第 36 卷 第 9 期 计 算 机 工 程 2010 年 5 月
Vol.36 No.9 Computer Engineering May 2010
·软件技术与数据库· 文章编号：1000—3428(2010)09—0085—02 文献标识码：A 中图分类号：TP311

基于 Linux 内核的动态内存管理机制的实现

杨 峰
(鞍山师范学院计算中心，鞍山 114005)

摘 要：在软件开发过程中，共享内存经常会遇到一个进程消耗太多内存导致其他进程无法得到需要内存的潜在问题，针对该问题，基于 Linux 内核实现一种动态内存管理机制，该机制能够限制每个进程所能申请的最大内存数，同时可以避免进程内存泄露造成的系统崩溃。实验结果表明，该机制效率高、且易用性好。

关键词：Linux 内核；内存管理；内存泄露

Implementation of Dynamic Memory Management Mechanism Based on Linux Kernel

YANG Feng
(Computer Center, Anshan Normal University, Anshan 114005)

【Abstract】During the development of software, the shared memory always meets potential problem, one process may consume too much memory so that other procsses may be starved. Aiming at the problem, this paper describes an implementation of dynamic memory management based on Linux kernel to limit the maximum memory size of each process that can be applied, avoids system crash caused by memory leak as well. Experimental results show that the management has a high effectiveness and is easy to be used.

【Key words】Linux kernel; memory management; memory leak

1 概述

操作系统中的内存管理单元负责管理整个系统的物理地址空间和虚拟地址空间，它是整个系统得以存在和运行的基础。动态内存分配，即从堆上分配内存，由程序员负责申请和释放。如何充分利用共享内存以及避免内存泄露成为内存管理的首要问题。针对以上问题，本文设计并实现了一种基于 Linux 内核的动态内存管理机制[1]。

2 Linux 动态内存分配机制

用户进程的内存管理是通过位于应用程序和 Linux 内核之间的 C 库维护的，本文使用的 C 库为 Glibc 库。应用程序通过 malloc, calloc 等函数申请动态内存，调用 Glibc 库中的分配函数，Glibc 库函数调用内核提供的系统调用向内核申请内存完成应用程序的动态内存分配；应用程序通过 free 函数释放动态内存，调用 Glibc 库中的释放函数，Glibc 库调用内核提供的系统函数向内核申请释放内存完成应用程序的动态内存释放。

2.1 Glibc 内存分配机制

Glibc 的 malloc 提供 2 种动态内存管理的方法：堆内存分配和 mmap 的内存分配。具体使用哪一种方式分配，取决于所需分配内存的大小，Glibc 中默认的分界线为 128 KB。

2.1.1 调用 brk 实现进程中的堆内存分配

在 Glibc 中，当进程所需要的内存较小时，该内存会从进程的堆中分配，但是堆分配出来的内存空间，系统一般不会回收，只有当进程的堆大小到达最大限额或者没有足够连续大小的空间为进程继续分配所需内存时，才会回收不用的堆内存。在这种方式下，Glibc 会维护一些固定大小的内存池以减少内存碎片。

2.1.2 mmap 的内存分配

在 Glibc 中，一般在比较大的内存分配时会使用 mmap 系统调用，它以页为单位分配内存，这不可避免地会带来内存浪费，但当进程调用 free 释放所分配的内存时，Glibc 会立即调用 munmap，把所分配的内存空间释放回系统。

2.1.3 munmap 内存空间的释放

应用程序通过 free 函数释放内存空间，Glibc 首先检查内存分配的方式。若为通过 brk 分配的内存空间，则 Glibc 并不向内核申请回收内存，而是标志此段内存为可用内存，插入 Glibc 的可用内存池链表；若为通过 mmap 分配的内存空间，则 Glibc 调用 munmap 向内核申请释放内存空间。

2.2 Linux 内存分配机制

Glibc 根据申请内存的大小调用不同的系统调用向内核申请内存。

2.2.1 brk 系统调用

当申请内存小于 128 KB 时，调用 sys_brk 申请内存。函数 sys_brk 的步骤如下[2]：(1)验证 brk 参数是否位于进程代码所在的线性区，如果是，则立即返回。(2)如果进程请求缩小堆，则调用 do_munmap()，然后返回。(3)如果进程请求扩大堆，则函数首先检查进程是否企图分配在其限制范围之外的内存，如果是，则返回 mm->brk 的原有值。(4)如果一切顺利，则函数调用 do_brk()函数，分配内存，并更新 current->mm->brk 指针。

应用程序在内存中的映射及 current->mm 对应位置见图 1。

作者简介：杨 峰(1976－)，男，讲师、硕士，主研方向：嵌入式系统

收稿日期：2009-12-14 **E-mail：**wxx_yyy_yxb@163.com

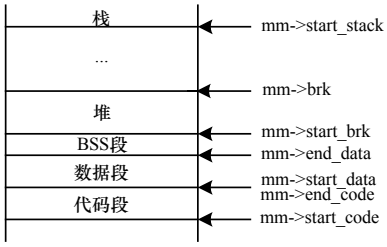

图 1　应用程序内存映射

2.2.2　mmap 系统调用

当申请内存大于 128 KB 时，调用 mmap()申请内存。函数 mmap 步骤如下[2]：(1)检查参数的正确性，若超出范围，则返回出错码。(2)获得新线性区的线性地址空间。(3)根据参数 prot 和 flags 设置线性区描述符的标志。(4)调用 find_vma_prepare()确定位于新区间之前线性区对象的位置以及在红-黑树中新线性区的位置。 (5)检查插入新的线性区是否引起进程地址空间大小超过所能分配的最大堆空间的阈值，如果是，则返回出错码。(6)检查前一个线性区是否可以扩展包含新的区间，若可以，则返回新线性区的线性地址。(7)把新线性区插入到线性区链表和红-黑树中，并返回新线性区的线性地址。

2.2.3　munmap 系统调用

当释放内存时，系统调用 sys_munmap()函数。

(1)扫描进程所拥有的线性区链表，并把包含在进程地址空间的线性地址区间中的所有线性区从链表中解除链接。

(2)更新进程页表，删除第 1 阶段找到并标识出的线性区。

Linux 动态内存分配结构如图 2 所示。

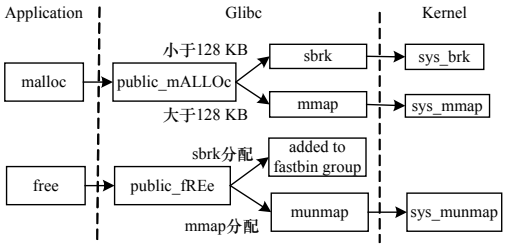

图 2　Linux 动态内存分配结构

3　Linux 进程资源限制

Linux 内核为每个进程设置一组相关的资源限制，限制指定了进程能使用的系统资源数量。对当前进程的资源限制存放在 current->signal->rlim 字段，该字段是类型为 rlimit 结构的数组，每个资源限制对应一个元素：

```
struct rlimit {
  unsigned long rlim_cur;
  unsigned long rlim_max;   };
```

其中，rlim_cur 字段是资源的当前资源限制；rlim_max 字段是资源限制所允许的最大值。

在 Linux 当前承认的资源限制中，与动态内存分配相关的有以下 2 个[3]：

(1)RLIMIT_DATA：堆大小的最大值。当内核调用 sys_brk 时，首先会检查进程是否企图分配在其限制范围之外的内存，如果是，则函数并不多分配内存，只是返回 mm->brk 的原有值。此资源限制只限制了通过 sys_brk 申请的内存分配，若应用程序通过 sys_mmap 申请大于 128 KB 的内存空间时，则起不到限制的作用。

(2)RLIMIT_AS：进程地址空间的最大数。当内核调用 do_brk 和 do_mmap_pgoff 时，都会检查插入新的线性区是否引起进程地址空间的大小超过存放在进程描述符 rlim[RLIMIT_AS].rlim_cur 字段中的阈值，如果是，则返回出错码-ENOMEM。此资源限制可将应用程序通过 malloc 等函数申请的动态内存总数加以限制，但应用程序在运行时产生的 BSS 段以及共享库都是通过 do_mmap_pgoff 映射到进程空间的，因此，此资源限制所限制的资源实际为 3 个部分：共享文件映射占用的内存资源、应用程序 BSS 段占用的内存资源以及应用程序动态申请的内存资源，而要限制的只是这 3 种中的一部分，所以，此资源限制也达不到本文所要求的效果。

4　Linux 内核动态内存管理机制的实现

根据以上分析可知，需要统计的为通过 brk 申请的全部内存以及通过 mmap 申请的属于应用程序动态申请的那部分内存。因此，要将共享文件映射占用的内存资源以及程序 BSS 段占用的内存资源从本文的统计中剔除。

4.1　文件映射与匿名映射

Linux 内核提供的内存映射可以分为 2 类：文件映射和匿名映射。本文中所指的共享文件映射主要包括共享库的数据段、代码段映射以及应用程序自身数据段、代码段映射；匿名映射主要包括共享库 BSS 段的映射、应用程序自身 BSS 段的映射以及动态内存申请所产生的映射。可通过函数 do_mmap_pgoff 中的第 1 个参数区分文件映射和匿名映射。当 file==NULL 时，表示此映射为匿名映射，否则为文件映射。

4.2　程序 BSS 段和应用程序动态内存申请

共享库的 BSS 段、应用程序的 BSS 段以及应用程序通过 mmap 申请的动态内存都属于匿名映射，需要统计的只有第 3 部分。共享库的 BSS 段以及应用程序的 BSS 段映射到虚拟内存的地址在链接时已经确定，而应用程序动态申请的内存地址是由程序运行时通过内核动态分配的。因此，可以通过函数 sys_mmap 的第 1 个参数 addr 区分程序 BSS 段和应用程序动态申请内存。当 addr==0 时，表示为应用程序动态申请的内存映射，否则为程序 BSS 段，置 MAP_LIBBSS(0x8000)标志，不计入之后的统计中。

4.3　动态内存管理机制的实现

4.3.1　在 Linux 内核中的实现

(1)在 mm_struct 结构体中添加 unsigned long total_heap 用来统计每个进程动态申请内存的总数。

(2)在 resource.h 文件下添加 RLIMIT_HEAP，用来限制用户进程动态申请内存的总数。

(3)调用 do_brk 时，先判断进程动态申请内存总数是否超过限制值。

```
if((mm->total_heap << PAGE_SHIFT) + len > current->signal->
rlim[RLIMIT_HEAP].rlim_cur)
    return -ENOMEM;
```

若超过限制，则返回-ENOMEM；若没超过限制，则分配内存空间，并更新当前进程申请内存总数。

```
mm->total_heap += len >> PAGE_SHIFT;
```

(4)当调用 do_mmap_pgoff 时，先判断进程动态内存申请总数是否超过限制值。

```
if((mm->total_heap << PAGE_SHIFT) + len > current->signal->
rlim[RLIMIT_HEAP].rlim_cur)
    return -ENOMEM;
```

若超过限制，则返回-ENOMEM；若没超过限制，则分

（下转第 89 页）

由于 $f(x_1,C)=f(x_2,C)$ 且 $f(x_1,D)\neq f(x_2,D)$, $f(x_3,C)=f(x_4,C)$ 且 $f(x_3,D)\neq f(x_4,D)$，因此 x_1 和 x_2, x_3 和 x_4 为不相容对象，表 1 为不相容的决策表信息系统，根据定义 8，增加属性列 β，有 $f(x_1,\beta)=f(x_1,D)=1$, $f(x_2,\beta)=f(x_2,D)=0$, $f(x_3,\beta)=f(x_3,D)=1$, $f(x_4,\beta)=f(x_4,D)=0$, $f(x_5,\beta)=$“$*$”，建立新的决策表 S_1', S_1'是相容的，如表 2 所示。

表 1　决策表信息系统 S_1

	a	b	c	D
x_1	1	0	1	1
x_2	1	0	1	0
x_3	0	0	1	1
x_4	0	0	1	0
x_5	1	1	1	1

表 2　决策表信息系统 S_1'

	a	b	c	β	D
x_1	1	0	1	1	1
x_2	1	0	1	0	0
x_3	0	0	1	1	1
x_4	0	0	1	0	0
x_5	1	1	1	*	1

在其基础上，根据定义 9 建立可分辨矩阵 $\boldsymbol{M}_1'$，如表 3 所示。

表 3　可分辨矩阵 $\boldsymbol{M}_1'$

	1	2	3	4	5
1		β	Φ	$a\beta$	Φ
2			$a\beta$	Φ	b
3				β	Φ
4					ab

分析可分辨矩阵 $\boldsymbol{M}_1'$，在 $\boldsymbol{M}_1'$中所有单属性元素有 $m_{12}=\{\beta\}$, $m_{25}=\{b\}$, $m_{34}=\{\beta\}$，只有 $m_{25}=\{b\}$且 $\beta\notin m_{25}$，由定理 1 得 $Core(C)=\{b\}$。此结果与文献[3]方法所得结果是一致的；而文献[2]方法获得的核为$\{a,b\}$，是不正确的。

实例 2 决策表 S_2 为不相容决策表，如表 4 所示，通过添加 β 列，转化为相容决策表 S_2'，如表 5 所示。

表 4　决策表信息系统 S_2

	a	b	c	D
x_1	1	0	1	0
x_2	0	0	1	0
x_3	0	0	1	1
x_4	1	1	1	1

表 5　决策表信息系统 S_2'

	a	b	c	β	D
x_1	1	0	1	*	0
x_2	0	0	1	1	0
x_3	0	0	1	0	1
x_4	1	1	1	*	1

根据定义 9，可创建可分辨矩阵 $\boldsymbol{M}_2'$，如表 6 所示。

表 6　可分辨矩阵 $\boldsymbol{M}_2'$

	1	2	3	4
1		Φ	a	b
2			β	ab
3				ab

在 $\boldsymbol{M}_2'$中所有单属性元素有 $m_{13}=\{a\}$, $m_{14}=\{b\}$, $m_{23}=\{\beta\}$，因此，$Core(C)=\{a,b\}$。此结果与文献[3]的方法所得结果是一致的。

5　结束语

为了解决决策表中因存在不相容性造成的求核错误，本文通过增加一个属性列 β，将决策表中不相容对象区分开，使不相容决策表转换为相容决策表。在新的决策表定义形式下建立可分辨矩阵，给出求核方法，并证明该核与基于正区域的核是等价的。实例表明，该求核方法既适用于不相容的决策表，也适用于相容的决策表。

参考文献

[1] 官礼和. 基于可辨识矩阵的属性约简算法[J]. 计算机工程, 2008, 34(3): 3-5.

[2] Hu Xiaohua, Cercone N. Learning in Relational Databases: A Rough Set Approach[J]. Computational Intelligence, 1995, 11(2): 323-337.

[3] 叶东毅, 陈昭炯. 一个新的差别矩阵及其求核方法[J]. 电子学报, 2002, 30(7): 1086-1088.

[4] 王国胤. 决策表核属性的计算方法[J]. 计算机学报, 2003, 26(5): 611-615.

[5] 刘文军, 谷云东, 冯艳宾, 等. 基于可辨别矩阵和逻辑运算的属性约简算法的改进[J]. 模式识别与人工智能, 2004, 17(1): 119-123.

[6] 杨　明, 孙志挥. 改进的差别矩阵及其求核方法[J]. 复旦大学学报: 自然科学版, 2004, 43(5): 865-868.

编辑　索书志

（上接第 86 页）

配内存空间，并更新当前进程申请内存总数(需要去除共享文件映射和共享文件 BSS 段的干扰)。

```
if(file == NULL && !(flags & MAP_LIBBSS))
    mm->total_heap += len >> PAGE_SHIFT;
```

(5)当调用 do_munmap 时，应该减去 unmap 的长度。在函数 unmap_fixup 中，判断是否为用户动态申请内存的释放，并更新当前进程申请内存总数。

```
if(area->vm_file == NULL)
    area->vm_mm->total_heap -= len >> PAGE_SHIFT;
```

(6)当调用 exit_mmap 时，将进程动态申请内存数置 0。

```
mm->total_heap = 0;
```

通过以上的修改可以在内核中精确地统计每个进程动态申请的内存总数，并且可以对每个进程设置 signal->rlim[RLIMIT_HEAP].rlim_cur 以限制每个进程所能动态申请的最大内存数。

4.3.2　用户接口的实现

若要在内核中对每个进程所能申请的动态内存最大值加以控制，就必须要修改 signal->rlim[RLIMIT_HEAP].rlim_cur 值，因此，需要给应用层提供接口修改内核中的资源限制值。

由用户空间向内核空间传递数据有多种方法，本文通过写/proc 目录下相应进程 PID 的 rlimit 文件实现对每个进程所能申请的最大动态内存的限制，通过读/proc 目录下相应进程 PID 的 rlimit 文件得到当前进程通过动态申请的内存总数[4]。

5　结束语

本文以 Linux 2.6 为基础实现了一套动态内存管理机制。本文机制从 Linux 内核中对应用进程进行控制，通过读写/proc 文件进行控制，相对于其他动态内存检测的应用软件，其易用性和通用性得到最大限度的提高。经过实验证明，本文管理机制能准确控制应用进程所能动态申请的最大内存数，对系统性能没有任何影响，并避免了因为内存泄露而导致的系统崩溃，具有很高的实用性。该机制对于嵌入式系统具有重要意义，可以使有限的内存资源得到合理分配和有效控制。在今后的研究中，可将该机制做成模块形式，提高其可移植性，最大限度地提高其实用性。

参考文献

[1] 高海昌, 冯博琴. Linux 下可执行文件的动态内存检测设计与实现[J]. 计算机工程, 2007, 33(1): 74-79.

[2] Bovet D P, Cesati M. 深入理解 Linux 内核[M]. 陈莉君, 译. 北京: 中国电力出版社, 2008.

[3] Gorman M. 深入理解 Linux 虚拟内存管理[M]. 白　洛, 李俊奎, 刘森林, 译. 北京: 北京航空航天大学出版社, 2006.

[4] 何杭军, 朱　利. 基于 Linux 的动态内存检测工具的设计与实现[J]. 计算机工程, 2005, 31(21): 69-71.

编辑　索书志