# Linux内核基于对称多处理机的实现分析

李 彬,任国林
(东南大学,江苏 南京 210096)

**摘 要**:SMP(对称多处理机)是一种紧耦合、共享存储的系统模型。Linux内核为了支持SMP体系结构,必须要修改基于UP(单处理机)的内核代码。文中详细分析了SMP的体系结构,并对Linux内核中SMP的启动以及如何实现多处理器的进程调度进行了详细的分析,最后阐述了SMP中出现的一致性问题及解决方案,解决了Linux内核在对称多处理机上的实现。

**关键词**:对称多处理机;进程调度;自旋锁;一致性
**中图分类号**:TP332 **文献标识码**:A **文章编号**:1005-3751(2006)01-0129-03

# Analysis of Linux Kernel Based on SMP

LI Bin, REN Guo-lin
(Southeast University, Nanjing 210096, China)

**Abstract**: SMP (symmetric multiprocessing) is a system model which is tighten coupling and shares memory. The Linux kernel has to modify the kernel code as to support the SMP architecture. The paper analyses the system architecture of SMP particularly. It also analyses the kernel how to boot SMP and how to implement the process scheduling in detail. In the end, it expatiates the problem of consistency in SMP and gives the resolvment, so realizes the Linux kernel running on SMP.

**Key words**: SMP; process scheduler; spin lock; consistency

## 0 引 言

Linux操作系统最初是由Linus Torvalds和经由Internet组织起来的开发小组编写的。起初是为单处理机设计的,以免费公开源代码和GPL方式发行而得到了广泛的应用和发展。随着硬件的发展,CPU的价格大大下降,而Linux也在不断的发展,内核从2.0版本就开始支持SMP[1]系统。文中以Linux2.4.0版为基础,分析阐述了Linux内核是如何支持SMP体系结构的。

## 1 SMP体系结构

SMP属于并行处理的范畴,是一种紧耦合、共享存储的系统模型。SMP中多个CPU共享内存和外围设备,所有的CPU在运行时(系统引导和初始化除外)都是对称的,没有主次之分,一台机器就具有多台机器的处理能力(SMP体系结构[2]见图1)。

SMP体系结构之所以为人们所青睐,是因为有以下几个优点:

①当一个任务可分成几个独立部分时,它们可以并行运行,比单处理机有更好的性能;

②在SMP模式中,所有处理器处于平等地位,执行相同功能。当一个处理器损坏时,不会引起整个系统崩溃,增加了可靠性;

③当用户希望增强系统的处理能力时,可通过增加处理器来达到目的。

Linux内核要支持SMP,必须要修改基于UP的内核代码以适应SMP,主要涉及部分就是基于SMP的启动和进程调度问题。

以下主要分析、研究Linux内核如何实现基于SMP的启动、进程调度以及如何解决出现的一致性问题。

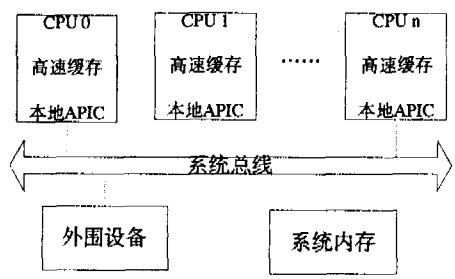


图1 SMP体系结构图

## 2 Linux内核的SMP的启动

SMP结构中的所有CPU都是平等的,没有主次之分,在系统中有多个执行的进程或"上下文"。而在SMP系统引导时只有一个可处理的"上下文",只能由一个引导

**收稿日期**:2005-04-28
**作者简介**:李 彬(1980—),男,安徽阜阳人,硕士研究生,研究方向为计算机系统结构;任国林,副教授,研究方向为计算机系统结构、嵌入式系统。

处理器(BP)运行。一个系统的启动包括以下几个步骤:

①系统加电,BIOS 启动、自检;

②BIOS 开始调入执行启动引导区程序,此程序负责具体调度并执行操作系统(Linux)启动部分;

③Linux 操作系统启动部分运行 Setup.S 和 Head.S,将全部内核解压到内存中;

④执行内核中的 start_kernel 函数,对 SMP 系统初始化,产生空闲进程。

SMP 的启动过程如图 2 所示。

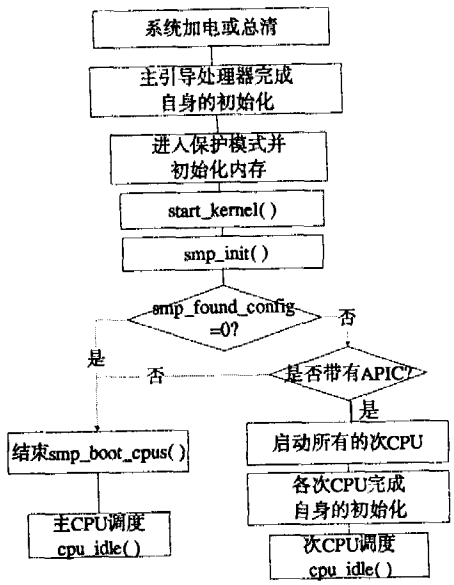


图 2 SMP 启动图

start_kernel 函数主要处理例如 cache、内存等初始化工作,smp_init 函数具体实现了 SMP 系统中各应用处理器(AP 引导启动时除引导处理器以外的处理器)的初始化。

```
static void init smp_init(void)
{
    smp_boot_cpus();
    smp_threads_ready=1;
    smp_commence();
}
```

在函数 smp_boot_cpus 中,建立并初始化各 AP,关键代码如下:

```
void init smp_boot_cpus(void)
{
.....
for (apicid=0;apicid < NR_CPUS;apicid++)
{
  if (apicid==boot_cpu_id)
  continue;
//如果是 BP,就不再需要初始化
  if (!(phys_cpu_present_map & (1 << apicid)))
  continue;
//如果 CPU 不存在,不需要初始化
  if((max_cpus >=0) && (max_cpus <=cpucount+1))
  continue;
//如果超过最大支持范围,不需要初始化
do_boot_cpu(apicid);
//对每个 AP 调用 do_boot_cpu 函数
  .....}
}
```

然后 do_boot_cpu( )为每个 CPU 建立一个 0 号(空闲)进程,这些进程共享内存,将空闲进程结构的 eip 设置为 start_secondary 函数的入口处。BP 得到 trampoline.S 代码的入口地址,将 trampoline.S 的入口地址写入热启动的中断向量(warm reset vector),发送 INIT IPI(inter-processor interrupt),发送 STARTUP IPI。而 AP 在接收到 IPI 之后,跳转到事先设置好的地址处执行 trampoline.S 和 head.S。在执行 head.S 的过程中直接跳入事先创建好的空闲进程,进入空闲状态,等待以后的系统调度,至此完成了 AP 的初始化。

## 3 Linux 中的 SMP 进程调度

Linux 刚出现时,是处理单 CPU 的操作系统,在每一个特定的时刻只有一个进程在运行。而基于 SMP 结构的系统,在同一时刻可以运行多个进程,是真正的并行处理。SMP 进程调度与 UP(单处理器)调度有明显的区别:

①需要调度的进程如何选择合适的 CPU;

②进程等待资源的处理方式。

### 3.1 SMP 系统进程调度

Linux 内核中的 SMP 进程调度的选择是双向的,用一张位图表来表示各个 CPU 是否正在运行,用 task_struct 字段中的 cpus_allowed 表示一个进程是否允许在某个 CPU 上运行(为 1 表示该任务可以在对应的 CPU 上运行),SMP 中的每个进程就可以根据调度策略在可运行的 CPU 集上选择最合适的 CPU 运行。

Linux 内核的调度[3]采取的是有条件可剥夺的方式,在操作系统运行时,一个任务的时间片用完或等待某时间发生时,CPU 处于空闲等待状态,就会引起新的进程调度。以下分析了进程主动放弃 CPU,CPU 空闲时的调度处理过程,具体流程如下:

①将 p 初始化为运行队列的第一个任务;

②用 c 记录了运行队列中所有进程最高优先级,具有最高优先级的进程是最易获得 CPU 的进程,优先级(weight)由 goodness()函数计算,计算公式如下:

$$\text{weight}=\begin{cases}\text{counter}+(20-\text{nice}) & (1)\\ \text{counter}+(20-\text{nice})+\text{PROC\_CHANGE\_PENALTY} & (2)\\ 1000+\text{rt\_priority} & (3)\end{cases}$$

上述公式中的 counter 为进程运行剩余的时间片、nice 为进程的优先级、rt_priority 为实时进程的优先级,情况(1)指非实时进程上次运行的 CPU 与当前 CPU 不同时的优先级;情况(2)指非实时进程上次运行的 CPU 与当前

CPU相同时的优先级;情况(3)指实时进程的优先级,其中Linux把PROC_CHANGE_PENALTY宏定义为20。调度程序遍历执行任务链表,跟踪具有最高优先级的进程。

③检查该任务的can_schedule( )是否为真,为真则选择该任务运行,为假重新执行查找。

④若查找结束后,c值为0,说明运行队列中的所有进程的goodness( )值都为0,schedule要重新计算进程的counter,新counter的值是原来值的一半加上进程的静态优先级。在counter的值计算完成后,重新开始执行该循环,找具有最大goodness( )的任务。

⑤如果选择的进程是先前运行的进程,不引起进程切换,否则切换到所选择的进程。

以上是对进程主动放弃CPU的处理过程,接下来分析进程被剥夺时放弃CPU后要做的处理[3],Linux内核调用schedule_tail( )进行调度后处理,SMP结构下的进程调度后的处理如图3所示。

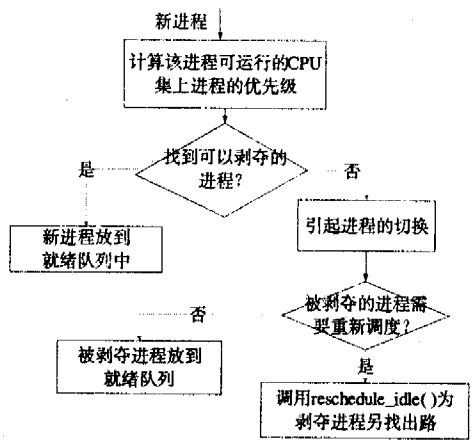


图3　进程调度后的处理图

如果在上图中的被剥夺的进程需要重新调度,系统将调度resched_idle( )试图将其转到其他CPU上运行,具体的流程如下:

①先前所处的CPU若空闲,该进程可以在原来的CPU上运行转⑤,否则转②;

②找出所有允许被剥夺的进程运行的CPU,如果没有则转⑥;

③计算以上CPU中允许运行该进程的CPU上的当前运行进程的优先级:c = goodness(pre,pre, this_cpu)找出一个优先级最低(运行资格最低)的进程,作为被剥夺对象;

④计算被剥夺进程p在最适合剥夺的那个进程所在CPU上的优先级weight = goodness(p,pre, this_cpu),若weight > c说明进程p的优先级高于当前运行的进程,可以在该CPU上运行,否则转⑥;

⑤把进程p的task_struct结构中的has_cpu置1,processor为该CPU的逻辑号,进程p在该CPU上抢占运行,调度结束跳过⑥;

⑥把该进程放在就绪队列等待调度。

### 3.2　SMP进程等待资源的处理方式

在UP结构中,一个进程等待某个资源时就进入睡眠状态,等待其它进程释放资源;而SMP系统中在一个任务等待某个资源时,Linux根据资源的特性,如果等待时间长则让进程睡眠,如果等待时间短,不让它进入睡眠,进入循环等待过程,这样可以节省进程切换的开销。为了实现这一点引入了自旋锁,自旋锁[4]就是在一个密封的循环里坚持反复尝试夺取一个资源(一把锁)直到成功为止,是一种短期互斥锁[5]。有如下两个函数实现:

①加锁函数。

```
#define spin_lock_string \
    "\n1:\t" \ //段1:
    "lock;decb %0\n\t" \ //将lock减1
    "js 2f\n" \ //为负则跳到段2:
    LOCK_SECTION_START("") \
    "2:\t" \ //段2:
    "cmpb $0,%0\n\t" \ //与0比较
    "rep;nop\n\t" \
    "jle 2b\n\t" \
    //如果小于等于0跳到上面段2处继续测试
    jmp 1b\n" \
    //lock大于0跳到上面段1处执行
    LOCK_SECTION_END
static inline void spin_lock(spinlock_t * lock)
    {
    ..........
    __asm__ __volatile__(
    spin_lock_string
    :"=m"(lock->lock) : :"memory");
    }
```

②解锁函数。

```
#define spin_lock_string \
movb "$1 %0" //把lock置1,释放锁
static inline void spin_unlock(spinlock_t * lock)
    {
    ..........
    __asm__ __volatile__(
    spin_unlock_string
    :"=m"(lock->lock) : :"memory");
    }
```

进程要获得自旋锁时先要调用spin_lock(),如果已经有进程获得了该资源,该进程就一直空转运行下去。一个进程获得自旋锁运行完毕后要通过调用spin_unlock()把自旋锁释放,让其他进程使用。

## 4　SMP结构中的一致性问题

SMP结构中的一致性问题[2]包括高速缓存一致性和存储一致性。高速缓存一致性指私有高速缓存中共享数据的副本和共享存储器中共享数据之间的一致性,存储一

(下转第153页)

作方式;文献[18]在研究多 Agent 仿真方法中采用了 HLA/RTI 作为多 Agent 仿真环境的支撑环境。这些都是将 HLA 和 MAS 相结合的尝试,也是把 AI 技术和仿真技术进行融合的有意义的探索。

## 3 结束语

HLA 和 MAS 在仿真领域的发展都颇受关注。文中对两者在个体结构、通信和模型建立方面进行的比较,以及对于比较结果的讨论,对于研究人员开发分布式仿真系统是有参考价值的。而把两者结合起来,在 HLA 规范框架下实现多 Agent 系统,将是未来分布式仿真的新方向。

**参考文献:**


[1] Crosbie R, Zenor J. High Level Architecture Module1 Basic Concepts(HLA1516)[R]. SCS, Mcleod Institute of Simulation Sciences, California State University,2001.

[2] Wooldridge M J, Jennings N R. Intelligent agent: Theory and Practice[J]. Knowledge Engineering Review, 1995,10(2):115-152.

[3] 王文杰,叶世伟.人工智能原理与应用[M].北京:人民邮电出版社,2004.

[4] 周 彦,戴剑伟.HLA 仿真程序设计[M].北京:电子工业出版社,2002.

[5] FIPA Agent Management Specification[EB/OL]. Foundation for Intelligent Physical Agents. http://www.fipa.org/, 2000.

[6] FIPA ACL Message Structure Specification[EB/OL]. http://www.fipa.org/specs/fipa00061/. 2002.

[7] 胡勤友. Web上的多主体系统若干关键技术研究[D].上海:复旦大学,2003.

[8] DMSO. RTI 1.3 - Next Generation Programmer's Guide Version 4[EB/OL]. http://www.dmso.mil, 2001.

[9] 魏晓斌,周盛宗,Bachmendo B,等. Agent 通信机制探讨[J].计算机工程与应用,2002,38(5): 66-70.

[10] 黄 健,黄柯棣. HLA 中的时间管理[J].计算机仿真,2000,17(4):69-73.

[11] 陈凌云,姜振东. HLA 中对象模型的研究[J].计算机仿真,2003,20(1):79-82.

[12] 李 军,苏国庆.基于 HLA OMT 的通信对抗建模仿真[J].舰船电子对抗,2004,27(5):27-30.

[13] Palmer M. HLA Simulations vs. Agents, Object Services and Consulting[J/OL]. http://www.objs.com/agility/tech-reports/9807-HLA-vs-ACL.html.1998-07.

[14] Lutz R. A Comparison of HLA Object Modeling Principles With Traditional Object-Oriented Modeling Concepts[A]. 97F-SIW-025, Simulation Interoperability Workshop[C]. Orlando, Florida:[s.n.],1997.

[15] 罗 批,司光亚,胡晓峰,等.基于 Agent 的复杂系统建模仿真方法研究进展[J].装备指挥技术学院学报, 2003,14(1):78-82.

[16] 高志年,邢汉承.基于 HLA 的多 Agent 系统体系结构研究[J].小型微型计算机系统,2003,24(13):336-339.

[17] 陈艳彪,李志刚,黄建明,等.基于多智能体的坦克分队对抗仿真模型研究[J].系统仿真学报,2004,16(4):705-707.

[18] 侯 锋,陈洪辉,罗雪山. 基于多 Agent 的 C4ISR 系统建模与仿真方法研究[J].光电技术应用,2004,19(3):25-30.


(上接第 131 页)

致性是指多个处理器程序的执行次序与共享存储器中共享数据存取次序之间的一致性。高速缓存与内存的数据一致性问题几乎都是由硬件完成的,Intel 在 Pentum CPU 提供了一种"窥探"机制,监听着系统总线上的操作,因而 SMP 结构中的高速缓存与内存的数据一致性问题是软件透明的。但页面映射目录和页面映射表的缓冲存储(TLB)的一致性,要通过软件来辅助实现。

当一个 CPU 改变内存某个页面目录或映射表中的内容,都可能引起其他 CPU 中的 TLB 内容不一致,就向系统中使用这个映射表的 CPU 发出 flush_to_others( ),毁弃 TLB 中的内容。如果修改的是一个页面表则调用_flush_tlb_one( )毁弃一个页面,若修改的是页面目录则调用_local_flush_tlb( )冲刷整个 TLB,以保证各 CPU 中 TLB 的一致性。

## 5 结束语

SMP 系统是一种利用共享内存空间与外围设备实现多进程的计算机体系结构,它最重要的机制在于对内存空间的共享。通过 SMP 所获得同等性能的提高,要比购买几台独立的机器把它们组合在一起更加便宜和简单,并且它比等待下一代 CPU 面世要快得多。SMP 系统能够以较小的成本换来更大效益,是一种经济的选择。但是,采用 SMP 提高性能也要付出代价的,为了支持 SMP 系统,Linux 内核复杂度和协同的开销大大增加,CPU 必须不能互相干涉彼此的工作,必须要解决 SMP 的启动与进程调度问题,以及一致性问题。随着 Linux 内核不断发展,Linux 操作系统对 SMP 的支持也会越来越完善。

**参考文献:**


[1] 杨孟辉,李 伟,廖建新,等. SMP 的结构分析研究[J].高技术通讯,2002(2):21-25.

[2] Schimmel C. 现代体系结构上的 Unix 系统[M].张 辉译.北京:人民邮电出版社,2003.

[3] Cesati B. 深入理解 Linux 内核[M].陈莉君译.北京:中国电力出版社,2001.

[4] 徐晓磊,董兆华,吴建峰,等. Linux 可抢占内核的分析[J].计算机工程,2003,29(15):115-117.

[5] 毛德超,胡希明. Linux 内核源代码情景分析[M].杭州:浙江大学出版社,2001.