理机构 深圳中一联合知识产权代理有限公司 44414
专利代理师 路亚芳
(51)Int.Cl.
G06F 9/50(2006.01)
(56)对比文件
CN 114443256 A,2022.05.06
CN 107450998 A,2017.12.08
CN 106383741 A,2017.02.08
CN 102033740 A,2011.04.27
CN 110413365 A,2019.11.05
CN 107608678 A,2018.01.19
审查员 沙爽

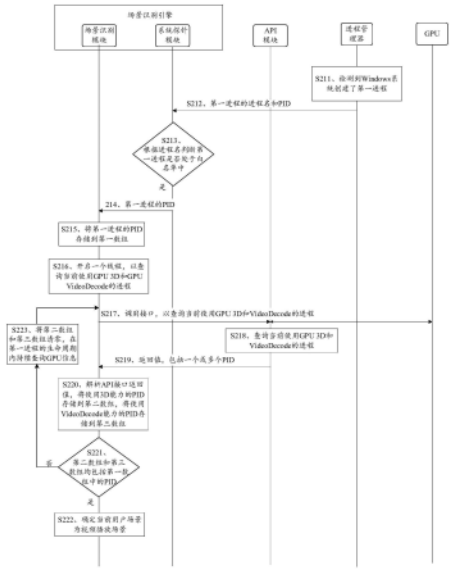
权利要求书3页 说明书31页 附图8页

(54)发明名称

资源调度方法、电子设备及存储介质

(57)摘要

本申请提供了一种资源调度方法、电子设备及存储介质，涉及计算机技术领域。通过该方案，可以先判断新创建的进程是否为预设视频类应用的进程，然后在新创建的进程是预设视频类应用的进程时，判断新创建的进程是否正在使用GPU的三维图形能力和视频解码能力，由此可以准确识别出电子设备当前所处的用户场景是否为视频播放场景，并且在视频播放场景时，根据电子设备当前执行任务的负载情况以及视频播放场景的资源需求对当前任务动态地进行更加精准的资源调度，在保证电子设备性能的同时，满足用户对于视频播放场景的流畅不卡顿体验。



CN 116028209 B

1.一种资源调度方法,其特征在于,应用于电子设备,所述电子设备包括图形处理器GPU及中央处理器CPU,所述方法包括:

响应于用户开启第一应用的第一操作,所述电子设备显示第一窗口,所述第一窗口为焦点窗口;

获取所述第一窗口对应的第一进程的进程信息,所述第一进程的进程信息包括第一进程名称和第一进程标识,所述第一进程名称对应于所述第一应用;

若根据所述第一进程名称确定所述第一应用为预设白名单内的应用程序,则将所述第一进程标识存储到第一数组中;若检测到所述第一进程被清理,则将所述第一进程标识从所述第一数组中删除;所述预设白名单内包括一个或更多个视频类的应用程序;

在判断所述第一数组不为空的情况下,调用Windows接口,获取GPU占用信息,所述GPU占用信息包括第二进程对应的进程标识,所述第二进程为正在使用所述GPU的三维图形能力和视频解码能力的进程;

当所述第二进程对应的进程标识中不包含所述第一进程标识时,确定所述电子设备所处的用户场景不是视频播放场景;

当所述第二进程对应的进程标识中包含所述第一进程标识时,确定所述电子设备所处的用户场景为视频播放场景;

根据所述电子设备的系统负载和所述视频播放场景确定第一调度策略;

根据所述第一调度策略调整所述电子设备的资源分配;

其中,所述调用Windows接口,获取GPU占用信息,包括:

调用所述Windows接口,返回N个第一结构体,所述N个第一结构体中的每个第一结构体包括进程标识PID信息和计数器值;

遍历所述N个第一结构体中的每个第一结构体,获取所述每个第一结构体中的所述计数器值;

当第二结构体中的计数器值大于零时,获取所述第二结构体中的PID信息,所述PID信息中包括正在使用所述GPU的三维图形能力和视频解码能力的进程的标识,所述第二结构体为所述N个第一结构体中的一个结构体;

解析所述PID信息,得到所述GPU占用信息。

2.根据权利要求1所述的方法,其特征在于,在所述调用Windows接口,获取GPU占用信息之前,所述方法还包括:

创建第一线程,所述第一线程用于查询所述GPU占用信息;

其中,所述调用Windows接口,获取GPU占用信息,包括:

响应于所述第一线程创建成功,调用Windows接口,获取所述GPU占用信息。

3.根据权利要求2所述的方法,其特征在于,所述创建第一线程,包括:

当判断第一数组不为空时,创建所述第一线程;

其中,所述第一数组用于存储属于所述预设白名单内的应用程序的进程标识。

4.根据权利要求1所述的方法,其特征在于,在所述调用Windows接口,获取GPU占用信息之后,所述方法还包括:

解析所述GPU占用信息,得到所述第二进程对应的进程标识,其中所述第二进程对应的进程标识包括正在使用所述GPU的三维图形能力的进程的标识,以及正在使用所述GPU的视

频解码能力的进程的标识；

将所述正在使用所述GPU的三维图形能力的进程的标识存储到第二数组；

将所述正在使用所述GPU的视频解码能力的进程的标识存储到第三数组。

5.根据权利要求4所述的方法，其特征在于，所述当所述第二进程对应的进程标识中包含所述第一进程标识时，确定所述电子设备所处的用户场景为视频播放场景，包括：

若所述第二数组和所述第三数组中均包括所述第一数组中的进程标识，则确定所述电子设备所处的用户场景为视频播放场景。

6.根据权利要求5所述的方法，其特征在于，所述方法还包括：

若所述第二数组或所述第三数组中不包括所述第一数组中的进程标识，则将所述第二数组和所述第三数组清零。

7.根据权利要求6所述的方法，其特征在于，在所述将所述第二数组和所述第三数组清零之后，所述方法还包括：

在所述第一进程的生命周期内，周期性地调用所述Windows接口来获取所述GPU占用信息；并根据所述GPU占用信息和所述第一进程标识，确定所述电子设备所处的用户场景是否为视频播放场景。

8.根据权利要求1所述的方法，其特征在于，所述Windows接口为性能数据库PDH函数。

9.根据权利要求1至8中任一项所述的方法，其特征在于，所述根据所述系统负载和所述视频播放场景确定第一调度策略，包括：

根据所述视频播放场景确定第二调度策略，所述第二调度策略包括所述第一进程的进程优先级A、第一输入/输出I/O优先级，所述CPU的第一长时睿频功耗PL1、第一短时睿频功耗PL2及第一能效比EPP；

根据所述系统负载、所述视频播放场景及所述第二调度策略得到所述第一调度策略，所述第一调度策略至少包括所述第一进程的进程优先级B、第二I/O优先级，CPU的第二PL1、第二PL2以及第二EPP；

其中，所述系统负载大于预设的第一数值，所述进程优先级B高于或等于所述进程优先级A，所述第二I/O优先级高于或等于第一I/O优先级，所述第二PL1大于所述第一长时睿频功耗PL1，所述第二PL2大于所述第二PL2，所述第二EPP小于所述第一能效比EPP。

10.根据权利要求1至8中任一项所述的方法，其特征在于，所述第一调度策略包括操作系统OS调度策略以及CPU功耗调度策略；

其中，所述根据所述第一调度策略调整所述电子设备的资源分配，包括：

根据所述OS调度策略调整所述第一进程的进程优先级、输入/输出I/O优先级；

根据所述CPU功耗调度策略调整所述CPU的功耗。

11.根据权利要求10所述的方法，其特征在于，所述方法还包括：

确定所述CPU的芯片平台类型，所述芯片平台类型包括第一类型和第二类型。

12.根据权利要求11所述的方法，其特征在于，所述CPU功耗调度策略包括第一子策略和第二子策略，所述第二子策略为根据所述第一子策略确定的动态调谐技术DTT策略；

其中，所述根据所述CPU功耗调度策略调整所述CPU的功耗，包括：

若所述芯片平台类型为所述第一类型，根据所述第一子策略调整所述CPU的功耗；

若所述芯片平台类型为所述第二类型，根据所述第二子策略调整所述CPU的功耗。

13.一种电子设备,其特征在于,所述电子设备包括:存储器和一个或多个处理器;

其中,所述存储器用于存储计算机程序代码,所述计算机程序代码包括计算机指令;当所述计算机指令被所述处理器执行时,使得所述电子设备执行如权利要求1至12中任一项所述的方法。

14.一种计算机可读存储介质,其特征在于,所述计算机可读存储介质存储有计算机程序,当所述计算机程序在电子设备上运行时,使得所述电子设备执行如权利要求1至12中任一项所述的方法。

资源调度方法、电子设备及存储介质

[0001] 本申请要求于2022年05月16日提交国家知识产权局、申请号为202210530859.8、申请名称为“基于Windows视频场景的资源调度方法及电子设备”的中国专利申请的优先权,其全部内容通过引用结合在本申请中。

技术领域

[0002] 本申请涉及计算机技术领域,尤其涉及一种资源调度方法、电子设备及存储介质。

背景技术

[0003] 随着电子设备性能的提升,电子设备的功耗也越来越高,用户对于电子设备的使用体验也越来越高。通过资源调度可以满足用户的使用体验。传统的资源调度方案是电子设备统计当前执行的所有任务在一段时间内的负载情况,然后根据统计得到的负载情况对当前执行的所有任务进行资源调度。例如,若当前执行的所有任务的负载较大,则可以增大中央处理器(central processing unit,CPU)的功率,但是这样导致CPU在绝大多数用户场景下运行在高性能状态,造成资源浪费和能耗过高的问题。

发明内容

[0004] 本申请提供一种资源调度方法、电子设备及存储介质,可以更准确地识别出Windows前台是否处于视频播放场景,并在判断为视频播放场景时基于视频播放场景的功耗需求对系统性能进行调优,保证整体使用体验的流畅,并降低电子设备的能耗。

[0005] 为达到上述目的,本申请采用如下技术方案:

[0006] 第一方面,本申请提供一种资源调度方法,应用于电子设备,电子设备包括图形处理器GPU及中央处理器CPU,方法包括:

[0007] 响应于用户开启第一应用的第一操作,电子设备显示第一窗口,该第一窗口为焦点窗口;获取第一窗口对应的第一进程的进程信息,该第一进程的进程信息包括第一进程名称和第一进程标识,该第一进程名称对应于第一应用;根据第一进程名称确定第一应用为预设白名单内的应用程序,该预设白名单内包括一个或更多个视频类的应用程序;调用Windows接口,获取GPU占用信息,GPU占用信息包括第二进程对应的进程标识,该第二进程为正在使用GPU的三维图形能力和视频解码能力的进程;当第二进程对应的进程标识中包含第一进程标识时,确定电子设备所处的用户场景为视频播放场景;根据电子设备的系统负载和视频播放场景确定第一调度策略;根据第一调度策略调整电子设备的资源分配。

[0008] 通过本申请提供的方案,可以先判断新创建的进程是否为预设视频类应用的进程,然后在新创建的进程是预设视频类应用的进程时,判断新创建的进程是否正在使用GPU的三维图形能力和视频解码能力,由此可以准确识别出电子设备当前所处的用户场景是否为视频播放场景,并且在视频播放场景时,根据电子设备当前执行任务的负载情况以及视频播放场景的资源需求对当前任务动态地进行更加精准的资源调度,在保证电子设备性能的同时,满足用户对于视频播放场景的流畅不卡顿体验。

[0009] 通过上述方案,当处于白名单中的第一进程启动时,立即去判断该第一进程是否使用GPU 3D和VideoDecode能力,由此判断当前场景是否处于视频播放场景。

[0010] 在另一些实施例中,可以先存储当前运行的且处于白名单中的第一进程,然后确定当前使用GPU 3D和VideoDecode能力的进程,并将该第一进程与当前使用GPU 3D和VideoDecode能力的进程进行比较,根据比较结果判断该第一进程是否正在使用GPU 3D和VideoDecode能力,由此判断当前场景处于视频播放场景。

[0011] 在一些可能实现方式中,在所述调用Windows接口,获取GPU占用信息之前,方法还包括:创建第一线程,该第一线程用于查询GPU占用信息。

[0012] 其中,所述调用Windows接口,获取GPU占用信息,包括:响应于第一线程创建成功,调用Windows接口,获取GPU占用信息。

[0013] 在一些可能实现方式中,如果进程PID数组A不为空,那么开启一个线程,依次查询数组中的进程PID是否正在使用GPU。具体地,调用Windows接口,解析Windows API接口返回值,获取到使用GPU 3D 或者GPU VideoDecode的进程的PID,并将获取到的该PID与待查询的进程PID进行比较,如果获取到的该PID与待查询的进程PID相同,那么确定当前用户场景为视频播放场景。如果获取到的该PID与待查询的进程PID不同,那么确定当前用户场景不是视频播放场景。

[0014] 在一些可能实现方式中,所述创建第一线程,包括:当判断第一数组不为空时,创建第一线程。其中,第一数组用于存储属于预设白名单内的应用程序的进程标识。

[0015] 在一些可能实现方式中,在根据第一进程名称确定第一应用为预设白名单内的应用程序之后,方法还包括:将第一进程标识存储到第一数组中。

[0016] 在一些可能实现方式中,在调用Windows接口,获取GPU占用信息之后,方法还包括:解析GPU占用信息,得到第二进程对应的进程标识,其中第二进程对应的进程标识包括正在使用GPU的三维图形能力的进程的标识,以及正在使用GPU的视频解码能力的进程的标识;

[0017] 将正在使用GPU的三维图形能力的进程的标识存储到第二数组;

[0018] 将正在使用GPU的视频解码能力的进程的标识存储到第三数组。

[0019] 在一些可能实现方式中,当第二进程对应的进程标识中包含第一进程标识时,确定电子设备所处的用户场景为视频播放场景,包括:

[0020] 若第二数组和第三数组中均包括第一数组中的进程标识,则确定电子设备所处的用户场景为视频播放场景。

[0021] 在一些可能实现方式中,方法还包括:若第二数组或第三数组中不包括第一数组中的进程标识,则将第二数组和第三数组清零。

[0022] 可以理解,在确定第一进程当前没有使用GPU 3D和GPU VideoDecode的情况下,将第二数组和第三数组清零,这样可以确保第二数组和第三数组每次更新存储最新的GPU占用信息,由此可以使得基于第二数组和第三数组的视频播放场景判断数据及时更新,因此可以更加准确地实时识别当前场景是否为视频播放场景。

[0023] 在一些可能实现方式中,在将第二数组和第三数组清零之后,方法还包括:在第一进程的生命周期内,周期性地调用Windows接口来获取GPU占用信息;并根据GPU占用信息和第一进程标识,确定电子设备所处的用户场景是否为视频播放场景。

[0024] 可以理解，在视频应用对应的第一进程存活时，视频应用可能在电子设备前台运行且正在播放视频资源，此时可以认为是视频播放场景；视频应用也可能退到电子设备后台或者视频资源未被点播，此时可以认为不是视频播放场景。因此，在视频应用对应的第一进程的生命周期内(即进程存活时)，有必要持续查询第一进程是否正在使用GPU 3D和GPU VideoDecode，以实时确定当前用户场景是否为视频播放场景。

[0025] 在一些可能实现方式中，方法还包括：在检测到第一进程被清理的情况下，将第一进程标识从第一数组中删除。

[0026] 需要说明的是，在用户触发关闭视频应用或者其他原因导致视频应用关闭的情况下，视频应用对应的第一进程被杀死(清理)，此时可以将第一进程的PID从第一数组(第一数组用于存储视频应用的进程标识)中删除，因为在视频应用被关闭后，其对应的第一进程被杀死，此时已经没有必要再查询该第一进程是否正在使用GPU 3D和GPU VideoDecode。

[0027] 在一些可能实现方式中，Windows接口为性能数据库PDH函数。

[0028] 在一些可能实现方式中，调用Windows接口，获取GPU占用信息，包括：

[0029] 调用PDH函数，返回N个第一结构体，该N个第一结构体中的每个第一结构体包括PID信息和计数器值；

[0030] 遍历N个第一结构体中的每个第一结构体，获取每个第一结构体中的计数器值；

[0031] 当第二结构体中的计数器值大于零时，获取第二结构体中的PID信息，PID信息中包括正在使用GPU的三维图形能力和视频解码能力的进程的标识，该第二结构体为N个第一结构体中的一个结构体；

[0032] 解析PID信息，得到GPU占用信息。

[0033] 在一些可能实现方式中，根据系统负载和视频播放场景确定第一调度策略，包括：

[0034] 根据视频播放场景确定第二调度策略，第二调度策略包括第一进程的进程优先级A、第一输入/输出I/O优先级，CPU的第一长时睿频功耗PL1、第一短时睿频功耗PL2及第一能效比EPP；

[0035] 根据系统负载、视频播放场景及第二调度策略得到第一调度策略，第一调度策略至少包括第一进程的进程优先级B、第二I/O优先级，CPU的第二PL1、第二PL2以及第二EPP。

[0036] 其中，系统负载大于预设的第一数值，进程优先级B高于或等于进程优先级A，第二I/O优先级高于或等于第一I/O优先级，第二PL1大于第一PL1，第二PL2大于第二PL2，第二EPP小于第一EPP。

[0037] 其中，系统负载是处于可运行状态的进程和不可中断状态的进程的平均数。可运行状态的进程指正在使用CPU或者等待使用CPU的进程。不可中断状态的进程为等待I/O访问(例如，磁盘I/O)的进程。其中，系统负载可划分为三个等级，分别为轻负载、中负载、重负载。电子设备可预先配置各种用户场景和各种系统负载对应的调度策略。

[0038] 可以理解地，负载越高，则第一进程的进程优先级和I/O优先级越高，可以保证第一进程可以优先占用CPU资源以及进行I/O访问，保证第一进程能够流畅运行。另外，在负载增加时，适当增加PL1、PL2，并降低EPP，以平衡电子设备的性能和功耗。

[0039] 通过上述方案，在保证终端设备性能可以流畅满足用户需求的情况下，降低电子设备的能耗，提升电子设备的续航能力。

[0040] 在一些可能实现方式中，第一调度策略包括操作系统OS调度策略以及CPU功耗调

度策略;其中,根据第一调度策略调整电子设备的资源分配,包括:根据OS调度策略调整第一进程的进程优先级、输入/输出I/O优先级;根据CPU功耗调度策略调整CPU的功耗。

[0041] 在一些可能实现方式中,方法还包括:确定CPU的芯片平台类型,芯片平台类型包括第一类型和第二类型。其中,第一类型的CPU可以为AMD ® 的CPU芯片,第二类型的CPU可以为英特尔 ® 的CPU芯片。

[0042] 在一些可能实现方式中,CPU功耗调度策略包括第一子策略和第二子策略,第二子策略为根据第一子策略确定的动态调谐技术DTT策略。其中,根据CPU功耗调度策略调整CPU的功耗,包括:若芯片平台类型为第一类型,根据第一子策略调整CPU的功耗;若芯片平台类型为第二类型,根据第二子策略调整CPU的功耗。也即,针对AMD ® 和英特尔 ® 的CPU,本申请可以适应性匹配不同的功耗调度策略。

[0043] 第一方面的一种可能设计方式中,第一进程的GPU占用信息包括第一进程的GPU占用率以及GPU引擎;根据进程信息及第一信息确定电子设备所处的用户场景,包括:

[0044] 根据进程信息确定第一进程的类型;若第一进程的类型为视频类,第一进程的GPU占用率大于0,且GPU引擎为GPU视频进程引擎,确定电子设备所处的用户场景为视频播放场景。可以理解地,若第一进程的类型为视频类,则可以先确定用户当前正在使用视频类应用。若第一进程的GPU占用率大于0,则表明第一进程运行过程中有占用GPU的资源。若第一进程的GPU引擎为GPU video processing(视频进程)引擎,则表明第一进程在运行过程中使用GPU进行解码操作。如此,可以确定用户大概率在使用电子设备进行播放视频,即电子设备所处的用户场景为视频播放场景。

[0045] 若第一进程的类型为视频类,第一进程的GPU占用率大于0,且GPU引擎为GPU 3D引擎,确定电子设备所处的用户场景为视频浏览场景。相应地,若第一进程的GPU引擎为GPU 3D引擎,则表明第一进程仅使用GPU进行2D或者3D渲染操作,可以推断用户在使用电子设备浏览视频资源,而非播放视频,即电子设备所处的用户场景为视频浏览场景。

[0046] 第一方面的一种可能设计方式中,方法还包括:若第一进程的类型为游戏类,电源模式为游戏模式,第一进程的GPU占用率大于0,且GPU引擎为GPU 3D引擎,确定电子设备所处的用户场景为游戏场景。

[0047] 可以理解地,若第一进程的类型为游戏类,则可以先确定用户当前正在使用游戏类应用。若第一进程的GPU占用率大于0,则表明第一进程运行过程中有占用GPU的资源。若第一进程的GPU引擎为GPU 3D引擎,则表明第一进程使用GPU进行2D或者3D渲染操作。如此,可以确定用户大概率在使用电子设备打游戏,即电子设备所处的用户场景为游戏场景。

[0048] 第一方面的一种可能设计方式中,外设事件包括键盘输入事件、鼠标输入事件、麦克风输入事件、摄像头输入事件中的一种或多种;根据进程信息及第一信息确定电子设备所处的用户场景,包括:根据进程信息确定第一进程的类型。

[0049] 若第一进程的类型为社交类,且检测到键盘输入事件,确定电子设备所处的用户场景为文字聊天场景。也即,若检测到用户在使用社交应用且同时在打字,则用户大概率在使用社交应用进行聊天,可以确定电子设备所处的用户场景为文字聊天场景。

[0050] 若第一进程的类型为社交类,检测到麦克风输入事件,且未检测到摄像头输入事件,确定电子设备所处的用户场景为语音聊天场景。也即,若检测到用户在使用社交应用且同时在语音输入,则用户大概率在使用社交应用进行语音聊天,可以确定电子设备所处的

用户场景为语音聊天场景。

[0051] 若第一进程的类型为社交类，检测到麦克风输入事件以及摄像头输入事件，确定电子设备所处的用户场景为视频聊天场景。也即，若检测到用户在使用社交应用且同时在视频输入，则用户大概率在使用社交应用进行视频聊天，可以确定电子设备所处的用户场景为视频聊天场景。

[0052] 第一方面的一种可能设计方式中，方法还包括：

[0053] 若第一进程的类型为办公类，且检测到键盘输入事件，确定电子设备所处的用户场景为文档编辑场景。若检测到用户在使用办公应用且同时在打字，则用户大概率在使用办公应用编辑文档，可以确定电子设备所处的用户场景为文档编辑场景。

[0054] 若第一进程的类型为办公类，检测到鼠标输入事件，且未检测到键盘输入事件，确定电子设备所处的用户场景为文档浏览场景。也即，若检测到用户在使用办公应用的过程中使用鼠标但未使用键盘，则用户大概率在使用办公应用浏览文档，可以确定电子设备所处的用户场景为文档浏览场景。

[0055] 若第一进程的类型为办公类，检测到麦克风输入事件以及摄像头输入事件，确定电子设备所处的用户场景为视频会议场景。也即，若检测到用户在使用办公应用的过程中使用摄像头，则用户大概率在使用办公应用进行视频会议，可以确定电子设备所处的用户场景为视频会议场景。

[0056] 第一方面的一种可能设计方式中，电子设备还包括场景识别引擎、系统事件驱动OsEventDriver节点及进程管理器，方法还包括：场景识别引擎向OsEventDriver节点发送第一请求；OsEventDriver节点向进程管理器发送第一请求；响应于第一请求，进程管理器在创建第二进程后，向OsEventDriver节点发送第二进程的进程信息；OsEventDriver节点向场景识别引擎发送第二进程的进程信息。

[0057] 第一方面的一种可能设计方式中，电子设备还包括场景识别引擎及API模块，方法还包括：场景识别引擎向API模块发送第二请求；响应于第二请求，API模块在检测到焦点窗口发生变化后，向场景识别引擎发送第一进程的进程信息。

[0058] 第一方面的一种可能设计方式中，电子设备还包括场景识别引擎、OsEventDriver节点及显卡驱动，方法还包括：场景识别引擎向OsEventDriver节点发送第三请求；OsEventDriver节点向显卡驱动发送第三请求；响应于第三请求，显卡驱动在检测到在GPU进行解码操作后向OsEventDriver节点上报GPU解码事件；OsEventDriver节点向场景识别引擎发送GPU解码事件。

[0059] 第一方面的一种可能设计方式中，电子设备还包括场景识别引擎、OsEventDriver节点及外设驱动，方法还包括：场景识别引擎向OsEventDriver节点发送第四请求；OsEventDriver节点向显卡驱动发送第四请求；响应于第四请求，外设驱动在检测到在外设操作后向OsEventDriver节点上报外设事件；OsEventDriver节点向场景识别引擎发送外设事件。

[0060] 第一方面的一种可能设计方式中，方法包括：响应于用户的第一操作，API模块获取第一进程的名称及第二进程的名称，第二进程为历史的焦点窗口对应的进程；若第一进程的名称与第二进程的名称不一致，向场景识别引擎发送第一进程的进程信息。

[0061] 第一方面的一种可能设计方式中，根据进程信息及第一信息确定电子设备所处的

用户场景，包括：场景识别引擎根据进程信息及第一信息确定电子设备所处的用户场景。

[0062] 第一方面的一种可能设计方式中，电子设备还包括调度引擎，根据系统负载、用户场景得到调度策略包括：场景识别引擎根据用户场景确定第二调度策略；场景识别引擎向调度引擎发送第二调度策略和用户场景；调度引擎向场景识别引擎发送第五请求；响应于第五请求，场景识别引擎获取系统负载，向调度引擎发送系统负载；调度引擎根据系统负载、用户场景及第二调度策略得到第一调度策略。

[0063] 第一方面的一种可能设计方式中，电子设备还包括进程管理器、I/O管理器，OS调度策略包括第一进程的进程优先级B、第二I/O优先级，根据OS调度策略调整第一进程的进程优先级、输入/输出I/O优先级，包括：调度引擎向进程管理器发送第一指令，第一指令携带有第一进程的进程优先级B；响应于接收到第一指令，进程管理器将第一进程的进程优先级调整为进程优先级B；调度引擎向I/O管理器发送第二指令，第二指令携带有第一进程的第二I/O优先级；响应于接收到第二指令，I/O管理器将第一进程的I/O优先级调整为第二I/O优先级。

[0064] 第一方面的一种可能设计方式中，确定CPU的芯片平台类型包括：调度引擎判断CPU的芯片平台类型为第一类型或者第二类型。

[0065] 第一方面的一种可能设计方式中，电子设备还包括电源管理器、系统和芯片OS2SOC驱动节点，第一子策略包括CPU的第二PL1、第二PL2及第二EPP，根据第一子策略调整CPU的功耗，包括：调度引擎向OS2SOC驱动节点发送第三指令，第三指令携带有CPU的第二PL1、第二PL2；OS2SOC驱动节点向CPU发送第三指令；响应于第三指令，CPU将PL1调整为第二PL1，将PL2调整为第二PL2；调度引擎向电源管理器发送第四指令，第四指令携带有CPU的第二EPP；电源管理器向CPU发送第四指令；响应于第四指令，CPU将EPP调整为第二EPP。

[0066] 第一方面的一种可能设计方式中，电子设备还包括英特尔DTT驱动，根据第二子策略调整CPU的功耗包括：调度引擎向英特尔DTT驱动发送第五指令，第五指令携带有第二子策略；英特尔DTT驱动向CPU发送第五指令；响应于第五指令，CPU基于第二子策略运行。

[0067] 通过上述方案，可以根据电子设备当前执行任务的负载情况以及用户场景的资源需求，对当前任务动态地进行更加精准的资源调度，在满足流畅不卡顿的使用体验的同时，降低电子设备的能耗。

[0068] 第二方面，本申请提供一种资源调度装置，该装置包括用于执行上述第一方面中的方法的单元。该装置可对应于执行上述第一方面中描述的方法，该装置中的单元的相关描述请参照上述第一方面的描述，为了简洁，在此不再赘述。

[0069] 其中，上述第一方面描述的方法可以通过硬件实现，也可以通过硬件执行相应的软件实现。硬件或软件包括一个或多个与上述功能相对应的模块或单元。例如，处理模块或单元、显示模块或单元等。

[0070] 第三方面，本申请提供一种电子设备，该电子设备包括存储器和一个或多个处理器；其中，所述存储器用于存储计算机程序代码，所述计算机程序代码包括计算机指令；当所述计算机指令被所述处理器执行时，使得所述电子设备执行第一方面中任一项所述的方法。

[0071] 第四方面，本申请提供一种计算机可读存储介质，该计算机可读存储介质包括计算机指令。当计算机指令在电子设备(如电脑)上运行时，使得该电子设备执行如第一方面

及其任一种可能的设计方式所述的方法。

[0072] 第五方面，本申请提供一种计算机程序产品，当所述计算机程序产品在计算机上运行时，使得所述计算机执行如第一方面及其任一种可能的设计方式所述的方法。

[0073] 第六方面，本申请提供一种芯片系统，该芯片系统包括一个或多个接口电路和一个或多个处理器。该接口电路和处理器通过线路互联。上述芯片系统可以应用于包括通信模块和存储器的电子设备。该接口电路用于从电子设备的存储器接收信号，并向处理器发送接收到的信号，该信号包括存储器中存储的计算机指令。当处理器执行该计算机指令时，电子设备可以执行如第一方面及其任一种可能的设计方式所述的方法。

[0074] 可以理解地，上述提供的第二方面所述的资源调度装置、第三方面所述的电子设备，第四方面所述的计算机可读存储介质，第五方面所述的计算机程序产品及第六方面所述的芯片系统所能达到的有益效果，可参考如第一方面及其任一种可能的设计方式中的有益效果，此处不再赘述。

附图说明

[0075] 图1为本申请实施例提供的资源调度方法的一种电子设备的结构示意图；

[0076] 图2为本申请实施例提供的资源调度方法的一种软件模块架构示意图；

[0077] 图3为本申请实施例提供的资源调度方法的软件模块间的交互示意图；

[0078] 图4为本申请实施例提供的资源调度方法的一种信号交互示意图；

[0079] 图5为本申请实施例提供的资源调度方法的一种界面图；

[0080] 图6为本申请实施例提供的资源调度方法的又一种信号交互示意图；

[0081] 图7为本申请实施例提供的资源调度方法的又一种信号交互示意图；

[0082] 图8为本申请实施例提供的资源调度方法对应的一种芯片结构示意图。

具体实施方式

[0083] 为使本申请实施例的目的、技术方案和优点更加清楚，下面将结合本申请实施例中的附图，对本申请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申请一部分实施例，而不是全部的实施例。

[0084] 本文中术语“和/或”，是一种描述关联对象的关联关系，表示可以存在三种关系，例如，A和/或B，可以表示：单独存在A，同时存在A和B，单独存在B这三种情况。本文中符号“/”表示关联对象是或者的关系，例如A/B表示A或者B。

[0085] 本文中术语“第一”、“第二”仅用于描述目的，而不能理解为指示或暗示相对重要性或者隐含指明所指示的技术特征的数量。由此，限定有“第一”、“第二”的特征可以明示或者隐含地包括一个或者更多个该特征。在本实施例的描述中，除非另有说明，“多个”的含义是两个或两个以上。

[0086] 为了下述各实施例的描述清楚简洁，首先给出相关概念或技术的简要介绍：

[0087] 焦点窗口(focus window)，指拥有焦点的窗口。焦点窗口是唯一可以接收键盘输入的窗口。焦点窗口的确定方式与系统的焦点模式(focus mode)关联。焦点窗口的顶层窗口被称为活动窗口(active window)。同一时间只有一个窗口可以是活动窗口。焦点窗口大概率为用户当前需要使用的窗口。

[0088] 焦点模式,可用于决定鼠标如何使一个窗口获得焦点。一般地,焦点模式可包括三种,分别为:

[0089] (1)点击聚焦(click-to focus),在这种模式下,鼠标点击的窗口即可获得焦点。即当鼠标点击一个可以获得焦点的窗口的任意位置,即可激活该窗口,该窗口便被置于所有窗口的最前面,并接收键盘输入。当鼠标点击其他窗口时,该窗口会失去焦点。

[0090] (2)焦点跟随鼠标(focus-follow-mouse),在这种模式下,鼠标下的窗口可以获取焦点。即当鼠标移到一个可以获得焦点的窗口的范围内,用户不需要点击窗口的某个地方就可以激活这个窗口,接收键盘输入,但该窗口不一定被置于所有窗口的最前面。当鼠标移出这个窗口的范围时,这个窗口也会随之失去焦点。

[0091] (3)草率聚焦(sloppy focus),这种焦点模式与focus-follow-mouse比较类似:当鼠标移到一个可以获得焦点的窗口的范围内,用户不需要点击窗口的某个地方就可以激活这个窗口,接收键盘输入,但该窗口不一定被置于所有窗口的最前面。与focus-follow-mouse不同的是,当鼠标移出这个窗口范围时,焦点并不会随之改变,只有当鼠标移动到别的可以接收焦点的窗口时,系统焦点才改变。

[0092] 进程包括多个线程,线程可以创建窗口。焦点进程为创建焦点窗口的线程所属的进程。

[0093] 长时睿频功耗(power limit1,PL1),指CPU在正常负载下的功耗,相当于热设计功耗,CPU绝大部分时间的运行功耗不超过PL1。

[0094] 短时睿频功耗(power limit2,PL2),指CPU在短时间内可达到的最高功耗,其具有持续时间限制。一般地,PL2大于PL1。

[0095] CPU能效比(energy performance preference,EPP),用于反映CPU的调度倾向,其取值范围为0~255。CPU能效比越小,则表明CPU趋向于高性能;CPU能效比越高,则表明CPU趋向于低功耗。

[0096] 本申请提供一种资源调度方法,提供内核层(kernel层)节点,该节点可以向应用层上报焦点窗口变化事件以及第一信息(包括焦点进程的进程信息、焦点进程对GPU的占用情况、外设事件以及电源模式等),应用层可以根据焦点窗口变化事件以及第一信息确定电子设备当前所处的用户场景,并结合用户场景和电子设备的系统负载确定第一调度策略,基于第一调度策略调整焦点进程的进程优先级、I/O优先级以及CPU的功耗,在流畅满足用户需求(保证焦点进程流畅运行)的情况下,降低电子设备的能耗。

[0097] 请参考图1,为本申请实施例提供的电子设备100的结构示意图。

[0098] 如图1所示,电子设备100可以包括:处理器110,外部存储器接口120,内部存储器121,通用串行总线(universal serial bus,USB)接口130,充电管理模块140,电源管理模块141,电池142,无线通信模块150,显示屏160等。

[0099] 可以理解的是,本实施例示意的结构并不构成对电子设备100的具体限定。在另一些实施例中,电子设备100可以包括比图示更多或更少的部件,或者组合某些部件,或者拆分某些部件,或者不同的部件布置。图示的部件可以以硬件,软件,或者软件和硬件的组合实现。

[0100] 处理器110可以包括一个或多个处理单元,例如:处理器110可以包括应用处理器(application processor,AP),调制解调处理器,图形处理器(graphics processing

unit,GPU),图像信号处理器(image signal processor,ISP),控制器,存储器,视频编解码器,数字信号处理器(digital signal processor,DSP),基带处理器,和/或神经网络处理器(neural-network processing unit,NPU)等。其中,不同的处理单元可以是独立的器件,也可以集成在一个或多个处理器中。

[0101] 控制器可以是电子设备100的神经中枢和指挥中心。控制器可以根据指令操作码和时序信号,产生操作控制信号,完成取指令和执行指令的控制。

[0102] 处理器110中还可以设置存储器,用于存储指令和数据。在一些实施例中,处理器110中的存储器为高速缓冲存储器。该存储器可以保存处理器110刚用过或循环使用的指令或数据。如果处理器110需要再次使用该指令或数据,可从所述存储器中直接调用。避免了重复存取,减少了处理器110的等待时间,因而提高了系统的效率。

[0103] 在一些实施例中,处理器110可以包括一个或多个接口。接口可以包括I2C接口,集成电路内置音频(inter-integrated circuit sound,I2S)接口,脉冲编码调制(pulse code modulation,PCM)接口,通用异步收发传输器(universal asynchronous receiver/transmitter,UART)接口,移动产业处理器接口(mobile industry processor interface,MIPI),通用输入输出(general-purpose input/output,GPIO)接口,用户标识模块(subscriber identity module,SIM)接口,和/或USB接口等。

[0104] 可以理解的是,本实施例示意的各模块间的接口连接关系,只是示意性说明,并不构成对电子设备100的结构限定。在另一些实施例中,电子设备100也可以采用上述实施例中不同的接口连接方式,或多种接口连接方式的组合。

[0105] 充电管理模块140用于从充电器接收充电输入。其中,充电器可以是无线充电器,也可以是有线充电器。充电管理模块140为电池142充电的同时,还可以通过电源管理模块141为电子设备供电。

[0106] 电源管理模块141用于连接电池142,充电管理模块140与处理器110。电源管理模块141接收电池142和/或充电管理模块140的输入,为处理器110,内部存储器121,外部存储器,显示屏160,以及无线通信模块150等供电。在一些实施例中,电源管理模块141和充电管理模块140也可以设置于同一个器件中。

[0107] 无线通信模块150可以提供应用在电子设备100上的包括WLAN(如Wi-Fi),蓝牙,全球导航卫星系统(global navigation satellite system,GNSS),调频(frequency modulation,FM),近距离无线通信技术(near field communication,NFC),红外技术(infrared,IR)等无线通信的解决方案。例如,本申请实施例中,电子设备100可以通过无线通信模块150与电子设备(如无线耳机)建立蓝牙连接。

[0108] 无线通信模块150可以是集成至少一个通信处理模块的一个或多个器件。无线通信模块150经由天线接收电磁波,将电磁波信号调频以及滤波处理,将处理后的信号发送到处理器110。无线通信模块150还可以从处理器110接收待发送的信号,对其进行调频,放大,经天线转为电磁波辐射出去。

[0109] 电子设备100通过GPU,显示屏160,以及应用处理器等实现显示功能。GPU为图像处理的微处理器,连接显示屏160和应用处理器。GPU用于执行数学和几何计算,用于图形渲染。处理器110可包括一个或多个GPU,其执行程序指令以生成或改变显示信息。

[0110] 显示屏160用于显示图像,视频等。该显示屏160包括显示面板。

[0111] 外部存储器接口120可以用于连接外部存储卡，例如Micro SD卡，实现扩展电子设备100的存储能力。外部存储卡通过外部存储器接口120与处理器110通信，实现数据存储功能。例如将音乐，视频等文件保存在外部存储卡中。

[0112] 内部存储器121可以用于存储计算机可执行程序代码，所述可执行程序代码包括指令。处理器110通过运行存储在内部存储器121的指令，从而执行电子设备100的各种功能应用以及数据处理。例如，在本申请实施例中，处理器110可以通过执行存储在内部存储器121中的指令，内部存储器121可以包括存储程序区和存储数据区。

[0113] 其中，存储程序区可存储操作系统，至少一个功能所需的应用程序(比如声音播放功能，图像播放功能等)等。存储数据区可存储电子设备100使用过程中所创建的数据(比如音频数据，电话本等)等。此外，内部存储器121可以包括高速随机存取存储器，还可以包括非易失性存储器，例如至少一个磁盘存储器件，闪存器件，通用闪存存储器(universal flash storage，UFS)等。

[0114] 上述电子设备100的软件系统可以采用分层架构，事件驱动架构，微核架构，微服务架构，或云架构。本申请实施例以分层架构的Windows系统为例，示例性说明电子设备100的软件结构。

[0115] 图2为本申请实施例的电子设备100的软件结构框图。

[0116] 分层架构将软件分成若干个层，每一层都有清晰的角色和分工。层与层之间通过软件接口通信。在一些实施例中，将Windows系统分为用户态和内核态。其中，用户态包括应用层以及子系统动态链接库。内核态自下而上分为固件层、硬件抽象层(hardware abstraction layer，HAL)、内核和驱动层及执行体。

[0117] 如图2所示，应用层包括音乐、视频、游戏、办公、社交等应用程序。应用层还包括环境子系统、场景识别引擎以及调度引擎等。其中，图中仅示出部分应用程序，应用层还可以包括其他应用程序，例如购物应用、浏览器等，本申请不做限定。

[0118] 环境子系统可以将基本的执行体系统服务的某些子集以特定的形态展示给应用程序，为应用程序提供执行环境。

[0119] 场景识别引擎可以识别电子设备100所处的用户场景，并确定与该用户场景匹配的基础调度策略(也可称为第二调度策略)。调度引擎可以获取电子设备100的负载情况，并结合电子设备100的负载情况及上述基础调度策略确定符合电子设备100实际运行情况的实际调度策略(也可称为第一调度策略)。其中，关于场景识别引擎和调度引擎的具体内容见后文，在此暂不描述。

[0120] 子系统动态链接库包括应用程序接口(application programming interface，API)模块，该API模块包括Windows API，Windows原生API等。其中，Windows API和Windows原生API均可以为应用程序提供系统调用入口及内部函数支持，区别在于Windows原生API为Windows系统原生的API。例如，Windows API可包括user.dll、kernel.dll，Windows原生API可包括ntdll.dll。其中，user.dll是Windows用户界面接口，可用于执行创建窗口、发送消息等操作。kernel.dll用于为应用程序提供访问内核的接口。ntdll.dll是重要的Windows NT内核级文件。当Windows启动时，ntdll.dll就驻留在内存中特定的写保护区域，使其他程序无法占用这个内存区域。

[0121] 执行体包括进程管理器、虚拟内存管理器、安全引用监视器、I/O管理器、Windows

管理规范(Windows management instrumentation,WMI)、电源管理器、系统事件驱动(operating system event driver,OsEventDriver)节点、系统与芯片驱动(operating system to System on Chip,OS2SOC)节点等。

[0122] 进程管理器用于创建及中止进程和线程。

[0123] 虚拟内存管理器实现“虚拟内存”。虚拟内存管理器也为高速缓存管理器提供基本的支持。

[0124] 安全引用监视器可在本地计算机上执行安全策略,它保护了操作系统资源,执行运行时对象的保护和监视。

[0125] I/O管理器执行独立于设备的输入/输出,并进一步处理调用适当的设备驱动程序。

[0126] 电源管理器可管理所有支持电源状态更改的设备的电源状态更改。

[0127] 系统事件驱动节点可以与内核和驱动层进行交互,例如与显卡驱动进行交互,在确定存在GPU视频解码事件后,向场景识别引擎上报该GPU视频解码事件。

[0128] 系统与芯片驱动节点可供调度引擎向硬件设备发送调整信息,例如向CPU发送调整PL1和PL2的信息。

[0129] 内核和驱动层包括内核以及设备驱动程序。

[0130] 内核是对处理器体系结构的抽象,将执行体与处理器体系结构的差异相隔离,保证系统的可移植性。内核可以进行线程安排和调度、陷阱处理和异常调度、中断处理和调度等。

[0131] 设备驱动程序运行在内核模式下,为I/O系统和相关硬件之间的接口。设备驱动程序可包括显卡驱动、Intel DTT驱动、鼠标驱动、音视频驱动、摄像头驱动、键盘驱动等。例如,显卡驱动可以驱动GPU运行,Intel DTT驱动可以驱动CPU运行。

[0132] HAL是一个核心态模块,可以隐藏各种与硬件有关的细节,例如I/O接口、中断控制器以及多处理器通信机制等,为运行Windows的不同硬件平台提供统一的服务接口,实现多种硬件平台上的可移植性。需要说明的是,为了维护Windows的可移植性,Windows内部组件和用户编写的设备驱动程序并不直接访问硬件,而是通过调用HAL中的例程。

[0133] 固件层可以包括基本输入输出系统(basic input output system,BIOS),BIOS是一组固化到计算机主板上一个只读存储器(read only memory,ROM)芯片内的程序,它保存着计算机最重要的基本输入输出的程序、开机后自检程序和系统自启动程序,它可从互补金属氧化物半导体(complementary metal oxide semiconductor,CMOS)中读写系统设置的具体信息。其主要功能是为计算机提供最底层的、最直接的硬件设置和控制。Intel DTT驱动可以通过BIOS向CPU发送指令。

[0134] 需要说明的是,本申请实施例仅以Windows系统举例来说明,在其他操作系统中(例如安卓系统,IOS系统等),只要各个功能模块实现的功能和本申请的实施例类似,也能实现本申请的方案。

[0135] 图3示出了电子设备100对资源进行调度时软件和硬件的工作流程示意图。

[0136] 如图3所示,应用层的场景识别引擎包括系统探针模块、场景识别模块及基础策略匹配管理器。场景识别模块可分别与系统探针模块及基础策略匹配管理器进行交互。场景识别模块可以向系统探针模块发送用于获取探针状态的请求。系统探针模块可以获取电子

设备100的运行状态。例如，系统探针模块可以包括电源状态探针、外设状态探针、进程负载探针、音视频状态探针、系统负载探针及系统事件探针等。

[0137] 其中，电源状态探针可以向内核态订阅电源状态事件，根据内核态反馈的回调函数确定电源状态，电源状态包括电池(剩余)电量、电源模式等，电源模式可包括交流电源(alternating current，AC)和直流电源(direct current，DC)。例如，电源状态探针可向执行体层的OsEventDriver节点发送订阅电源状态事件的请求，由OsEventDriver节点向执行体层的电源管理器转发该请求。电源管理器可通过该OsEventDriver节点向电源状态探针反馈回调函数。

[0138] 外设状态探针可以向内核态订阅外设事件，根据内核态反馈的回调函数确定外设事件。外设事件包括鼠标滚轮滑动事件、鼠标点击事件、键盘输入事件、麦克风输入事件、摄像头输入事件等。

[0139] 进程负载探针可以向内核态订阅进程负载，根据内核态反馈的回调函数确定进程(例如，第一进程)的负载。

[0140] 系统负载探针可以向内核态订阅系统负载，根据内核态反馈的回调函数确定系统负载。

[0141] 音视频状态探针可向内核态订阅音视频事件，根据内核态反馈的回调函数确定电子设备100当前存在的音视频事件。音视频事件可包括GPU解码事件等。例如，音视频状态探针可以向执行体层的OsEventDriver节点发送用于订阅GPU解码事件的请求，由OsEventDriver节点向内核和驱动层的显卡驱动转发该请求。显卡驱动可以监控GPU的状态，在监控到GPU在进行解码操作后，通过该OsEventDriver节点向音视频状态探针反馈回调函数。

[0142] 系统事件探针可以向内核态订阅系统事件，根据内核态反馈的回调函数确定系统事件。系统事件可包括窗口变化事件、进程创建事件、线程创建事件等。例如，系统事件探针可以向执行体层的OsEventDriver节点发送订阅进程创建事件的请求，由OsEventDriver节点向进程管理器转发该请求。进程管理器可在创建进程后，通过该OsEventDriver节点向系统事件探针反馈回调函数。又例如，系统事件探针还向API模块发送订阅焦点窗口变化事件，API模块可监控电子设备100的焦点窗口是否发生变化，并在监控到焦点窗口发生变化时，向系统事件探针反馈回调函数。

[0143] 可见，系统探针模块通过向内核态订阅电子设备100的各种事件，再根据内核态反馈的回调函数确定电子设备100的运行状态，即得到探针状态。系统探针模块得到探针状态后，可向场景识别模块反馈该探针状态。场景识别模块接收到探针状态后，可根据该探针状态确定电子设备100所处的用户场景。该用户场景可包括视频播放场景、游戏场景、办公场景及社交场景等。用户场景可以反映用户当前的使用需求。例如，场景识别引擎在识别出焦点窗口为视频应用的窗口时，确定出电子设备100处于视频播放场景，这说明用户需要使用视频应用观看、浏览视频。又例如，场景识别引擎识在识别出焦点窗口为微信™的聊天窗口时，确定电子设备100处于社交场景。场景识别模块还可向基础策略匹配管理器发送该用户场景。基础策略匹配管理器可根据该用户场景确定基础调度策略(也可称为第二调度策略，具体可以参见下文S301、S302中的说明)。基础策略匹配管理器可向场景识别模块反馈该基础调度策略。场景识别模块可向应用层的调度引擎发送该基础调度策略及用户场景。

[0144] 如图3所示，调度引擎包括负载管控器、芯片策略融合器以及调度执行器。其中，负载管控器可接收场景识别模块发送的基础调度策略及用户场景。负载管控器还可从系统探针模块获取系统负载，并根据系统负载和用户场景对该基础调度策略进行调整，得到实际调度策略(也可称为第一调度策略，具体可以参见下文S310中的说明)。实际调度策略中包括OS调度策略和第一CPU功耗调度策略(也可称为第一子策略)。其中，负载管控器可向调度执行器发送该OS调度策略，由调度执行器基于该OS调度策略进行调度。OS调度策略用于调整焦点进程的进程优先级及I/O优先级。示例性的，调度执行器可向进程管理器发送调整焦点进程的进程优先级的指令，响应于该指令，进程管理器对焦点进程的进程优先级进行调整。又例如，调度执行器可向I/O管理器发送调整焦点进程的I/O优先级的指令，响应于该指令，I/O管理器对焦点进程的I/O优先级进行调整。

[0145] 负载管控器还可向芯片策略融合器发送第一CPU功耗调度策略，芯片策略融合器可基于CPU的芯片平台类型及该第一CPU功耗调度策略，得到第二CPU功耗调度策略(也可称为第二子策略，具体可以参见下文S317~S325中的说明)。CPU的芯片平台类型主要分为两种，分别为超威半导体公司®(advanced micro devices，AMD)的CPU和英特尔®(Intel®)的CPU，这两类CPU对于CPU功耗的调整方式并不相同，因此需要进行区分。

[0146] 若CPU的芯片平台类型为AMD(也可以称为第一类型)，调度执行器可以向电源管理器发送调整EPP的指令，以调整CPU的EPP。另外，调度执行器还可以向OS2SOC驱动节点发送调整PL1、PL2的指令，以调整CPU的PL1和PL2。

[0147] 若CPU的芯片平台类型为英特尔®，调度执行器可以通过WMI插件向Intel DTT驱动发送该第二CPU功耗调度策略，第二CPU功耗调度策略可包括PL1的最小值、PL1的最大值、PL2、PL2的持续时间及EPP，由Intel DTT驱动CPU基于该第二CPU功耗调度策略运行。

[0148] 本申请实施方式所提供的资源调度方法，主要分为两个过程，分别为：(1)确定电子设备所处的用户场景；(2)根据电子设备所处的用户场景及电子设备的系统负载进行资源调度。下面将结合附图分别说明上述两个过程。

[0149] 下面将以电子设备处于视频播放场景为例，结合图4，对图3所示的电子设备中部分模块的交互过程进行说明。如图4所示，本申请实施例提供的一种资源调度方法，其确定电子设备所处的用户场景的流程如下：

[0150] S101、系统探针模块向OsEventDriver节点发送订阅进程创建事件的请求。

[0151] 如图3所示，场景识别引擎包括系统探针模块，系统探针模块包括系统事件探针。在本申请实施例中，可以由系统事件探针向位于执行体层的OsEventDriver节点发送订阅进程创建事件的请求。其中，订阅进程创建事件的请求也可以称为第一请求。

[0152] 在一种可选的实施方式中，订阅进程创建事件的请求可以携带有进程名称。即场景识别引擎可以仅订阅指定进程的创建事件，减少不相干进程的创建事件的干扰。例如，指定进程可以为视频应用的进程、游戏应用的进程、办公应用的进程、社交应用的进程等等。当然，在其他实施方式中，场景识别引擎也可以不对订阅的进程创建事件做出限制。

[0153] S102、OsEventDriver节点向进程管理器发送订阅进程创建事件的请求。

[0154] 进程创建事件的请求可以参考S101的描述，在此不做赘述。

[0155] 也就是说，场景识别引擎的系统事件探针可以通过OsEventDriver节点向进程管理器发送订阅进程创建事件的请求。

[0156] 可以理解地，OsEventDriver节点会向进程管理器注册一个回调，注册该回调的作用是当进程管理器创建进程后，可以向OsEventDriver节点返回该进程创建事件。

[0157] S103、系统探针模块向OsEventDriver节点发送订阅GPU解码事件的请求。

[0158] 仍然如图3所示，系统探针模块还包括音视频状态探针。在本申请实施例中，可以由系统探针模块的音视频状态探针向OsEventDriver节点发送订阅GPU解码事件的请求。其中，订阅GPU解码事件的请求也可以称为第三请求。

[0159] S104、OsEventDriver节点向显卡驱动发送订阅GPU解码事件的请求。

[0160] 也就是说，场景识别引擎的音视频状态探针可以通过OsEventDriver节点向显卡驱动发送订阅GPU解码事件的请求。同样地，OsEventDriver节点可向显卡驱动注册一个回调，注册该回调的作用是当显卡驱动监控到GPU进行解码操作后，可以向OsEventDriver节点返回该GPU解码事件。

[0161] S105、系统探针模块向API模块发送订阅焦点窗口变化事件的请求。

[0162] API模块可包括由user32.dll实现的windows用户界面接口，该接口可用于创建窗口。在一种可选的实施方式中，可以由系统探针模块的系统事件探针向API模块的windows用户界面接口发送订阅焦点窗口变化事件的请求。其中，订阅焦点窗口变化事件的请求也可以称为第二请求。

[0163] 同样地，该系统事件探针可向API模块注册一个回调，注册该回调的作用是当API模块(的windows用户界面接口)监控到焦点窗口发生变化时，可以向系统事件探针返回该焦点窗口变化事件。

[0164] 焦点窗口为拥有焦点的窗口，大概率为用户当前需要使用的窗口。因此，通过监控焦点窗口，可以确定用户的使用需求。例如，焦点窗口为视频应用的窗口，则表明用户需求浏览、播放视频。又例如，焦点窗口为游戏应用的窗口，则表明用户需求打游戏。通过监控焦点窗口是否发生变化，可以确定用户需求是否发生改变。例如，焦点窗口由视频应用的窗口变为游戏应用的窗口，则表明用户当前的需求由看视频变成了打游戏。

[0165] 需要说明的是，上述S101、S103及S105之间没有严格的先后顺序，其可以按照图4中所示的顺序依次执行，也可以同时执行，也可以按照S103、S101、S105的顺序依次执行、按照S103、S105、S101的顺序依次执行、按照S105、S101、S103的顺序依次执行或者按照S105、S103、S101的顺序依次执行。相应地，S102、S104及S106之间也没有严格的先后顺序，只要满足S102在S101之后执行、S104在S103之后执行以及S106在S105之后执行即可，在此不做具体限制。

[0166] S106、响应于接收到用户开启视频应用的操作，视频应用向进程管理器发送创建进程请求。

[0167] 其中，创建进程请求包括视频应用程序的存储地址。

[0168] 视频应用可以通过API模块的kernel32.dll接口及Ntdll.dll接口向进程管理器发送创建进程的请求(图未示)。

[0169] S107、进程管理器创建视频应用进程。

[0170] 具体的，进程管理器可以通过该存储地址查询到视频应用程序的二进制文件。通过加载视频应用程序的二进制文件，可以创建进程运行的环境，启动视频应用进程。

[0171] 其中，Windows操作系统将一个应用程序的一次运行定义为一个进程。一个进程可

以拥有多个线程。窗口是窗口结构的实例，是一种图形用户界面(graphical user interface,GUI)资源，窗口是由线程创建的，线程可以拥有它所创建的所有窗口。在本申请实施例中，电子设备运行视频应用，则进程管理器需创建该视频应用的进程，即视频应用进程(即第一进程)。视频应用进程包括多个线程，多个线程包括线程1，线程1可用于创建视频应用的主窗口，主窗口为集成有视频应用全部功能按键的窗口。

[0172] S108、进程管理器向OsEventDriver节点上报进程创建事件。

[0173] 其中，进程创建事件可包括进程管理器所创建的进程的名称。在本申请实施例中，该进程的名称为视频应用进程的名称。当然，若进程管理器创建的是其他应用的进程，该进程的名称也对应为其他应用进程的名称。

[0174] 前文已经说明，OsEventDriver节点向进程管理器发送了订阅进程创建事件的请求，且注册了回调。因此，进程管理器在创建视频应用进程后可向OsEventDriver节点上报进程创建事件。

[0175] S109、OsEventDriver节点向系统探针模块上报进程创建事件。

[0176] 其中，关于进程创建事件的描述见S108，在此不再赘述。

[0177] 在本申请实施例中，该OsEventDriver节点可向系统探针模块的系统事件探针上报该进程创建事件。

[0178] S110、系统探针模块向场景识别模块发送进程创建事件。

[0179] S111、响应于线程1的调用请求，API模块创建窗口1。

[0180] 进程管理器创建视频应用进程后，视频应用进程的线程1主动调用API模块的windows用户界面接口创建窗口1。示例性的，如图5中(a)所示，电子设备可以显示窗口101，该窗口101可以为桌面，也可以称为主界面。该窗口101包括视频应用的图标102。电子设备可以接收用户点击该视频应用的图标102的操作，响应于该操作，如图5中的(b)所示，电子设备显示窗口103(即窗口1，也可以称为第一窗口)。在上述过程中，焦点窗口由原本的窗口101变为窗口103。

[0181] S112、API模块向系统探针模块上报焦点窗口事件。

[0182] 在本申请实施例中，API模块的windows用户界面接口创建窗口1后，可以获取第一进程(即焦点进程)的名称及第三进程的名称，第一进程为当前的焦点窗口(即窗口1)对应的进程，第三进程为上一个焦点窗口(例如，窗口2)对应的进程。示例性的，窗口1对应的进程为视频应用进程(第一进程)，该进程的名称例如为hlive.exe，窗口2对应的进程为windows程序管理器的进程(第三进程)，该进程的名称例如为explorer.exe。由于第一进程的名称与第三进程的名称不一致，API模块确定焦点窗口发生变化，向系统探针模块的系统事件探针上报焦点窗口事件。其中，焦点窗口变化事件包括第一进程(即焦点进程)的名称。示例性的，第一进程为视频应用进程，焦点窗口变化事件携带有视频应用进程的名称。

[0183] 需要说明的是，在电子设备已经启动视频应用的情况下，电子设备可以不用执行S106~S111。在系统探针模块向API模块发送订阅焦点窗口变化事件的请求后，若用户将焦点窗口切换为视频应用的窗口，API模块同样可以检测到焦点窗口发生变化，并向系统探针模块上报焦点窗口事件。

[0184] S113、系统探针模块向场景识别模块发送焦点窗口事件。

[0185] S114、场景识别模块确定第一进程所属的类型为视频类。

[0186] 电子设备可以预先配置有应用名单，场景识别模块可以查询应用名单中是否包括第一进程。若应用名单中包括第一进程，场景识别模块可以确定第一进程所属的类型。其中，应用名单包括每个应用的进程名称及应用所属的类型。示例性的，应用名单可以如表1所示：

[0187] 表1

[0188]

应用	进程名称	类型
荣耀视频	hlive.exe	视频类
word	word.exe	办公类
射击游戏	shot.exe	游戏类
微信™	wechat.exe	社交类
……	……	……

[0189] 例如，第一进程的名称为hlive.exe，则场景识别模块可以确定第一进程所属的类型为视频类。又例如，第一进程的名称为wechat.exe，则场景识别模块可以确定第一进程所属的类型为社交类。需要说明的是，上述表1仅作为示例，实际上表1还可包括更多应用的进程名称及其所属的类型。

[0190] 需要说明的是，此步骤的目的在于初步判断电子设备所处的用户场景。电子设备所处的用户场景可包括视频播放场景、游戏场景、社交场景、办公场景、浏览器场景等等。其中，视频播放场景进一步可包括视频播放场景、视频浏览场景。社交场景进一步可包括文字聊天场景、语音聊天场景、视频聊天场景等。办公场景进一步可包括文档编辑场景、文档浏览场景、视频会议场景等。浏览器场景可包括浏览网页场景及播放视频场景等。

[0191] 在本步骤中，通过第一进程所属的类型，可以确定电子设备所处的用户场景的类型。例如，若确定第一进程所属的类型为视频类，则可以确定电子设备处于视频播放场景；又例如，若确定第一进程所属的类型为游戏类，则可以确定电子设备处于游戏场景。为了进一步分析用户需求，场景识别模块还可以进一步结合其他参数(例如，外设事件、GPU运行状态等)来分析电子设备所处的具体场景，以达到分析结果更加准确的效果，其具体内容参见后文，在此暂不描述。

[0192] S115，响应于接收到用户播放视频的操作，视频应用向API模块发送视频播放指令。

[0193] 具体的，视频应用可向API模块的DirectX API发送该视频播放指令。该视频播放指令可包括视频的缓存地址。

[0194] S116、API模块读取视频文件。

[0195] API模块可根据视频播放指令中携带的缓存地址，读取对应的视频文件。

[0196] S117、API模块向显卡驱动发送解码指令。

[0197] S118、显卡驱动向GPU发送启动指令。

[0198] S119、GPU进行解码。

[0199] 具体的，GPU可通过GPU video processing引擎对该视频文件进行解码操作。

[0200] S120、GPU向显卡驱动上报解码事件。

[0201] S121、显卡驱动向OsEventDriver节点上报解码事件。

[0202] S122、OsEventDriver节点向系统探针模块上报解码事件。

[0203] 具体的,OsEventDriver节点向系统探针模块的音视频状态探针上报该解码事件。

[0204] S123、系统探针模块向场景识别模块发送解码事件。

[0205] S124、场景识别模块向系统探针模块发送指令1。

[0206] 其中,指令1指示系统探针模块获取第一进程的GPU占用率。该指令1可携带有第一进程的名称。

[0207] S125、系统探针模块向进程管理器发送获取第一进程的GPU占用率的请求。

[0208] 其中,该获取焦点进程的GPU占用率的请求可以包括第一进程的名称。

[0209] 在一种可选的实施方式中,可以由系统探针模块的音视频状态探针向进程管理器发送获取第一进程的GPU占用率的请求。

[0210] S126、进程管理器采集第一进程的GPU占用率。

[0211] 具体的,进程管理器可以通过显卡驱动的图像核心(graphics kernel)接口采集第一进程的GPU占用率。

[0212] S127、进程管理器向系统探针模块发送第一进程的GPU占用率。

[0213] 进程管理器可向系统探针模块的音视频状态探针发送第一进程的GPU占用率。

[0214] S128、系统探针模块向场景识别引擎发送第一进程的GPU占用率。

[0215] S129、场景识别模块判断第一进程的GPU占用率是否大于0。

[0216] 其中,若第一进程的GPU占用率大于0,则执行S130。

[0217] 通过第一进程的GPU占用率,可以确定第一进程在运行过程中是否使用GPU,若第一进程的GPU占用率大于0,则可以认为第一进程在运行过程中使用了GPU;若第一进程的GPU占用率为0,则表明第一进程在运行过程中未使用GPU。

[0218] S130、场景识别模块向系统探针模块发送指令2。

[0219] 其中,指令2指示系统探针模块获取第一进程的GPU引擎。该指令2可携带有第一进程的名称。

[0220] S131、系统探针模块向进程管理器发送获取第一进程的GPU引擎的请求。

[0221] 其中,可以由系统探针模块的音视频状态探针向进程管理器发送获取第一进程的GPU引擎的请求。获取第一进程的GPU引擎的请求包括第一进程的名称。

[0222] GPU引擎包括GPU 3D引擎、GPU copy引擎、GPU video encode引擎、GPU video processing引擎。其中,GPU 3D引擎主要负责处理2D或者3D图形。GPU copy引擎主要用于传输数据。GPU video encode引擎主要用于进行编码操作。GPU video processing引擎主要进行解码操作。在一些实施例中,GPU video processing引擎也可以由GPU video decode引擎代替。

[0223] S132、进程管理器获取第一进程的GPU引擎。

[0224] 具体的,进程管理器可以通过显卡驱动的graphics kernel接口获取第一进程的GPU引擎。

[0225] S133、进程管理器向系统探针模块发送消息1,消息1指示第一进程的GPU引擎为GPU video processing引擎。

[0226] 具体的,进程管理器可向系统探针模块的音视频状态探针发送该消息,再由音视频状态将该消息转发给场景识别模块。

[0227] S134、系统探针模块向场景识别模块发送消息1。

[0228] S135、场景识别模块判断第一进程的GPU引擎是否为GPU video processing引擎。

[0229] 若第一进程的GPU引擎为GPU video processing引擎,则执行S129;若第一进程的GPU引擎不为GPU video processing引擎,则执行S130。

[0230] 在S114步骤中,场景识别引擎已经确定第一进程所属的类型为视频类,即可以确定电子设备处于视频播放场景。通过步骤S135,场景识别引擎可以确定第一进程通过GPU执行的具体操作,进而确定用户在使用视频应用的具体操作。例如,若第一进程的GPU引擎为GPU video processing引擎,则表明第一进程在使用GPU进行解码操作,可以认为用户在使用视频应用播放视频。又例如,第一进程的GPU引擎不为GPU video processing引擎,则表明第一进程没有在使用GPU进行解码操作,那么用户大概率在视频应用上浏览视频资源,还未播放视频。

[0231] S136、场景识别模块根据第一进程的进程信息确定用户场景为视频播放场景。

[0232] 其中,第一进程的进程信息包括第一进程的名称、第一进程所属的应用类型、第一进程的GPU占用率及第一进程使用的GPU引擎等信息。

[0233] 综合上述内容可知,若第一进程(焦点进程)的类型为视频类、第一进程的GPU占用率大于0且第一进程的GPU引擎为GPU video processing引擎,则可以确定电子设备处于视频播放场景。

[0234] 除了本申请上述实施例提供的识别视频播放场景的方式之外,本申请实施例还提供了另一种识别视频播放场景的方式,如图6所示,该方法包括下述S211至S223。由于电子设备所处的用户场景可能会频繁变化或切换,相应地需要动态地识别用户场景。下面给出的这种识别方式能够动态地、准确地识别用户场景是否为视频播放场景。

[0235] S211、进程管理器检测到Windows系统创建了新的进程(称为第一进程)。

[0236] 其中,响应于用户开启第一应用的第一操作(例如点击第一应用的图标),电子设备显示第一窗口,第一窗口为焦点窗口。相应地,电子设备创建第一进程,该第一进程为第一应用的进程。可以理解,第一进程为第一窗口对应的进程。第一进程也称为焦点进程。

[0237] S212、进程管理器将该第一进程的名称(简称进程名)和进程标识(process identifier,PID)发送给系统探针模块。

[0238] 其中,系统探针模块中包括音视频状态探针。具体地,进程管理器将第一进程的进程名和PID发送给系统探针模块的音视频状态探针。

[0239] 示例性地,在爱奇艺®视频应用被开启时,系统为其创建的进程的名称可以为QyClient.exe。

[0240] 其中,每个进程对应一个PID。PID是各进程的身份标识,当应用程序开始运行时系统会为该应用程序的进程自动分配一个唯一的PID。需要说明的是,一个应用程序的一次运行为一个进程,因此当该应用程序的进程中止后,PID会被系统回收。当应用程序再次开始运行时,系统会为该应用程序的进程重新分配另一个PID。

[0241] 可以理解,在创建第一进程的情况下,第一进程对应有一个进程名和一个PID。

[0242] S213、系统探针模块根据进程名判断第一进程是否处于白名单中。

[0243] 在一些实施例中,白名单中可以包括预设的视频类应用的信息以及对应的可用进程名。在另一些实施例中,白名单中可以包括多个视频类应用的进程名。

[0244] 具体到本申请方案,若第一进程的进程名在白名单中,或者说在白名单中查找到第一进程的进程名,则可以判断第一进程为视频类应用的进程。

[0245] 下面以爱奇艺®视频应用为例,示例性地示出了白名单内容:

[0246] <!--爱奇艺-->

[0247] <Application id="4001" name="爱奇艺" sceneType="4">

[0248] <process num="0" name="QyClient.exe" />//进程名称

[0249] <process num="1" name="QyPlayer.exe" />//进程名称

[0250] <process num="2" name="QyPlayerCore.exe" />//进程名称

[0251] </Application>

[0252] S214、如果第一进程处于白名单中,那么系统探针模块将该第一进程的PID发送给场景识别模块。

[0253] S215、场景识别模块接收第一进程的PID,并将第一进程的PID存储到第一数组。

[0254] 其中,第一数组可以称为数组A。

[0255] 在一些实施例中,第一进程的PID可以采用数组形式进行存储。示例性地,可以将第一进程的PID保存到数组A中。例如,假设第一进程的PID为112233,那么数组A = {112233}。此时数组A不为空。

[0256] 在一些实施例中,除了将第一进程的PID存储到第一数组中之外,当检测到创建了新的进程,且该新的进程也处于白名单中时,可以将该新的进程的PID也存储到第一数组中。例如,假设创建的新进程的PID为234,那么数组A = {112233, 234}。也就是说,第一数组中可以存储当前运行的多个进程的PID,且该多个进程均处于白名单中。

[0257] S216、场景识别模块开启一个线程,以查询当前使用GPU 3D和GPU视频解码(VideoDecode)能力的进程。

[0258] 若第一数组不为空,则场景识别模块开启一个线程,以查询哪些进程当前正在使用GPU 3D和GPU VideoDecode能力。

[0259] S217、场景识别模块调用API接口,以查询当前使用GPU 3D和GPU VideoDecode能力的进程。

[0260] 该线程可以调用Windows接口,即API接口,来查询哪些进程当前正在使用GPU 3D和GPU VideoDecode能力。

[0261] S218、API接口查询当前使用GPU 3D和GPU VideoDecode能力的进程。

[0262] S219、API接口向场景识别模块发送返回值,该返回值包括一个或多个PID。

[0263] 其中,该一个或多个PID包括当前使用GPU 3D的进程的PID和当前使用GPU VideoDecode的进程的PID。

[0264] S220、场景识别模块解析API接口返回值,将使用3D能力的PID存储到第二数组,将使用VideoDecode能力的PID存储到第三数组。

[0265] 其中,第二数组可以称为数组B,第三数组可以称为数组C。

[0266] 在本实施例中,可以调用Windows API接口函数来获取计数器数据,具体可以通过API集来访问系统事件和性能数据的众多计数器。既可以实时地得到计数器的值,也可以从一个日志文件中读取计数器数据。例如,API集可以用于监控记录当前GPU 占用率,例如GPU 3D当前使用情况和GPU VideoDecode当前使用情况。

[0267] 例如可以使用性能数据库(performance data helper,PDH)获取系统性能数据,例如GPU占用率。PDH提供用于收集当前性能数据的 API、将性能数据保存到日志文件以及从日志文件中读取数据。PDH为高级 API,可简化收集性能计数器数据。其有助于查询分析、元数据缓存、在示例之间匹配实例、从原始值计算格式化值、从日志文件读取数据以及将数据保存到日志文件。若要使用 PDH 函数收集性能数据,可以执行以下步骤:创建查询;向查询添加计数器;收集性能数据;显示性能数据;关闭查询。

[0268] 下面示例性地示出了场景识别模块调用的Windows接口函数。

[0269] PDH_FUNCTION

[0270] PdhGetFormattedCounterArrayW(

[0271] _In_PDH_HCOUNTER hCounter,

[0272] _In_DWORDdwFormat,

[0273] _Inout_ LPDWORDlpdwBufferSize,

[0274] _Out_LPDWORDlpdwItemCount,//返回lpdwItemCount个PPDH_FMT_COUNTERVALUE_ITEM_W结构体

[0275] _Out_writes_bytes_opt_(* lpdwBufferSize) PPDH_FMT_COUNTERVALUE_ITEM_W ItemBuffer

[0276]);

[0277] 该API返回lpdwItemCount个PPDH_FMT_COUNTERVALUE_ITEM_W结构体,表示可能使用GPU 能力的进程,遍历每个PPDH_FMT_COUNTERVALUE_ITEM_W结构体内容。

[0278] 上述PPDH_FMT_COUNTERVALUE_ITEM_W结构体示例性地表示为:

[0279] typedef struct _PDH_FMT_COUNTERVALUE_ITEM_W {

[0280] LPWSTRszName;//szName包含进程PID

[0281] PDH_FMT_COUNTERVALUEFmtValue;

[0282] } PDH_FMT_COUNTERVALUE_ITEM_W, * PPDH_FMT_COUNTERVALUE_ITEM_W;

[0283] 进一步调用下述结构体,上述PDH_FMT_COUNTERVALUE对应的FmtValue可以通过下述的结构体中的doubleValue得到。

[0284] typedef struct _PDH_FMT_COUNTERVALUE {

[0285] DWORDCStatus;

[0286] union {

[0287] LONG longValue;

[0288] doubledoubleValue;// GPU占用率

[0289] LONGLONGlargeValue;

[0290] LPCSTR AnsiStringValue;

[0291] LPCWSTRWideStringValue;

[0292] };

[0293] } PDH_FMT_COUNTERVALUE, * PPDH_FMT_COUNTERVALUE;

[0294] 其中,doubleValue表示正在使用GPU能力的百分比数值。

[0295] 如果FmtValue.doubleValue的值大于0,进一步解析szName的值,例如szName为(L"pid_10308_luid_0x00000000_0x0000CA13_phys_0_eng_11_engtype_

VideoProcessing″)，通过解析可知进程ID(10308)，最终得到使用GPU 3D或者GPU VideoDecode能力的进程id。

[0296] 在一些可选的实施方式中，调用上述Windows接口，分别获取到使用GPU 3D和GPU VideoDecode能力的进程数组。

[0297] 如果判断到进程使用GPU 3D，那么将该进程PID存入数组B中，最终得到使用GPU 3D能力的进程PID数组，例如数组B= {112233，112235，...}。

[0298] 如果判断到进程使用GPU VideoDecode，那么将该进程PID存入数组C中。最终得到使用GPU VideoDecode能力的进程PID数组，例如数组C= {112233，112555，..}。

[0299] S221、场景识别模块判断第二数组和第三数组是否包括第一数组中的PID。

[0300] 可以理解，若判断第二数组和第三数组均包括第一数组中的PID，则可以判断第二数组和第三数组均中包括第一进程的PID。

[0301] 例如，若数组A== {112233，234}，数组B={112233，112235，...}，数组C={112233，112555，..}，则可以判断第二数组和第三数组均包括第一数组中的PID(112233)。其中，该PID(112233)为视频应用对应的第一进程的进程标识。

[0302] 一方面，如果第二数组和第三数组均包括第一进程的PID，说明第一进程正在占用GPU的视频播放资源，那么继续执行下述的S222。

[0303] 另一方面，如果第二数组或第三数组不包括第一进程的PID，说明视频应用对应的第一进程目前未占用GPU的视频播放资源，那么继续执行下述的S223。

[0304] S222、场景识别模块确定当前用户场景为视频播放场景。

[0305] 也就是说，如果第二数组和第三数组中均包含第一数组的进程PID，说明视频应用对应的第一进程正在占用GPU的视频播放资源，那么可以确定当前用户场景为视频播放场景，即用户正在通过视频APP观看视频的场景。

[0306] S223、当第二数组或第三数组不包括第一数组中的PID时，场景识别模块将第二数组和第三数组清零，并在第一进程的生命周期内持续查询GPU信息。

[0307] 可以理解，在确定第一进程当前没有使用GPU 3D和GPU VideoDecode后，将第二数组和第三数组清零，这样可以确保第二数组和第三数组每次更新存储最新的GPU占用信息，由此可以使得基于第二数组和第三数组的视频播放场景判断数据及时更新，因此可以更加准确地实时识别当前场景是否为视频播放场景。

[0308] 在S223之后，返回继续执行上述S217至S221，这样可以实时监测电子设备当前所处场景是否为视频播放场景。

[0309] 可以理解，在视频应用对应的第一进程存活时，视频应用可能在电子设备前台运行且正在播放视频资源，此时可以认为是视频播放场景；视频应用也可能退到电子设备后台或者视频资源未被点播，此时可以认为不是视频播放场景。因此，在视频应用对应的第一进程的生命周期内(即第一进程存活且第一进程为焦点进程时)，有必要持续查询第一进程是否正在使用GPU 3D和GPU VideoDecode能力，以实时确定当前用户场景是否为视频播放场景。

[0310] 需要说明的是，在用户触发关闭视频应用或者其他原因导致视频应用关闭的情况下，视频应用对应的第一进程被杀死(清理)，此时可以将第一进程的PID从第一数组(第一数组用于存储视频应用的进程标识)中删除，因为在视频应用被关闭后，其对应的第一进程

被杀死,此时已经没有必要再查询该第一进程是否正在使用GPU 3D和GPU VideoDecode能力。

[0311] 在另一些可选的实施方式中,如果进程PID数组A不为空,那么开启一个线程,依次查询数组中的进程PID是否正在使用GPU。具体地,调用Windows接口,解析Windows API接口返回值,获取到使用GPU 3D 或者GPU VideoDecode的进程的PID,并将获取到的该PID与待查询的进程PID进行比较,如果获取到的该PID与待查询的进程PID相同,那么确定当前用户场景为视频播放场景。如果获取到的该PID与待查询的进程PID不同,那么确定当前用户场景不是视频播放场景。

[0312] 本申请给出了两种方案,一种方案是当处于白名单中的第一进程启动时,立即去判断该第一进程是否使用GPU 3D和VideoDecode能力,由此判断当前场景是否处于视频播放场景。另一种方案是先存储当前运行的且处于白名单中的第一进程,然后确定当前使用GPU 3D和VideoDecode能力的进程,并将该第一进程与当前使用GPU 3D和VideoDecode能力的进程进行比较,根据比较结果判断该第一进程是否正在使用GPU 3D和VideoDecode能力,由此判断当前场景处于视频播放场景。

[0313] 下面整体详细地说明本申请实施例调用Windows接口来获取正在使用Video 3D能力和Video Decode能力的进程PID的过程。

[0314] 可以通过调用两次PdhGetFormattedCounterArrayW()接口来获取Video 3D 数组和Video Decode数组。

[0315] PDH_FUNCTION

[0316] PdhGetFormattedCounterArrayW(

[0317] _In_PDH_HCOUNTERhCounter,

[0318] _In_DWORDdwFormat,

[0319] _Inout_ LPDWORDlpdwBufferSize,

[0320] _Out_LPDWORDlpdwItemCount,

[0321] _Out_writes_bytes_opt_(* lpdwBufferSize) PPDH_FMT_COUNTERVALUE_ITEM_W ItemBuffer

[0322]);

[0323] 其中,获取Video 3D 数组和Video Decode数组,具体可以通过PDH 函数中的hCounter来区分。

[0324] 对于3D,hCounter的构造方式可以为:

[0325] PdhAddCounter(hQuery, L"\\GPU Engine(*engtype_3d)\\Utilization Percentage",0,&h3DCounter)。

[0326] 对于VideoDecode,hCounter的构造方式可以为:

[0327] PdhAddCounter(hQuery, L"\\GPU Engine(*engtype_Video*)\\Utilization Percentage",0,&hCounter)。

[0328] 将上述3D对应的hCounter和VideoDecode对应的hCounter分别传入PdhGetFormattedCounterArrayW()后,即可获取到使用GPU 3D和GPU VideoDecode能力的信息。

[0329] 其中,上述API返回lpdwItemCount个PPDH_FMT_COUNTERVALUE_ITEM_W结构体,表

示可能使用GPU 能力的进程,遍历每个PPDH_FMT_COUNTERVALUE_ITEM_W结构体内容。

[0330] 上述PPDH_FMT_COUNTERVALUE_ITEM_W结构体示例性地表示为:

[0331] typedef struct _PDH_FMT_COUNTERVALUE_ITEM_W {

[0332] LPWSTRszName;

[0333] PDH_FMT_COUNTERVALUEFmtValue;

[0334] } PDH_FMT_COUNTERVALUE_ITEM_W, * PPDH_FMT_COUNTERVALUE_ITEM_W;

[0335] 上述PPDH_FMT_COUNTERVALUE_ITEM_W结构体进一步调用下述结构体,上述PDH_FMT_COUNTERVALUE对应的FmtValue可以通过下述的结构体中的doubleValue得到。

[0336] typedef struct _PDH_FMT_COUNTERVALUE {

[0337] DWORDCStatus;

[0338] union {

[0339] LONG longValue;

[0340] doubledoubleValue;//使用GPU 百分比

[0341] LONGLONGlargeValue;

[0342] LPCSTR AnsiStringValue;

[0343] LPCWSTRWideStringValue;

[0344] };

[0345] } PDH_FMT_COUNTERVALUE, * PPDH_FMT_COUNTERVALUE;

[0346] 如果FmtValue. doubleValue的值大于0,那么进一步解析szName(L"pid_10308_luid_0x00000000_0x0000CA13_phys_0_eng_11_engtype_VideoProcessing")的值,取出进程PID(10308),最终得到使用GPU 3D或者GPU VideoDecode能力的进程PID。

[0347] 上述API通过调用对应的结构体来查询进程PID,然后根据查询结果将当前使用GPU 3D和GPU VideoDecode能力的进程的PID 存储到对应的3D 和VideoDecode数组中。

[0348] 下面结合实际应用场景示例性地举例说明视频播放场景的识别过程。

[0349] 白名单内容:

[0350] <!--爱奇艺-->

[0351] <Application id="4001" name="爱奇艺" sceneType="4">

[0352] <process num="0" name="QyClient.exe" />

[0353] <process num="1" name="QyPlayer.exe" />

[0354] <process num="2" name="QyPlayerCore.exe" />

[0355] </Application>

[0356] 进程管理器检测到进程名为QyClient.exe、QyPlayer.exe、QyPlayerCore.exe的进程运行,并判断这三个进程存在于白名单中,因此进程管理器将进程名和PID发送给音视频状态探针。

[0357] 音视频状态探针将进程PID存到数组A中,A={2988, 12668, 2188}。

[0358] 音视频状态探针开启一个线程,查询当前使用GPU 3D和GPU VideoDecode能力的进程。下述的表1'示意性地示出了查询结果:

[0359] 表1'

[0360]

PID	3D(%)	VideoDecode(%)
1572	7.1	0
2988(QyClient.exe)	1.7	0
12668(QyPlayer.exe)	3.5	0
2188(QyPlayerCore.exe)	2.8	1.9

[0361] 音视频状态探针根据查询结果,将当前使用GPU 3D和GPU VideoDecode能力的进程的PID 存储到对应的3D数组B和VideoDecode数组C中。

[0362] 例如,数组B={1572, 2988, 12668, 2188},数组C={2188}。

[0363] 可以判断出数组A中的内容存在于3D数组B和VideoDecode数组C中,因此可以确定当前用户场景为视频播放场景。

[0364] 需要说明的是,上述步骤仅以电子设备处于视频播放场景下的视频播放场景为例进行说明。实际上,电子设备还可以处于其他用户场景(例如,游戏场景、办公场景、社交场景、视频浏览场景等)。

[0365] 在一种可选的实施方式中,若场景识别引擎确定第一进程(焦点进程)的类型属于游戏类、CPU的电源模式变为游戏模式(game mode)、第一进程的GPU占用率大于0且第一进程的GPU引擎为GPU 3D引擎,则可以确定电子设备处于游戏场景。

[0366] 其中,系统探针模块的电源状态探针可以向电源管理器发送订阅电源模式变化事件的请求。电源管理器可以在电源模块转换为游戏模式(game mode)时,向系统探针模块的电源状态探针上报该电源模式变化事件。如此,场景识别引擎可通过电源模式变化事件确定CPU的电源模式是否为game mode。

[0367] 另外,场景识别引擎获取第一进程的类型的过程可以参阅图4中S101、S102、S105、S106~S114,场景识别引擎判断第一进程的GPU占用率是否大于0且第一进程的GPU引擎是否为GPU 3D引擎的过程参阅S124~S135。区别在于将视频应用更换为游戏应用,在此不再赘述。

[0368] 上述内容说明了如何识别电子设备所处的用户场景,在确定电子设备所处的用户场景后,电子设备还可根据自身所处的用户场景和系统负载进行资源调度,使得电子设备的CPU可以按照用户的实际需求进行运行,达到在不影响用户体验的情况下避免CPU出现性能过剩的效果。

[0369] 下面,继续以电子设备处于视频播放场景为例,说明电子设备的资源调度过程。如图7所示,本申请实施例提供的一种资源调度方法,其进行资源调度的流程如下:

[0370] 如图7所示,本申请实施方式提供的资源调度方法还包括:

[0371] S301、场景识别模块向基础调度策略匹配管理器发送场景信息。

[0372] 其中,场景信息用于指示电子设备所处的用户场景。示例性的,电子设备可以预先为不同的用户场景分配唯一标识,该场景信息则可包括用户场景所的唯一标识。例如,该标识(例如为V01)可指示电子设备处于视频播放场景。又例如,该标识(例如为V02)可指示电子设备处于视频浏览场景。

[0373] 关于场景识别模块确定电子设备所处的用户场景的过程,具体参阅S101~S136,在此不再赘述。

[0374] S302、基础策略匹配管理器根据场景信息得到调度策略1。

[0375] 其中，调度策略1包括OS调度策略1及CPU功耗调度策略1。OS调度策略1包括第一进程的进程优先级A及第一I/O优先级。其中，调度策略1也可称为第二调度策略。

[0376] 第一进程的优先级用于衡量第一进程抢占CPU的能力，其优先级越高，则可以优先满足其对CPU资源的占用需求，从而使得第一进程的运行流畅度越高。在一种可选的实施方式中，焦点进程的优先级从高到低依次包括等级：实时、高、高于正常、正常、低于正常、低。其中，第一进程的优先级也可以理解为焦点进程优先级(focus process priority，FPP)。

[0377] 第一进程的I/O优先级用于衡量系统对第一进程的磁盘和I/O请求的响应度，期优先级越高，则第一进程的磁盘和I/O请求的响应度越高，即响应速度越快。在一种可选的实施方式中，焦点进程I/O优先级从高到低依次包括等级：关键、高、正常、低、非常低。其中，第一进程的I/O优先级也可以理解为焦点进程I/O优先级(focus process IO priority，FPP_IO)。

[0378] CPU功耗调度策略1包括CPU的第一PL1、第一PL2以及第一EPP。

[0379] 可见，调度策略1可以调整第一进程的进程优先级、I/O优先级以及CPU功耗。

[0380] 在一种可选的实施方式中，电子设备可以预先配置各种用户场景和其对应的调度策略。示例性的，各种用户场景和其对应的调度策略的对应关系可以如表2所示。

[0381] 示例性的，若确定电子设备所处的用户场景为社交场景下的文字聊天场景，则调度策略1包括：第一进程的进程优先级A为正常，第一进程的第一I/O优先级为正常，CPU的第一PL1为12W，第一PL2为60W，第一EPP为220。需要说明的是，表2中的调度策略仅作为示例，在实际应用中，进程优先级、I/O优先级、PL1、PL2及EPP的值可以与表2中的值不一致。另外，表2仅示出了部分场景的调度策略，实际电子设备还可配置比表2更多的调度策略。

[0382] 需要说明的是，上述调度策略为默认电子设备处于轻负载状态时的调度策略，其可以为电子设备预先统计每个应用在对应的负载特征下的CPU功耗，并根据统计得到的负载特征及CPU功耗配置的。因此，基础策略匹配管理器得到的调度策略1可作为电子设备进行调度的策略的参考方案，电子设备还可根据该调度策略1并结合实际的系统负载来得到实际的调度策略。

[0383] 表2

[0384]

用户场景	子场景	OS 调度		CPU 功耗调度		
		FPP	FPP_IO	PL1（瓦/W）	PL2（瓦/W）	EPP
社交	文字聊天	正常	正常	12	60	220
	语音聊天	正常	正常	15	60	210
	视频聊天	正常	正常	20	60	180
办公	文档编辑	正常	正常	12	60	220
	文档浏览	正常	正常	12	60	220
	视频会议	正常	正常	20	60	180
浏览器	浏览网页	正常	正常	12	60	220
	播放视频	正常	正常	20	60	180
游戏	游戏中	高于正常	高	40	90	50
视频	视频播放	正常	正常	18	60	200
	视频浏览	正常	正常	12	60	220
.....	.....	.....	.....	.....	.....	.....

[0385] S303、基础策略匹配管理器向场景识别模块发送调度策略1。

[0386] S304、场景识别模块向负载管控器发送调度策略1和场景信息。

[0387] 也即，基础策略匹配管理器确定调度策略1后，通过场景识别模块向负载管控器转发调度策略1。在一种可选的实施方式中，场景识别模块可以分两个步骤分别向负载管控器发送调度策略1和场景信息。

[0388] S305、负载管控器向系统探针模块发送获取系统负载的请求。

[0389] 其中，系统负载是处于可运行状态的进程和不可中断状态的进程的平均数。可运行状态的进程指正在使用CPU或者等待使用CPU的进程。不可中断状态的进程为等待I/O访问(例如，磁盘I/O)的进程。

[0390] S306、系统探针模块向进程管理器发送获取系统负载的请求。

[0391] 如图3所示，系统探针模块包括系统负载探针，可以由系统负载探针向进程管理器发送获取系统负载的请求。在一种可选的实施方式中，也可以由OsEventDriver节点向进程管理器转发系统负载探针的获取系统负载的请求(图未示)。

[0392] S307、进程管理器获取系统负载。

[0393] S308、进程管理器向系统探针模块发送系统负载。

[0394] 具体的，进程管理器可以向系统探针模块的系统负载探针发送该系统负载。在一种可选的实施方式中，也可以由OsEventDriver节点向系统负载探针转发该系统负载(图未示)。

[0395] S309、系统探针模块向负载管控器发送系统负载。

[0396] S310、负载管控器根据系统负载、场景信息及调度策略1，得到调度策略2。

[0397] 调度策略2可包括OS调度策略2(也可以称为OS调度策略)和CPU功耗调度策略2(也可以称为第一子策略)。CPU功耗调度策略2包括PL1`、PL2`、EPP`，PL1`为负载管控器调整后的PL1，也可以称为第二PL1。PL2`为负载管控器调整后的PL2，也可以称为第二PL2。EPP`为负载管控器调整后的EPP，也可以称为第二EPP。其中，调度策略2也可以称为第一调度策略。

[0398] 在一种可选的实施方式中，负载管控器可将系统负载划分为三个等级，分别为轻负载、中负载、重负载。电子设备可预先配置各种用户场景和其对应的调整策略。例如，调整

策略可以如表3所示：

[0399] 表3

[0400]

用户场景	子场景	系统负载	OS 调度		CPU 功耗调度		
			FPP	FPP_IO	PL1（瓦/W）	PL2（瓦/W）	EPP
社交	文字聊天	轻	正常	正常	12	60	220
		中	正常	正常	+8（20）	+30（90）	-70（150）
		重	高于正常	高	+43（55）	+45（105）	-100（120）
视频	视频播放	轻	正常	正常	18	60	200
		中	正常	正常	+22（40）	+30（90）	-50（150）
		重	正常	高	+37（55）	+45（105）	-100（100）
.....	.....		.....	.....	.....	.....	.....

[0401] 示例性的，若电子设备处于视频播放场景，且根据表2可知调度策略1为：视频应用进程的进程优先级为正常，视频应用进程的I/O优先级为正常，CPU的PL1（即第一PL1）为18W，PL2（即第一PL2）为60W，EPP（即第一EPP）为200。在这种情况下，若系统负载为轻负载，则不需要调整调度策略，即调度策略2为调度策略1。若系统负载为中负载，则需保持视频应用进程的进程优先级为正常，视频应用进程的I/O优先级为正常，PL1在18W的基础上增加22W，PL2在60W的基础上增加30W，EPP在200的基础上减少50，即调度策略2为：视频应用进程的进程优先级为正常、视频应用进程的I/O优先级为正常（OS调度策略2），PL1`为40W，PL2`为90W，EPP`为150（CPU调度策略2）。若系统负载为重负载，则需保持视频应用进程的进程优先级为正常，调整视频应用进程的I/O优先级为高，PL1在18W的基础上增加37W，PL2在60W的基础上增加45W，EPP在200的基础上减少100，即调度策略2为：视频应用进程的进程优先级为正常，视频应用进程的I/O优先级为高，PL1`为55W，PL2`为105W，EPP`为100。

[0402] 需要说明的是，表3仅示出了部分用户场景及其对应的调整策略，电子设备还可配置比表3更多的调整策略，在此不做具体限制。

[0403] 在一种可选的实施方式中，系统负载与CPU功耗之间满足特定的映射关系（例如通过特定算式进行映射），负载管控器也可以通过该特定算式和系统负载计算得到CPU功耗，进而得到调度策略2。

[0404] S311、负载管控器向调度执行器发送OS调度策略2。

[0405] 其中，OS调度策略2包括第一进程的进程优先级B、第二I/O优先级。

[0406] S312、调度执行器向I/O管理器发送指令1。

[0407] 其中，指令1携带有第一进程的第二I/O优先级。另外，如图3所示，调度执行器包括I/O优先级接口，可以由该I/O优先级接口向I/O管理器发送指令1。其中，该指令1也可以称为第二指令。

[0408] S313、响应于指令1，I/O管理器调整第一进程的I/O优先级。

[0409] 也即，I/O管理器可将第一进程的I/O优先级调整为第二I/O优先级。如此，可以保证第一进程可以优先进行I/O访问，减少第一进程在I/O访问过程中的响应时间。

[0410] S314、调度执行器向进程管理器发送指令2。

[0411] 其中，指令2携带有第一进程的进程优先级B。另外，如图3所示，调度执行器还包括进程优先级接口，可以由该进程优先级接口向进程管理器发送指令2。其中，该指令2也可以

称为第一指令。

[0412] S315、响应于接收到指令2，进程管理器调整第一进程的进程优先级。

[0413] 也即，进程管理器可将第一进程的进程优先级调整为进程优先级B。如此，第一进程可以优先占用CPU资源，保证第一进程能够流畅运行。

[0414] 可见，通过调整第一进程的I/O优先级和进程优先级，可以优先保证第一进程的I/O访问以及对CPU资源的消耗，使得第一进程可以正常、流畅地运行，保证用户有良好的体验。

[0415] 需要说明的是，S312与S314之间不存在严格的先后顺序，可以先执行S312再执行S314，也可以先执行S314再执行S312，也可以同时执行S314和S312。

[0416] S316、负载管控器向芯片策略融合器发送CPU功耗调度策略2。

[0417] S317、芯片策略融合器判断CPU的芯片平台类型为AMD®或者Intel®。

[0418] AMD®公司的CPU芯片和Intel®公司的CPU，对于CPU功耗的调整方式并不相同，因此需要进行区分。其中，若CPU的芯片平台类型为AMD®（也可以称为第一类型），则执行S318；若PU的芯片平台类型为Intel®（也可以称为第二类型），则执行S325。

[0419] S318、芯片策略融合器向调度执行器发送CPU功耗调度策略2。

[0420] 其中，CPU功耗调度策略2包括PL1`、PL2`、EPP`。

[0421] S319、调度执行器向OS2SOC驱动节点发送指令3。

[0422] 其中，指令3携带有PL1`、PL2`。也即，指令3用于调整CPU的PL1和PL2。其中，指令3也可以称为第三指令。

[0423] 在一种可选的实施方式中，可以由调度执行器的CPU功耗调度接口向OS2SOC驱动节点发送指令3。

[0424] S320、OS2SOC驱动节点向CPU发送指令3。

[0425] S321、响应于指令3，CPU调整PL1和PL2。

[0426] 也即，CPU可将PL1调整为PL1`，将PL2调整为PL2`。

[0427] S322、调度执行器向电源管理器发送指令4。

[0428] 其中，指令4携带有EPP`。也即，指令4用于调整CPU的EPP。指令4也可以称为第四指令。

[0429] S323、电源管理器向CPU发送指令4。

[0430] S324、响应于指令4，CPU调整EPP。

[0431] 也即，CPU可将EPP调整为EPP`。

[0432] S325、芯片策略融合器根据CPU功耗调度策略2确定动态调谐技术策略号。

[0433] 动态调谐技术(dynamic tuning technology，DTT)是英特尔®公司在英特尔®处理器和英特尔®独立显卡之间自动并动态分配功耗，以优化性能并延长电池续航时间的技术，其可以使CPU和GPU的性能得到提升，智能混合工作负载功率平衡。

[0434] 可以理解地，DTT策略号与CPU功耗调度策略2可以存在映射关系。在BIOS中会构建一张DTT策略表，任何一条CPU功耗调度策略2都能通过其内的参数(PL1`、PL2`、EPP`)映射到DTT策略表中某个DTT策略号，如表4所示。

[0435] 其中，DTT策略号可用于标识一种DTT策略(也可以称为第二子策略)，DTT策略号对应的DTT策略用于调整CPU的PL1_MINI、PL1_MAX、PL2、PL2_TIME、EPO Gear。PL1_MINI为PL1

的最小值,PL1_MAX为PL1的最大值,PL2_TIME为PL2的持续时间。能效-性能优化挡位(Energy Performance Optimize Gear,EPO Gear)用来表征DTT调节CPU能效比(EPP)的力度,取值范围1~5,值越大,调节EPP时越倾向能效;值越小,调节EPP时越倾向性能。

[0436] 表4

[0437]

CPU 功耗调度策略 2			DTT 策略表					
PL1'	PL2'	EPP'	DTT 策略号	PL1_MINI	PL1_MAX	PL2	PL2_TIME	EPO Gear
-1	-1	-1	0	30	40	95	28	3
14	32	153	1	16	36	32	4	4
8	25	230	2	8	28	25	2	4
...	...	...	...					

[0438] 需要说明的是,表4仅示出部分PL1'、PL2'、EPP'和DTT策略号的对应关系,实际上还可包括比表4更多的信息。示例性的,若CPU功耗调度策略2指示PL1`为-1,PL2`为-1且EPP`为-1,则可以确定DTT策略号为0,其对应的PL1_MINI为30,PL1_MAX为40,PL2为95,PL2_TIME为28,EPO Gear为3。

[0439] S326、芯片策略融合器向调度执行器发送DTT策略号。

[0440] 在一种可选的实施方式中,芯片策略融合器也可直接向调度执行器发送该DTT策略号对应的功DTT策略(即第二子策略)。

[0441] S327、调度执行器向Intel DTT 驱动发送DTT策略号。

[0442] S328、Intel DTT 驱动向CPU发送DTT策略号。

[0443] 可以理解地,该Intel DTT 驱动可通过BIOS向CPU发送DTT策略号。

[0444] S329、CPU基于DTT策略号运行。

[0445] 可见,若CPU的芯片平台类型为AMD®,芯片策略融合器可以通过调度执行器向电源管理器发送调整EPP的指令,电源管理器可调整CPU的EPP。另外,调度执行器还可以向OS2SOC驱动节点发送调整PL1、PL2的指令,OS2SOC驱动节点驱动CPU的PL1和PL2。

[0446] 若CPU的芯片平台类型为英特尔®,则芯片策略融合器可以确定CPU功耗调度策略2得到DTT策略号,并通过调度执行器通过bios向Intel DTT 驱动发送DTT策略号,使得CPU基于该DTT策略号运行,达到调整功耗的效果。

[0447] 可以理解地,本申请可以获取焦点窗口变化事件以及第一信息(包括焦点进程的进程信息、焦点进程对GPU的占用情况、外设事件以及电源模式等),并根据焦点窗口变化事件以及第一信息确定电子设备当前所处的用户场景,并结合用户场景和电子设备的系统负载确定第一调度策略,基于第一调度策略调整焦点进程的进程优先级、I/O优先级以及CPU的功耗,在流畅满足用户需求(保证焦点进程流畅运行)的情况下,降低电子设备的能耗。

[0448] 本申请实施例还提供一种电子设备,所述电子设备包括存储器和一个或多个处理器。

[0449] 其中,所述存储器用于存储计算机程序代码,所述计算机程序代码包括计算机指令;当所述计算机指令被所述处理器执行时,使得所述电子设备执行上述方法实施例中的各个功能或者步骤。该电子设备的结构可以参考图1所示的电子设备100的结构。

[0450] 本申请实施例中的电子设备可以为笔记本电脑或者个人计算机(personal computer,PC)等,本申请实施例不作具体限定。

[0451] 本申请实施例还提供一种芯片系统，如图8所示，该芯片系统包括至少一个处理器801和至少一个接口电路802。处理器801和接口电路802可通过线路互联。例如，接口电路802可用于从其它装置（例如电子设备的存储器）接收信号。又例如，接口电路802可用于向其它装置（例如处理器801）发送信号。示例性的，接口电路802可读取存储器中存储的指令，并将该指令发送给处理器801。当所述指令被处理器801执行时，可使得电子设备执行上述实施例中的各个步骤。当然，该芯片系统还可以包含其他分立器件，本申请实施例对此不作具体限定。

[0452] 本申请实施例还提供一种计算机存储介质，该计算机存储介质包括计算机指令，当所述计算机指令在上述电子设备上运行时，使得该电子设备执行上述方法实施例中手机执行的各个功能或者步骤。

[0453] 本申请实施例还提供一种计算机程序产品，当所述计算机程序产品在计算机上运行时，使得所述计算机执行上述方法实施例中手机执行的各个功能或者步骤。

[0454] 可以理解的是，本申请实施例提供的电子设备为了实现上述功能，其包含了执行各个功能相应的硬件结构和/或软件模块。本领域技术人员应该很容易意识到，结合本文中所公开的实施例描述的各示例的单元及算法步骤，本申请实施例能够以硬件或硬件和计算机软件的结合形式来实现。某个功能究竟以硬件还是计算机软件驱动硬件的方式来执行，取决于技术方案的特定应用和设计约束条件。专业技术人员可以对每个特定的应用来使用不同方法来实现所描述的功能，但是这种实现不应认为超出本申请实施例的范围。

[0455] 本申请实施例可以根据上述方法示例对上述电子设备进行功能模块的划分，例如，可以对应各个功能划分各个功能模块，也可以将两个或两个以上的功能集成在一个处理模块中。上述集成的模块既可以采用硬件的形式实现，也可以采用软件功能模块的形式实现。需要说明的是，本申请实施例中对模块的划分是示意性的，仅仅为一种逻辑功能划分，实际实现时可以有另外的划分方式。

[0456] 通过以上实施方式的描述，所属领域的技术人员可以清楚地了解到，为描述的方便和简洁，仅以上述各功能模块的划分进行举例说明，实际应用中，可以根据需要而将上述功能分配由不同的功能模块完成，即将装置的内部结构划分成不同的功能模块，以完成以上描述的全部或者部分功能。

[0457] 在本申请所提供的几个实施例中，应该理解到，所揭露的装置和方法，可以通过其它的方式实现。例如，以上所描述的装置实施例仅仅是示意性的，例如，所述模块或单元的划分，仅仅为一种逻辑功能划分，实际实现时可以有另外的划分方式，例如多个单元或组件可以结合或者可以集成到另一个装置，或一些特征可以忽略，或不执行。另一点，所显示或讨论的相互之间的耦合或直接耦合或通信连接可以是通过一些接口，装置或单元的间接耦合或通信连接，可以是电性，机械或其它的形式。

[0458] 所述作为分离部件说明的单元可以是或者也可以不是物理上分开的，作为单元显示的部件可以是一个物理单元或多个物理单元，即可以位于一个地方，或者也可以分布到多个不同地方。可以根据实际的需要选择其中的部分或者全部单元来实现本实施例方案的目的。

[0459] 另外，在本申请各个实施例中的各功能单元可以集成在一个处理单元中，也可以是各个单元单独物理存在，也可以两个或两个以上单元集成在一个单元中。上述集成的单

元既可以采用硬件的形式实现，也可以采用软件功能单元的形式实现。

[0460] 所述集成的单元如果以软件功能单元的形式实现并作为独立的产品销售或使用时，可以存储在一个可读取存储介质中。基于这样的理解，本申请实施例的技术方案本质上或者说对现有技术做出贡献的部分或者该技术方案的全部或部分可以以软件产品的形式体现出来，该软件产品存储在一个存储介质中，包括若干指令用以使得一个设备(可以是单片机，芯片等)或处理器(processor)执行本申请各个实施例所述方法的全部或部分步骤。而前述的存储介质包括:U盘、移动硬盘、只读存储器(read only memory，ROM)、随机存取存储器(random access memory，RAM)、磁碟或者光盘等各种可以存储程序代码的介质。

[0461] 以上内容，仅为本申请的具体实施方式，但本申请的保护范围并不局限于此，任何在本申请揭露的技术范围内的变化或替换，都应涵盖在本申请的保护范围之内。因此，本申请的保护范围应以所述权利要求的保护范围为准。

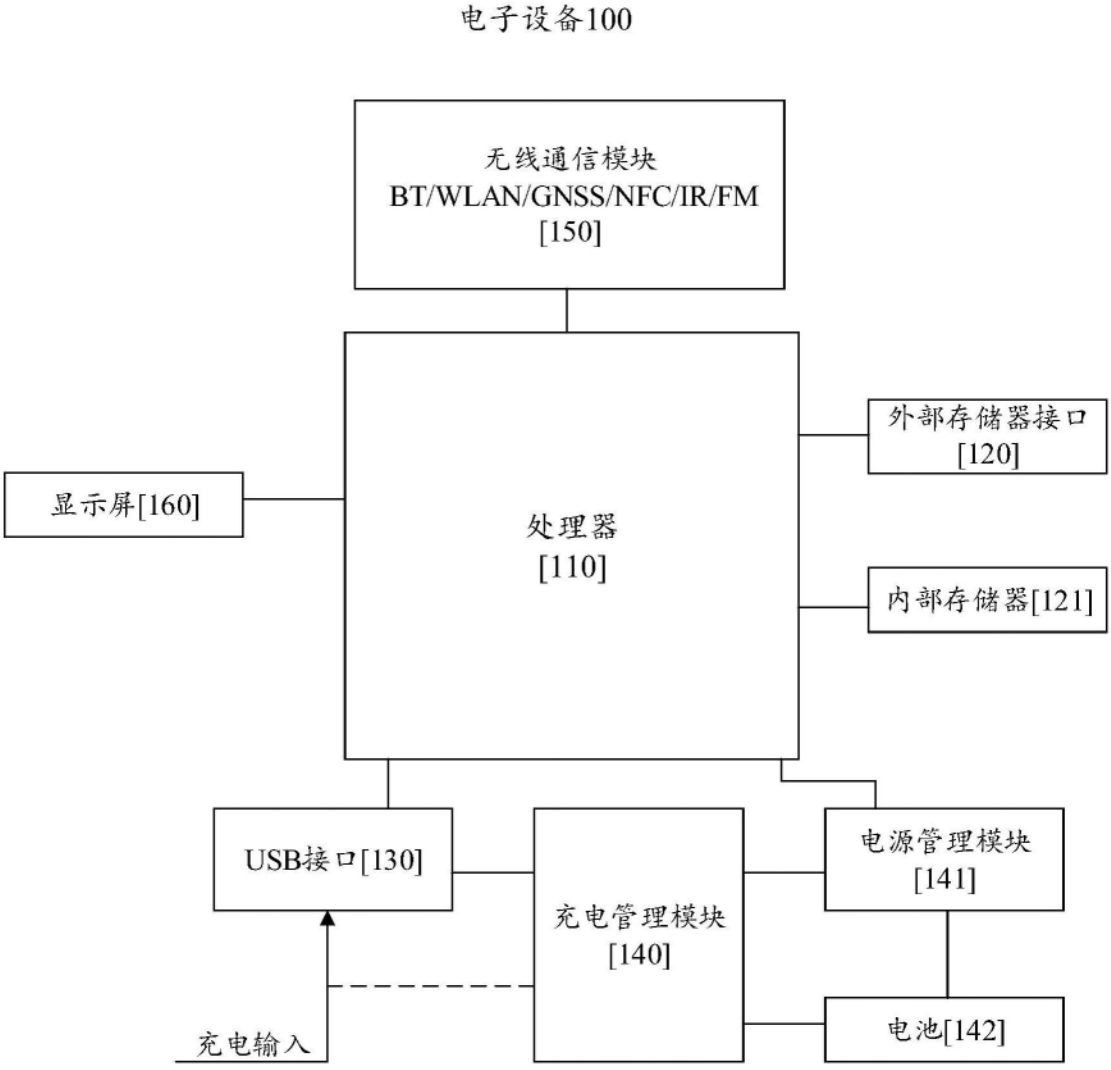

电子设备100
无线通信模块
BT/WLAN/GNSS/NFC/IR/FM
[150]
外部存储器接口
[120]
显示屏[160]
处理器
[110]
内部存储器[121]
USB接口[130]
充电管理模块
[140]
电源管理模块
[141]
电池[142]
充电输入

图1

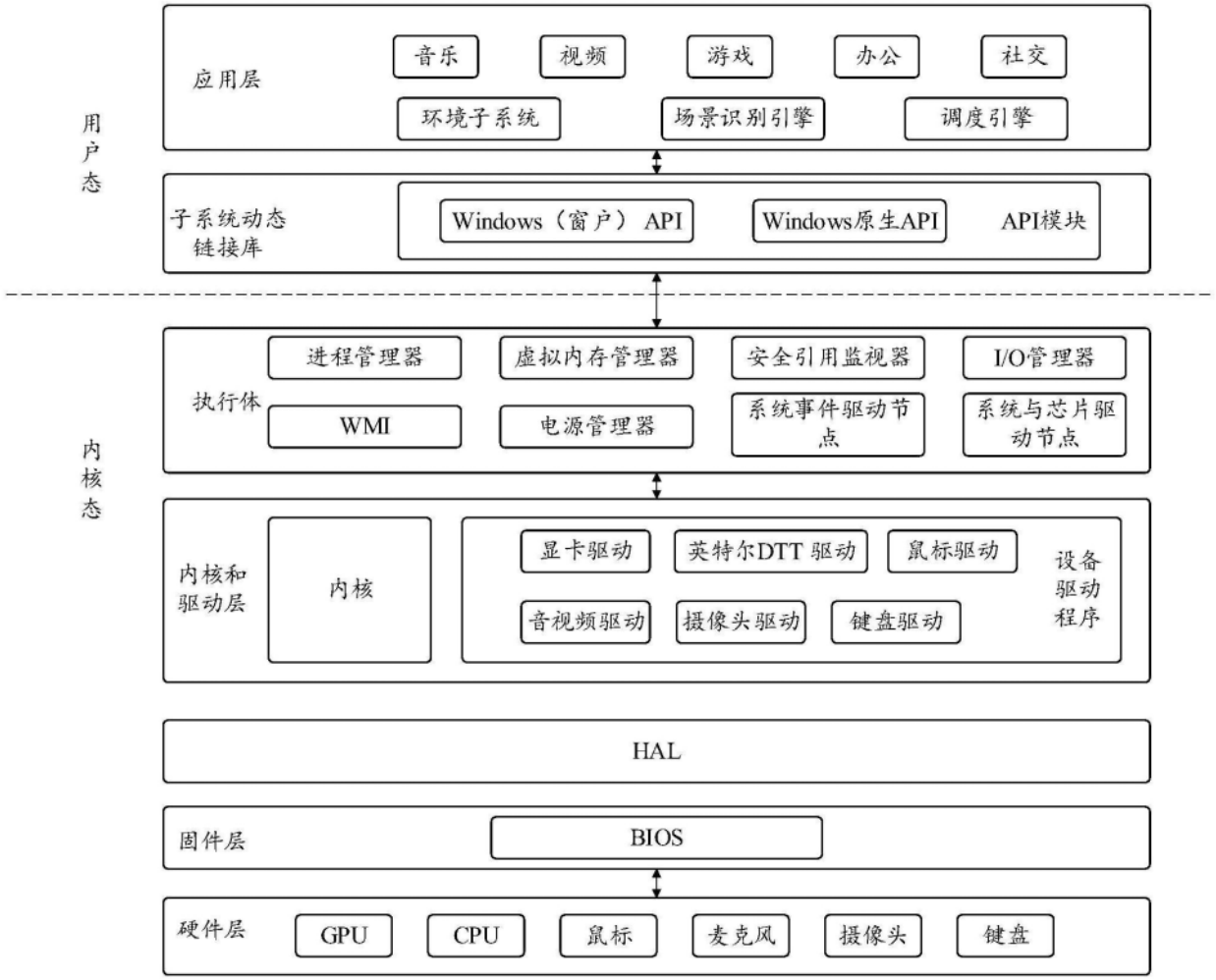

用户态
应用层
音乐
视频
游戏
办公
社交
环境子系统
场景识别引擎
调度引擎
子系统动态链接库
Windows（窗户）API
Windows原生API
API模块
内核态
执行体
进程管理器
虚拟内存管理器
安全引用监视器
I/O管理器
WMI
电源管理器
系统事件驱动节点
系统与芯片驱动节点
内核和驱动层
内核
显卡驱动
英特尔DTT 驱动
鼠标驱动
音视频驱动
摄像头驱动
键盘驱动
设备驱动程序
HAL
固件层
BIOS
硬件层
GPU
CPU
鼠标
麦克风
摄像头
键盘

图2

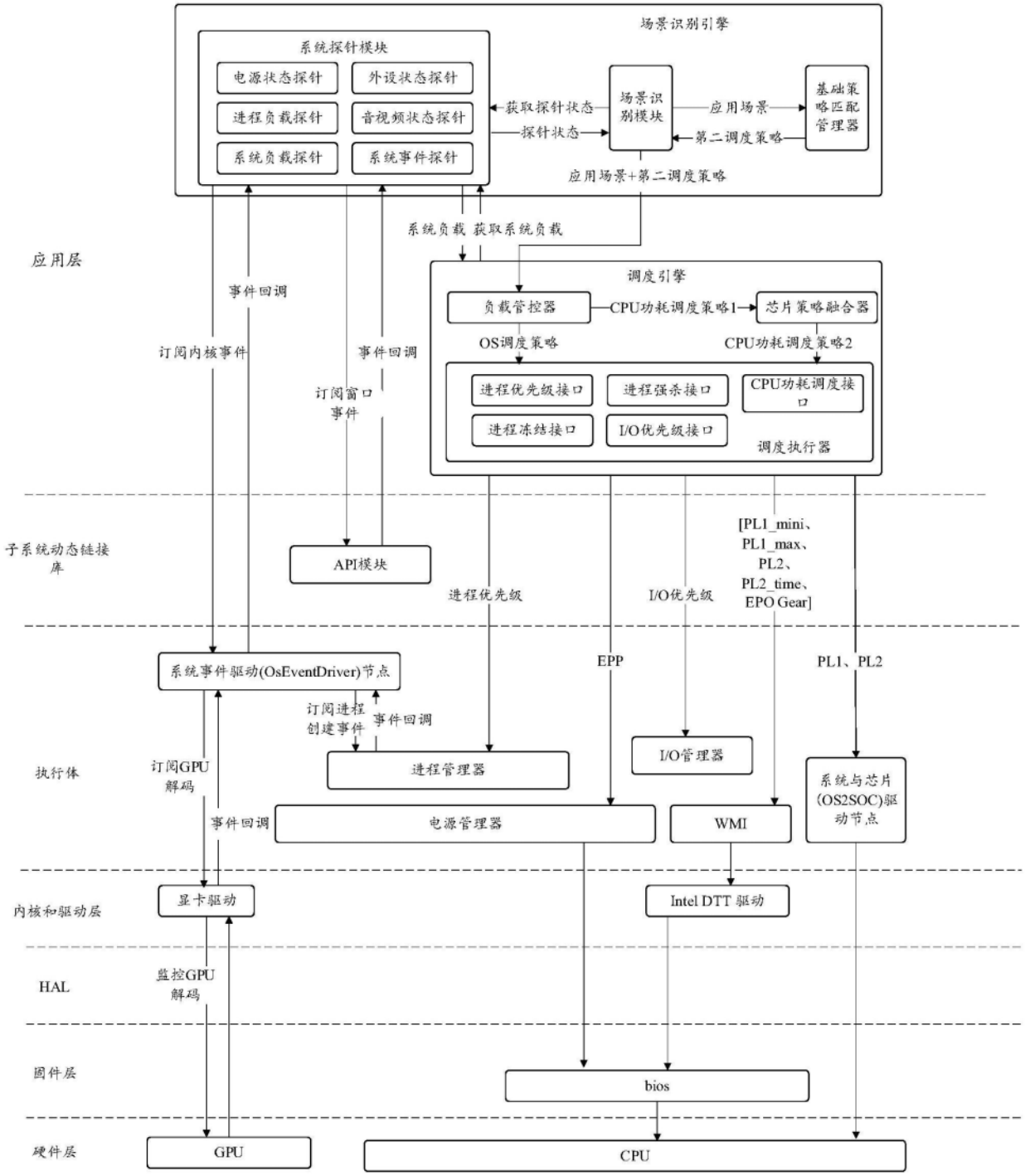

场景识别引擎
系统探针模块
电源状态探针
外设状态探针
进程负载探针
音视频状态探针
系统负载探针
系统事件探针
获取探针状态
探针状态
场景识别模块
应用场景
第二调度策略
基础策略匹配管理器
应用场景+第二调度策略
系统负载 获取系统负载
应用层
事件回调
调度引擎
负载管控器
CPU功耗调度策略1
芯片策略融合器
OS调度策略
CPU功耗调度策略2
订阅内核事件
事件回调
进程优先级接口
进程强杀接口
CPU功耗调度接口
订阅窗口事件
进程冻结接口
I/O优先级接口
调度执行器
子系统动态链接库
API模块
[PL1_mini、PL1_max、PL2、PL2_time、EPO Gear]
进程优先级
I/O优先级
系统事件驱动(OsEventDriver)节点
EPP
PL1、PL2
订阅进程创建事件
事件回调
执行体
订阅GPU解码
进程管理器
I/O管理器
系统与芯片(OS2SOC)驱动节点
事件回调
电源管理器
WMI
内核和驱动层
显卡驱动
Intel DTT 驱动
HAL
监控GPU解码
固件层
bios
硬件层
GPU
CPU

图3

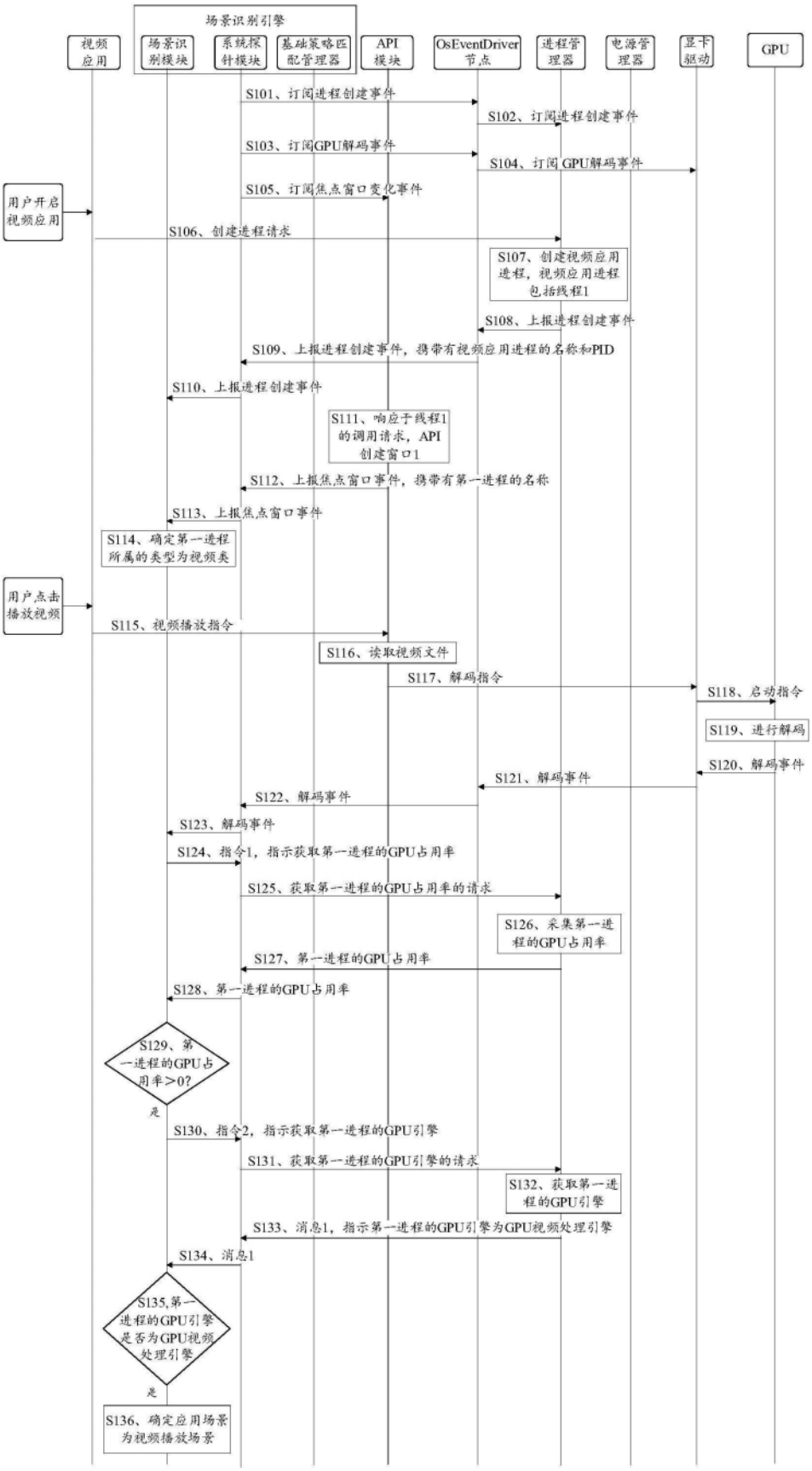

场景识别引擎
视频应用
场景识别模块
系统探针模块
基础策略匹配管理器
API模块
OsEventDriver节点
进程管理器
电源管理器
显卡驱动
GPU
S101、订阅进程创建事件
S102、订阅进程创建事件
S103、订阅GPU解码事件
S104、订阅GPU解码事件
S105、订阅焦点窗口变化事件
用户开启视频应用
S106、创建进程请求
S107、创建视频应用进程，视频应用进程包括线程1
S108、上报进程创建事件
S109、上报进程创建事件，携带有视频应用进程的名称和PID
S110、上报进程创建事件
S111、响应于线程1的调用请求，API创建窗口1
S112、上报焦点窗口事件，携带有第一进程的名称
S113、上报焦点窗口事件
S114、确定第一进程所属的类型为视频类
用户点击播放视频
S115、视频播放指令
S116、读取视频文件
S117、解码指令
S118、启动指令
S119、进行解码
S120、解码事件
S121、解码事件
S122、解码事件
S123、解码事件
S124、指令1，指示获取第一进程的GPU占用率
S125、获取第一进程的GPU占用率的请求
S126、采集第一进程的GPU占用率
S127、第一进程的GPU占用率
S128、第一进程的GPU占用率
S129、第一进程的GPU占用率>0?
是
S130、指令2，指示获取第一进程的GPU引擎
S131、获取第一进程的GPU引擎的请求
S132、获取第一进程的GPU引擎
S133、消息1，指示第一进程的GPU引擎为GPU视频处理引擎
S134、消息1
S135,第一进程的GPU引擎是否为GPU视频处理引擎
是
S136、确定应用场景为视频播放场景

图4

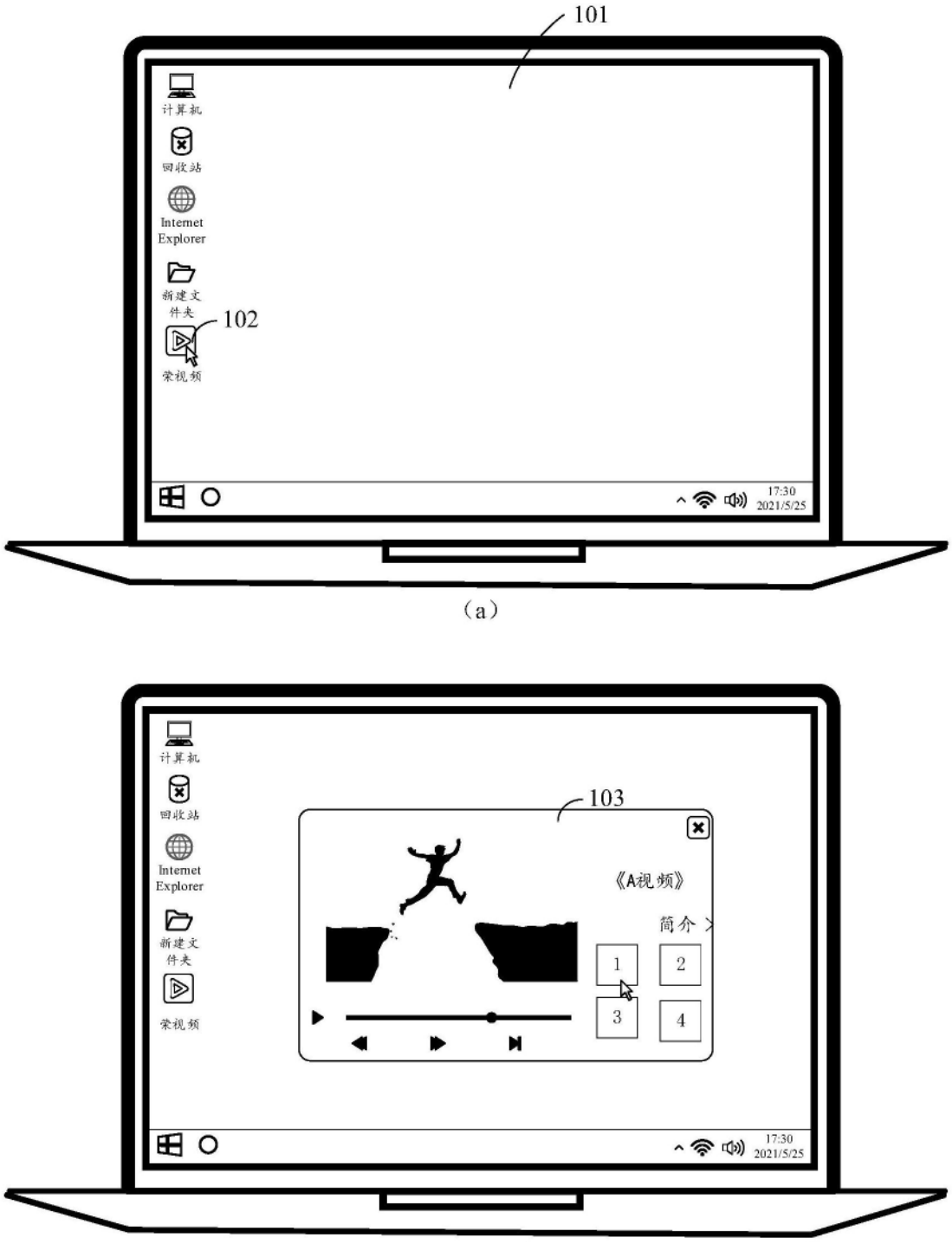

101
计算机
回收站
Internet Explorer
新建文件夹
102
荣视频
17:30
2021/5/25
(a)
计算机
回收站
Internet Explorer
新建文件夹
荣视频
103
《A视频》
简介 >
1
2
3
4
17:30
2021/5/25
(b)

图5

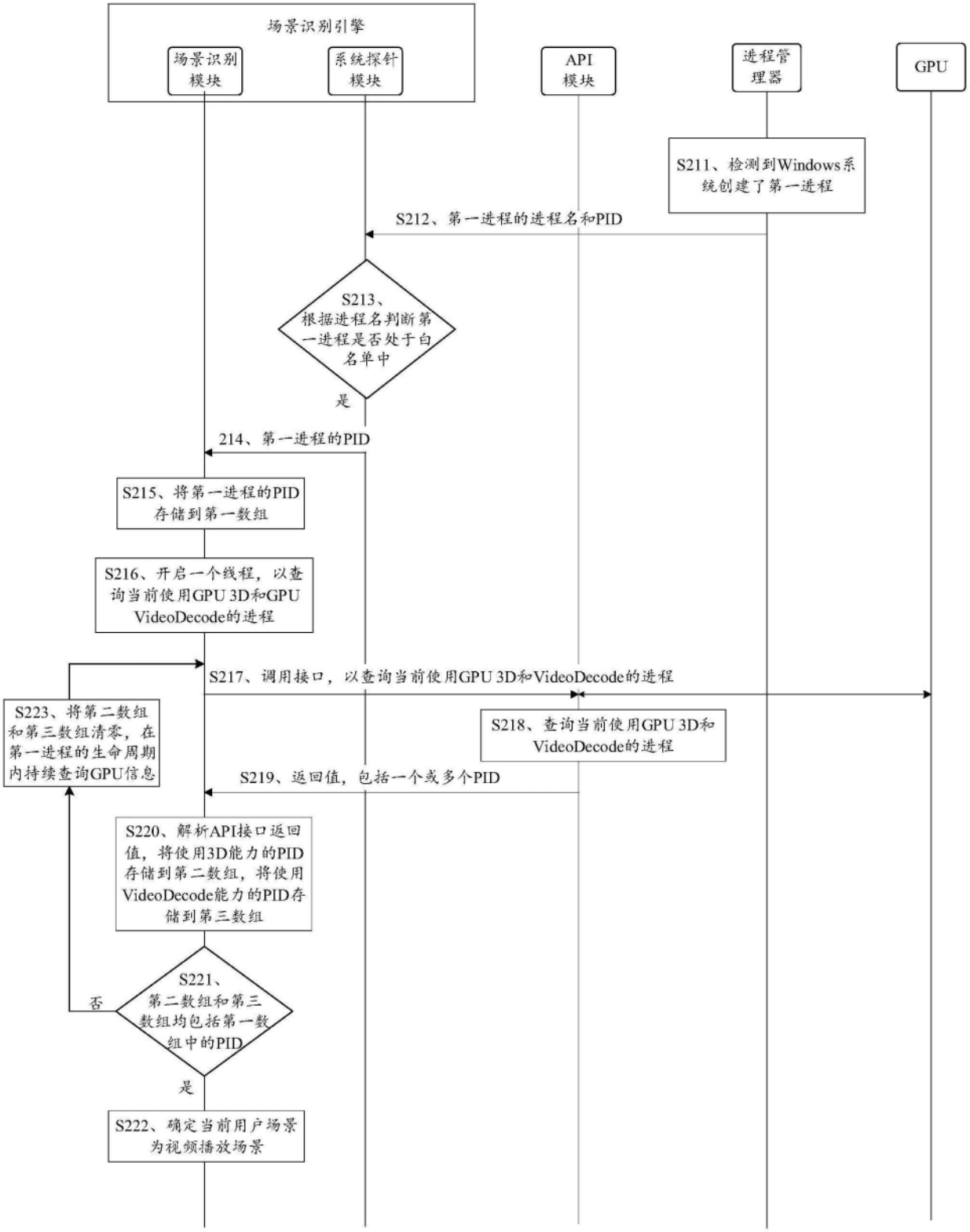

场景识别引擎
场景识别模块
系统探针模块
API模块
进程管理器
GPU
S211、检测到Windows系统创建了第一进程
S212、第一进程的进程名和PID
S213、根据进程名判断第一进程是否处于白名单中
是
214、第一进程的PID
S215、将第一进程的PID存储到第一数组
S216、开启一个线程，以查询当前使用GPU 3D和GPU VideoDecode的进程
S217、调用接口，以查询当前使用GPU 3D和VideoDecode的进程
S218、查询当前使用GPU 3D和VideoDecode的进程
S219、返回值，包括一个或多个PID
S220、解析API接口返回值，将使用3D能力的PID存储到第二数组，将使用VideoDecode能力的PID存储到第三数组
S221、第二数组和第三数组均包括第一数组中的PID
否
是
S222、确定当前用户场景为视频播放场景
S223、将第二数组和第三数组清零，在第一进程的生命周期内持续查询GPU信息

图6

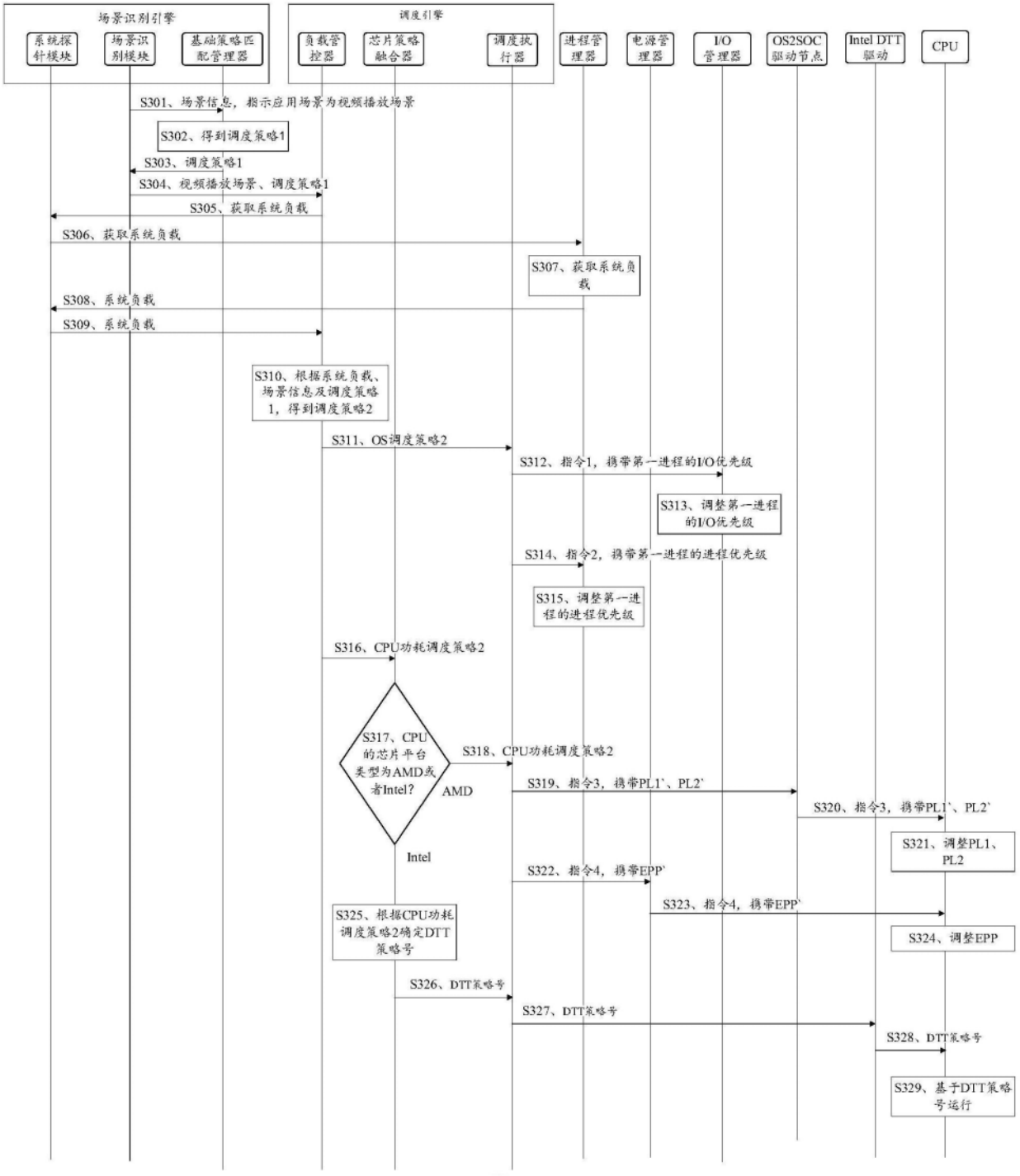

场景识别引擎
系统探针模块
场景识别模块
基础策略匹配管理器
调度引擎
负载管控器
芯片策略融合器
调度执行器
进程管理器
电源管理器
I/O管理器
OS2SOC驱动节点
Intel DTT驱动
CPU
S301、场景信息，指示应用场景为视频播放场景
S302、得到调度策略1
S303、调度策略1
S304、视频播放场景、调度策略1
S305、获取系统负载
S306、获取系统负载
S307、获取系统负载
S308、系统负载
S309、系统负载
S310、根据系统负载、场景信息及调度策略1，得到调度策略2
S311、OS调度策略2
S312、指令1，携带第一进程的I/O优先级
S313、调整第一进程的I/O优先级
S314、指令2，携带第一进程的进程优先级
S315、调整第一进程的进程优先级
S316、CPU功耗调度策略2
S317、CPU的芯片平台类型为AMD或者Intel?
AMD
Intel
S318、CPU功耗调度策略2
S319、指令3，携带PL1`、PL2`
S320、指令3，携带PL1`、PL2`
S321、调整PL1、PL2
S322、指令4，携带EPP`
S323、指令4，携带EPP`
S324、调整EPP
S325、根据CPU功耗调度策略2确定DTT策略号
S326、DTT策略号
S327、DTT策略号
S328、DTT策略号
S329、基于DTT策略号运行

图7

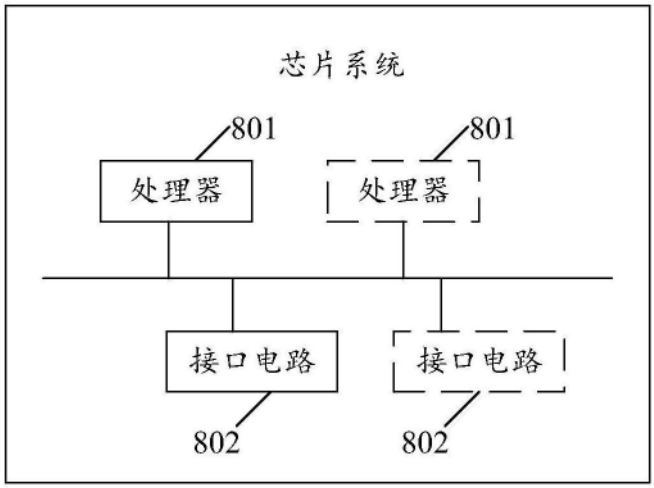


图8