第24卷 第1期
2006年3月

干 旱 气 象
ARID METEOROLOGY

Vol.24 NO.1
Mar,2006

文章编号:1006-7639(2006)-01-0071-04

预加载共享动态链接库算法在桌面Linux环境下的设计与实现

王仁塘,王仁斌

(兰州商学院计算机系,甘肃 兰州 730020)

摘 要:针对桌面Linux操作系统在支持桌面交互应用程序上存在着系统响应慢和交互能力差等问题,从内存管理角度提出了旨在缩短系统响应时间、增强系统交互能力的预加载共享动态链接库算法,并在桌面Linux环境下设计和实现了它。该算法从Linux应用程序频繁地调用共享动态链接库这一特点出发,根据共享动态链接库被调用的频率,动态地选择被调用最频繁的一些共享动态链接库,并预先将它们加载到物理内存空间,以加快进程的执行速度,缩短系统的响应时间,最终提升系统的交互能力。

关键词:预加载;共享动态链接库;启发式学习;桌面Linux操作系统

中图分类号:TP393.02 **文献标识码**:A

引 言

近几年来,Linux在桌面操作系统市场上越来越受到重视,但它在支持桌面交互应用程序上仍然存在着系统响应慢、交互能力差和交互界面不友好等问题。能否有效地支持桌面应用程序是桌面Linux系统能否在桌面操作系统市场上生存下去,直至成为Windows操作系统强有力竞争对手的决定性因素之一,因此如何增强桌面Linux系统支持桌面应用程序优化运行的能力是一个亟待解决的重要问题。

桌面Linux系统平台上运行的应用程序有一个重要特征:频繁地调用共享动态链接库,使共享动态链接库代码成为除内核代码外执行最频繁的代码[1]。Linux平台上应用程序的执行遵循"能拖尽量拖"的按需调页原则:在程序运行过程中才将需要的代码或数据从磁盘空间加载到物理内存空间。由于磁盘I/O操作相对CPU直接访问物理内存操作是非常耗时的,这使得程序的执行在时间上有一定的延迟。因为共享动态链接库的执行也遵循按需调页原则,而被所有进程共享的内核代码却常驻物理内存空间,在程序执行过程中CPU直接从物理内存空间读取它们,所以应用程序运行速度的提高主要依赖于共享动态链接库运行速度的提高。

由于现代操作系统运行的硬件平台上物理内存空间比较大,不再是非常紧缺的资源,并且应用程序执行按需调页原则本身具有延迟进程执行的缺点,因此,在应用程序执行前,预先将应用程序需要调用的共享动态链接库加载到物理内存空间,将加快应用程序的执行速度,这实际上就是预加载共享动态链接库算法的雏形思想。本文提出的预加载共享动态链接库算法是在这些基本思想基础上,考虑了不同用户使用的桌面应用程序可能不同,并且用户使用桌面应用程序的偏好可能随着时间的推移而发生变化的因素,动态地选择被调用比较频繁的一些共享动态链接库,并预先将它们加载到物理内存空间,以加快进程的执行速度,缩短系统的响应时间,最终提升系统的交互能力。

1 算法基本思想

由于Linux发展历史原因,桌面Linux操作系统设计中按需调页原则在本质上存在缺陷,而且如果系统物理内存空间越大,这个设计原则的缺陷就越明显。

现代操作系统运行的硬件平台上物理内存空间通常比较大,因此将进程执行过程中需要的代码或数据预先装入物理内存空间是加快进程执行速度的重要途径[2]。

然而,Linux操作系统只将内核代码常驻物理内存空间,以缩短系统的响应时间和提高系统的事务吞吐量。事实上,对于桌面Linux操作系统而言,共享动态链接库是除内核代码外执行最频繁的代码。共享动态链接库的执行是以按需调页方式在执行过程中才将需要的代码从磁盘加载到物理内存空间,

收稿日期:2006-01-27;**改回日期**:2006-02-24

作者简介:王仁塘(1982-),甘肃白银人,本科,现从事LINUX系统开发与应用.

这使得执行时间有一定的延迟。如果动态地选择被频繁调用的共享动态链接库并预先加载到物理内存空间,那么将极大地缩短系统的响应时间。因此预加载共享动态链接库到物理内存空间是缩短桌面Linux操作系统响应时间的重要途径。

由于物理内存资源的限制,不可能将所有的共享动态链接库预先加载到物理内存空间,因此使用预加载共享动态链接库算法缩短系统响应时间引发了另外一个问题:如何选择最常用的共享动态链接库预先加载到物理内存空间?

对于内核子系统而言,它提供的内核系统调用代码是所有进程共享的,并且是常驻物理内存的,因此针对不同的用户和进程不存在选择哪些内核代码预先加载到物理内存空间的问题。然而,对于桌面应用程序子系统而言,不同类型的用户频繁使用的桌面应用软件很有可能不同。如果使用预加载共享动态链接库方法缩短系统响应时间,则有一个潜在的问题需要解决:如何针对不同类型的用户选择最常用的共享动态链接库?

从上面提出的2个问题可以看出:对于不同类型的用户,由于经常使用的桌面应用程序有很大的不同,导致桌面应用程序频繁调用的共享动态链接库也是不同的,并且随着操作系统运行时间的迁移很可能在动态地变化着。然而,桌面Linux操作系统是一个通用桌面系统,不可能针对不同类型的用户群体发行不同的版本,因此需要一个能够自动学习用户类型信息和用户喜好动态变化信息的算法,该算法能够根据学习到的信息,对自己的行为进行动态调整,以更好地为用户服务。

启发式学习模式是著名的解决多样性和动态变化性问题的有效方法。针对桌面Linux系统中预加载共享动态链接库存在的问题[3],可以提出解决这些问题的基于启发式学习模式的预加载共享动态链接库算法。

算法基本思想如下:

将应用程序加载共享动态链接库的次数作为获取用户类型和用户喜好信息的基点,操作系统自动地学习不同类型用户和同一用户在不同时间段内使用桌面应用程序的情况,收集共享动态链接库的使用信息,然后根据学习到的信息动态地预先选择最常用的共享动态链接库并加载到物理内存空间,以缩短系统的用户响应时间。

2 算法设计与实现

2.1 基本数据结构

(1)代表共享动态链接库的结构型数据结构(ShareLibraryNode)[3]。

对于每一个在计算机启动以来被加载过并且被加载次数符合条件的共享动态链接库都需要创建一个ShareLibraryNode的实例,以记录该共享动态链接库的信息:被加载次数、引用计数和状态等等。在运行过程中,系统根据需要动态地创建和删除ShareLibraryNode实例。ShareLibraryNode的定义如下:

```
struct ShareLibraryNode
{  //链接到 DoubleSortLinkedList 有序链表
int  *SortPrev,  *SortNext;
//链接到散列共享动态链接库的 Hash 表
int  *HashPrev,  *HashNext;
//互斥访问锁
Mutex MutextAccessInstance;
//共享动态链接库被加载的次数
Unsigned Int AccessCount;
//共享动态链接库被引用的次数
Unsigned Int ReferenceCount;
//共享动态链接库的状态
Unsigned short ShareLibraryStatus ;
//共享动态链接库的名字
Char  *ShareLibraryName;
//共享动态链接库在操作系统中的路径
Char  *ShareLibraryPath;  };
```

在系统运行时,一个ShareLibraryNode实例同时链接到2个链表:DoubleSort LinkedList和Hash表。

(2)有序双向链表(DoubleSortLinked List)

记录本次计算机启动以来所有被加载过并满足一定条件的共享动态链接库,根据被加载的次数从大到小排序。链表节点是ShareLibraryNode的实例。在系统运行过程中DoubleSortLinked List被动态地添加和删除ShareLibraryNode的实例。

(3)Hash表

通过它能够快速地散列到共享动态链接库在Hash表中的位置。所有在DoubleSortLinkedList表中的节点同时在Hash表中。

2.2 需要实现的系统调用、内核函数和内核线程

(1) Hash函数HashShareLibraryName:以共享动态链接库名字为参数调用它可以获取共享动态链接库在Hash表中的索引下标,以进一步查找共享动态链接库在DoubleSortLinkedList中的位置。该函数被调用的频率比较高,函数性能要求比较高。

(2)系统调用LoadShareLibraryToVM(char *ShareDLLLibraryName):将共享动态链接库加载到虚拟内存空间。

（3）系统调用 UnLoadShareLibrary FromVM（char *ShareDLLLibraryName）：将共享动态链接库从虚拟内存空间卸载。

（4）内核函数 LoadShareLibraryToPM（char *ShareDLLLibraryName）：为共享动态链接库分配所需的所有物理内存空间。

（5）内核函数 UnLoadShareLibrary FromPM（char *ShareDLLLibraryName）：将共享动态链接库占用的物理页面释放。

（6）内核线程 KernelThread：以 Daemon 方式运行。在系统运行过程中，根据桌面应用程序运行的变化，动态地选择共享动态链接库并预先加载到物理内存空间。

根据算法的基本思想，上述系统调用、桌面进程、内核线程和内核函数间的交互如图 1 所示。

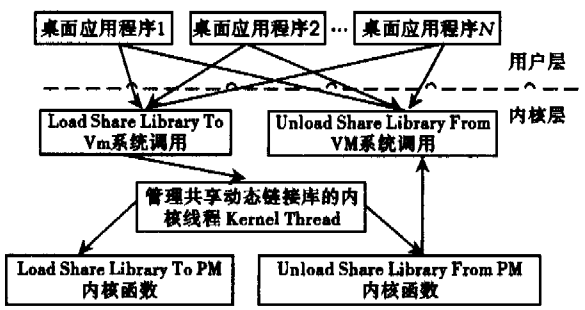

图 1　算法各个部分的交互示意图

Fig. 1　Algorithm interaction graph

在图 1 中，有如下几种可能的交互序列：

①桌面应用程序调用 LoadShareLibrary ToVM 系统调用加载共享动态链接库到虚拟内存空间，整个执行过程如图 2 所示。

②桌面应用程序调用 LoadShareLibraryToVM 系统调用，导致 DoubleSortLinkedList 链表中节点位置有重要的变化，LoadShareLibraryToVM 通过信号量机制唤醒 KernelThread，如果有足够的物理内存空间，内核线程调用 LoadShareLibraryToPM 将选中的共享动态链接库加载到物理内存空间。

③桌面应用程序调用 LoadShareLibraryToVM 系统调用，导致 DoubleSortLinkedList 表中节点位置有重要的变化，LoadShareLibraryToVM 通过信号量机制唤醒 KernelThread，如果没有足够的物理内存空间，内核线程首先依次调用 UnLoadShareLibrary-FromPM 和 UnLoadShareLibraryFromVM 释放引用计数等于零的共享动态链接库所占用的物理内存空间，最后调用 LoadShareLibraryToPM 将选中的共享动态链接库加载到物理内存空间。

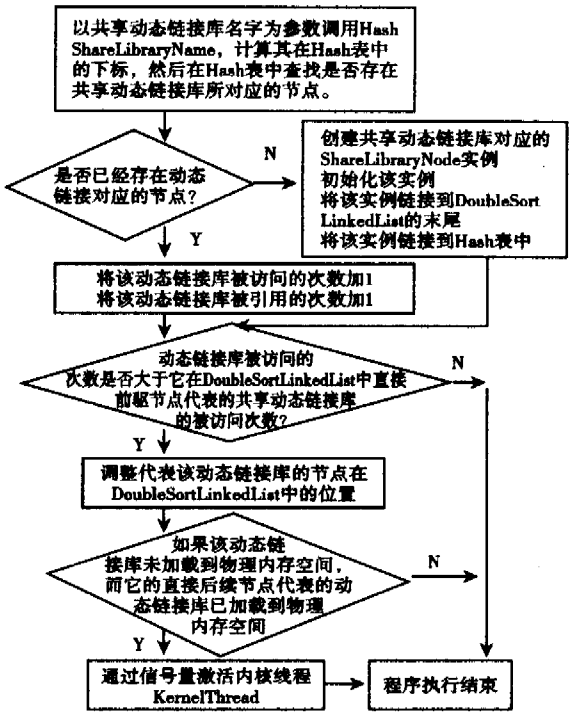

图 2　应用程序加减动态链接库的过程图

Fig. 2　Process of application programs loading DLL

在上述各种可能的调用序列下，共享动态链接库的状态动态地进行迁移。共享动态链接库可能处于的状态有 3 种：未加载到虚拟内存空间、已加载到虚拟内存空间和已加载到物理内存空间。共享动态链接库的状态迁移如图 3 所示。

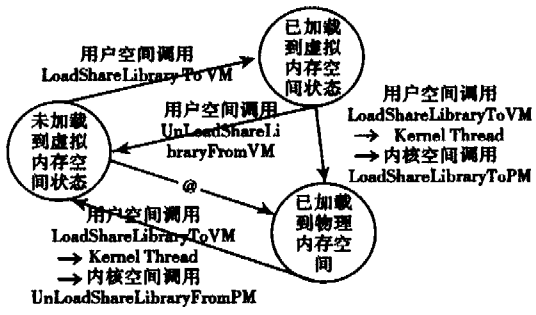

图 3　共享动态链接库状态迁移示意图

（@ 表示一个调用序列：用户空间调用 Load Share LibraryToVM→Kernel Thread→内核空间调用 Load ShareLibraryToPM；LoadShare LibraryToVM 是通过信号量机制激活内核线程 Kernel Thread 的）

Fig. 3　DLL status transmission graph

2.3　动态地选择共享动态链接库并预先加载其到物理内存空间的过程

系统启动阶段，KernelThread 根据系统配置文

件建立 DoubleSortLinkedList 链表和 Hash 表，预先加载上次系统运行过程中被频繁调用的共享动态链接库到物理内存空间；系统正常运行时，KernelThread 通常阻塞在信号量上，当桌面应用程序调用 LoadShare LibraryToVM 加载共享动态链接库，导致 DoubleSortLinkedList 链表中代表未加载到物理内存空间的共享动态链接库的节点移动到代表已加载到物理内存空间的共享动态链接库的节点的前面时，LoadShareLibraryToVM 通过信号量唤醒 KernelThread，以对加载到物理内存空间的共享动态链接库进行调整。

加载到物理内存空间的的共享动态链接库占用的物理内存大小有一定的限制，通常为系统物理内存空间大小的 15% ~20% 左右。

KernelThread[4] 的执行过程如图 4 所示。

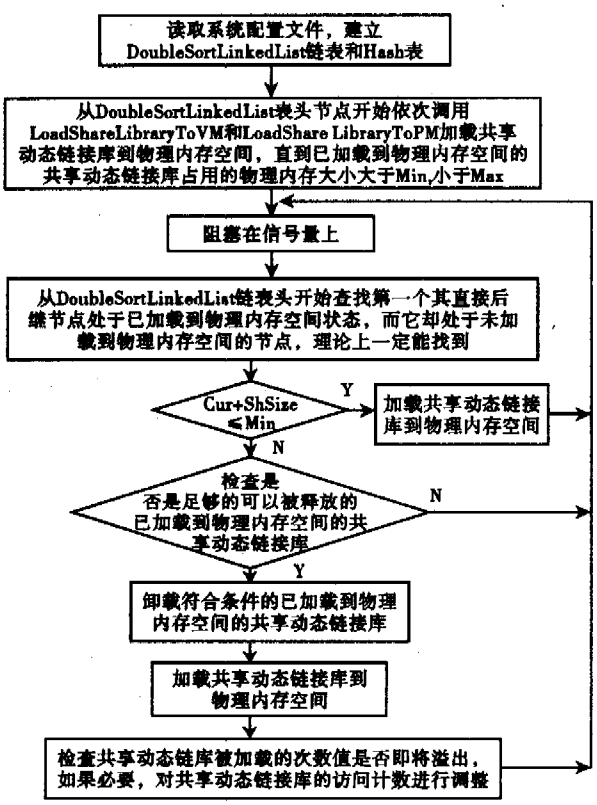

图 4 选择共享动态链接库并加载到物理内存空间的流程图
（Cur：已加载到物理内存空间的共享动态链接库占用的物理内存大小；Min：系统总的物理内存空间大小的 15%；Max：系统总的物理内存空间大小的 20%；ShSize：要加载的共享动态链接库需要占用的物理内存空间）

Fig. 4 Flow chart of selecting and loading DLL to physical space

2.4 系统配置文件设置

在系统配置文件中需要记录系统上次运行过程中被调用比较频繁的一些共享动态链接库信息。当系统再次启动时，从配置文件中读取这些信息，以加载共享动态链接库到物理内存空间。

在安装桌面 Linux 操作系统的过程中，安装程序缺省地将 Office 等桌面应用软件频繁调用的共享库动态链接库信息写入系统配置文件。当系统第一次启动时使用系统配置文件的缺省信息加载共享动态链接库。以后每次启动计算机时，都是根据系统上次运行过程中调用共享动态链接库的信息加载共享动态链接库到物理内存。

在关闭系统时，将 DoubleSortLinkedList 链表中所有代表没有加载到物理内存空间的共享动态链接库的 ShareLibraryNode 实例节点删除，只在系统配置文件中记录本次系统运行过程中被频繁调用的已加载到物理内存空间的共享动态链接库信息。

同时，在系统第一次启动时需要根据系统物理内存空间大小，设置加载到物理内存空间共享动态链接库可占用的物理内存大小的最小值和最大值。通常最小值设置为系统物理内存大小的 15% 左右，最大值设置为 20% 左右[4]，并将这 2 个值写入系统配置文件，以后每次启动计算机的时候，直接从配置文件中读取。

3 结束语

本文提出的预加载共享动态链接库算法从按需调页原则的缺陷出发，考虑到现代操作系统运行的硬件平台上物理内存空间不再是紧缺资源的实际情况，利用 Linux 应用程序频繁地调用共享动态链接库这一特征，根据共享动态链接库被调用的频率，动态地选择被调用最频繁的一些共享动态链接库，并预先将它们加载到物理内存空间，以加快进程的执行速度，缩短系统的响应时间，最终提升系统的交互能力。

具有启发式学习特性的预加载共享动态链接库算法有如下优点：

（1）通过将被调用最频繁的一些共享动态链接库预先加载到物理内存空间，减少磁盘 I/O 操作，加快进程的执行速度。

（2）通过在物理内存空间上共享动态链接库，提高物理内存空间的利用率。

在下一步工作中，针对算法设计和实现的全局性特点，细化粒度，在进程级别上增加一些特性，例如将指定应用程序调用的共享动态链接库加载到物理内存空间和将指定的共享动态链接库从物理内存空间中卸载等，提供应用程序加载和卸载共享动态链接库的接口，增强算法的灵活性和可收缩性。

（下转第 89 页）

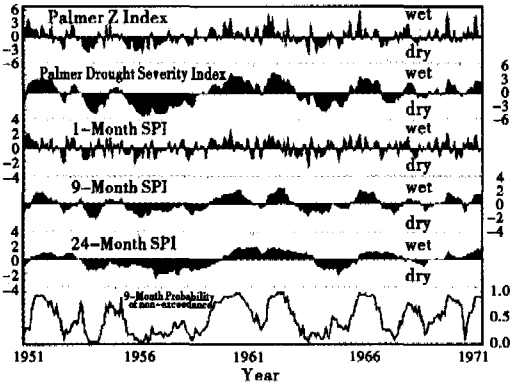

图 7　1951 年 1 月 1 日至 1970 年 12 月 31 日爱荷华州中东部 2 个逐月 Palmer 指数(Z 指数和 PDSI)与几个 SPI 产品的对比(用 1 个月的 SPI 和 Palmer Z 指数来度量短期(即月的)干旱,9 个月的 SPI 近似与 PDSI 一致。在这里可见 SPI 的多样性具有描述长期干旱(24 个月的 SPI)和把降水短缺用随机概率表示的能力。特殊指数的应用取决于使用者要表达什么。任何人都不能简单地说某一指数在所有方面比另一指数好)

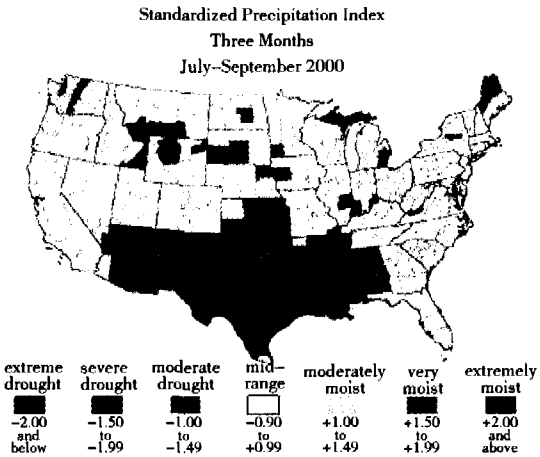

图 8　2000 年 7,8,9 三个月的 SPI(表明期间美国东南部平原到西南部地区发生的极度干旱情况。SPI 的值可比较不同地点的标准降水偏差)

(上接第 74 页)

参考文献:

[1] Jeffrey C Mogul. Big Memories on the Desktop[A]. In: Proceedings of the Fourth Workshop on Workstation Operating Systems[C]. List of publications from the DBLP Bibliography Server - FAQ in united states. 1993. 110 - 115.

[2] Lee D, Crowley P, Baer J, et al. "Execution Characteristics of Desktop Applications on Windows NT". 25th Annual International Symposium on Computer Architecture (ISCA98), USA, 1998.

[3] Rik van Riel. Page Replacement in Linux 2.4 Memory Management. 2001 FREENIX Track Technical Program. Boston, Massachusetts, USA, 2001.

[4] Juan E Navarro, Alan Cox Mitosis. a high performance, scalable virtual memory system[A]. Technical report TTR01 - 378[C], CS Dept., Rice University in America, 2001. 35 - 36.

The Design and Implementation of Preloading DLL Algorithm under Linux Platform

WANG Ren - tang, WANG Ren - bin

(Department of Computer Science, Lanzhou Commercial College, Lanzhou 730020, China)

Abstract: *Aiming at problems such as slow system response, bad interaction that exist in desktop Linux OS supporting application programs, we present preloading DLL algorithm in order to shorten system response time and enhance system interaction capability, meanwhile design and implement it under desktop Linux platform. Based on the characteristic that application programs invoke DLL heavily, the algorithm selects some DLLs that are run comparatively frequently and preloads them to physical memory so as to speed up running process and step up system interaction capabilities.*

Key words: preloading; sharing DLL; heuristic study; desktop Linux OS