基于同构多核处理器的任务调度①

许雍祯, 陈香兰, 李 曦, 周学海

(中国科学技术大学 计算机科学与技术学院, 合肥 230001)

摘 要: 随着现代应用对计算机性能要求的提高, 计算机主频不断提升. 由于功耗和半导体工艺的限制, 仅靠提高单核主频难以继续维持“摩尔定律”, 同构多核处理器(Homogeneous Multi-core)应运而生. 在同构多核处理器的支持下, 一个芯片汇集多个地位对等、结构相同的通用处理器核, 以最小的代价满足了提高系统性能、负载均衡、处理器容错的需要. 并行体系结构需要结合与之适应的软件实现性能效益的倍增. 本文从操作系统层面, 针对处理器结构的变化, 研究并实现多核任务调度. 系统采用混合调度策略, 簇间独立调度, 簇内统一调度. 从调度模式、调度算法、分配算法、调度时机等方面详细分析了多核调度的原理和实现机制. 最后通过模拟实验证明功能正确性及算法可调度性.

关键词: 同构多核处理器; 任务调度; 分簇混合调度; 查找算法; 分配算法

Scheduling on Homogeneous Multi-Core System

XU Yong-Zhen, CHEN Xiang-Lan, LI Xi, ZHOU Xue-Hai

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230001, China)

Abstract: The frequency of CPU has boosted in recent years to meet the requirement of modern application on compute performance. However, the “Moore Law” can’t be maintained only by increasing the frequency of single chip, the homogeneous multi-core system has appeared to compensate this insufficient. With the support of homogenous multi-core processors, the chip puts multiple processors with same status and structure together to minimize the cost of performance improving, load balancing and fault tolerance. The performance of parallel system is doubling when combining with appropriate software. In this paper, our research is focused on the multi-core task scheduling along with the changes in processor architecture from operating system level. The system uses hybrid scheduling which is composited of independent inter-cluster scheduling and unified inner-cluster scheduling. We do some deep analyzing on multi-core scheduling theory and implementation strategy from different aspects, including scheduling model, scheduling algorithm, dispatch algorithm and scheduling occasion. The experiment results prove the correctness and schedulability of the algorithms.

Key words: homogeneous multi-core processor; task scheduling; hybrid scheduling; search algorithm; distribute algorithm

1 引言

随着应用需求的扩展和软件技术的进步, 处理器性能提升需求激增. 通过提高处理器频率来提升系统性能的方法受到功耗、成本及体积的限制, 单核处理器验证以继续维持“摩尔”定律[1], 同构多处理器,(Homogeneous Multi-core)应运而生. 同构多核处理器是指在一块芯片上使用多个地位对等、结构相同的的通用处理器核, 各核执行相同或相似任务, 整个芯片作为一个统一的结构对外提供服务, 输出性能, 以最小的代价满足提高性能、实现负载均衡的需要[2].

① 基金项目:国家自然科学基金(61272131)

收稿时间:2014-03-02;收到修改稿时间:2014-04-02

单靠多核处理器平台并不能实现性能的倍增效益, 需要与多核硬件相适应的软件, 才能真正发挥多核的性能. 操作系统作为计算机的系统软件, 能否针对处理器结构的改变, 管理硬件并且为应用软件提供基础平台, 是多核处理器结构应用的基础问题.

操作系统的任务调度对充分发挥多处理器结构的并行性起到决定性的作用. 如何通过恰当的调度机制和策略保证多核的负载均衡, 并充分发挥各处理器的性能特点、提高整个系统的吞吐量, 是多核任务调度的研究核心. 本文以多核处理器为具体硬件环境, 研究并实现同构多核上的任务调度.

2　调度模式的选择

调度模式是指运行调度程序的处理器选择, 即哪个处理器来执行调度, 有 “集中调度模式”和 “分布调度模式”两种模式.

2.1 集中调度模式

集中调度模式指定一个特定的处理器作为主处理器(Master), 负责整个系统中所有任务的调度, 包括任务的初始化、任务的切分、任务的分配、任务的合并等工作, 如图 1. 系统中其他剩余的处理器作为从处理器(Slave), 用于执行主处理器分配的任务.

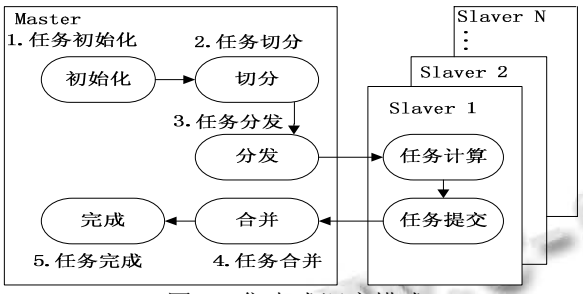

图 1　集中式调度模式

若集中式调度模式应用在大规模并行处理器 MPP 结构中, 主处理器很可能成为整个系统的瓶颈. 另一方面, 若系统中仅存在较少的处理器, 主处理器可能成为较闲的处理器, 造成资源的浪费.

2.2 分布式调度模式

本系统采取分布式调度模式, 所有处理器处于平等地位, 每个处理器都可以运行调度程序.

在分布式调度模式中, 需要保证各核对共享资源(如: 公共就绪队列)的互斥访问, 因此存在多核同步问题. 同时, 由于该模式中所有核对等运行调度程序, 假设核 A 选择核 B 执行新的就绪任务, 此时需要核 A 通知核 B 进行任务切换, 因此存在多核通信问题.

2.2.1 多核同步

本系统设计自旋锁(spin lock)保证各核对共享资源的互斥访问. 以公共就绪队列为例, 访问公共就绪队列前, 需要获得自旋锁. 某核申请自旋锁时, 若自旋锁没有被占用, 就占用该自旋锁, 执行代码; 若自旋锁已经被另一 CPU 占用, 则该 CPU 采取 “忙等”方式, 循环执行申请锁的指令, 直到锁被释放. 申请自旋锁的伪代码如下:

```
_SMP_lock_spinlock_Obtain(lock){
    do {
        内存屏障;
        SMP_CPU_SWAP( lock, previous );
        内存屏障;
    } while (previous == 1);
}
```

其中, SMP_CPU_SWAP 等价下列汇编语言片段:

```
lock;
xchgl *address, Per_CPU_Control->lock;
```

SMP_CPU_SWAP 暂时锁住总线保证操作的原子性, 并利用 xchgl 指令一直读入 lock 变量所在内存 address 中的新值.

2.2.2 多核通信

本系统中通过彼此发送核间中断来实现多核之间的通信. 进一步地, 通过给核间中断附加动作, 不同的 CPU 可以在某种程度上彼此进行控制.

与单核处理器不同, 多核处理器系统并不采用一个外部中断控制器, 而采用高级可编程中断控制器 APIC (Advanced Programmable Interrupt Controller)[3]. 如图 2, APIC 分为两个部分: I/O APIC 和本地 APIC.

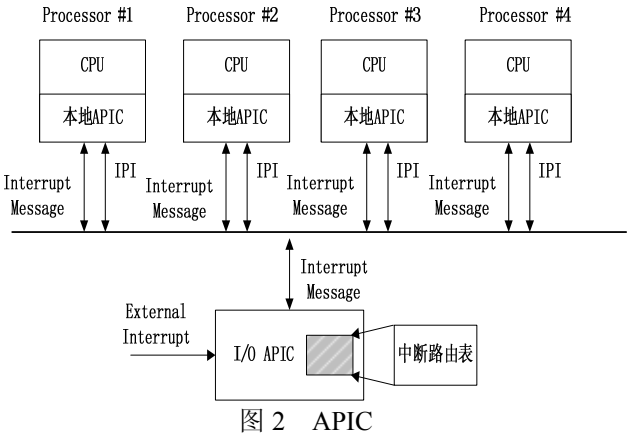

图 2　APIC

① I/O APIC 接收由 I/O 设备引起的中断, 利用中断重定向表, 查找中断的目的 CPU, 将外部中断请求通过 APIC 总线转发到目标 CPU 的本地 APIC.

② 本地 APIC 负责收集 I/O APIC 转发的中断以及 CPU 之间产生的核间中断, 再将其发送给 CPU 处理.

上述的核间中断只能够打断目的核的原执行序列, 并不能够让其了解具体的下一步工作. 需要设置核间通信消息, 用以给核间中断附加动作.

本系统为每个核配置一个消息池, 用来存放核间通信消息. 如图 3 所示, 发生核间通信时, 通信发起方往目标核的消息池中插入一条消息, 并向目标核发送核间中断. 收到中断的通信响应方暂停当前的执行序列, 进入中断处理程序, 中断处理程序通过中断向量号判断收到的中断为核间中断后, 调用核间中断处理程序. 核间中断服务例程读取本核消息池中的消息, 并根据消息的内容作出相应的工作. 特别地, 消息 SMP_CONTEXT_SWITCH_NECESSARY 用来通知目的核执行重调度和任务切换.

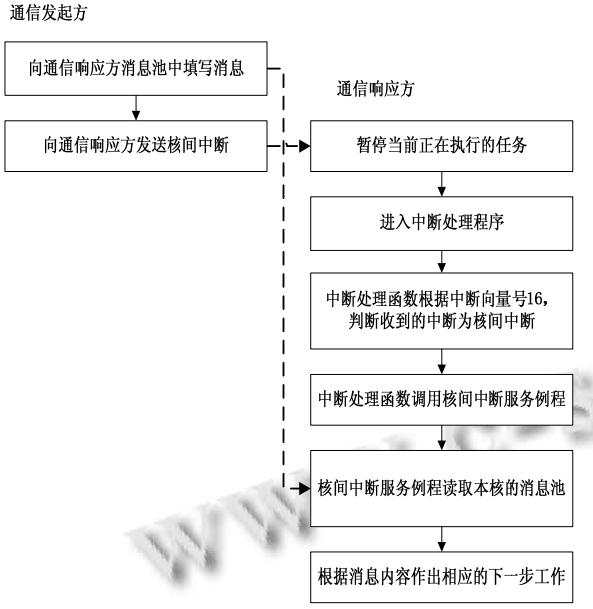

图 3 多核通信流程

3 调度策略的选取

调度策略按照就绪队列的组织方式可以分为: “全局调度”和“局部调度”[4].

全局调度是指系统中只存在唯一的就绪队列, 由所有 CPU 核共同维护, 如图 4. 调度时机产生时, 调度程序选择全局就绪队列中具有最高优先级的任务执行. 当系统规模很大时, 任务数的增多会使全局调度队列的开销急剧增大, 全局就绪队列容易成为瓶颈影响多核系统的并行性.

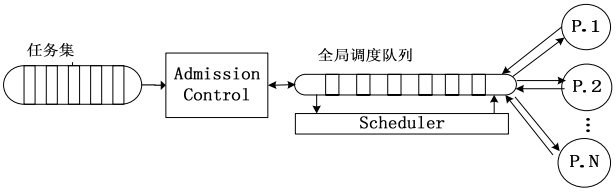

图 4 全局调度就绪队列组织方式

局部调度是指为每个处理器核配置一个就绪队列. 任务按照一定的规则分配给处理器核后, 各核独立调度本地就绪队列中的任务, 如图 5.

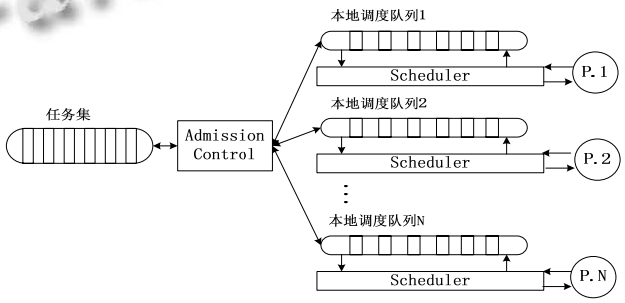

图 5 局部调度就绪队列组织方式

局部调度可能会造成某些 CPU 忙, 而某些 CPU 却处于空闲状态[5,6]. 为了实现负载平衡, 系统需要每隔一段时隙, 检查所有就绪队列的负荷, 选择负荷最大的就绪队列, 将该队列上部分任务迁移到另一负荷值较小的就绪队列[7]. 负载平衡算法较为复杂, 时间复杂度与任务的个数相关, 预测性不强, 有不确定性.

本系统采用分簇混合调度, 将系统中的所有可用核分组, 共享高速缓存的多个核作为一个簇集, 每个簇集内建立一个本地调度队列. 混合调度策在簇间独立调度, 类似与局部调度策略, 首先将任务静态的分配到各个处理器簇集. 然后各簇集各自调度本地队列中的就绪任务, 簇内各核统一调度. 在基于簇集的混合调度算法调度方案如图 6 所示.

不同于局部调度算法, 混合调度算法中每个簇集包含有两个或两个以上的处理器核, 当任务分配算法将利用率很高的任务分配到一个处理器核后, 该处理器核剩余的处理能力可与同一簇集内的其他处理器核所剩余的处理能力合并, 因此, 混合调度算法中处理器核的处理能力可得到更充分的利用.

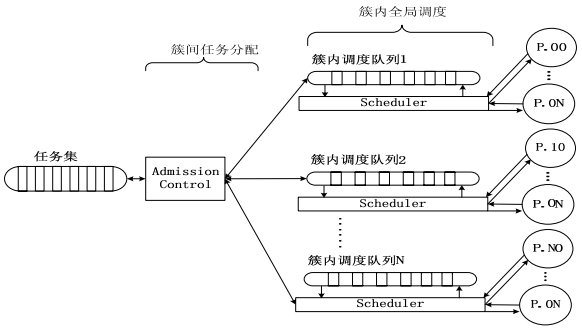

图 6　混合调度就绪队列组织方式

不同于全局调度策略, 采用混合调度策略的系统中, 每个簇内都有一个调度队列, 当系统规模很大时, 任务数的增多并不会使调度开销急剧增大. 另一方面, 分簇混合调度策略仅允许任务在簇内迁移, 而不能够在簇间迁移, 因此在一定程度上减少了由于任务迁移引起的系统开销.

4　簇内全局调度

采用分簇混合调度时, 将系统中共享高速缓存的多个核作为一个簇集, 可将存在数据依赖的任务分配到同一簇集, 以减少缓存失效率. 本系统由用户在调度程序运行前, 通过对簇集任务入口点 main_entry 的填写, 将任务静态的分配到各个簇集, 然后再簇内进行全局调度. 因此, 下文重点研究簇内的全局调度.

4.1 调度算法

根据任务对实时性的要求, 可以将任务分为普通任务和实时任务. 进一步的, 根据任务是否周期性释放, 可分为周期任务和非周期任务. 根据对任务的截止时限(Deadlines)要求, 可分为硬实时任务和软实时任务. 调度是目的是为了能够更好的使用处理器和分配资源, 需要针对操作系统所要执行任务的特点选择合适的调度方法.

本系统将任务划分为普通任务和实时任务, 普通任务拥有普通优先级(高于 RPRIO), 实时任务拥有实时优先级(低于 RPRIO). 针对普通任务, 本系统提供基本的静态优先级调度算法, 基本思想为: 任务创建时按照用户指定的参数为每个任务赋予一个优先级, 调度程序按照高优先级任务先调度, 不同优先级可抢占, 同优先级可轮转的调度策略进行任务的调度, 以确保系统内任何一个处理节点上运行的任务是处于就绪状态中优先级最高的那一个任务.

针对周期任务, 系统提供了根据任务执行周期长短决定优先级的 RM(Rate-Monotonic)调度算法[8], 周期越短优先级越高. 另一方面, 系统支持典型的动态优先级算法 EDF(Earliest Deadline First)算法[9], 根据就绪队列中的各个任务根据它们的截止期限(Deadline)来分配优先级, 具有最近的截止期限的任务具有最高的优先级.

为了同时支持上述不同调度算法, 本系统设计如图 7 所示的通用调度框架, 利用面向对象的思想, 以模块化的方式实现这些策略. 每个任务增设如下几项属性作为任务调度的依据:

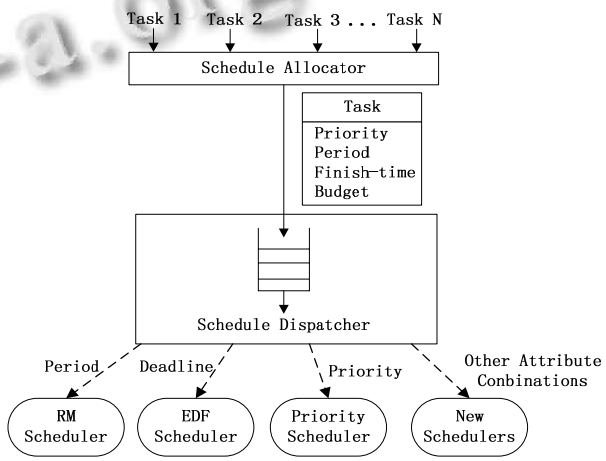

图 7　调度框架

① Priority: 任务的优先级

② Finish-Time: 任务的结束时间

③ Budget: 任务在运行期间所要使用的资源量

④ Scheduler_info: 指明任务使用的调度策略

每种调度策略都需要提供以下几种接口:

① Scheduler_Assign: 初始化就绪队列.

② Scheduler_Enqueue: 就绪队列插入就绪任务.

③ Scheduler_Extract: 就绪队列剔除就绪任务.

④ Scheduler_Yield: 进程主动放弃处理器控制权.

⑤ Scheduler_Block: 任务阻塞.

⑥ Scheduler_Unblock: 任务解除阻塞.

⑦ Scheduler_Schedule: 激活调度.

4.2 查找算法

多核环境下, 查找算法的主要任务是: 从就绪队列队首开始遍历, 挑选出前 N 个(N 为可用核个数)没有分配处理器核并且优先级最高的任务[10]. 最高优先

级查找算法应尽量选择 O(1)复杂度的算法[11].

若簇内仅设置唯一的公共就绪队列, 那么将任务插入或者剔除就绪队列的操作, 都需要从前往后遍历整个就绪队列, 时间复杂度都为 0(N)(N 为就绪任务个数). 为了满足大部分实时应用的的需求, 将簇内的全局队列划分为如图 8 所示的 256 个就绪队列. 任务按照自身优先级并结合 FIFO 原则插入对应的队列. 此时, 插入和剔除任务的操作时间复杂度变为 O(1).

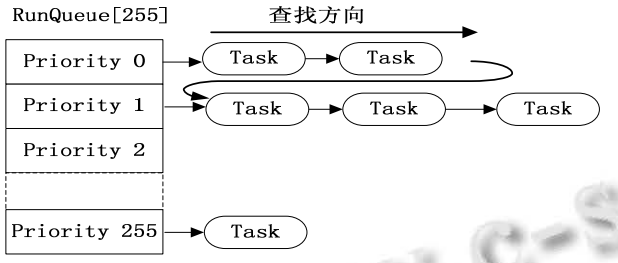

图 8 多级就绪队列

查找算法首先从队头往后遍历优先级最高的队列, 如果此队列中有未分配处理器核的任务, 则选择此任务为后备任务. 否则, 从下一优先级队列继续查找.

另一方面, 加入位图, 使查找后备任务这个过程的效率更高. 本系统使用_Priority_Major_bit_map 与_Priority_Bit_map 数组一起表示含 256 个优先级的位图, 指明给定优先级列表上是否存在就绪任务, 如图 9. _Priority_Major_bit_map 一共 16bits, 其中每个 bit 对应_Priority_Bit_map 数组中的一个元素. _Priority_Bit_map 数组一共有 16 个元素, 每个_Priority_Bit_map 元素对应 16 个优先级. 当有任务就绪, _Priority_Bit_map 位图的相应位置 1, _Priority_Bit_map 中 16 个优先级都为 0 时, _Priority_Major_bit_map 置 0, 否则置 1.

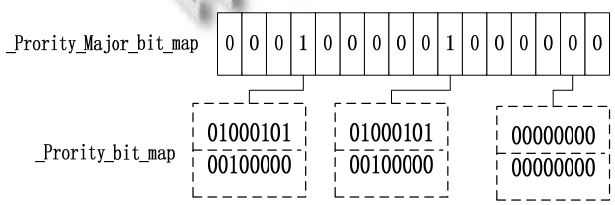

图 9 任务优先级位图

此时, 查找系统中最高优先级的任务这一操作转化为了查找位图中被设置的第一个位. 因此, 查找最高优先级的 N 个任务, 所需要的时间并不依赖于就绪任务的个数, 而是恒定的常数, 集杂度为 O(1).

4.3 分配算法

多核环境下, 分配算法的主要任务是根据系统中所有处理器核的负载情况, 以及任务优先级, 逐一地为查找算法提交的 N 个就绪任务分配合适的处理器核资源, 新的就绪任务将成为被选中核上的后备任务(heir 任务, 下一次调度时机占有 CPU 资源的任务), 等待该处理器产生调度时机调度执行.

假设 CPU 核 A 从就绪队列中取出任务 R, 分配算法思想如下:

① 遍历所有可用 CPU 核, 若有空闲核

直接将空闲核的 heir 任务指针修改为指向就绪任务 R(原指向本核的 idle 任务), 该空闲核将作为合适的处理器核资源分配给该就绪任务.

② 若没有空闲的 cpu

i. 比较各核上 heir 进程, 选择优先级最低的任务所在的核作为备选核;

ii. 若各核的优先相同, 选择已运行时间最长的任务所在的核作为备选核;

iii. 若各核的已运行时间最长, 选择不可抢占的任务所在的核作为备选核;

③ 找到合适的备选核后, 将该核的 heir 任务指针指向就绪任务 R;

综合考虑任务优先级, 可抢占性, 已运行时间等因素, 选定某核分配给新的就绪任务后, 该任务将成为被选中核上的备选任务. 此时, 若更新后的 heir 任务比该 CPU 核正在执行的任务优先级更高, 且可抢占, 则设置该核的重调度标志位 dispatch_necessary 为 1, 表明本核需要重调度, 而内核会在接下来的适当时机完成该请求, 将正在执行的任务切换出去, 而执行新分配的任务. 分配算法流程图 10 如下:

4.4 调度时机

4.4.1 主调度机制

在内核中许多地方, 需要将 CPU 分配给与当前活动任务不同的另一个任务, 如:

①当前任务主动执行可能引起阻塞的操作, 例如, 磁盘 I/O 操作或睡眠. 该任务转换为阻塞状态后, 调度程序将选择当前最高优先级的任务执行.

②当前进程并没有执行任何可能引起阻塞的操作, 由于分配给该任务的时间片用完, 或是高优先级的任务等待的 I/O 事件发生, I/O 中断唤醒了等待队列

中的更高优先级任务. 此时, 当前任务将被迫让出CPU, 重新插入就绪队列. 在中断返回前夕, 由调度程序选择就绪队列中的更高优先级进程占有 CPU.

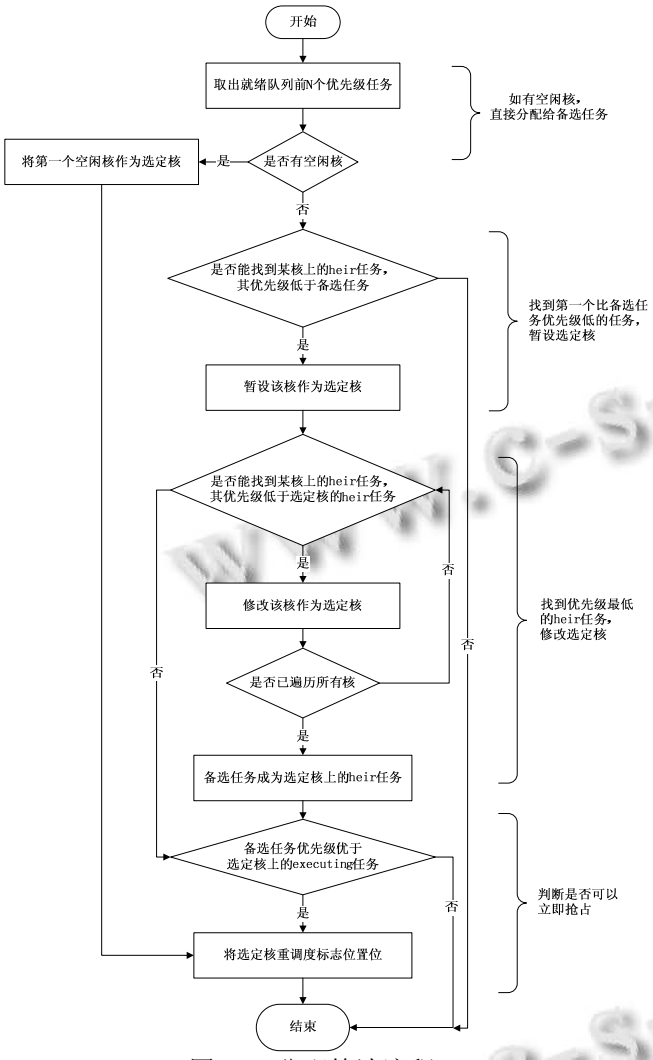

图 10　分配算法流程

本系统设计_Schedule_Schedule( )接口, 在上述情况下, 直接激活调度程序, 伪代码如下:

```
_Schedule_Schedule( ){
for(任务 R=查找算法提交的第一个备选任务; 任务 R=查找算法提交的第 N+1 个备选任务;任务 R=查找算法提交的下一个备选任务){
        if ( !分配算法( R ) )
                break;
        if ( ++已遍历的 CPU 个数 >= 簇内可用核数)
                break;
        }
}
```

_Schedule_Schedule( )逐一地为查找算法提交的 N 个就绪任务分配合适的处理器核资源, 直到某一次分配核资源失败, 说明此时系统中正在执行的任务已经是优先级最高的任务.

4.4.2 周期机制

另一种通过周期机制激活调度程序. 本系统设计 Schedule_Tick()接口以固定的频率(tick)激活调度程序, 该接口检查重调度标志位, 判断是否有必要进行进程切换. 为了实现 Schedule_Tick()接口的固定频率 tick, 将其内嵌在时钟中断服务例程 clock_tick() 中. Schedule_Tick()伪代码如下:

```
_Scheduler_Tick {
        for ( cpu=0 ; cpu <可用的核数 ; cpu++ )
                _Scheduler_Tick_helper( cpu );
        _Schedule_Schedule( );
}
```

具体完成下面两个任务:

① 利用_Scheduler_Tick_helper( )更新各核与调度相关的统计量, 如任务时间预算、已用时间等.

② 利用_Scheduler_Schedule, 检查重调度标志位, 如果重调度标志位为 1, 重新选择需要激活的任务.

4.5 任务切换

在分配算法中, 仅仅选择了合适的核分配给某一就绪任务, 改变了该核的重调度标志位. 如何让被选中的核感知自己被选中, 中断原执行序列, 真正切换到新的就绪任务是本节需要解决的问题.

如上文所述, 本系统通过 CPU 核彼此发送核间中断, 并且定义核间通信消息给核间中断附加动作, 来实现多核之间的控制.

设计 SMP_Request_other_cores_to_dispatch( )接口发送核间中断. 该接口遍历所有的核, 如果某核的重调度标志位为 1, 将向选中的核发送核间通信消息 CONTEXT_SWITCH_NECESSARY , 伪代码如下:

```
SMP_Request_other_cores_to_dispatch( ){
    for (i=1 ; i <  可用核数; i++ ) {
        if (  本核的重调度标志位为零)continue;
        SMP_Send_message(i,
                CONTEXT_SWITCH_NECESSARY);
    }
}
```

其中, _SMP_Send_message()完成两个任务:

①在目的 CPU 的消息池中填入核间通信消息, 消息内容为 CONTEXT_SWITCH_NECESSARY .

②向目的 CPU 发送一次核间中断.

目的 CPU 接受到核间中断后, 进入中断处理程序, 根据中断向量号判断该中断为核间中断后, 调用核间中断服务例程, 该中断服务例程主要三个工作:

①应答核间中断;

②读取本核消息池中的消息;

③根据消息的内容作出相应的工作.

与 CONTEXT_SWITCH_NECESSARY 消息对应的工作即为任务切换. 具体地, 调用 _Thread_Dispatch()更新相关任务的时间变量, 并调用 _Context_Switch 执行进程上下文切换, 它的工作是保存当前进程的上下文, 恢复新进程的上下文.

综上, 多核环境下, 任务切换的具体过程如图 11:

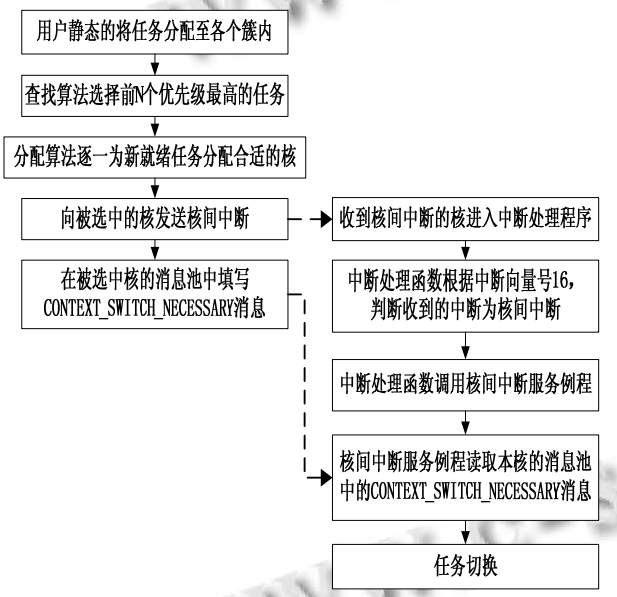

图 11 任务切换流程

5 实验

本文使用QEMU模拟包含4个核的同构多核结构, QEMU 是一款面向完整 PC 系统的开源仿真器. 首先设计功能性测试, 证明系统能够正确进行任务调度和任务切换, 验证了任务调度模块功能的正确性. 其次, 设计可调度性测试, 对比全局调度和局部调度, 证明系统的可调度上限性能的提高.

5.1 功能性测试

簇间任务分配由用户在系统运行前通过对任务入口点 main_entry 的填写, 静态地将任务分配到各个簇. 因此, 功能测试主要是对簇内 4 个同构核的全局调度, 编写如下测试用例, 其中任务 TA1 代码段 Test_task1( ) 填入簇集 1 的 main_entry 作为初始任务, 优先级为 2.

```
Test_task1( ){
    create_task( “TA2”,Test_task2, 1);
    //TA1 创建任务 TA2, 入口点为 Test_task2, 优先级为 1;
    while(1);
}
Test_task2( ){
   create_task( “TA3”,Test_tasks, 2);
   create_task( “TA4”,Test_tasks, 2);
   create_task( “TA5”,Test_tasks, 1);
//TA2 创建任务 TA3、TA4、TA5, 代码入口点为 Test_tasks, 优先级分别为都为 2、2、1;
   create_task( “TA6”,Test_task6, 1);
   //创建任务 TA6, 入口点为 Test_task6, 优先级为 1;
   while(1) ;
}
Test_tasks( ){
   while(1) ;
}
Test_task6( ){
   shut_down ( );
}
```

运行结果图如图 12, 可以看出, 任务 TA5 被创建时, 各核正在执行的任务优先级最低的为核 1 上的 TA1, 因此 TA5 抢占 TA1 的处理器资源执行. 任务 TA6 被创建时, TA3 和 TA4 的优先级都为 2, 但 TA3 的运行时间较长, 所以 TA6 选择抢占 TA3 的处理器执行.

通过该实验, 证明了该系统在多核环境下, 以下内容成立:

①所有核处于平等地位, 都能够执行调度程序;

②调度程序能够综合考虑优先级、已运行时间以及任务的抢占性等因素, 分配合适的核给就绪任务;

③执行调度程序的核能够正确的通知被选中核进行任务切换;

④被选中的核能够中断原执行程序, 切换到新分配的就绪任务.

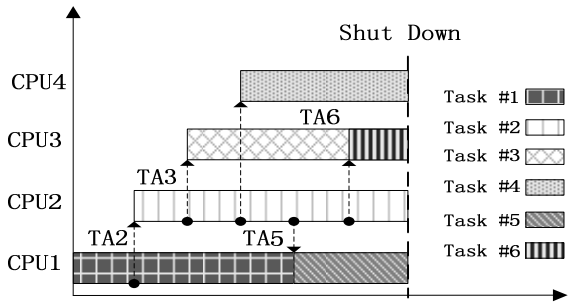

图 12　功能性测试运行结果

5.2 可调度性测试

可调度性测试使用 QEMU 模拟包含 4 个核的同构多核结构，并将其平均分成两个簇集，分别为 Cluster1={CPU1, CPU2}和 Cluster2={CPU3, CPU4}. 给定如表 1 任务集待调度:

表 1　待调度任务集

任务	周期	执行时间	截止时间
1	3	2	3
2	3	2	3
3	3	2	3
4	3	2	3
5	6	4	6
6	6	3	6

将任务 1、任务 2、任务 3 的代码段写入 Cluster1 的任务入口点 main_entry1, 表明任务 1、任务 2、任务 3 静态分配给簇集 1. 同样地, 将任务 4、任务 5、任务 6 分配给簇集 2. 另一方面, 考虑该任务集中中任务皆为具有截止时间的实时任务, 将所有任务的属性 Scheduler_info 指向 EDF 调度器类, 指明该任务集使用 EDF 最早截止时间调度策略.

运行调度算法后, 该任务集一次调度的情况如图所示, 其中横坐标表示时间, 纵坐标表示 CPU 核资源.

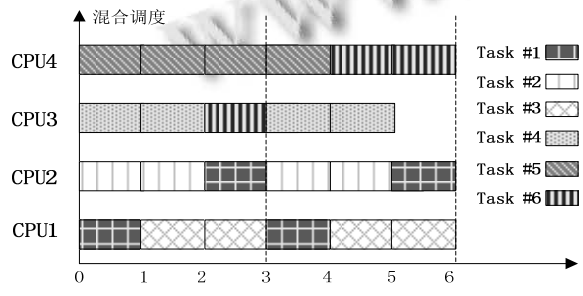

图 13　混合调度运行结果

可知, 混合调度算法可调度该任务集, 所有任务均在截止时限前完成. 另一方面, 修改配置, 将系统中的所有 CPU 核分配在同一簇中, 即 Cluster1 = {CPU1, CPU2, CPU3, CPU4}, 并将所有任务的代码段写入簇集 1 的 main_entry1, 以混合调度的这一特殊情况模拟全局调度. 再次运行调度算法, 该任务集的调度情况如图 14 所示:

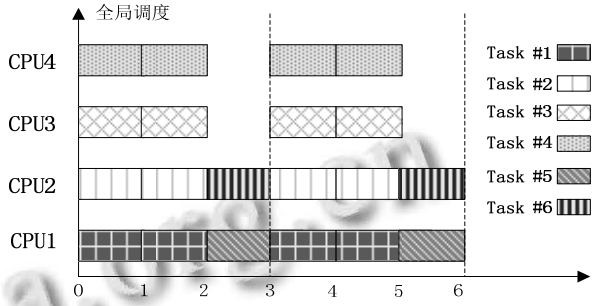

图 14　全局调度运行结果

可知, 任务 5、任务 6 都错过了截止时限. 同样地, 局部调度算法不能调度该任务集, EDF 算法调度硬实时周期任务集是, 可调度的充分必要条件是任务集的总负载≥1. 每个核被分配一个任务后, 没有足够的容量容纳其他剩余的任务, 否则将使处理中容纳的任务其总利用率大于 1, 超过处理器的处理能力.

通过本实验可以看出, 混合调度算法在一定条件下, 综合了局部调度和全局调度算法的优点, 使得系统的可调度上限性能提高了.

6　总结

本文基于同构多核处理器, 研究并实现了多核任务调度. 就绪队列方面, 采用分簇混合调度策略, 将共享数据缓存的多个核分为一个簇集, 簇间独立调度, 簇内设置公共就绪队列实现簇内全局调度. 调度模式方面, 采用本系统采取分布式调度模式, 所有处理器对等运行调度程序, 并解决了分布式调度模式中存在通信和同步问题. 然后从调度算法、查找算法、分配算法等方面研究了簇内全局调度的实现. 最后通过模拟实验验证了调度模块功能的正确性、分析了调度算法的可调度性.

参考文献

1 Schaller RR. Moore's law: Past, present and future. Spectrum, IEEE, 1997, 34(6): 52–59.

2 何军,王飙.多核处理器的结构设计研究.计算机工程, 2007,33(16):208–210.

3 何翔,任晓瑞.支持多核的嵌入式操作系统关键技术研究.航空计算技术,2013,43(4):86–90.

4 张惠娟,翟鸿鸣,周利华.多处理器系统的实时调度算法研究.计算机工程与设计,2004,25(8):1233–1235.

5 Ding W, Guo R. An effective RM-based scheduling algorithm for fault-tolerant real-time systems. International Conference on Computational Science and Engineering(CSE'09). IEEE. 2009, 2. 759–764.

6 Luo W, Qin X, Tan X C, Qin K. Exploiting redundancies to enhance schedulability in fault-tolerant and real-time distributed systems. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 2009, 39(3): 626–639.

7 He Y, Liu J, Sun H. Scheduling functionally heterogeneous systems with utilization balancing. Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International. IEEE. 2011. 1187–1198.

8 叶明,罗克露,陈慧.单调比率(RM)调度算法及应用.计算机应用,2005,25(4):889–890.

9 邢群科,郝红卫,温天江.两种经典实时调度算法的研究与实现.计算机工程与设计,2006,27(1):117–119.

10 刘怀,胡继峰.实时系统的多任务调度.计算机工程,2002,28(3):43–44.

11 李娟,任晓瑞,叶宏,时磊.一种基于实时性考虑的对称多处理机任务调度策略的设计.航空计算技术,2007,37(2):49–52.