文章编号: 1001-2486 (2000) 01-0020-04

实时应用中的内存锁定技术研究*

吴纯青, 李 钢
(国防科技大学计算机学院, 湖南 长沙 410073)

摘 要: 内存锁定是一种保证某进程驻留在内存而不需换页的方法。在实时环境中, 系统应保证将某进程锁定在内存中, 以减少数据访问、指令读取、进程间缓冲区切换等等引起的延迟。将一个进程的地址空间锁定在内存中, 就为应用的响应时间满足实时需要提供了保障。一般来说, 对时间要求苛刻的进程应锁定在内存中。本文主要以实时环境为背景, 阐述了内存锁定和解锁函数以及在多进程下内存锁定技术的应用。

关键词: 实时; 换页; 交换; 内存锁定; 共享内存
中图分类号: TP316.2 文献标识码: B

Study on the Technology of Memory Locking in Realtime Applications

WU Chun-qing, LI Gang
(College of computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Memory locking is a way to ensure that a process stays in main memory and is exempt from paging. In a realtime environment, a system must be able to guarantee that it will lock a process in memory to reduce latency for data access, instruction fetches, buffer passing between processes, and so forth. Locking a process's address space in memory helps ensure that the application's response time satisfies realtime requirements. As a general rule, time-critical processes should be locked into memory. This paper describes the items about "Memory Locking and Unlocking Functions" and "Memory Locking in multiprocesses environment" in a realtime environment

Key words: realtime; paging; swaping; memory-locking; share memory

在多程序 (multiprogramming) 环境中, 对操作系统而言, 能够使多个进程有效共享可用内存是至关重要的。如果某进程驻留在内存中, 则该进程的核必须分配足够的内存。如果仅进程的一部分需驻留在内存中, 则内存管理再配合调度策略即可做到对资源的优化使用。虚地址空间被划分为固定大小的页面, 每个进程通常占用一定数量的页面。在进程执行过程中, 这些页面独立地换页 (paging) 或交换 (swaping)。在进程执行时, 通常有一部分页面是常驻在内存中的。

由于系统的可用内存的数量有限, 所以在进程运行时需进行换页和交换操作。换页和交换操作通常由虚存管理系统执行, 对用户进程是透明的。在进程很大或多进程的情况下, 可能导致系统进行频繁换页或交换操作, 由此引起的系统开销会降低系统效率。

对于实时性要求非常苛刻的任务, 换页或交换引起的延迟往往是不能接受的。虽然通过增加内存容量可减少换页或交换频度, 但仍不能保证实时进程不被换出。因此, 在实时应用中, 需要将对时间要求苛刻的进程, 甚至其需访问的数据都锁定在内存中, 不让其换出, 以满足实时要求。

1 内存锁定和解锁函数

1.1 锁定函数和解锁函数

在实时应用中, 内存锁定通常是程序初始化的一部分。许多实时应用在执行期间锁定在内存中, 但有时候也希望在执行应用时, 时而锁定内存, 时而解锁内存。

* 收稿日期: 1999-06-07
作者简介: 吴纯青 (1964-), 女, 副研究员, 硕士。

内存锁定应用到某进程的地址空间。只有那些映射到进程地址空间的页面才能锁定在内存中。进程退出时, 页面从进程地址空间中删除, 从而锁定也就失效了。

函数 mlock 和 mlockall 被用来锁定内存。mlock 函数允许调用进程有选择地锁定某段地址空间。mlockall 函数导致整个进程地址空间被锁定到内存中。被锁定的内存一直有效直至该进程退出或应用调用相应的 munlock 或 munlockall 函数。

内存锁定不能通过 fork 被继承, 且所有与某进程[1]相关的内存锁定可以被 exec 函数解锁或在进程终止时解锁。

表 1 列出了内存锁定和解锁函数。

表 1 内存锁定和解锁函数

Tab. 1 Memory locking/unlocking functions

函数	参数	描述
Mlock	(内存区首地址, 内存区大小)	锁定进程地址空间的指定区域
Munlock	(内存区首地址, 内存区大小)	解锁进程地址空间的指定区域
Mlockall	(标志)[注1]	锁定整个进程地址空间
Munlockall	无	解锁整个进程地址空间

注 1: 标志为 MCL_CURRENT 时, 锁定当前被映射的所有页面; 标志为 MCL_FUTURE 时, 锁定将来被映射的所有页面。

1.2 锁定和解锁指定内存区域

mlock 函数可以锁定预先分配的指定内存区域。mlock 的地址和大小两参数决定了预分配区域的边界。对 mlock 的成功调用, 指定内存区域就被锁定了。内存锁定是按系统定义的页面来锁定的。如果地址和大小两参数指定的区域比一个页面小, 则系统锁定一页。mlock 函数锁定包含指定区域的任何部分的所有页面, 所以有可能被锁定的地址比指定区大。

重复调用 mlock 可能会申请到比可用内存更多的物理内存。在这种情况下, 子进程 (subsequent process) 必须等到被锁定的内存变成可用的。实时应用经常不能忍受由进程等待可用内存而引入的延迟。因此, 在实时应用中, 一般采用预分配并锁定策略。

如果进程需要锁定的内存比系统实际内存大, 则系统自动报错。

2 多进程下内存锁定技术的应用

共享内存和内存映射文件允许进程通过将数据直接送到进程地址空间来通讯。进程利用一部分地址空间的共享来达到通讯的目的。当一个进程数据写入共享区的某个位置, 该数据立即对其它共享这一区域的进程有效。这是一种快捷的通讯方式, 它不需调用任何系统调用, 数据的传递也减少到最少。

一个进程可以通过映射部分内存对象到它的地址空间。当多个进程映射相同的内存对象, 他们就能共享其中的数据。

2.1 内存对象

内存映射和共享内存功能函数提供了一些控制对共享内存的访问的手段, 这样, 应用程序可以协调对共享地址空间的使用。

对一个共享的映射文件, 使用相同路径并打开了对该内存对象连接的所有进程都获得了该文件的共享映射, 各进程引起的改变将立即反映到该映射文件中。如果该映射允许写, 则由一个进程写入的数据立即被所有拥有该映射的进程共享。

内存映射对象是永久的: 其名字和内容一直保留到所有访问该对象的进程都与这个文件脱链。共享内存和内存映射文件的一般用法如下:

- 通过函数 open 或 shm_open 打开某内存对象并得到一个文件描述符;
- 通过函数 mmap 将该对象映射到内存;

• 通过函数 munmap 撤消这个对象的映射;
• 通过 close 调用关闭这个对象;
• 撤消共享内存对象用 shm_unlink 调用, 或撤消内存映射文件用 unlink 调用。

共享内存对象仅在应用程序被执行时产生和使用, 而文件则可以在应用程序每次运行时保存起来或重新使用。函数 unlink 和 shm_unlink 删除文件及其内容。如果需要保存一个共享文件, 则每次使用后只需关闭这个文件, 不要用 unlink 删除它。

可以使用内存映射文件而不使用共享内存, 本文假设两个都使用。表 2 概括了打开和删除共享内存的调用 (shm_open, shm_unlink); 表 3 列出了产生和控制内存映射对象的函数 (mmap, munmap)。

表 2 打开和删除共享内存

Tab. 2 Open/delete shared-memory

函 数	参 数	描 述
shm_open	(共享内存对象名, flag[注2], mode[注3])	打开一个共享内存对象, 返回一个文件描述符
shm_unlink	(共享内存对象名)	删除共享内存对象的名字

注 2: flag——状态标志, 共四种:
O_RDONLY: 该共享内存对象为只读;
O_RDWR: 该共享内存对象为可读可写;
O_CREAT: 该共享内存对象不存在时创建它;
O_EXCL: 当使用标志 O_CREAT 时创建一个与该共享内存对象的互斥连接。
注 3: mode: 访问状态。

表 3 内存映射函数

Tab. 3 Memory mapping functions

函 数	参 数	描 述
mmap	(addr, length, prot, flag, fd, offset)[注4]	将内存对象映射到内存
munmap	(内存区首地址, 内存区大小)	撤消以前的映射区

注 4: addr——起始地址, 需为页面大小的整数倍。
length——区域大小, 单位为字节, 需为页面大小的整数倍。
prot——访问权限标志, 共四种: PROT_READ: 可读; PROT_WRITE: 可写;
PROT_EXEC: 可执行; PROT_NONE: 不能访问。
flag——被映射区域属性, 共三种: MAP_SHARED: 共享; MAP_PRIVATE: 私有;
MAP_FIXED: 精确解释地址。
fd——文件描述符。
offset——地址偏移量, 需为页面大小的整数倍。

用 shm_open 调用可以产生和打开一个内存对象, 然后该对象就可以被映射到进程地址空间, 文件控制功能可以控制访问权限, 比如读、写权限等。

由一个进程地址空间写入对象的数据对所有映射相同区域的进程都是可用的。子进程继承这个地址空间和所有父进程映射的区域。一旦对象被打开, 子进程可以用 mmap 函数与它建立一个映射。如果对象已被映射, 则子进程也继承映射区。

无关的进程也可以使用这个对象, 但必须先调用 open 或 shm_open 并用 mmap 函数建立与共享内存的连接。

2.2 锁定共享内存

可以锁定和解锁一个共享内存段到物理内存以消除换页。下例显示了如何将一个文件映射到进程地址空间。撤消文件映射后, 对该地址的锁定也被撤除。

例 # include 〈unistd. h〉
include 〈sys/ types. h〉

```
# include 〈stdio.h〉
# include 〈sys/mman.h〉
# include 〈sys/stat.h〉
# include 〈errno.h〉
main ()
{   int fd, size= 5000, mode= S_IRWXO| S_IRWXG| S_IRWXU;
    caddr_tpg_addr;
    fd= shm_open ( "example", O_RDWR| O_CREAT, mode);
    if (fd< 0) {perror ( "open error"); exit ();}
    if ( (ftruncate (fd, size)) = = - 1) {perror ( "ftruncate failure"); exit ();}
    pg_addr= (caddr_t) mmap (0, size, PROT_READ| PROT_WRITE| PROT_EXEC,
                                        MAP_SHARED, fd, 0);
    if (pg_addr= = (caddr_t) - 1) {perror ( "mmap failure"); exit ();}
    if (mlock (pg_addr, size)! = 0) {perror ( "mlock failure"); exit ();}
    if (munmap (pg_addr, size) < 0) peeror ( "unmap error");
    close (fd);
    shm_unlink ( "example");
}
```

上例中, 共享内存对象 example 映射到的内存区首地址为 pg_addr, 该内存被锁定后, 可供多个进程进行实时访问。

3 结束语

在单进程的实时应用中, 内存锁定时不需涉及共享内存的概念; 对于单进程、多线程的应用, 被锁定内存作为该进程的全局数据区即可达到多线程间通讯的目的; 对于多进程的实时应用, 将共享内存概念应用于内存锁定技术, 则是一种简单而高效的满足实时要求的手段。

另外, 如果同时有多个线程或进程被映射到被锁定内存, 则在对该被锁定内存进行互斥访问时需用到线程互斥变量或进程间信号灯等技术, 限于篇幅, 在本文中不作阐述, 见文献 [1, 2]。

参考文献:

[1] Guide to Realtime Programming [R]. Digital Equipment Corp., March 1996.
[2] 聊鸿斌. UNIX 高级教程系统技术内幕 [M]. 北京: 清华大学出版社, 1999.
[3] Guide to DECthreads [R]. Digital Equipment corp., March 1996.
[4] POSIX 1003.1c (pthread) Routines Reference [R]. Digital Equipment Corp, March 1996.
[5] Programmer's Guide [R]. Digital Equipment Corp., March 1996.