第15卷 第1期 2015年1月
1671—1815(2015)01-0078-06

科 学 技 术 与 工 程
Science Technology and Engineering

Vol. 15 No. 1 Jan. 2015

计算机技术

基于进程执行时间的多级反馈队列调度算法

邱相存 臧 洌 杨 丹 董国良
(南京航空航天大学计算机科学与技术学院,南京 210016)

摘 要 针对多级反馈队列(MLFQ)调度算法在时间片大小选取上对系统性能的关键影响,提出了一种基于进程执行时间的多级反馈队列调度算法。算法结合动态时间量子思想,通过进程的执行时间动态确定队列以及时间片来完成调度。同时为了减少队列进程的切换次数,利用中位数的方法来决定时间片的大小。实验结果表明,与传统的多级反馈队列调度算法相比,改进的算法不仅缩短了进程的平均周转时间和平均等待时间,也减少了进程切换次数,为操作系统领域处理机调度智能化提供了有效的参考价值。

关键词 多级反馈队列调度 动态时间量子 中位数 智能化
中图法分类号 TP316; 文献标志码 A

在多道程序环境下,主存中有多个进程,其数目往往多于处理机数目。操作系统[1,2]通过处理机调度程序,按照某种调度算法动态地把处理机分配给就绪队列中的一个进程,使之执行。CPU调度[3]是重要的计算机资源,提高CPU的利用率及改善系统性能(吞吐量、响应时间),很大程度上取决于CPU调度性能的好坏,因此其调度算法设计是非常关键的。

分配进程的CPU执行算法需要考虑到系统进程的公平性,避免CPU空闲。通常调度算法的评判标准为:进程周转时间、响应时间、进程平均等待时间和进程上下文切换次数。到目前为止,各种调度技术已经逐步成熟,它们都有自己的优点和缺点。先来先服务(FCFS)调度算法[4,5]是根据进程到达就绪队列的先后顺序依次进行调度,它是一个非优先权的调度算法,系统操作简单,开销小,一旦进程获得CPU的执行,一直运行该进程到结束或者I/O中断请求;其缺点就是进程的平均等待时间较长,因此不适合实时性要求严格的系统。短作业(进程)优先[SJ(P)F]调度算法[6]是对短作业或者短进程优先调度的算法,它是从就绪队列中选出一个估计运行时间最短的进程,将处理机分配给它,并一直执行到完成或发生某事件而被阻塞放弃处理机。短进程优先调度算法问题是它需要事先知道该进程精确的执行时间,而这是不可预测的。最高优先权优先(FPF)调度算法[7]是把处理机分配给就绪队列中优先权最高的进程,影响其性能的关键因素就是优先级的确定。轮转法(RR)调度算法是系统将所有的就绪进程按先来先服务的原则排成一个队列,每次调度时,把CPU分配给队首进程,并令其执行一个时间片。时间片的大小从几毫秒到几百毫秒。当执行的时间片用完时,由一个计时器发出时钟中断请求,调度程序便据此信号来停止该进程的执行;并将它送往就绪队列的末尾;然后,再把处理机分配给就绪队列中新的队首进程,同时也让它执行一个时间片。这样就可以保证就绪队列中的所有进程在给定的时间内均能获得一时间片的处理机执行时间。轮转法调度关键是时间片的选取,过长使得每个进程都能在一个时间片完成,时间片轮转算法就退化为FCFS算法,无法满足交互式用户的需求;过短将有利于短作业,因为它能较快地完成;但会频繁地发生中断、进程上下文的切换,从而增加系统的开销。多级反馈队列(MLFQ)调度算法是所有就绪队列首先被分配到第一个队列中,而第一个队列根据每个进程的执行时间和时间片的大小依次执行。若进程的执行时间多于时间片的大小,则这些进程将被送到第二个队列中,这样依次循环,直到完成;这种调度算法关键在于队列个数和时间片的大小确定,传统的方法不能动态地反映当前进程的特点,导致出现

2014年8月8日收到
第一作者简介:邱相存,男,硕士研究生。研究方向:软件工程。E-mail:qxc.0211@163.com。

CPU 的利用率降低、进程的平均等待时间激增、上下文切换次数增加和系统的吞吐量较低等现象,从而增加系统开销,影响系统的交互操作。本文利用动态时间量子的方法,根据就绪队列中的进程执行时间[8]动态确定时间片的大小,提出了一种智能的多级反馈队列调度算法——基于进程执行时间的多级反馈队列调度算法。实验表明,该算法减少了系统进程的平均周转时间、平均等待时间和进程间的切换次数,提高了系统吞吐量。

1　多级反馈队列

多级反馈队列(MLFQ)调度算法[9,10]结合了先来先服务(FCFS)、轮转调度法(RR)、优先级算法和短作业优先(SJF)算法。该算法有多个队列,同一个队列中的进程优先级相同,不同队列中的进程优先级不同; 最高优先级上的进程运行一个时间片,次高级上的进程运行两个时间片,再下一级运行四个时间片,以此类推; 每当一个进程在一个优先级队列中用完它的时间片后,就下移一级,进入另一个队列; 在低优先级队列中等待时间过长的进程,将被移入高优先级队列; 调度程序将该进程从等待队列中释放后提高其优先级。多级反馈队列调度算法如图 1 所示[11,14]。由于基于多级反馈队列的调度算法能够较好地满足全部进程的 CPU 运行时间,因此该方法具有很大的潜力。

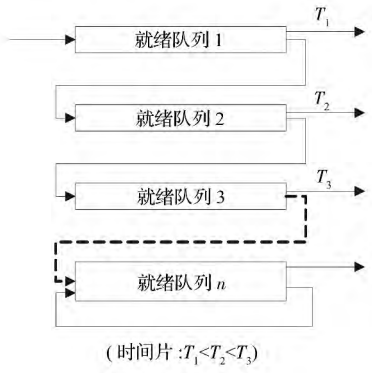


图 1　多级反馈队列调度算法

Fig. 1　Multi-level feedback queue scheduling algorithm

多级反馈队列调度算法具有更好的性能,能更好地满足各种类型用户的需要。

对终端型作业用户,由于其所提交的作业大多属于交互作业,作业通常较小,系统只要能使这些作业(进程)在第一队列所规定的时间片内完成,便可使终端型作业用户都感到满意。

对短批处理作业用户,很短的批处理型作业,开始时像终端型作业一样; 如果仅在第一队列中执行一个时间片即可完成,便可获得与终端型作业一样的响应时间。稍长的作业,通常也只需要在第二队列和第三队列各执行一个时间片即可完成,其周转时间仍然较短。

对长批处理作业用户,长作业将依次在第 1,2,…,n 队列中运行,然后再按轮转方式运行,用户不必担心其作业长期得不到处理。

多级反馈队列调度算法需要考虑队列的个数和每个队列的时间片的选取。

2　基于进程执行时间的多级反馈队列调度算法

时间片的大小利用动态时间量子方法来确定。队列优先级逐级降低,开始所有的进程被调度到第一个队列中,若其未完成,则被调度到第二个队列中; 同时进程的执行时间也得到更新,依次循环,当进程被放至最低级队列后,对所有的进程根据其剩余执行时间进行升序排列,然后将这些进程调度到不同的队列中。其中进程的剩余执行时间越少,其优先级越高。

2.1　动态时间片

当进程初次执行时,解析该进程来获取其执行时间,这种解析仅仅需要执行一次,除非进程改变或者得到了修改。通过解析获得进程的执行时间,利用进程的执行时间来确定当前队列的时间片。

具体的方法为: 初始的时候操作系统给队列分配一个默认的时间片; 通过解析获得进程的执行时间后,动态调节时间片大小; 当一个新的进程到达就绪队列时,操作系统先检查其状态标识位。状态标识位若为 0,则表示该进程第一次进入系统中执行或虽然执行过,但是已经得到了修改,这时操作系统分配一个计数器来统计该进程的执行时间,然后该队列继续使用当前队列的时间片运行。状态标识位若为 1,则表示该进程不是新来的进程,操作系统根据该队列的每个进程剩余执行时间,重新计算当前队列的时间片。

中位数是指将统计总体当中的各个变量值按大小顺序排列起来,形成一个数列,处于变量数列中间位置的变量值就称为中位数。当变量值的项数 N 为奇数时,处于中间位置的变量值即为中位数; 当 N 为偶数时,中位数则为处于中间位置的两个变量值的平均数。一个数集中最多有一半的数值小于中位数,也最多有一半的数值大于中位数。中位数是以它在所有标志值中所处的位置确定的全体单位标志值的代表值,不受分布数列的极大或极小值影响 。

鉴于中位数的上述特点,时间片计算方法采用

中位数的方法[18]，如果在就绪队列中的进程执行时间的中位数小于 Q_0，则为了减少上下文切换次数，时间片的大小就设置为 Q_0（其中 Q_0 为操作系统根据用户行为获取的）。公式（1）为计算时间片的公式。

$$Q = \bar{x} \equiv \begin{cases} Y_{(N+1)/2}, & N \text{ 为奇数} \\ \dfrac{Y_{(N/2)} + Y_{(1+N/2)}}{2}, & N \text{ 为偶数} \end{cases} \quad (1)$$

式（1）中，Y 是进程按升序排列后位于中间的数值；N 为当前队列中的进程个数；Q_0 为系统默认时间片大小。

对公式（1），可以用区间的方式表示，如公式（2）所示。

$$Q = \begin{cases} \bar{x}, & \bar{x} \geqslant Q_0 \\ Q_0, & \bar{x} < Q_0 \end{cases} \quad (2)$$

从公式（1）和公式（2）可知，系统中的进程经过第一级队列执行后，有近 50% 的进程执行结束；经过第二级队列执行后，余下进程中的近 50% 的进程执行结束。重复执行，经过最多六级队列执行，系统中的进程几乎执行完毕。每一级队列调度后，进程剩余数量百分比如图 2 所示。

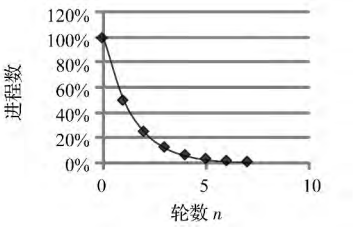


图 2 队列进程的数量

Fig. 2 The number of queue processes

由于每级队列进程数目越少，进程间的切换次数就越少，一定程度上减少了系统调度的额外开销。进程的上下文切换次数可用公式（3）计算。

$$Q_T = \left(\sum_{i=1}^{n} k_i\right) - 1 \quad (3)$$

式（3）中 Q_T 是总共的上下文切换次数，n 是队列的个数，k_i 是每个队列中的进程个数。

在其他的时间片确定调度算法中，即使当就绪队列中只有一个进程，一旦系统分配的时间片用尽，进程仍需被中止，这就增加了额外的进程切换次数。然而，改进的多级反馈队列调度算法中，这种情况是不会发生的，因为在这种情况下，当前队列的时间片就是该进程的剩余执行时间。

基于动态时间片的调度算法具有进程响应时间快和交换次数少等优点，同时还能对系统进程运行时间精确预估。采用中位数确定时间片大小根据就绪队列中进程的执行时间情况实时确定当前队列时间片的值，避免了当队列中仅有一个进程存在时，由于时间片的值小于进程的剩余执行时间，而产生不必要的进程切换情况。

2.2 算法描述

提出的基于进程执行时间的多级反馈队列调度算法采用动态时间片方法，首先通过默认的时间片大小来执行初次进入系统的进程，利用计时器解析出该进程的执行时间；其次，根据队列中进程的执行情况，队列的个数从无依次创建，直到所有进程都得到执行（根据动态时间片方法最终的队列个数不大于 6）。当队列个数大于 5 时，把未完成的进程重新分配到前面创建的队列中调度。

算法步骤描述如下（假设就绪队列中进程个数为 n）：

步骤 1 对进入就绪队列的进程分别进行执行时间的分析。通过检查进程状态标识（1 或 0），采用系统默认时间片大小，分配计数器得到 n 个进程的预估执行时间。

步骤 2 利用中位数方法计算每个队列时间片大小 Q_i。在第一级队列中，对就绪队列中的 n 个进程，依次分配 CPU 执行且时间片大小为 Q_1；若有未完成的进程，则系统创建第二级队列，同样依次分配 CPU 执行且时间片大小为 Q_2。依次类推，直到所有进程都执行完毕。若队列个数大于 5 时，把就绪队列中的进程升序排列后，重新分配调度。

每个队列进程算法流程如下：

```
1. while ( ready queue ! = null)
2. if new process P_i arrived then
      //check P_i status
3. if P_i status = 0 then assign new counter C_i for this process
4. end if
5end if
      //calculate new quantum
6. Q_i = median ( remaining burst time of process in ready queue with
status = 1)
7. if ( Q_i < Q_0)
8. Q_i = Q_0
9. end if
      //continuing executing the processes using new quantum Q_i
10. for i = 1 to n loop
11. assign the CPU to process P_i and give Q_i
12. if P_i terminated normally and P_i status = 0 then save C_i( P_i)
13. let P_i status = 1
14. end if
15. end for
16. end while
```

2.3 算法分析

相对于基于传统时间片的多级反馈队列调度算法,基于进程执行时间的多级反馈队列调度算法体现的优势如下:

1) 传统的时间片确定方法(如 EMLFQ 和 Power MLFQ)[16] 未考虑进程的特征;若进程执行时间在时间片的取值周围分布不均匀时,会导致大量的进程长久存活在多级队列中,不仅增加了进程的周转时间,而且还产生了大量的进程切换次数。而基于执行时间的多级反馈队列调度算法,首先预估就绪队列的执行时间,再通过中位数的方法确定时间片的值,这样每经过一次执行有近一半的进程执行结束离开队列,降低了进程的平均周转时间,减少了进程间的切换次数。

2) 对于就绪队列中进程个数为 n 的数据集。传统的时间片确定方法(如 EMLFQ 和 Power MLFQ),由于其是固定时间片大小,假设其进程全部完成需要 L_1 级队列,因此时间复杂度是 $O(L_1)$。而对于本文提出的算法解决预估进程的执行时间的时间复杂度为 $O(n)$。而计算每个队列时间片的时间复杂度为 $O(n\lg n)$。假设在这种情况下其进程完成需要 L_2 级队列,我们知道 $L_2 \leqslant L_1$,构造队列的个数在减少,从而减少进程切换次数,缩减 CPU 的开销。

3　实验和结果分析

为了验证改进算法的有效性,选取了几组不同执行时间和到达时间的进程,通过对比多级反馈队列采用固定时间片方法(EMLFQ)以及多级反馈队列采用成倍增加时间片方法(Power MLFQ),用进度表的方式模拟它们的执行情况。从平均周转时间和平均等待时间[17] 两个方面来考察实验结果。

平均周转时间 TAT 如公式(4)所示。

$$TAT = \frac{1}{n}\left[\sum_{i=1}^{n} T_i\right] \tag{4}$$

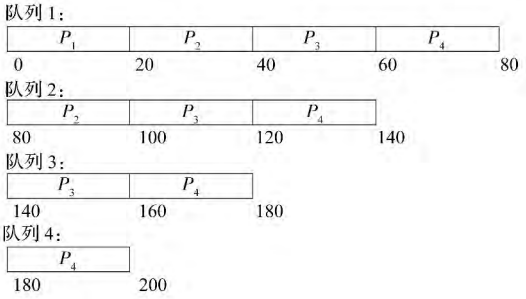
平均等待时间 AWT 如公式(5)所示。

$$AWT = \frac{1}{n}\left[\sum_{i=1}^{n} t_i\right] \tag{5}$$

式中 n 为进程个数,T_i 为每个进程完成时间和到达时间之差,t_i 为每个进程的等待时间。

样例 1: 假设有 4 个进程,其到达时间都是 0,执行时间为 $P_1 = 20$,$P_2 = 40$,$P_3 = 60$,$P_4 = 80$。(系统默认时间片大小为 $Q_0 = 25$)

对于 EMLFQ($Q = 20$)算法,其进程的进度表流程如下。



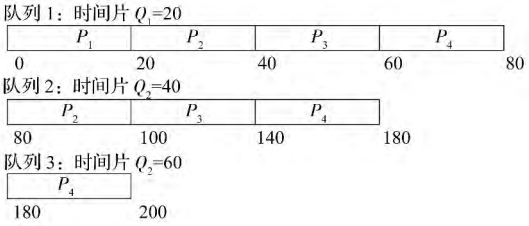
该方法进程的平均周转时间和平均等待时间为:

$$TAT = (20 + 100 + 160 + 200)/4 = 120;$$

$$AWT = (0 + 60 + 100 + 120)/4 = 70;$$

$$Q_T = (4 + 3 + 2 + 1) - 1 = 9。$$

对于 Power MLFQ($Q = 20$)算法,其进程的进度表流程如下。



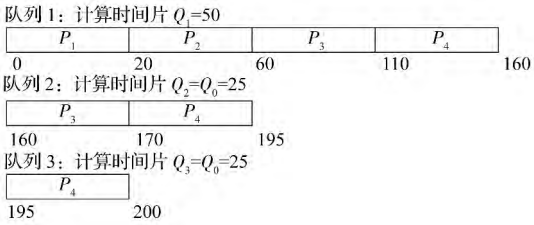
该方法进程的平均周转时间和平均等待时间为:

$$TAT = (20 + 100 + 140 + 200)/4 = 115;$$

$$AWT = (0 + 60 + 80 + 120)/4 = 65;$$

$$Q_T = (4 + 3 + 1) - 1 = 7。$$

对于本文改进的 MLFQ 算法,其进程的进度表流程如下。



该方法进程的平均周转时间和平均等待时间为:

$$TAT = (20 + 60 + 170 + 200)/4 = 112.5;$$

$$AWT = (0 + 20 + 110 + 120)/4 = 62.5;$$

$$Q_T = (4 + 2 + 1) - 1 = 6。$$

表 1 是样例一在 EMLFQ、Power MLFQ 和改进 MLFQ 算法上得出的比较值。

样例二: 假设 4 个进程的到达时间分别为 0,4,8,16,而执行时间 $P_1 = 18$,$P_2 = 22$,$P_3 = 70$,$P_4 = 74$。(系统默认时间片大小为 $Q_0 = 25$)

表 2 是样例二在三种方法上得出的比较值。

表 1 样例一的三种方法对比

Table 1 Comparison of the three methods in sample one

	EMLFQ ($Q=20$)	PowerMLFQ ($Q=20$)	改进 MLFQ
时间片	20	20,40,60	50,25,25
平均周转时间	120	115	112.5
平均等待时间	70	65	62.5
切换次数	9	7	6

表 2 样例二的三种方法对比

Table 2 Comparison of the three methods in sample two

	EMLFQ ($Q=20$)	PowerMLFQ ($Q=20$)	改进 MLFQ
时间片	20	20,40,80	(25,70),25
平均周转时间	106	81	81
平均等待时间	60	60	35
切换次数	10	8	4

样例三: 假设 4 个进程的到达时间分别为 0,6,13,21,而执行时间 $P_1=10$,$P_2=14$,$P_3=70$,$P_4=120$。(系统默认时间片大小为 $Q_0=25$)

表 3 是样例三在三种方法上得出的比较值。

表 3 样例三的三种方法对比

Table 3 Comparison of the three methods in sample three

	EMLFQ ($Q=20$)	PowerMLFQ ($Q=20$)	改进 MLFQ
时间片	20	20,40,80	(25,25,95),25
平均周转时间	90.5	90.5	75.5
平均等待时间	37	37	22
切换次数	11	7	4

从表 1、表 2 和表 3 对比中,可以明显看出基于进程执行时间的多级反馈调度算法与固定时间片多级反馈队列(EMLFQ)调度算法和时间片大小成倍增长的多级反馈队列(PMLFQ)调度算法相比具有很大的优势,尤其是在减少进程间的切换次数、周转时间和等待时间方面。改进的方法能根据进程的情况动态调整时间片的大小,保证了 CPU 的高效工作,提高系统的吞吐量。图 3、图 4 和图 5 展示了上述三个样例中三种方法在平均周转时间、平均等待时间和交换次数上差异情况对比。

综合平均周转时间、平均等待时间和进程切换次数三方面,可以看到基于进程执行时间的多级反馈队列调度算法在操作系统领域中 CPU 处理机调度提供了有效的参考价值。

4 结语

鉴于动态时间片在进程响应时间和交换次数上的优势,并对系统进程运行时间进行精确预估,提出了基于进程执行时间的多级反馈队列调度算法,该

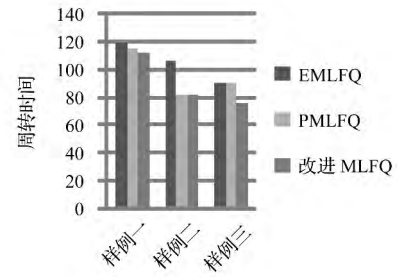


图 3 周转时间对比

Fig. 3 Comparison of turnaround time

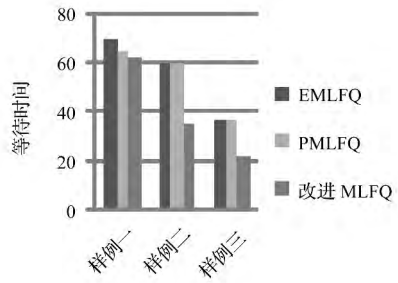


图 4 等待时间对比

Fig. 4 Comparison of waiting time

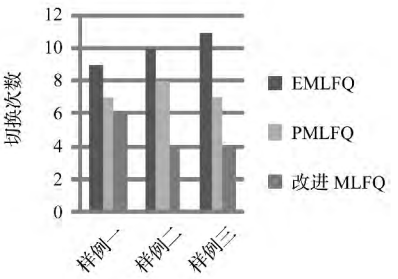


图 5 切换次数对比

Fig. 5 Comparison of switching times

算法根据队列中进程的执行时间实时动态地确定时间片大小,降低了进程的平均周转时间和平均等待时间,一定程度上减少了进程的切换次数。实验结果表明,改进的算法在性能上优于 EMLFQ 算法和 Power MLFQ 算法。另外,这种算法也为我们提供了一种动态时间片的思想,结合这种思想操作系统根据用户的行为来确定时间片而非人为指定。然而在时间复杂度上,本文提出的算法还有一些不足,下一步可以结合神经网络的思想,通过用户行为训练多级反馈队列每个队列的时间片,从而得出一个更加智能的、易学习的和适应性更好的操作系统调度算法。

参 考 文 献

1 Tanenbaum A S. Modern operating systems. Prentice Hall Press, 2007

2 Boyd-Wickizer S, Chen H, Chen R, *et al*. Corey: an operating system for many cores. OSDI, 2008

3 Soltesz S, Pötzl H, Fiuczynski M E, *et al*. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. ACM SIGOPS Operating Systems Review, ACM, 2007

4 Sharma R, Soni V K, Mishra M K, *et al*. An agent based dynamic resource scheduling model with FCFS-Job grouping strategy in grid computing. International Conference on Cluster and Grid Computing Systems (ICCGCS-2010), Italy, Rome 2010

5 Adan I, Weiss G. Exact FCFS matching rates for two infinite multitype sequences. Operations Research, 2012; 60(2): 475—489

6 **金瑛棋, 吴俊敏, 赵小雨. 公平性考虑的短作业优先内存调度策略. 计算机工程,** 2012; 38(20): 243—246
Jin Yingqi, Wu Junmin, Zhao Xiaoyu. Fairness-considered shortest job first strategy for memory scheduling. Computer Engineering, 2012; 38(20): 243—246

7 **涂　刚, 阳富民, 卢炎生. 基于动态优先级策略的最优软非周期任务调度算法. 计算机研究与发展,** 2004; 41(11): 2026—2034
Tu Gang, Yang Fumin, Lu Yansheng. An optimal scheduling algorithm for soft aperiodic tasks. Journal of Computer Researth and Development, 2004; 41(11): 2026—2034

8 **丛龙水. 动态优先级作业调度算法与实现. 计算机工程与应用,** 2013; 49(10): 267—270
Cong Longshui. Realization of dynamic priority scheduling algorithm. Computer Engineering and Applications. 2013; 49(10): 267—270

9 Parvar M R E, Parvar M E, Safari S. A starvation free IMLFQ scheduling algorithm based on neural network. International Journal of Computational Intelligence Research, 2008; 4(1): 27—36

10 Hoganson K. Reducing MLFQ scheduling starvation with feedback and exponential averaging. Journal of Computing Sciences in Colleges, 2009; 25(2): 196—202

11 **张尧学, 方存好, 王　勇. 非精确计算中基于反馈的 CPU 在线调度算法. 软件学报,** 2004; 15(04): 0616—0623
Zhang Yaoxue, Fang Cunhao, Wang Yong. A feedback-driven online scheduler for processes with imprecise computing. Journal of Software, 2004; 15(04): 0616—0623

12 **陈源龙, 马培军, 李　东. 硬实时系统中自适应反馈软件容错动态调度算法研究. 宇航学报,** 2010; 31(11): 2591—2596
Chen Yuanlong, Ma Peijun, Li Dong. Dynamic scheduling algorithm with adaptive feedback software fault-tolerance in hard real-time system. Journal of Astronautics, 2010; 31(11): 2591—2596

13 **秦承刚, 于　东, 吴文江. 一种面向数控系统的动态反馈调度模型. 小型微型计算机系统,** 2010; 31(4): 788—792
Qin Chenggang, Yu Dong, Wu Wenjiang. Dynamic feedback-based scheduling model for CNC system. Journal of Chinese Computer Systems, 2010; 31(4): 788—792

14 **金　宏, 王宏安, 傅　勇, 等. 模糊反馈控制实时调度算法. 软件学报,** 2004; 15(06): 0791—0798
Jin Hong, Wang Hongan, Fu Yong, *et al*. A fuzzy feedback control real-time scheduling algorithm. Journal of Software, 2004; 15(06): 0791—0798

15 Behera H S, Naik R K, Parida S. Improved multilevel feedback queue scheduling using dynamic time quantum and its performance analysis. International Journal of Computer Science and Information Technology (IJCSIT), 2012; 3(2): 3801—3807

16 EffatParvar M R, EffatParvar M, Haghighat A T, *et al*. An intelligent MLFQ scheduling algorithm (IMLFQ). PDPTA, 2006: 1033—1036

17 Kashif M, Helmy T, El-Sebakhy E. A priority-based MLFQ scheduler for CPU power saving. Computer Systems and Applications, IEEE International Conference on, IEEE, 2006: 130—134

MLFQ Scheduling Algorithm Based on Burst Time of the Now Running Process

QIU Xiang-cun, ZANG Lie, YANG Dan, DONG Guo-liang

(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, P. R. China)

[Abstract] The time quantum in Multi-level feedback queue has a great influence on the performance of operating system. To solve the problem, a self-adjustment time quantum in MLFQ was proposed based on burst time of the now running process. The idea of dynamic time quantum was combined with, and the queues and time quantum were dynamicly adjusted according to the now running process. To reduce the switch times of the processes, the median was used to determine the value of time quantum. The experimental results show that the proposed algorithm has lower average turnaround time and average waiting time than other traditional MLFQ and a certain reduction in the switch times of processes. It is valuable for intelligent scheduling algorithm in the field of operating system.

[Key words] multi-level feedback queue (MLFQ)　dynamic time quantum　median　intelligence