实时系统调度算法的优化设计

毛　佳[1]　张振花[2]　戴　红[3]　苑森淼[2]
[1]吉林大学通信工程学院,长春 130012)
[2]吉林大学计算机科学与技术学院,长春 130012)
[3]吉林工程技术师范学院信息工程学院,长春 130052)
E-mail:maojia@cc.cngb.com

摘　要　文章介绍了 Linux 操作系统实时调度算法的简化模型,并提出了一种优化改进调度算法。该算法以进程的重要性为基础,兼顾截止期内完成进程的紧迫程度,建立了进程的优先级队列。算法可通过双链表来实现。对比实验结果表明,优化后的算法与优化前相比,特别是 CPU 正常负载时,可以实现更高的价值完成率和进程完成率,从而有效地提高了操作系统的实时性能。

关键词　调度算法　进程　优先级　实时

文章编号 1002-8331-(2003)15-0112-04　**文献标识码** A　**中图分类号** TP316

Optimum Design to Scheduling Algorithm for Real-Time Systems

Mao Jia[1]　Zhang Zhenhua[2]　Dai Hong[3]　Yuan Senmiao[2]
[1](College of Communication Engineering,Jilin University,Changchun 130012)
[2](College of Computer Science & Technology,Jilin University,Changchun 130012)
[3](College of Information,Jilin Teachers' Institute of Engineering & Technology,Changchun 130052)

Abstract: In this paper,a simplification model of real-time scheduling algorithm based on Linux OS is introduced.This paper gives an optimal scheduling algorithm.The optimal algorithm is on the basis of the value of real-time process,also concerns urgent process before Deadline,and processing queue is found on priority.The algorithm comes up with doubly linked list.Simulation results show that the improved algorithm can get higher complete value ratio and process ratio than before.The responsive ability of RTOS can be improved effectually.

Keywords: Scheduling algorithm,Process,Priority,Real-time

近年来,实时系统在航空航天、军事、工业控制、仪器仪表、信息家电等行业得到广泛应用。在实时系统中,系统输出的正确性不仅仅依赖于计算的逻辑结果而且依赖于结果产生的时间[6]。实时系统一个重要的要求就是系统必须在一个事先定义好的时限内,对外部或内部的事件进行响应和处理,笔者称这种事先定义好的时限为截止期(Deadline)。

随着计算机硬件的发展,人们对实时系统在性能和安全等方面均提出了更高的要求。实时系统可分为硬实时和软实时(Hard Real-time and Soft Real-time)系统。硬实时系统就是系统必须及时地对事件做出反应,绝对不能发生错过事件处理或超出截止期的情况。例如控制火箭发射的系统,如果没有对突发事故做出及时的处理,将造成巨大的损失;而在软实时系统中,当系统负载较高时允许发生少数事件处理错过截止期的情况。进程调度算法是影响系统实时性能的直接因素。大多数实时系统(RTOS)的调度算法是由单调率算法(RM)和最早期限优先算法(EDF)[5]变化而来。在不同的系统状态下两种算法各有优劣,实时系统采用的实际调度算法是考虑综合影响因素的折中。

目前世界上比较有影响力的实时操作系统主要有 QNX、LynxO、RT-Linux 等。QNX 是 QNX 软件系统有限公司开发的分布式、嵌入式、可规模扩展的实时操作系统,它支持抢占式的、基于优先级的调度策略,其核心仅提供 4 种系统服务,更适用于分布式系统。LynxOS 也是一个分布式的实时操作系统,它由 Lynx 实时系统公司开发,目前还不是一个微内核结构的操作系统,支持硬实时优先级调度。相比以上两种商业 RTOS 而言,Linux 由于内核可剪裁性好、易于移植、源代码开放等优点而具备更广阔的发展空间。将 Linux 改造成实时系统,是国内外计算机界的研究热点之一。美国的 Victor Yodaiken 和 Michael Barabanov 等人最早运用在原有内核的底层加载实时内核的方法实现了 Linux 实时改造,开发出 RT-Linux 实时系统,其实时性能达到微秒级,符合硬实时要求[1]。后来 Paolo Mantegazza 又在此基础上设计了实时性能更好的 RTAI[2]。RT-Linux 在操作系统之下实现了一个简单的实时核心,Linux 本身作为一个可抢占的任务在核内运行,其优先级最低,随时会被高优先级任务抢占,其支持基于优先级的抢占式调度和 EDF 调度,对实时周期性进程的调度采用单调率算法。但这类双内核实时系统由于不能直接利用 Linux API,其在 PC 机领域的应用前景受到限制,同时这类实时操作系统的可靠性、容错性和可维护性

基金项目:国家 863 高技术研究发展项目(编号:2002AA742044)
作者简介:毛佳,博士生,主要研究方向为实时操作系统和嵌入式软件。苑森淼,教授,博士生导师,主要研究方向为智能网络、数据挖掘和实时嵌入式系统。

还有待提高。另一种改进方法是以 uClinux 为代表的嵌入式 Linux，有着极其广阔的应用前景[3]，嵌入式 Linux 本身并不支持实时调度，可以通过使用 RT-Linux 的 patch 增强实时性能。目前嵌入式 Linux 在实现过程中，主要研究方向有中断延迟、任务切换时间、最小内存开销和进程调度算法等，其中进程调度算法是影响系统实时性能的重要因素，其改进可以由重新编译 Linux 内核来实现。该文将在分析 Linux 系统的进程调度算法的基础上，提出一种改进后的调度算法，并通过实验验证此算法的有效性和优异性。

1 Linux 操作系统进程调度算法简介

Linux 一共有三类可执行进程：

(1)SCHED_OTHER：是普通的非实时进程，进程调度采用基于动态优先级的先进先出(FIFO)调度策略，系统选择优先级最高的进程来运行。非实时进程在 CPU 运行时，可以被高优先级的进程抢占。只要 CPU 中有实时进程在运行，则任何 SCHED_OTHER 进程都不能运行。

(2)SCHED_FIFO：是一种实时进程，遵守 POSIX1.b 标准的 FIFO 调度规则。它一直运行，直到主动释放 CPU，或者是 CPU 被另一个具有更高优先级(rt_priority)的进程抢占。在 Linux 实现中，SCHED_FIFO 进程只有当时间片用完时才会被迫释放 CPU。

(3)SCHED_RR：也是一种实时进程，遵守 POSIX1.b 标准的循环(round-robin，RR)调度规则，其调度策略与 SCHED_FIFO 类似。当 SCHED_RR 进程的时间片用完后，就被放到 SCHED_FIFO 和 SCHED_RR 队列的末尾，让具有相同优先级的其他就绪进程先运行。

Linux 提供了两种实时进程调度策略，即基于优先级的选择可抢占式调度和固定时间片轮转调度(Time-slice Round-Robin，TRR)。在将 Linux 改造成实时系统的过程中，其调度算法应考虑以下问题：

(1)Linux 在用户态支持可抢占调度，而在核心态却不支持抢占式调度。这不符合实时系统的要求，并且容易引起进程的优先级反转。实时系统是允许高优先级的进程(如系统调用)抢占 CPU 的，改进后的调度算法应无条件支持可抢占式调度，取消原有调度机制中的进程在用户态和核心态的限制。

(2)硬实时系统是不支持时间片轮转调度策略的。在不改变时间片长度的前提下，改进后的 Linux 实时系统将是软实时系统。为使下面的对比实验具备可比性，仍将沿用 Linux 原有的时间片机制。

(3)Linux 调度算法实际上是由 UNIX 调度算法继承而来的，其建立等待进程优先级的主要依据是进程所拥有的价值(反映进程的重要性)，然后系统选择优先级最高的进程进入 CPU 运行。在实时系统中应采用调度效率更高的优化算法。

2 优化改进调度算法

2.1 建立模型

作为实时系统调度算法应综合考虑进程的价值和截止期两个概念，以保证实时进程在截止期内尽可能多地完成。因此该文提出新的调度算法中实时进程的优先级数计算公式为：

$$v_i=w_i+\frac{p_i}{d-T_i}*k \qquad (1)$$

公式中各参数定义：v 表示进程的优先级数；w 表示该进程的价值(代表进程的重要程度)；P 表示进程的估计执行时间；d 表示绝对截止期；T 表示该进程提交时间；$(d-T)$ 是相对截止期，即该进程最大容忍的等待时间，在相对截止期内，这个进程就应该被执行，否则这个进程夭折；$P/(d-T)$ 这里称之为紧迫度，即这个值越大，说明从时间上看这个任务越紧迫；k 是系数因子。

改进后的调度算法(以下称为优化后调度算法)仍以进程的价值为基础，同时也关注了完成进程的紧迫度，对于优先级相同的进程，采用 FIFO 调度策略。进程的价值越大说明该进程越重要，CPU 越应完成它，可是对一些价值和它相差不多，而紧迫度要比它大得多的进程来说，就不公平了。例如，有两个进程 A、B 同时提交，A 的价值是 1001，估计执行时间是 1ms，相对截止期是 5ms，B 的价值是 1000，估计执行时间是 1ms，相对截止期是 2ms，假设这里的系数因子 k 是 10，则更应该执行进程 B。这是因为进程 A、B 的价值相近，而 B 的紧迫度要比 A 的大一些，A 的优先级=1001+1/5*10=1003，B 的优先级=1000+1/2*10=1005，因此笔者选择 B 先运行，以防止 B 的夭折(注：Linux 中实时进程的价值设为从 1000 到 1099，非实时进程的价值设为从 1 到 99，因此选择系数因子 k 为 10)。

优化后的调度算法依然采用时间片轮转策略，依照 Linux，分配给进程的时间片为 20 次时钟滴答，也就是 200ms=20*10ms。

2.2 结构定义

该文只给出实时进程的结构定义，非实时进程依然采用原有的动态优先级调度策略，其结构定义略去。

```
# define SCHED_RR
typedef struct TaskNode{
  int dtime;//进程的截止期
  int T;//进程提交时间
  int ptime;//进程尚未完成时间，初值等于任务的执行时间估计
  int flag;//其值为 1 时说明是实时进程，为 0 时说明是非实时进程
  int v;//进程的优先级数
  int w;//进程的价值
  struct TaskNode *next;
  }
  TaskNode *prior,*next;
```

2.3 链表定义

整个调度算法可以用双链表来描述。其中实时进程就绪等待队列用一个带空的头节点(从1)循环单链表就可以完成。这个链表分为两部分，即两级队列，分别用两个指针指向。最初，这两部分的头指针都指向 0 空的头节点，表明这两个队列均为空。

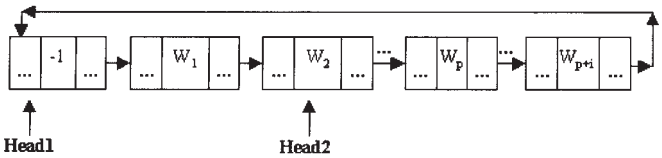
开始时的一级队列是新到来的比当前运行进程的优先级低的实时进程。有新的进程到来，它与当前运行的进程比较优先级数，若比其大则抢占 CPU，否则根据其优先级在就绪链表的第一级队列排序。

如果由于时间片到或被更高任务抢占的进程，根据它的优先级将其插入到第二级队列。

若当前进程的时间片到时，CPU 便在就绪等待链表中选择第二个节点的进程来判断(第一个是空的头节点)，如果它的紧迫度大于 1 的话，说明这个进程在规定时间内不能完成，它必

须夭折，此时再选择链表的第三个节点判断，如果紧迫度小于等于 1 便运行它。

当一级队列为空时，二级队列便升成一级队列，而此后由于时间片到或被更高任务抢占的进程则变成二级队列。



注：这里的 $W_1>W_2>...>W_p...>W_{p+i}$

图 1　优化后调度算法的实时进程优先级表

图 1 是实时进程的等待链表，分为二级。Head1 指向等待队列的空的头节点，Head2 指向该等待队列的一级队列的尾节点，即从 Head2 节点后便是二级队列。其中 pnew 指向新来的实时进程，pnow 指向当前运行的实时进程。

非实时进程的调度也通过单链表来实现。如果新来的进程属非实时进程则根据优先级高低插入非实时队列。只有实时队列（一级和二级队列）为空时，非实时队列才能被调度。

2.4　实时进程的调度策略算法描述

2.4.1　实时就绪队列的初始化

```
#define LEN sizeof(TaskNode)
TaskNode*creat_new_line()   /* 创建空链表 */
{TaskNode*head;
     head=(TaskNode*)malloc(LEN);
     head->w=-1;
     Head1=Head2=head;
     Head1->next=Head2->next=head;
     return(head);
}
```

2.4.2　实时进程接收策略

```
#define K 10
void*pnow;
if((*pnew).flag==1)//新来的进程是实时进程
{
if((*pnow).flag==1)//当前运行的进程是实时进程
{w=(*pnew).v+((*pnew).ptime)/((*pnew).dtime-(*pnew).T))*K;
if(h>(*pnow).v+((*pnow).ptime/((*pnow).dtime-(*pnow).T))*K
 {
  move_last_runqueue(Head2,pnow);//将当前运行的进程插入二级队列
   pnow=pnew;//新来的进程抢占 CPU
  }
 else//否则将新来的进程插入到等待链表的一级队列
   move_last_runqueue(Head1,pnow);
 }
else//当前运行的进程是非实时进程
move_last(pnow);//将当前进程插入非实时队列（非实时进程的插入算法 move_last 略）
}
else//新来的进程是非实时进程
   move_last(pnew);//将新来的进程插入非实时队列（非实时进程的插入算法略）
```

2.4.3　实时进程插入策略

```
move_last_runqueue(h,p)
TaskNode*h,*p;
{
TaskNode*p1,*p2,*t;
p1=h;
p2=p1->next;
if(h==Head1)//插入一级队列
{
while((*p).w<=(*p2).w&&p1!=Head2)//该处等于说明了优先级相同的进程采用 FIFO 调度策略
 {
  p1=p2;
  p2=(*p2).next;
 }
 (*p).next=p2;
 (*p1).next=p;
if(p1==Head2)//该节点插入到一级队列的末尾，则该节点便成了一级队列的末尾
  Head2=p;
}
else
{while((*p).w<=(*p2).w)//插入二级队列
  {
   p1=p2;
   p2=(*p2).next;
  }
  (*p).next=p2;
  (*p1).next=p;
}
```

说明：根据传来的 h 值，决定在一级队列（h 的值是 Head1 时）还是在二级队列（h 的值是 Head2 时）中查找插入的合适位置。

当 p 是新来的任务时，h 的值是 Head1，p 被插入一级队列。$p2$ 是 $p1$ 的后继节点。在循环体中，当新来的进程 p 高于一级队列的进程 $p2$ 时，停止循环，将 p 插入到 $p1$ 的后面；当 $p1$ 等于 Head2 时，说明一级队列的节点的优先级都比 p 节点的高，停止循环，把 p 插在 $p1$ 的后面，则该节点便成了一级队列的末尾。

当 p 是被抢占的任务时，h 的值是 Head2，p 被插入二级队列。在循环体中，当 p 的优先级高于二级队列的进程 $p2$ 时，停止循环，将 p 插入到 $p1$ 的后面；如果 p 高于二级队列的所有进程时，也会在 $p2$ 指向 Head1 时，因为 $(*p).w<=(*p2).w$ 而停止，则该节点便成了二级队列的末尾。因为 $(*Head1).w=-1$，而任何进程的优先级数都不会小于 0，因此当 $p1$，$p2$ 在这二级队列中遍历时，一定能有机会停止。

2.4.4　当前进程完成或时间片到时，调度等待链表中的一级队列中最前面的实时进程

```
PnowTime;//当前的时间
run_list(pnow)
TaskNode pnow;
{
if((*pnow).ptime!=0)//该进程未完成
  {if((*pnow).flag==1)
    if(PnowTime-(*pnow).T>=(*pnow).dtime)
      move_last_runqueue(Head2,pnow);//将未完成的进程插入到二级队列
      else//不能完成
        pnow 被夭折
  else
```

```
        move_last (now);//将未完成的非实时进程插入到非实时队列
      }
    p=get_node ();//从就绪链表中获得优先级最大的进程
    while ()
     {
      if (p!=NULL)
       {
         if ((*p).ptime<= (*p).dtime-PnowTime))//实时进程并且没超过截止期
             {pnow=p ;break ;}
         else//该进程已经不能完成，所以重新从就绪链表中获得优先级最大的进程
             p=get_node ();
       }
      else//实时就绪链表中无等待的进程
         调用非实时就绪队列；
     }
     return p；
   }
```

2.4.5 实时进程删除策略

```
  curtime 是当前的时间
  get_node ()
  {
    p= (*Head1).next；
    while ()
      if ((*Head1).next!= Head1)//有进程就绪等待
      {
        p= (*Head1).next；
        if ((*p).ptime> (*p).dtime-PnowTime)//进程未完成的时间大于相对截止期，该进程夭折。
          {
           (*Head1).next= (*p).next；
          if (p==Head2)//删除的节点是一级队列的尾节点时
            {
              Head2=Head3//二级队列荣升为一级队列
              Head3=Head2 ;//新二级队列为空
            }
          }
        else//p 进程可以在规定时间完成
          return p；
        }
      else//无进程等待
      return NULL；
  }
```

3 实验结果与讨论

为了验证该文所提出的调度算法，将两种算法进行了仿真对比实验。每次实验数据都是 50 个独立实验样本的运算结果的算术平均值，样本中相关参数定义如下：

(1)进程集共有 100 个进程，包括实时进程和非实时进程，在截止期内全部随机产生；

(2)进程集中实时进程产生的概率小于 0.1，非实时进程产生的概率大于 0.9；

(3)实时进程的价值由 1000 到 1099 之间随机选择，非实时进程的价值由 1 到 99 之间随机选择，都服从均匀分布；

(4)绝对截止期 d 为 18000 个时间单元；

(5)时间片为 200 个时间单元，每个时间片拥有 20 次时钟滴答；

(6)每次时针滴答产生进程的概率小于 0.005；

(7)负载系数 v 取值区间为 (0,4.8)，初始值 $v_0=0$，$v_{I+1}=0.2+v_I$；(I=0,1,2…24)；

(8)初次实验中每个进程的估计执行时间 P 是从最小估计执行时间 (1 次时钟滴答，10 个时间单元)到最大估计执行时间 (15 个时间片，300 个时间单元)之间的随机变量；

(9)第 I 次实验中每个进程的估计执行时间 P_I 是从最小估计执行时间 (仍为 10 个时间单元)到最大估计执行时间 (100+v_I*10 个时间单元)之间的随机变量；

(10)当系统严重过载时，实时调度算法的执行效率都很低，失去现实研究意义，这里不做讨论。

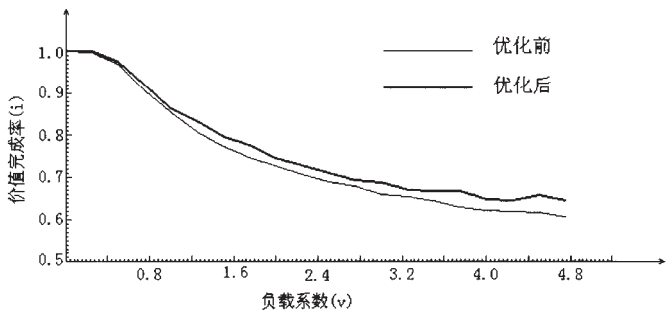


图 2 价值完成率

价值完成率 i 是第 I 次实验中截止期内执行完成的进程总价值与产生的进程总价值之比。从图 2 中可以看出，在 CPU 负载较低时 (即 v<1 时)，优化前、后的两条曲线基本一致，两种算法都表现出较好的性能；当 CPU 正常负载及负载稍高时，优化后的算法曲线下降速率明显低于优化前，其价值完成率仍高于优化前的算法 6%左右。通过对比曲线说明，优化后的实时调度算法在 CPU 正常负载及负载稍高时，具有更好的性能。原因是 CPU 负载正常时，优化前的实时调度算法只以进程的价值来决定优先级数，未充分考虑到完成进程的截止期，容易使价值稍低但紧迫度较高的进程长期等待而得不到执行。优化后的算法则有效防止了这一情况的发生。

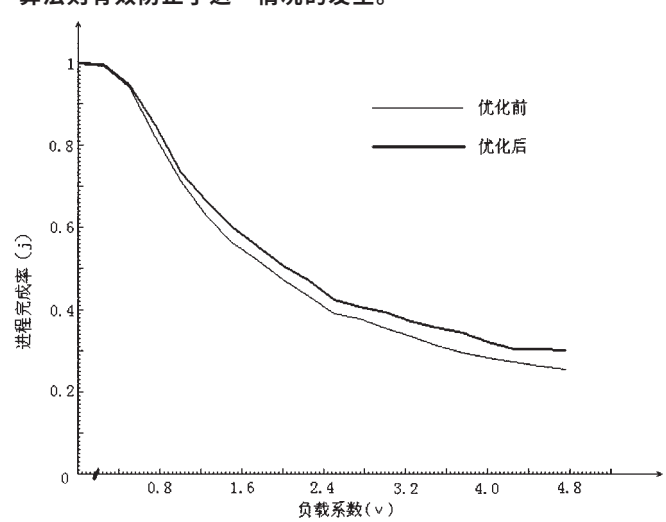


图 3 进程完成率

进程完成率 j 就是指截止期内已完成的进程数与产生的总进程数之比。通过图 3 可以看出，优化后的算法在 CPU 正常

(下转 207 页)

图2 轴心轨迹监测运行界面

```
lc.pszText=str.GetBuffer(str.GetLength());
m_List.InsertColumn(i,&lc);
//……其它操作,如调用GetFieldCount(),GetFieldValue()等函数来显示和操作各列数据
m_set->close();//关闭记录集
m_pCon->close();//关闭链接
}
```

轴心轨迹监测运行界面如图2(示例数据)。

5 结语

ADO的底层是OLE DB,这是现在最快速的数据库访问中间层。ADO类库可以完成大部分数据库操作,使得程序员可以从具体的DBMS中解脱出来,减少了应用程序开发的工作量。在轧机监测系统中使用ADO技术不仅可以满足系统对数据访问速度、数据交换等方面的要求,它还实现了登录用户名、口令的非硬代码方式,使得登录用户名、口令不必用硬代码方式写进程序,提高了数据库的安全性。(收稿日期:2002年7月)

参考文献

1.胡峪,刘静.VC++高级编程技巧与示例[M].西安:西安电子科技大学出版社,2001
2.虞和济,寇惠,原培新.轧钢机状态监测与故障诊断[M].北京:冶金工业出版社,1993

(上接115页)

负载及负载稍高时相比优化前具有较高的进程完成率。对实时系统而言,时间性是最重要的。人们不但要考虑系统的响应速度(即从某个事件发生到系统对此做出反应开始执行有关进程之间所需的时间),而且还要考虑到有关进程能否在截止期内完成。通常PC机系统中非实时进程在数量上要远远大于实时进程,如出现I/O阻塞等引起的中断(或异常),就会使正在运行的非实时进程被频繁调度而始终无法完成,这也是人们不希望发生的情况。CPU执行速度的提高并不能从根本上解决以上问题。在优化后的实时调度算法中,注意到对进程执行时间的估计,可以减少出现上述问题的几率,避免更多的低优先级进程无法运行。

4 结论

针对目前Linux实时系统调度算法中仅用进程的价值来确定优先级的现象,该文提出了综合考虑进程的重要性和紧迫度来决定优先级的调度算法。算法将进程的截止期和价值两个不相关的概念,通过公式(1)结合在一起,用来计算就绪等待队列中进程的优先级数。该算法通过双链表来实现。该文进行了优化前后两种算法的仿真实验,实验结果表明,优化后的算法在价值完成率和进程完成率两个调度效率评价指标上都好于优化前的算法,在CPU正常负载的情况下,优化后的调度算法体现了更优的实时性能。(收稿日期:2003年4月)

参考文献

1.http://www.rtlinux.com
2.http://www.rati.org/
3.http://www.uclinux.org/
4.Epplin J.Linux as an Embedded Operating System.http://www.embedded.com/97/fe39710.htm
5.Liu C L,Layland J W.Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment[J].Journal of ACM,1973,20:46~61
6.Cecilia Ekelin,Jan Jonsson.Real-Time System Constrains:Where do They Come From and Where do They Go?[C].In:Proceedings of the Int'l Workshop on Real-Time Constrains,1999,10:53~57
7.Giorgio C Buttazzo,Fabrizio Sensini.Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Hard Real-Time Environments[J].Proceedings of IEEE Transactions on Computers,1999,48(10):1035~1051
8.毛德操,胡希明.Linux内核源代码情景分析[M].杭州:浙江大学出版社,2001
9.Scott Maxwell.Linux Core Kernel Commentary[M].北京:机械工业出版社,2000

(上接149页)

8.J Broch et al.A Performance Comparison of Multi-hop Wireless Ad Hoc Networks[C].In:Proceedings of the 4th Int Conference on Mobile Conmputing and Networking,1998-10:85~97
9.Lei Wang,Lianfang Zhang,Yantai Shu et al.Multipath Source Routing in Wireless Ad Hoc Networks[C].In:Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering CCECE,Halifax,Nova Scotia,Canada,2000-05:479~483
10.P Johansson et al.Scenario-based Performance Analysis of Routing Protocols for Mobile Ad Hoc Networks[C].In:Proceeding of the ACM/IEEE International Conference on Mobile Computing and networking,1999-08:195~206
11.S Corson,J Macker.Mobile Ad Hoc Networking(MANET):Routing Protocol Performance Issues and Evaluation Considerations[S].RFC 2501,1999-01
12.The CMU Monarch Project's Wireless and Mobility Enhancements to ns.http://www.monarch.cs.cmu.edu
13.Ya Xu,John heidemann,Deborah Eatrin.Adaptive energy_conserving routing for multihop ad hoc networks[R].USC/ISI Research Report 527,2000-10