国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第14卷第4期 1992年12月 Vol. 14 No. 4

多机环境下系统程序并行化的实现方法*

姚益平 靳远宏 杨桃栏

（电子计算机系）

摘 要 本文给出了多处理机环境对系统程序的要求，阐述了系统程序并行化研制过程中遇到的问题和解决办法，给出了实现临界段互斥的三种处理方法及其性能评价。

关键词 多处理机系统，系统程序并行化，可再入，临界段互斥

分类号 TP311. 11

软件并行化是多处理机系统的关键技术。作者在多机环境下进行了并行化系统程序的研制工作。该多机系统包含多个中央处理机CPU，所有CPU共享一个通用全局存储器，每一个CPU有一个局部存储器LM，有各自的计算和控制部件，通过指令还可以访问系统中的共享数据；系统设有信号灯，并设置了“测试与置定”，“无条件置定”等硬件指令。系统还支持栈式动态存储分配。本文提出了多机环境下对系统程序的要求，概述了多处理机环境下遇到的问题和解决方法，给出了实现临界段互斥的几种处理方法，并提出了并行化系统程序设计的一般原则。

1 系统程序并行化所要解决的问题

对多机系统而言，系统程序要能被多个任务调用，必须是可再入的，它要求满足：(1) 程序模块本身在其执行过程中不能改变，即为纯代码；(2) 调用它的各程序应自带参数工作区。

第 (2) 点是容易实现的。因为每个CPU都有独立的寄存器，我们可以约定一个寄存器，调用程序申请一片栈式内存，存放要传送的参数，然后将该栈式内存的首地址送入约定寄存器中。系统程序将根据该寄存器中的地址获取相应的参数。

对第 (1) 点，由于每个CPU都有各自的计算和控制部件，除涉及内存存取部分外的代码都可视为纯代码，在一个CPU上运行时完全独立于其他CPU，不受多机环境的影响。而系统程序涉及的内存，可概括为下面的三种使用方式：

第一种：用CON定义的常数单元，所有程序只对它进行读而不写。显然，它是可再入的。如：

* 国家自然科学基金资助课题
1991年11月20日收稿

TEN CON 10 ；强置字界，将常数 10 装配于内存，TEN 为其地址。

第二种：用 CON 和 BSS [Z] 定义，程序并不使用它的初值，程序结束时其值也将不再被使用。我们不妨称之为临时工作单元。在程序中它们大都用作保护单元，临时变量或缓冲区等。如：

X CON O

Y BSS exp1；强置字界，分配 exp1 个相继内存，Y 为其首地址。

Z BSSZ exp2；含义同上，但产生的是 exp2 个 0 的字块。

第三种：用 CON 定义，程序将使用其初值，而且程序（别的子程序）中可能修改，程序结束后，其值仍将有用，即程序下一次进入后使用。我们称这类单元为共享单元，如：

LONG CON exp

对于临时工作单元，由于系统支持栈式存储分配，我们可以在程序中申请栈空间以取代之，在程序出口处释放该空间。

但不能用栈式分配法开辟共享单元，因为它们必须永久驻存，其初值和终值对当前及以后程序均有用。我们可视共享单元为临界资源（即一次仅允许一个进程使用的资源），并称对这些单元进行读写操作的代码段为临界段。有一类方法，将临界段代码分散在欲实现对共享资源进行操作的程序中完成。为解决读写或写操作冲突，多个程序必须互斥执行临界段。

实现临界段互斥的方法很多，其中最基本的一种方法就是提供一把锁和置锁、开锁的操作，按如下的步骤实现互斥：(1) 测试锁是否被置。(2) 如果锁被置，等待直到它被开锁后转 (3)，否则直接转 (3)。(3) 置锁并进入临界段。(4) 执行临界段。(5) 退出临界段时开锁。因为不被允许进入临界段的进程必须等待，所以应使临界段代码尽可能短。

2 临界段互斥的处理方法

我们知道，直接为应用程序服务的系统程序，其共享内容多为用户设置或利用省缺值（即用 CON 定义的值）。当共享单元被使用时，不失一般性，我们假设对共享单元内容的设置（写）与获取（读）的程序是分开的，对每个单元的设置也可以独立成一个子程序。这样，如共有 n 个共享单元 BS_1，BS_2，…，BS_n（显然，我们可以使它们分在连续的单元），则有 P_0，P_1，…，P_n 共 n+1 个程序，其中 P_0 完成获取工作，P_i (i=1，2，…，n) 完成对共享单元 BS_i 的设置，因而存在如图 1 所示的共享关系。

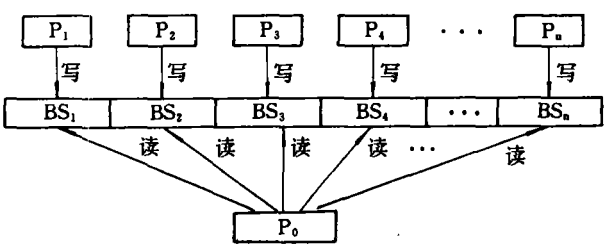

图 1 各单元间的共享关系

分析图 1，我们不难看出：在互斥使用共享单元（如 P_0 与 P_i，i=1，2，…，n）的同时，也存在可以并行的成分，如 P_i 与 P_j (i≠j，i，j 均为从 1 到 n 的整数)。因此有下列临界段互斥的处理方法的最佳目标，并将作为我们评价后述算法的主要依据：

(1) 对必须互斥使用临界段的进程保证互斥；(2) 保证能并行成分的完全并行；(3) 避免死锁；(4) 方法简单而且高效。

2.1 硬件指令方法

利用系统提供的信号灯及相应操作，尽量组织代码短、执行时间省的临界段，实现互斥地使用共享单元。

程序 P_i 对 BS_i (i=1，2，…，n) 的写总可以一次完成：先将要写的内容送入某个寄存器 S_i，然后将 S_i 的内容写入。程序 P_i 编写如下：

```
Pi：非 临 界 段          ；将要写入的内容送 Si.
    SM  1，  TS          ；测试信号灯 SM，当 SM 为 0 时指令流出，SM 置 1，否
                           则等待流出。
临界段：BSi，0   Si      ；(Si) →BSi
    SM       0           ；无条件置 SM 为 0.
    非 临 界 段
```

对程序 P_0，可以将读取共享单元的操作集中起来完成：分配给 P_0n 个局部存储器 LM_j，LM_{j+1}，…、LM_{j+n-1}，首先将共享单元的值读到 LM 中，要用到时再从 LM 相应的单元中取出就行了。这样，读取共享单元的内容到局部存储器中就成为临界段操作。可如下编程实现 P_0：

```
P0：非 临 界 段
    SM  1，  TS
临界段：A0          BS1     ；共享单元首地址送 A0.
        A1          n       ；共享单元个数送 A1.
        LM，j，A1   0，A0   ；将共享单元内容成组传送到 LM 中.
    SM              0
    非 临 界 段
```

通过分析，不难看到：

(1) 该方法保证了临界段的互斥使用。对 P_0 与 P_i (i=1，2，…，n)，当 P_0 在执行临界段时 SM=1，此时 P_i 必须等待直到 P_0 执行完临界段且置 SM 为 0 后才能继续执行。反之亦然。

(2) 由于充分利用硬件指令的特性，该方法可以避免死锁。

(3) 该方法思路清晰，逻辑简单，临界段执行时间很短，效率较高，有效地使用了系统提供的硬件指令。通过指令分析，得知：P_i (i=1，2，…，n) 的临界段执行时间仅 1 拍；而 P_0 的临界段执行时间最少时，仅需 $1+1+(15+[\frac{A_1}{2}])=17+[\frac{n}{2}]$ 拍。

(4) 上述方法不能保证能并行的进程的完全并行。例如，P_0 的进程 P_{01}和 P_{02}分别同时进入两个 CPU 运行，若 P_{01}首先测试 SM 为 0，指令流出并置 SM 为 1，转入临界段，此时 P_{02}也开始测试，须等待 P_{01}从临界段退出并置 SM 为 0 后才可执行临界段。不过，由于我们的临界段执行时间很短，因此这种停等的时间也是很短的。

2.2 转移临界段法

方法的指导思想是：利用系统提供的硬件指令，引入新的临界段来实现临界段互斥，并尽量减少能并行进程的停等时间。用类汇编语言描述如下：

设 A、B 为全局变量，初值都为 0，

程序 P_0：

```
L1      SM      1,TS
        A       A+1     ;(A)+1→A，表示 P0 访问临界段。
        JBZ     L2      ;若(B)=0，表示无 Pi 在使用临界段，流出。
临界段：A       A-1     ;若(B)≠0，表示有 Pi 在运行，循环等待。
        SM      0
        J       L1
L2      SM      0
        P0处理程序      ;包括读取共享单元内容。
        SM      1,  TS
临界段：A       A-1     ;(A)-1→A，表示 P0 执行结束.
        SM      0
```

程序 $P_i(i=1,2,\cdots,n)$：

```
L3  SM  1,TS
    B       B+1     ;(B)+1→B，表示 Pi 访问临界段。
    JAZ     L4      ;如(A)=0，表示无 P0 在执行，流出。
临界段：B   B-1     ;若(A)≠0，表示 P0 正在运行，循环等待。
    SM      0
    J       L3
L4      SM      0
        Pi处理程序
        SM      1,TS
临界段：B   B-1     ;(B)-1→B，表示 Pi 执行结束。
        SM      0
```

本方法思想新颖。当完成读写的程序某一方使用频率较大时，它能减少其并行时的停等时间，同时也能完全保证互斥，并避免死锁。

2.3 改进的 T. Dekker 算法

我们分析并改进了能实现两个进程间互斥使用临界段的 T. Dekker 算法（见下），保证并行成分完全并行和杜绝死锁，同时当完成读写的程序某一方使用概率较大时，其效果更佳。例如，我们假设 P_0 的调用概率较大，那么，当 P_0 的一个进程进入临界段时，若此时没有 P_i 的进程欲进入临界段，则由于 flag [1]、…、flag [n] 都为假，因此 P_0 的其它进程也可同时进入临界段。方法用类 Pascal 语言描述如下（V 表示“或”）：

```
VAR     flag [0 · · n] of boolean;  (初值为假)
        turn: 0 · · 1;              (初值为 0 或 1)
```

```
程序P0:
    flag [0]: =true;
    while  flag [1] Vflag [2] v…vflag [n] do
                if turn=1  then
                  begin
                      flag [0]: =false;
                      while turn=1  do wait;
                      flag [0]: =true
                end;
          临界段
          turn: =1;
          flag [0]: =false;
          非临界段
```

```
程序 Pi (i=1, 2, …, n):
        flag [i]: =true;
        while  flag [0]  do
            if turn=0  then
                begin
                    flag [i]: =false;
                    while turn=0 do wait;
                    flag [i]: =true
                end;
          临界段
          turn: =0;
          flag [i]: =false;
          非临界段
```

关于算法的正确性，须证明三点：(1) 该算法能保证 P_0 与 P_i 互斥；(2) 该算法不会发生死锁；(3) 该算法能保证 P_0 的多个进程、P_i 与 P_j ($j\neq i$) 的多个进程并行。具体的证明，结合参考文献 [2] 不难给出，限于篇幅，不再详述。

3 结束语

通过前面的分析，我们发现，系统程序并行化方法的重点在于内存问题解决，而其中的难点无疑是多个可再入程序共享单元的互斥使用。本文提出的实现临界段互斥的三种方法，其中硬件方法比较理想，特别适用于读写操作频率相近以及共享单元比较少的情况，转移临界段法和改进的 T. Dekker 算法适用于临界段较长以及读写操作频率相差悬殊的情况。

需要指出的是，本文所给出的处理临界段的三种方法虽然是基于系统程序并行化而提出来的，但是，它们同样可运用于具有相似特点的应用程序并行化。

参 考 文 献

1 尤晋元. UNIX 操作系统教程. 西北电讯工程学院出版社

2 邹鹏等. 操作系统原理与方法. 学苑出版社

3 李勇., 刘恩林. 计算机体系结构. 国防科技大学出版社

Methods for Implementing Parallel System Program Under Multiprocessor Environment

Yao Yiping Jin Yuanchong Yang Taolan

(Department of Computer Science)

Abstract

The requirements to system program under multiprocessor environment are presented. The Problems encountered in developing system program parallelization are presented and their solutions are discussed. Three methods for implementing the mutual exclusion of critical section and their performance evaluation are given in it.

Key words multiprocessor system, system program parallelization, reentry, critical section mutual exclusion