Linux 内核完全公平调度器的分析及模拟

张桂兰[1,2] 王飞超[3]
1.济南广播电视大学信息技术学院 250001
2.山东大学网络信息安全研究所 250001
3.山东教育学院网络分院 250001

摘 要
调度算法的优劣对操作系统性能起着重要作用，linux 最新发布的完全公平调度器 CFS，其整体性能比以往大大提高，本文重点分析 CFS 调度器的工作原理，并介绍了调度算法的模拟器 schedsim 及其工作原理，并对 schedsim 模拟器进行改进以使其更好的支持 CFS 调度算法，最后给出一个模拟实例。

关键词
CFS 调度器；Schedsim 模拟器；操作系统

Abstract
The scheduler' superior and inferior has important influence to the operating system' s whole function. Linux issues the latest Completely Fair Scheduler (CFS) which' s capability is increased by a big margin . Especially we aspected principle of cfs and analyzed common scheduling schedsim simulators, we improved schedsim simulator to better support the linux' s cfs scheduling algorithm. Finally we provide some source code about schedsim.

Key words
CFS Scheduler; Schedsim Simulator; Operating System

1 引言

操作系统内核调度算法历来是人们改进系统性能的研究热点。作为主流操作系统之一的 linux，它的调度算法几经改进，表现出优异的性能，在越来越多的领域逐渐占据重要地位。纵观 linux 调度器的发展，大致经历了三个阶段：最早期的 0.11 内核中的 O(n)调度算法，并一直到 2.4 内核都没有大的改变；随后在 2.6 内核中发布了由 Ingo Molnar设计并实现的O(1)调度器，该调度器与过去调度器相比获得了长足的进步；最后就在2.6.23内核中发布新的完全公平调度器 CFS (Completely Fair Scheduler)，它采用了与以往调度器完全不同的设计理念，具有革命性的意义。其调度复杂度依然为 O(1)，并使得进程更加公平的共享处理器资源，这也是 cfs 调度器最终被采纳的主要原因。调度算法的开发与实现并不是一件容易的事，很多情况下需要先对算法进行模拟测试，然后才能决定是否有必要在真实平台上实现。目前在模拟调度算法方面，基于 linux 调度算法的模拟器则少之又少，已有的模拟器总是在某些方面表现不足，无法很好的模拟 linux 的调度过程本文重点介绍了开源项目 schedsim 模拟器，并对其性能加以改进，使其支持CFS调度器，最后并给出一个模拟实例，并可以很直观的看到模拟效果。

2 新一代调度器 CFS 的工作原理

cFS 调度器是由 HignMolnar 设计实现的，在2.6.23内核版本中发布。其设计初衷是让任务更加公平的共享 CPU 资源。CFS 50% 的设计可以总结为一句话：CFS 在真实硬件上实现了一个“理想精确的多任务 CPU”。理想多任务 CPU 的含意是 CPU 具有 100% 的处理能力并且能以相同速率并行运行每一个任务。例如有两个任务在执行，则每个任务利用CPU50%的处理能力。在新内核中添加的调度器的主要特性包括三个方面：模块化的调度器接口，模块化的意思并不是说调度器能够以可加载模块的方式动态添加，而是在内核代码中添加 CFS调度器，这是新内核中最为核心的内容，它确保进程公平共享CPU；CFS组调度，组调度是为了使用户能公平的共享 CPU。

2.1 相关数据结构

CFS 为每个 CPU 使用一个按时间排序的红黑树结构。之所以使用红黑树，是因为：红黑树总是平衡的；对红黑树的操作时间复杂度为 O(logn)，当进程数少时其性能表现并不差，只有当进程数比较大时才会有一定性能损失，但对其最左边节点的存取可以通过 cache 来高效实现。对于每一个运行队列都有一个数据结构来与红黑树相关联，这也是CFS中最为重要的一个数据结构，它就是 struct cfses rq。其定义如下：

```
struct cfses rq{
struct load_ weight load;
    unsigned long nr_ running;
    u64 exec_clock;
    u64 min_ vruntime;
    struct rb_ root tasks timeline;
    struct rb _node *rb leftmost;
    struct rb_ node *rb load balance-
curr;
    struct sched_ ntity *curr;
    unsigned long nr_ spreades over;
#ifdef CONFIG _FAIR_ GROUP_
SCHED
    struct rq *rq;/* cpu runqueue to
which this cfs rq is attached*/
    struct list_head leaf _cfs rq_list;
    struct task_ group *tg;/* group
that "owns" this runqueue*/
#endif
};
```

2.2 实现要点

(1)实现 pick next：CFS 抛弃了 active/expire 数组，而使用红黑树选取下一个被调度进程。所有状态为RUNABLE的进程都被插入红黑树。在每个调度点，CFS调度器都会选择红黑树的最左边的叶子节点作为下一个将获得 cpu 的进程。

(2)tick 中断：在 CFS 中，tick 中断首先更新调度信息。然后调整当前进程在红黑树中的位置。调整完成后如果发现当前进程不再是最左边的叶子，就标记need_resched标志，中断返回时就会调用 scheduler()完成进程切换。否则当前进程继续占用 CPU。从这里可以看到 CFS 抛弃了传统的时间片概念。Tick中断只需更新红黑树，以前的所有调度器都在 tick 中断中递减时间片，当时间片或者配额被用完时才触发优先级调整并重新调度 。

(3)红黑树键值计算：该键值由三个因子计算而得：一是进程已经占用的 CPU 时间；二是当前进程的nice值；三是当前的cpu负载。进程已经占用的 CPU 时间对键值的影响最大，该值越大，键值越大，从而使得当前进程向红黑树的右侧移动。另外CFS规定，nice 值为 1 的进程比 nice 值为 0 的进程多获得 10% 的 CPU 时间。在计算键值时也考虑到这个因素，因此nice值越大，键值也越大。红黑树是平衡树，调度器每次总最左边读出一个叶子节点，该读取操作的时间复杂度是 O(logn)。

(4)调度器管理器：为了支持实时进程，CFS 提供了调度器模块管理器。2.6.23 中，CFS实现了两个调度算法，CFS算法模块和实时调度模块。

以上的讨论看出 CFS 对以前的调度器进行了很大改动。用红黑树代替优先级数组；用完全公平的策略代替动态优先级策

略；引入了模块管理器；它修改了原来Linux2.6.0调度器模块70%的代码。结构更简单灵活，算法适应性更高。

3 Schedsim模拟器简介

3.1 Schedsim模拟器工作原理

本文所信赖的模拟器是schedsim，它是一个开源软件项目，实现了通用调度算法的模拟平台，提供了良好的接口及几种常见的调度算法，可以容易的实现新的调度算法并可立刻获得直观的结果。Schedsim完全由Java语言实现，这带来的好处是其可移植性大大增强。并且利用Java丰富的图形接口，提供了良好的可视化界面。Schedsim模拟过程中需要实现的最重要三部分是：进程信息的模拟，进程是调度的实体，因此对这部分的模拟是必不可少的，这主要包括进程的到达时间、进程的优先级、进程的执行时间以及进程的截止时间等；处理器信息的模拟，处理器是进程执行的载体，因此也是模拟过程中必不可少的一部分，这包括处理器的处理能力，时钟中断等信息；调度算法模拟，调度算法是进程分配的核心部分，在需要重新调度时，调度算法按照一定原则从就绪队列中选择合适的进程运行，不同的调度算法体现了不同的调度思想。其类结构图如图1所示：

3.2 加载CFS调度器实现

Linux的快速发展使得它在操作系统领域开始占据越来越重要的地位。在服务器和嵌入式领域已经成为主流平台。作为一个调度器的模拟平台，对linux的支持自然是必不可少的。我们通过扩充schedsim的接口以实现它对linux最新调度器CFS的支持。

(1)添加调度器基本信息

对于CFS调度器，每个进程需要记录额外的信息以供调度时使用。GeneralProcess所提供的信息已经明显不够，因此通过继承GeneralProcess，在保留原有内容的基础上增加CFS所需的信息。同时提供对这些信息存取的接口。以下就是其实现方式：

```
Public class CompleFairProcess extends eneralProcess {
    Private long load=1024;
    Private long vruntime=0;
    Publie Boolean first add=true;
    private long start exec=-1;//the time it's selected to run
    public Comple FairProcess (GeneralProcess gp)
    Public void setLoad(long l)
    Public long getLoad()
    Public long getVRuntime()
    Public void setVRuntime(long vr)
    public void setStartTime(long start)
    public long getStartTime() }
```

(2)创建红黑树的数据结构

CFS调度器中每个处理器的运行队列与以往大为不同，它采用红黑树的数据结构将进程联系起来。这样在查找下一个可执行进程时只要获得树中最左边的节点就可以，大大提高了效率。运行队列还保存了当前队列负载等信息，它为CFS调度器的实现提供了方便的接口，是整个CFS调度器的核心部分。其实现方式如下所示：

```
Public class CFSProcessList extends ProcessList{
    private long load_ weight=0;
    private long min _vruntime=0;
    private int num_ running=0;
    static public long sysctl _sched latency=20;
    static public long schedse _nr _latency=5;
    private CompleFairProcess curr=null;
    publicCFSProcessList(SchedulerInterface Interface)
    public CompleFairProcess getCurr()
    public void setCurr(CompleFairProcess rocess)
    publicBooleanAddProcess (GeneralProcessProcess, Boolean setid)
    public void updateCurr( )
    public long calculateVR (CompleFairProcess cfp)
    public long getSchedPeriod(int nr)
    public long getSchedSlice (CompleFairProcess cfp)
    public long getVRuntime()
    public void setVRuntime(long vruntime)
    public CompleFairProcess getFirstRunnable()
    public CompleFairProcess getLastRunnable()
    public Boolean AllUpdateStatistics()
    public int getNumRunnable()
    public void dumpCFSProcessList()
    }
```

(3)最后需要添加CFS调度器，该调度器继承自GeneralScheduler，实现了其Schedule接口。该接口通过调用CFSProcessList的接口从中选择合适的进程执行。此外，还需要在SchedulerGenerator中将其添加进去。CFS调度器的处理流程如图2所示。

4 测试结果

设定两个用户组进程，程序运行共需要19个时间片。选取其中第16个时间片的程序运行结果，显示如下：

```
ProeessID:4034 ParentID:403
ProeessID:4036 ParentID:403
ProeessID:4033 ParentID:403
ProeessID:4035 ParentID:403
ProeessID:4037 ParentID:403
4033Usertime:3.520seeonds
4033Children, susertime:0.000seeonds
4034User-time:3.500seeonds
4034 Children's user-time:0.000seeonds
4037User-time:5.140seeonds
4037Children's user- time:0.000seeonds
```

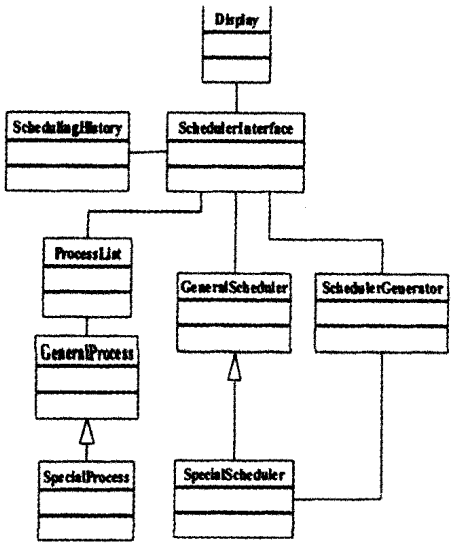

图1 Schedsim类结构图

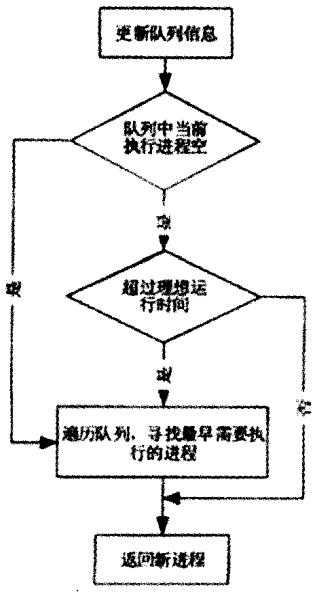

图2 CFS调度器处理流程

下转第137页

变化改变了与逻辑的输入值，由此产生内部调节行为，根据与逻辑关系运算产生相关结果并显示在输出引脚上，经发送数据到与之相连接的虚拟器件，作为接收者的器件的ExternalCond随之发生变化，那么它将根据外部条件和自身条件进行调节，重复刚才的过程。虚拟实验系统就是通过这种方式实现实验过程的。

四．虚拟器件建模举例

微机接口实验所需的器件种类繁多，功能各异，应具体分析各类外观属性与内部逻辑关系，建立不同的模型。本文以常用的8255为例。

8255为集成芯片，它的特点是芯片通过引脚与外界传递信息，所以通过定义每个引脚以及引脚间的逻辑关系即可描述该器件模型。

8255外部有40根引脚，除电源和地以外，其他信号分为两组：和CPU相连的有D_0-D_7、CS、RD、WR、A_0 A_1、RESET，与外设相连的有3组8位的数据信号线PA_0-PA_7、PB_0-PB_7、PC_0-PC_7。分析8255的工作原理，其工作方式由用户的方式命令字确定。对于每一种可能的工作方式，产生的控制信号是不同的，当输入引脚接收到CPU传来的不同控制信号时，内部硬件逻辑产生不同的动作，并在相关引脚产生输出。表1描述了8255由输入控制信号所确定的工作状态表，反映了CPU作为主动者对8255的影响以及8255根据外部条件改变和自身内部逻辑而做出的调节。

通过以上分析，归纳其模型应包括如下一些主要内容：

1．属性：

器件ID：唯一确定器件的ID号；Width,height,Left,Top:元件外观尺寸,分别是宽、高、位置（Left,Top）;Pin(i)(i=1,…,PinCount):虚拟元件引脚对象，8255芯片需定义40个引脚；Ram(i):芯片内部的寄存器或存贮器。Attri(i=1,…,n)n个属性定义；GND、Vcc和CS作为器件工作的控制开关。

2．行为：

表1　接口8255工作状态选择表

A_0	A_1	RD	WR	CS	工作状态
0	0	0	1	0	A口数据→数据总线
0	1	0	1	0	B口数据→数据总线
1	0	0	1	0	C口数据→数据总线
0	0	1	0	0	数据总线→A口
0	1	1	0	0	数据总线→B口
1	0	1	0	0	数据总线→C口
1	1	1	0	0	数据总线→控制字寄存器
x	x	x	x	1	数据总线→三态
1	1	0	1	0	非法状态
x	x	1	1	0	数据总线→三态

Inititail():器件初始化方法；Draw():器件外形绘制方法；PinFnti():引脚之间的逻辑关系描述函数，根据工作方式控制字和工作状态表决定输入引脚与输出引脚的逻辑关系。此外，引脚间连接时存在约束关系，如Vcc与GND不能连在一起等，作为电路检查的依据。逻辑关系不止一个，需要用多个函数来描述。MessaggeRespond():消息响应方法Access():外部访问方法；Send():信号发送方法等。其中，引脚类由以下属性组成：PinNo:引脚在芯片中的编号；PinPoint:引脚在窗口中的坐标位置；BitValue:当前引脚的值：0或1；Wire:与引脚相连的连线。存贮单元类由以下属性组成；RamNo:Ram在芯片中的编号；RamAddr:Ram的地址；RamValue:Ram的值。

五．结束语

虚拟器件模型构建后，就可以建立虚拟连线模型。器件和连线作为虚拟实验系统中主要的实体对象，其内在属性、外观形状、功能特点、行为表现都各不相同，通过前面的分析可分别建立各自的虚拟原型。微机接口实验作为一个真实存在的实体，包含了学生完成实验的全过程。从它本身的属性和功能方面出发，提取出与实验相关的数据进行分析，也可以建立起虚拟实验的模型。微机接口实验的虚拟原型旨在为用户构建一个完成虚拟实验的实验平台，在上面有各种虚拟器件供用户选择，可以进行器件连接、复制、删除、移动、存储等操作。因此，有了虚拟器件的模型，就可以为虚拟实验室的建模研究提供重要的前期工作基础。

参考文献

[1] 齐欢，王小平．系统建模与仿真．北京：清华大学出版社．2004：229-261．

[2] 王永武，王咏刚．面向对象实践指南．北京：电子工业出版社．2004．

[3] 张雯，黄新燕，于蕾，《微机原理》及《微机接口技术》实验教学实施．南京理工大学学报．1997年第06期．

[4] 朱刚，微机接口虚拟实验的研究．华中理科大学图书馆．2004．硕士学位论文．

[5] 刘瑞叶，任洪林，李志民．计算机仿真技术基础．北京：电子工业出版社．2004：8-23．

作者简介

黄秀丽，汉族，1974年12月生，现为辽宁省营口职业技术学院计算机系教师，主要讲授微机原理及接口技术课程，并从事相关学科领域的研究。

上接第135页

4036User-time:5.110seeonds

4036Children's usertime:0.000seeonds

ProeessID:4032 ParentID:4030

4032 User-time: 0.000seeonds

4032 Children's usertime:10.250seeonds

1035 User-time:3.520seconds

4035 Children's usertime:0.000seeonds

proeessID:4031 parentID:4030

4031User-time:0.000seeonds

4031 Children's time:10.530seeonds

用户userl的CPU占用率：10.25/10.25+10.53=49.32%；

用户user2的CPU占用率：53/10.25+10.53=50.68%

从处理器对进程的调度过程，我们可以看到，在CFS调度算法下，处理器公平的在进程间共享。这种共享是以优先级为基础的，优先级越高，在一个调度周期内所获得的执行时间越多。可以看到，在每个调度周期内，所有进程都得到机会执行，并没有出现进程一直等待得不到机会运行的状况。这充分体现了CFS调度算法的核心思想，即公平性。

参考文献

[1] 许占文．李欲．Liuux2．6内核的实时调度的研究与改进[J]沈阳工业人学学报．2006(8):438-441

[2] 李善平，陈文智．边干边学——Linux内核指导．杭州：浙江大学出版社．2002

[3] C．L．Liu，James W Layland．Scheduling Algorithm for Multiprogramming in a Hard-Read-Time Environment．J.ACM.1973，20(1)

[4] Linux调度器内幕．http://www.ibm.com/developerworks/cn/linux/1-schedule.2008

[5] Linux2.6调度系统分析． http://www.ibm.com/developerworkslcn/linux/kernel/1-kn26schlindex.htm1.2007

作者简介

张桂兰(1980.05)，女（汉族），山东单县人，助教，硕士研究生，主要研究方向Linux操作系统内核原理；

王飞超，山东菏泽人，助教，硕士研究生，主要研究方向：计算机网络安全。