# 多任务下 I/O 设备的动态功耗管理

戚隆宁，张　哲，黄少珉
（东南大学国家专用集成电路系统工程技术研究中心，南京 210096）

[摘要]　减少 I/O 设备功耗已越来越被嵌入式系统设计者所关注。传统动态功耗管理(DPM)策略在实际的多任务环境下无法得到预期的节能效果。提出基于堆栈的预测性超时(SBPT)策略。该策略通过分析任务的调用和堆栈信息来预测任务对 I/O 设备的访问模式，并采用多请求源(MSR)模型进行多任务的联合预测。然后根据预测结果分组统计，采用超时技术决策。基于实际负载的仿真实验表明 SBPT 策略能够适应多任务的应用环境，更稳定更有效地降低了功耗。

[关键词]　动态功耗管理；预测性超时；堆栈；多请求源
[中图分类号]　TP316　[文献标识码]　A　[文章编号]　1009－1742(2008)02－0060－06

## 1　前言

动态功耗管理（DPM，dynamic power management)是一种系统级低功耗技术，通过对系统或功耗可控部件(PMC，power manageable component)的空闲状态进行实时的分析，动态地调整系统或设备的状态，从而降低系统或设备在欠载时的功耗[1]。在这一领域已经提出了多种策略，如超时策略[2]、预测策略[3]、随机策略[4]等。超时策略是在设备空闲一段固定时间后关闭设备，或者根据设备空闲时间的历史记录调整超时时限超时策略，前者称为固定超时策略[5]而后者称为自适应超时策略[6]，超时策略实现机制最简单，应用也最为广泛。预测策略的提出是为了弥补超时策略在超时时间浪费掉的能量，所以一旦预测到设备进入空闲状态的时间能够弥补状态转换所带来的额外消耗时就将设备关闭。随机策略则通过概率模型(如 Markov 模型)描述系统的随机行为，从而能够更准确地选择设备将要进入的低功耗状态。

硬盘和无线网卡等 I/O 设备都是典型的功耗可控部件，I/O 设备的 DPM 策略受到很多不确定因素的影响，负载(即 I/O 操作请求)的分布正是其中重要因素之一，能否准确地分析设备的使用规律是策略成败的关键。传统策略都单纯地从设备的角度考虑，而忽略了 I/O 操作请求所反映的应用特征。在实际的多任务系统中，不同的任务使用设备的方式不同，系统运行任务的规律也不同，所以实际负载是非平稳的(non-stationary)。这使得基于负载平稳性假设的传统 DPM 策略在实际的多任务环境下无法得到预期的节能效果。

笔者提出了基于堆栈的预测性超时(SBPT，stack based predictive timeout)策略。该策略分为预测和超时决策两部分。前者通过应用任务的调用和堆栈信息来分析任务对 I/O 设备的访问模式，并采用多请求源(MSR，multiple service requesters)模型进行多任务下的联合预测。后者根据预测结果进行分组，在各个分组上采用基于概率统计的超时技术进行决策，控制设备进入低功耗模式。该策略充分利用应用程序以及操作系统提供的信息，有效预测了多任务下的 I/O 设备访问模式，并把预测和超时两种 DPM 技术相结合，使得策略能够自动适应不同的应用环境。

## 2　相关工作

嵌入式系统中许多 I/O 设备，如存储设备、网络设备等，其负载具有明显的簇集和波动等非平稳特性[7,8]。这使得许多基于负载平稳性假设的传统 DPM 策略的效果受到了削弱，为此不断有研究者提

[收稿日期]　2007－03－05；修回日期　2007－04－25
[作者简介]　戚隆宁(1979－)，男，浙江临安市人，东南大学博士研究生，主要研究领域为嵌入式系统的仿真和低功耗技术

出更好的方法和策略来自动适应这类负载变化。

E. Y. Chung, S. Irany 和吴琦等人都采用了滑动窗口这一方法，认为从全局看负载的分布是非平稳的，但在局部的分布却可以视作是平稳的，所以可以根据有限时间窗口内的负载样本，用概率估计的方法确定局部时间段内的负载分布。他们的方法不同：E. Y. Chung 提出的是基于离散时间 Markov 模型的随机策略，采用最大似然估计（MLE, maximum likelihood estimation）来确定转移概率矩阵[7]；S. Irany 提出的是基于概率统计的超时策略，用窗口内的样本直方图近似计算分布从而确定最佳超时门限[9]；而吴琦则是针对 Pareto 分布，采用截尾均值估计（TME, trimmed mean estimation）的方法来确定分布的参数从而计算出最佳超时门限[8]。

滑动窗口的方法使得策略能够自动适应概率分布的变化，具有一定的局限性。假设负载在 $T_1$ 和 $T_2$ 相邻的两段时间内（$T_1$ 和 $T_2$ 内的样本数远大于窗口大小）分别服从分布 $F_1$ 和 $F_2$。在从 $T_1$ 过渡 $T_2$ 的时间内，滑动窗口在学习分布 $F_2$ 的过程中仍保留了分布 $F_1$ 的影响直到窗口完全进入 $T_2$，因此对过渡区内的分布估计有较大误差。这种现象称为自适应延迟（adaptation delay），窗口越大延迟的效果越明显[7]。要降低这种延迟，直接的方法是减小窗口大小，但是这样会导致样本容量减小，增大分布的估计误差。

设备负载的这种非平稳特性是不难理解的。任何设备总是被任务所操纵的，不同任务使用设备的方式有区别，任务的执行顺序会变化，即使是同一个任务，每次执行也可能会存在差异。在多任务的环境下，因任务的多样性和任务执行的差异，导致了设备负载的随机和非平稳的特性。传统 DPM 策略局限于观察设备负载本身，而忽略了负载背后的不同任务的使用特征。

Lu Yung-Hsiang 提出基于任务的功耗管理（TBPM, task based power management）策略[10]，将每个任务视为独立的设备请求源（requester），使用设备任务利用率矩阵（device-requester utilization matrix）来描述各个任务对设备的使用情况。每个任务采用指数平均（exponential average）的预测方法对设备的请求时间间隔（TBR, time between request）进行独立预测，然后以设备－任务利用率矩阵中对应的系数（即任务的 CPU 执行时间所占比例）作为权重，加权求和各任务的预测值，从而得出整个系统的预测结果。C. Gniady 等人提出基于程序计数器的访问预测（PCAP, program counter based access predictor）策略[11]，以设备 I/O 的 API 函数被应用任务调用时的程序计数器作为标志，区分出不同的设备请求。并认为在一系列特定的 I/O 操作之后，设备会进入空闲状态，所以可以根据这些 I/O 操作的特征（即程序计数器之和）进行预测。

TBPM 策略和 PCAP 策略对负载的分析和划分仍然是比较粗糙的，许多任务的 CPU 执行时间与对设备的使用并没有很强的相关性，而设备的 I/O 操作路径的特征并不十分明显，还存在子路径（sub-path）重复的问题，这些都影响了策略的预测效果。笔者通过对设备负载的进一步的分析，提出了多任务下的基于堆栈的预测性超时（SBPT, stack based predictive timeout）策略。

## 3 基于堆栈的多任务预测技术

在嵌入式系统中，进程是任务（程序）的一次执行过程，线程是进程的最小执行单元。图 1 显示了 2 个进程对设备的使用，每个进程各自包含了 2 个线程。在时间轴上的方块表示进程对设备的 I/O 操作请求（request），不同颜色的请求表示属于不同的线程。可以看到，从设备的角度而言，到达的请求毫无规律，而从不同的进程或线程看过去，请求的规律是十分明显的。所以，将设备的负载分解到各个任务的线程上。

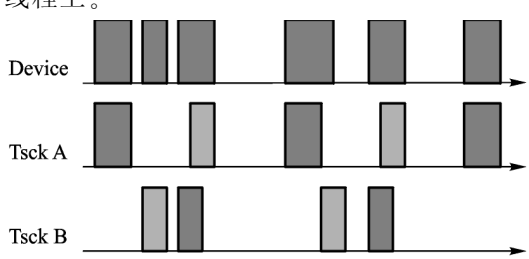


图 1 多任务下的设备 I/O 操作

**Fig.1 Device I/O operations under multi-tasks**

由于任务可以被多次执行，这些属于同一任务的进程拥有相同的执行代码，具有相似的行为特征，将其归入同一个进程组 $G_P$。同样，因为执行入口地址相同的线程其行为也具有相似性，也将其归入同一个线程组 $G_T$。这样在设备的 I/O 操作时获得执行操作的线程的 ID 及其所属进程的 ID，根据线程的 ID 和进程的 ID，该 I/O 操作被映射到组分类（$G_P$, $G_T$）上。因为每个任务的执行代码都是确定的，假

设对某个设备有需求的任务的总数为 $N$，第 $i$ 个任务中包含对该设备操作的线程的入口地址总数为 $n_i$，则该设备的组分类的总数为 $n_G = \sum_{i=1}^{N} n_i$。考虑系统提供的线程创建和退出信息，那么一段时间内的设备负载就可以按照线程的影响被明确的划分为若干区间，每段区间用 $A_i$ 表示。假设区间 $A_i$ 的负载仅受线程集合 $TS_i$ 的影响，那么有 $TS_i \neq TS_{i+1}$，$TS_i$ 中所有线程在区间 $A_i$ 内的负载的叠加就是设备在该区间内的负载。以图 2 所示为例，3 个线程 $t_1$，$t_2$，$t_3$ 分割出 5 段区间 $A_1$，…，$A_5$，设备的负载表现为 5 种不同的特征。假设每个线程在设备上的负载是平稳的，那么设备在各个区间上的负载也是平稳的。因为明确了解了负载变化的区间边界，所以 DPM 策略就可以避免滑动窗口等方法所固有的自适应延迟的缺陷。

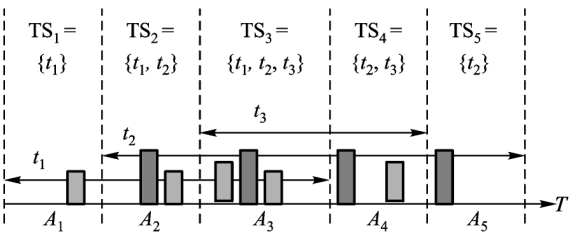


图 **2**　根据线程的影响划分负载的时间区间

**Fig.2　Districts of the workload affected by threads**

参考 C. Gniady 所提出的 PCAP 方法，线程的执行还能根据设备的 API 函数被调用地址进一步划分。但是，假设某个应用程序的函数调用过程如图 3 所示，实线箭头表示函数调用关系，虚线箭头表示函数的调用路径(call path)。设备的 API 函数仅被函数 func1 调用，此时用 PCAP 的方法无法区分图 3 中所示的 4 条不同的调用路径。栈(stack)反映了线程的调用过程，图 3 中 4 条不同的调用路径对应着 4 个不同的栈内容，通过分析栈的内容就能够容易的区分出它们。文献[12]在分析应用程序对 Buffer 的使用情况时采用了栈展开(stack unwind)的方法，但嵌入式系统广泛使用的 ARM 和 MIPS 架构在栈帧(stack frame)中并未保存帧指针(frame pointer)，因此栈展开需要通过分析代码来确定栈帧的大小从而遍历整个栈。这样势必会影响设备 I/O 的性能，所以采用栈深(stack depth)和返回地址(return adress)来联合区分调用路径。其中，栈深是当前栈指针(stack pointer，即当前 sp 寄存器的内容)与初始栈指针的差值，返回地址是当前 lr 或 ra 寄存器的内容，这些可以很容易地从线程保存的上下文(context)中获取。

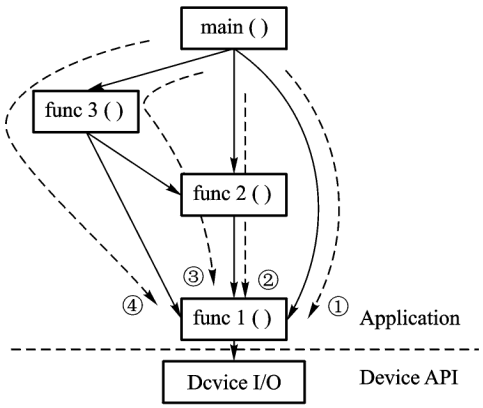


图 **3**　应用程序对设备的不同调用路径

**Fig.3　Different call paths of the application for using a device**

这样设备的 I/O 操作被细分到堆栈，I/O 操作的分类扩展为($G_P$，$G_T$，$S_R$，$S_D$)，其中 $S_R$，$S_D$ 分别表示返回地址和栈深。这样从单个线程来看，可以根据函数调用的返回地址和栈深来预测 2 次 I/O 操作的间隔时间。首先建立预测查找表，其中保存了长间隔时间的 I/O 操作所对应的 $S_R$ 和 $S_D$ 以及间隔时间的分布，每次操作请求到来时根据 $S_R$ 和 $S_D$ 匹配查找表，若命中则根据对应项的间隔时间分布的期望来预测是否进入低功耗模式，并根据实际的间隔时间更新分布。如果不命中则预测为不进入低功耗模式，一旦实际间隔时间超过了损益平衡时间(break-even time)[13]，则将这次操作对应的 $S_R$ 和 $S_D$ 加入到查找表中。如前所述，属于相同进程和线程组分类($G_P$，$G_T$)的操作序列具有相似性，因此可以共享同一张查找表。

因为各进程对 I/O 设备的访问相互独立，可以视为多个独立的请求源(service requester)，根据文献[14]所提出的 MSR 模型，假设进程 $P_j$ 的 I/O 操作时间间隔服从分布 $F_j$，当 I/O 操作请求到来时，各个进程的距离各自上次操作请求的时间间隔为 $t_j$，则从设备角度来看，操作请求间隔时间服从分布

$$F_{\min}(t) = 1 - \prod_{i=0}^{n-1} \frac{1 - F_j(t + t_j)}{1 - F_j(t_j)} \qquad (1)$$

查找表中保存的分布信息即为所需的各进程 I/O 操作间隔时间的分布，这样每次操作请求完成之后，就可以根据式(1)计算出的分布再次预测是否进入低

功耗模式。

预测结果往往会有错误，一些很长的空闲时间被预测得过短(underprediction)，而很短的空闲时间被预测得过长(overprediction)。前者导致设备长时间处于高功耗状态，浪费了能量，而后者则造成请求处理被延后，降低了性能。为此需要在决策过程中引入超时机制来减少误预测带来的危害，所以根据预测的功耗模式不同进行分组决策，而不是武断地直接按照预测结果操作。功耗模式相同的预测进入相同的决策流程，每个决策流程独立统计进入流程的操作的实际间隔时间，假设模式 $i$ 对应的决策分组下的间隔时间分布为 $F^{(i)}(t)$。根据文献[9]若设备处模式 $i$ 时的平均功耗为 $a_i$，从模式 $i$ 进入正常工作状态的转换能耗为 $b_i$，设备操作的间隔时间服从概率密度函数为 $F_D(t)$的分布，设置从模式 $i$ 转换到状态 $i+1$ 的超时门限为 $k$，则空闲时的能耗期望为

$$E(k)=\int_0^k(a_it+b_i)\mathrm{d}F_D(t)+\int_k^{\infty}(a_it+a_{i+1}(t-k)+b_{i+1})\mathrm{d}F_D(t) \quad (2)$$

因此对于确定的间隔时间分布 $F^{(i)}(t)$，可以计算出最佳的超时门限 $k^{(i)}$，使得总的空闲时能耗最小。在当前的决策流程中，依次计算各个模式下的最佳超时门限 $k^{(i)}$ 和最低能耗，选择能耗最低的模式和门限作为决策的最佳门限 $k_{opt}$。当空闲时间超过决策的最佳门限 $k_{opt}$ 之后才会让设备进入对应的低功耗模式。

## 4 策略评估与比较

为了评估策略的节能效果和对性能的影响，在 Intel PXA255 处理器和 Hitachi Travelstar C4K60 微硬盘构建的嵌入式系统上运行 5 个应用程序(文本编辑器、图片浏览器、多媒体播放器和 2 个日志程序)，共计进程 430 个，线程3 400个，在 60 h 左右的时间内对硬盘进行了约 $3\times10^6$ 次读写操作。这些读写操作记录即为仿真实验的测试集，表 1 是其中 5 组典型的测试用例。

表 1 中 Active Idle，Unload 和 Standby 是 C4K60 硬盘空闲时的三种功耗模式，各模式下平均功耗依次为 480 mW，340 mW，80 mW，唤醒延迟分别为 0 s，0.2 s，1.15 s，根据各模式下的损益平衡时间(分别为 0 s，2.7 s，4.3 s)可以确定空闲时间 idle 对应的最优功耗模式[15]。从表 1 可以看出，硬盘 75 % 以上的操作间隔是集中在 2.7 s 内的短 idle，而在测试集 4 和测试集 5 中表现出明显的重尾(heavy tail)特征[8]，操作间隔超过 4 s 的长 idle 都大于 10 %。

表 1 测试用例的硬盘空闲时间分布

**Table 1 Distribution of the hard-disk idle time under different test cases**

| 测试集 | 空闲时间的分布(按照最优功耗模式划分) | | |
|---|---|---|---|
| | Active Idle /% | Unload /% | Standby /% |
| 1 | 99.41 | 0.10 | 0.48 |
| 2 | 99.39 | 0.06 | 0.55 |
| 3 | 81.78 | 17.88 | 0.34 |
| 4 | 83.32 | 2.22 | 14.45 |
| 5 | 79.21 | 0 | 20.79 |

参与评估比较的策略有指数平均预测策略(EA)[16]、PCAP 策略[11]、基于概率统计的超时策略(OPBA)[9] 和 SBPT 策略。采用差分竞争率(DCR，differential competitive ratio)作为功耗评估指标，差分竞争率为当前策略的能耗 $E_A$ 和最优离线策略下的能耗 $E_{opt}$ 与无策略时的能耗 $E_{no}$ 的差值比，$DCR=(E_{no}-E_A)/(E_{no}-E_{opt})$，$(E_{opt}\neq E_{no})$。DCR 反映了当前策略相对于最优策略降低功耗的效率，值越高表示节能效果越接近最优策略，上限值为 1。性能指标采用延迟率(DR，delay ratio)，即当前策略下的总时间比无策略时的总时间多出的比例，$DR=(T_A-T_{no})/T_{no}$。DR 反映了策略对硬盘响应操作请求的影响，值越低表示影响越小，下限值为 0。策略的预测效率则通过分级命中率(CHR，classified hit ratio)和分级失误率(CMR，classified miss ratio)来评估。这两种指标都是根据功耗模式来划分级别，功耗越低级别越高。CHR 为当前策略模式 $i$ 预测正确的次数 $hit_A(i)$与最优策略下处于模式 $i$ 的次数 $n_{opt}(i)$ 的比值，$CHR_A(i)=hit_A(i)/n_{opt}(i)$，$(n_{opt}(i)\neq0)$。而 CMR 为当前策略模式 $i$ 误预测的次数与预测总次数 $n_A(i)$之比，$CMR_A(i)=(n_A(i)-hit_A(i))/n_A(i)$，$(n_A(i)\neq0)$。高命中率和低失误率的策略能够更有效的降低功耗。

表 2 比较了 5 组测试集下不同策略的 CHR 和 CMR。

EA 策略仅在第 1 组测试中命中率表现较好，而失误率一直居高不下。PCAP 策略的命中率起伏不定，低功耗模式(级别 3)下的失误率除第 5 组测试外均高于 70 %。OPBA 策略和 SBPT 策略的预测相对稳定，而 SBPT 策略在低功耗模式下具有更高的

表 2　分级命中率和失误率的比较

**Table 2　Classified hit ratio and classified miss ratio for policies under tests**

| 测试集 | 策略 | 分级命中率(CHR)/% | | | 分级失误率(CMR)/% | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 1 | 2 | 3 |
| 1 | EA | 93.37 | 53.85 | 53.33 | 3.23 | 76.67 | 53.62 |
| | PCAP | 99.73 | 0 | 25.00 | 0.45 | 0 | 70.00 |
| | OPBA | 99.76 | 0 | 70.00 | 0.15 | 100 | 47.50 |
| | SBPT | 99.77 | 0 | 71.67 | 0.15 | 100 | 46.25 |
| 2 | EA | 91.39 | 20.00 | 36.17 | 1.97 | 96.15 | 86.82 |
| | PCAP | 99.91 | 0 | 0 | 0.60 | 0 | 100 |
| | OPBA | 99.75 | 0 | 85.11 | 0.08 | 100 | 38.46 |
| | SBPT | 99.75 | 0 | 85.11 | 0.09 | 100 | 37.50 |
| 3 | EA | 69.39 | 0 | 16.55 | 33.69 | 100 | 67.26 |
| | PCAP | 41.74 | 37.27 | 88.66 | 5.15 | 14.29 | 41.47 |
| | OPBA | 91.49 | 0.62 | 54.65 | 8.23 | 99.49 | 41.65 |
| | SBPT | 94.82 | 33.54 | 92.74 | 1.39 | 22.86 | 36.49 |
| 4 | EA | 99.51 | 0.41 | 0 | 31.29 | 72.73 | 100 |
| | PCAP | 99.97 | 0 | 0 | 18.21 | 0 | 100 |
| | OPBA | 99.94 | 0 | 7.14 | 18.15 | 100 | 66.67 |
| | SBPT | 92.48 | 88.90 | 7.14 | 1.95 | 30.18 | 75.00 |
| 5 | EA | 42.25 | 100 | 9.27 | 28.57 | 100 | 91.45 |
| | PCAP | 84.66 | 100 | 90.73 | 2.68 | 0 | 39.35 |
| | OPBA | 84.02 | 100 | 83.47 | 0 | 0 | 48.12 |
| | SBPT | 95.56 | 100 | 95.16 | 1.10 | 0 | 15.71 |

命中率和更低的失误率。命中率和失误率对功耗的影响可以从图 4 对各个策略的 DCR 的比较可以看出。EA 策略除了在第 1 组测试中达到 0.7 外，其余 4 组测试均未超过 0.3。而 PCAP 策略波动很大，好的时候可以达到 0.7，差的时候甚至出现负增长(也就是说还不如没有策略)。OPBA 策略和 SBPT 策略则相对稳定，5 组测试均超过 0.6。在第 1 组和第 2 组测试中，两者不相上下，但其后 3 组重尾特征明显的测试中，SBPT 策略均达到 0.9 以上，要超过 OPBA 策略25 %～30 %。所有 DPM 策略都是对功耗和性能的折衷，降低功耗的同时也就意味着一定的性能损失，图 5 DR 的比较很好的说明了这一点，基本上策略的 DCR 越高，DR 也就越高。例外的是最后两组中 SBPT 策略的 DR 反而降低了，这要归功于 SBPT 策略在这两组测试中的低失误率。从整体上看，后 3 组测试的 DR 要高于前 2 组。这是因为后 3 组测试的重尾特征，使得低功耗模式的预测次数增多，而低功耗模式下的唤醒延迟很长，所以造成了高延迟率。

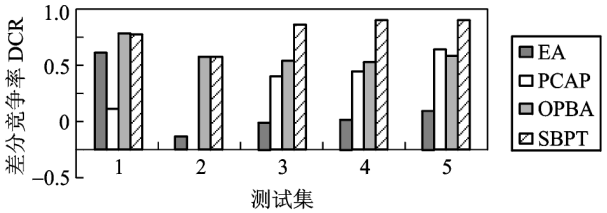


图 4　测试集下各策略的差分竞争率比较

**Fig.4　Differential competitive ratios for policies under tests**

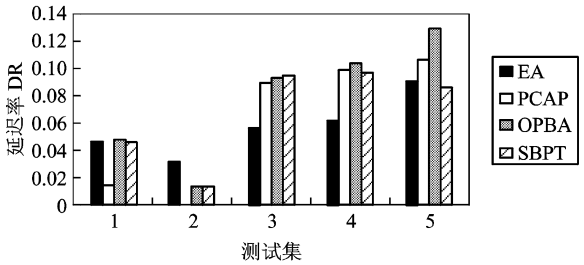


图 5　测试集下各策略的延迟率比较

**Fig.5　Delay ratios for policies under tests**

## 5　结语

针对嵌入式系统的多任务环境，提出了基于堆栈的 I/O 设备的动态功耗管理策略 SBPT，通过任务的调用和堆栈信息分析任务对 I/O 设备的访问模式，采用 MSR 模型进行多进程多线程下的联合预测，再根据预测结果进行分组，在各个分组上采用基于概率统计的超时技术进行决策。该策略充分利用应用程序以及操作系统提供的信息，有效预测了多任务下的 I/O 设备访问模式，并把预测和超时两种 DPM 技术相结合，使得策略能够自动适应不同的应用环境。实验表明，在多任务环境下，该策略相比于 EA 策略、PCAP 策略以及 OPBA 策略，在有限的性能损失下(延迟率小于 0.1)，具有更稳定更优越的节能效果。

参考文献


[1] Benini L, Bogliolo A, Micheli G D. A survey of design techniques for system-level dynamic power management [J]. IEEE Transactions on Very Large-scale Integration Systems, 2000, 8(3): 299～316

[2] Helmbold D P, Long D D E, Sconyers T L, et al. Adaptive disk spin-down for mobile computers [J]. Mobile Networks and Applications, 2000, 5(4): 285～297

[3] Chung E Y, Benini L, Micheli G D. Dynamic power management using adaptive learning tree [A]. International Conference on

Computer Aided Design [C]. San Jose, CA, USA: ACM Press, November 1999. 274～279

[4] Qiu Qinru, Pedram M. Dynamic power management based on continuous-time Markov decision processes [A]. Design Automation Conference [C]. New Orleans, Los Angeles, USA: ACM Press, June 1999. 555～561

[5] Golding R, Bosch P, Staelin C, et al. Idleness is not sloth [A]. In: Winter USENIX Technical Conference [C]. New Orleans, Los Angeles, USA, January 1995. 201～212

[6] Helmbold D P, Long D D E, Sherrod B. A dynamic disk spin-down technique for mobile computing [A]. In: International Conference on Mobile Computing and Networking [C]. Rye, New York, USA: ACM Press, November 1996. 130～142

[7] Chung E Y, Benini L, Bogliolo A, et al. Dynamic power management for nonstationary service requests [J]. IEEE Transactions on Computers, 2002, 51(11): 1345～1361

[8] 吴 琦,熊 光泽. 非平稳自相似业务下自适应动态功耗管理[J],软件学报,2005,16(8),1499～1505

[9] Irani S, Shukla S, Gupta R. Online strategies for dynamic power management in systems with multiple power-saving states [J]. ACM Transactions on Embedded Computing Systems, 2003, 2(3): 325～346

[10] Lu Yung-Hsiang, Benini L, Micheli G D. Operating-system directed power reduction [A]. Proceedings of the International Symposium on Low-power Electronics and Design [C]. July 2000. 37～42

[11] Gniady C, Hu Y C, Lu Yung-Hsiang. Program counter based techniques for dynamic power management [A]. Proceedings of the 10th International Symposium on High-performance Computer Architecture [C]. February 2004. 24～35

[12] Gniady C, Butt A R, Hu Y C. Program-counter-based pattern classification in buffer caching [A]. Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation [C]. December 2004. 395～408

[13] Irani S, Shukla S, Gupta R. Competitive analysis of dynamic power management strategies for systems with multiple power saving states [A]. Design, Automation and Test in Europe Conference [C]. Paris, France: IEEE Computer Society Press, March 2002. 117～123

[14] 戚隆宁,张 哲,胡 晨,等,基于MSR模型的动态功耗管理策略[J],电路与系统学报,2006, 16(5): 88～92

[15] Hitachi. Hard Disk Drive Specifications [EB/OL]. Travelstar C4K40 — 40/20. www. hitachigst. com/tech/techlib. nsf/techdocs/B452324297C3078886256D9600695E87/$ file/10fa_ e7. pdf, 2003 —12—11

[16] Hwang Chi-Hong, Wu A C-H. A predictive system shutdown method for energy saving of event-driven computation [J]. ACM Transactions on Design Automation of Electronic Systems, 2000, 5(2): 226～241

# Dynamic Power Management for I/O Devices Under Multi-task Environment

Qi Longning, Zhang Zhe, Huang Shaomin

(*Nation ASIC System Engineering Research Center, Southeast University, Nanjing 210096, China*)

[**Abstract**] More embedded system designers pay attention to how to reduce the power consumption of I/O devices. Traditional dynamic power management (DPM) policies only focus on the device requests, and neglect the application features behind the workload. Because of the assumption about the stationary workload, traditional DPM policies can not reach their expected goal under the multi-task environment. The paper presents a stack-based predictive timeout strategy (SBPT). It can predict the access pattern of the device I/O operations by analyzing the calling and stack information of tasks and combine predictions of multiple tasks to form the global prediction according to the multiple-service-requester model. At last, classify the I/O request by the global prediction and then make the decision with the timeout technique based on the distribution of the grouped requests. An evaluation study of SBPT using the trace-driven simulation is performed. The results show that SBPT can adapt the non-stationary multi-task environment and reduces power consumption more efficiently than other policies.

[**Key words**] DPM; predictive timeout; stack; multiple service requesters