一种 Linux 内存管理机制

谢文娣,冷先进,金 建

(安徽新华学院 电子通信工程学院,合肥 230088)

摘 要:以 Linux 内存管理研究为基础,阐述了内存管理模型、伙伴系统 slab 分配流程和内存回收策略,提出了通过修改内存管理器和调整 Linux 水位线的方法,以避免 OOM(out of memory)事件的发生和内存耗尽问题的出现,并对内存变化过程进行了分析。

关键词:内存管理;内存回收策略;水位线;OOM

中图分类号:TP316.85 文献标识码:A 文章编号:2095-7726(2016)12-0031-06

内存是 Linux 系统中重要的资源,在运行过程中,系统内核和所有进程的运行都需要使用有限的物理内存,因此内存的分配和释放策略制约着系统的运行效率。高效的内存管理策略不仅要有效地管理系统内存,减少内存频繁分配,避免出现内存耗尽的情况,而且要尽量提高分配和回收的速度以提高系统的运行效率。

1 Linux 内存管理模型

Linux 系统的内存区域(zone)分为 ZONE_DMA、ZONE_NORMAL 和 ZONE_HIGHMEM 三种,系统给每种 zone 都分配有 buddy system。在 buddy system 中,物理内存空间是按 2 的整数次幂(称为 order)放于 zone_free_area 中的。系统会按照需求先拆分物理内存空间进行内存分配,再合并和释放内存,即在 zone allocator 里调用 alloc_page () 函数分配内存,调用 free_page()函数释放内存。虽然 buddy system 的效率非常高,但它存在内部碎片问题,这是因为只有从 2 的 0 次方到 MAX_ORDER 次方的页才会被分配出来,且对于硬件 cache(缓存)来说,这种使用方法也不方便。在 Linux 内核中,固定分配的内存经常会被申请使用,如果每次申请都必须通过 buddy system 就会浪费时间和空间,而引入 slab allocator(分配器)则可以节省时间和空间。slab allocator 模型图如图 1 所示。

slab allocator 会把内存资源放到 cache 里按定值进行缓存。如果系统需要使用内存,就可直接从此 cache 里调取,不使用时就释放,而内存也会被重新放回 cache 里。cache 以 slab 为单位来划分区域,几个连接的物理页常常包含在一个 slab 中。多个 cache 在 slab allocator 中共同维护,而每个 cache 里可能包含同一类型的 object(对象)。系统通过调用 kmalloc()函数或 kmem_cache_alloc()函数分配 slab allocator 中的内存资源,调用 kfree()函数或 kmem_cache_free()函数释放内存,如果在通过调用 kmem_cache_alloc()函数分配内存时 object 不在 cache 中,那么就通过调用 cache_grow () 函数在 cache 中增加一个 slab。cache_grow()函数给 object 分配物理内存 (cache_grow ()=>kmem_getpages()),而分配 kmem_getpages 还要通过 buddy system 实现 (kmem_getpages () => alloc_pages_node ()__alloc_pages ())。由此可见,slab allocator的出现并不意味着 buddy system 完全消失,而是对后者的一种增强。

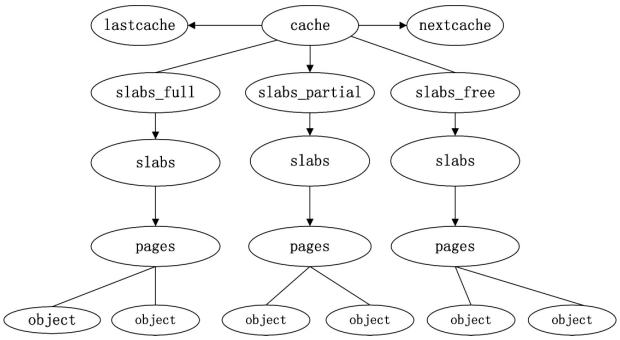

图 1 slab allocator 模型

收稿日期:2016-06-01

基金项目:安徽高校自然科学研究项目(KJ2016A306)

作者简介:谢文娣(1987-),女,硕士,讲师,主要研究方向:嵌入式系统、计算机网络和移动通信。

2 Linux 伙伴系统

2.1 分配流程

(1)正常分配(或叫快速分配)

为提升性能,对分配的是单个页面的情况,系统会在每个内存管理区中都定义一个“perCPU”页框高速缓存。所有的“perCPU”高速缓存中都包含一些预先分配的页框,这些页框可以用来满足本地 CPU 发出的单个页内存请求。Linux 内核为每个内存管理区和 CPU 都提供了两种高速缓存:一种是热高速缓存,它存放的页框中所包含的内容很可能在 CPU 硬件高速缓存中也存在;另一种是冷高速缓存。如果 perCPU 缓存中不存在页面,则可通过伙伴系统来提取页面进行增补。

对多个页面的分配,系统先分配指定类型的页面,若指定类型没有足够的页面,就转而分配链表中备用的类型,然后将类型链表保留下来。

(2)慢速(允许等待和页面回收)分配

慢速分配的一般流程是:唤醒内存页面回收线程→尝试低水位分配→忽略水位分配→压缩内存分配→直接回收内存分配→杀死线程分配(称为 OOM killer)→压缩内存分配。

在 Linux 伙伴系统中,每个 order 都被分为五种不同的类型,它们被统称为 MIGRATE_TYPES,即迁移类型。MIGRATE_TYPES 是反碎片的一种机制,其原理是将伙伴系统的内存页分为可移动、不可移动和可回收等几种类型,同一类型的页只能放在同一个区域,如不可回收的页不能放在可移动类型区域。

MIGRATE_UNMOVABLE 是不可移动页,在内存中有固定位置,不能移动,核心内核分配的大部分内存属于此类。MIGRATE_RECLAIMABLE 是可回收页,不能移动,但可以删除,Kswapd 内核线程在有交换页面需要时对此区域进行操作,如内存欠缺,Kswapd 会将某些处于进程中的页面与 swap 空间交换。MIGRATE_MOVABLE 是可移动又可回收页,用户空间程序使用此类,通过页表映射实现,若应用程序虚拟地址空间有变化,只需变化页表即可。MIGRATE_RESERVE 是在系统剩余内存很少且要求又比较紧急时才用到的区域。MIGRATE_ISOLATE 在非一致内存访问 NUMA(non-uniform memory access)架构上使用,它是一种特殊的虚拟区域,用于跨越 NUMA 节点移动物理内存页,系统不能通过这个区域申请内存。

图 2 是从内存为 2 GB 的 Linux 系统中得到的内存分配信息。在图 2 中,前面几行是每个迁移类型可用内存的页数,最后一行是每个迁移类型的总页数。

```
# cat /proc/pagetypeinfo
Page block order: 10
Pages per block:  1024

Free pages count per migrate type at order       0      1      2      3      4      5      6      7      8      9     10
Node    0, zone   Normal, type    Unmovable     46     35     53     51     35     16      3      4      0      0      0
Node    0, zone   Normal, type  Reclaimable      0     19     14      4      6      2      0      1      1      1      0
Node    0, zone   Normal, type      Movable     24     62     30     14      1      4      1      1      1      0     13
Node    0, zone   Normal, type      Reserve      0      0      0      0      0      0      0      0      0      0      2
Node    0, zone   Normal, type      Isolate      0      0      0      0      0      0      0      0      0      0      0

Number of blocks type     Unmovable  Reclaimable      Movable      Reserve      Isolate
Node 0, zone   Normal            7            1           25            2            0
```

图 2 2 Gb 内存分配图

2.2 内存回收策略

(1)定期检查

系统的内存使用量是由 kswapd 进程定期检查的,kswapd 进程在后台运行,当它检测到系统固定的阈值大于空闲的物理页面数时,系统就会进行页面回收。

(2)内存不足

在某些情况下,操作系统会突然需要通过伙伴系统为用户进程分配大量内存,或是需要新建一个较大的缓冲区,而此时若没有满足要求的物理内存,操作系统就需要尽快执行页面回收操作,释放出部分内存空间。这种页面回收方式也被称作“直接页面回收”,它采用页面回收算法(PFRA)。PFRA 以获取页框并使它空余为目的,它根据页框所包含的内容而使用不同的处理方式。页框的内容分为不可回收页面、可交换页面、可同步页面和可丢弃页面。许多种属于进程的用户态、磁盘和高速缓存内存的页必须通过 PFRA 处理,处理时都根据试探法的几条准则进行,处理流程如图 3 所示。

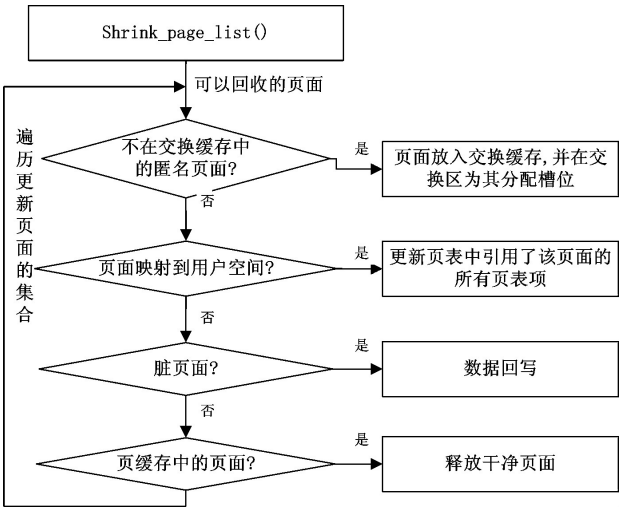

图 3　PFRA 处理流程

系统在通过内存回收策略回收页面后也无法满足内存的需求时,会做出 OOM(out of memory) killer 的决策。这是一种内存耗尽时管理内存的处理机制:操作系统会挑选正在运行中最恰当的一个进程,将其杀掉并将它所占用的页面全部释放。

内存回收机制主要依赖于 pages_min、pages_low 和 pages_high 三个字段。这三个字段在每个 zone 的描述符中都有各自的定义:pages_min 是区域预留的页面数目,pages_low 是控制进行页面回收的最小阈值,pages_high 是控制进行页面回收的最大阈值。如果 pages_min 大于空闲的物理页面数,那么系统运行会有较大的压力,需要进行页面回收。如果 pages_low 高于空闲的物理页面数,那么页面回收操作就会在系统内核中开始进行。pages_high 低于空闲的物理页面数则是内存区域最理想的状态。

3　问题与解决方案

3.1　内存管理问题

系统在初始化时会根据内存的大小计算出一个回收内存的阈值,用来控制系统的空闲内存。阈值越低,内存回收开始的越晚,空闲内存越小。其计算规则是:

min_free_kbytes=sqrt (lowmem_kbytes*16)=4*sqrt(lowmem_kbytes)

其中 lowmem_kbytes 是指系统内存大小。这种规则计算出来的阈值的范围为 128 K~64 M。由于 min_free_kbytes 不是随着内存的增大而线性增大,因此也无需按线性预留出过多的内存,只要能保证系统的紧急使用量即可。

Linux 为内存的使用设置了三种内存水位标记,watermark[high]、watermark[low]和 watermark[min]。它们所标记的含义分别为:剩余内存在 high 以上,表示内存剩余较多,目前内存使用压力不大;在 high 到高于 low 的范围内,表示目前剩余内存使用存在一定压力;在 low 到高于 min 的范围内,表示内存开始有使用压力,剩余内存不多;min 是最小的水位标记,当剩余内存达到这个状态时,就说明使用面临很大压力。小于 min 的这部分内存是内核保留给特定情况使用的,一般不会分配。内存回收行为是基于剩余内存的水位标记进行决策的:当系统剩余内存低于 watermark[low]的时候,内核的 kswapd 开始起作用,进行内存回收,直到剩余内存达到 watermark[high]的时候停止。如果内存消耗导致剩余内存达到或超过 watermark[min],就会触发直接回收(direct reclaim)。

min_free_kbytes 的主要用途是计算影响内存回收的三个参数 watermark [min/low/high],等同于 page_min、page_low 和 page_high。

page_min=watermark[min]=min_free_kbytes

page_low=watermark[low]=watermark[min]*5/4

page_high=watermark[high]=watermark[min]*3/2

min_free_kbytes 的值设得越大,watermark 的水平标记参数值越高,三个标记参数值之间的 buffer 量也越大,这会使系统较早启动 kswapd 进行回收,且只有内存回收至 watermark[high]时才会停止,导致系统预留过多空闲内存,降低了应用程序可使用的内存量。在极端情况下,min_free_kbytes 设置的值接近物理内存大小,就可能会导致因应用程序可用的内存太少而频繁地 OOM。而 min_free_kbytes 设得过小,则会导致系统预留内存过小。

kswapd 回收的过程中也会有少量内存的分配行为标志 PF_MEMALLOC,这是一个进程标志,这个标志可允许 kswapd 使用预留内存。被 OOM 选中杀死的进程在退出的过程中,如果需要申请内存也可以使用预留的内存。在以上两种情况下,使用预留内存可以避免系统进入 deadlock 状态。

系统通过网络接收一段升级包后内存的变化如图 4 所示。在图 4 中,一段 34 M 的 mtd 工具升级包被分为三个部分:rootf.bin、kernel.bin 和 app.bin。系统通过网络先接收 rootfs.bin,并以 mallco rootfs.bin 大小的内存存放它,再以 fwrite 的方式,将这块内存写入文件节

点/tmp 目录下，写完之后，再释放这块内存，然后将写完的/tmp/rootfs.bin 通过烧写命令 nandwrite 写入某个/mnt/mtdblock 分区下。在 nandwrite 操作完成后，系统会删除 rootfs.bin 并执行 rm-rf rootfs.bin。接着，系统开始接收下一个升级包 app.bin+kernel.bin，然后以 fwrite 方式将包含 kernel.bin 大小的内存写入/tmp 目录下，再将 kernel.bin 写入 nand 分区节点下，之后释放掉 kernel.bin。接下来再处理 app.bin，处理方式是用 fwrite 先将 app.bin 写入挂载 nand 分区下，再对其进行解压。最后，系统删除 app.bin，整个流程结束。

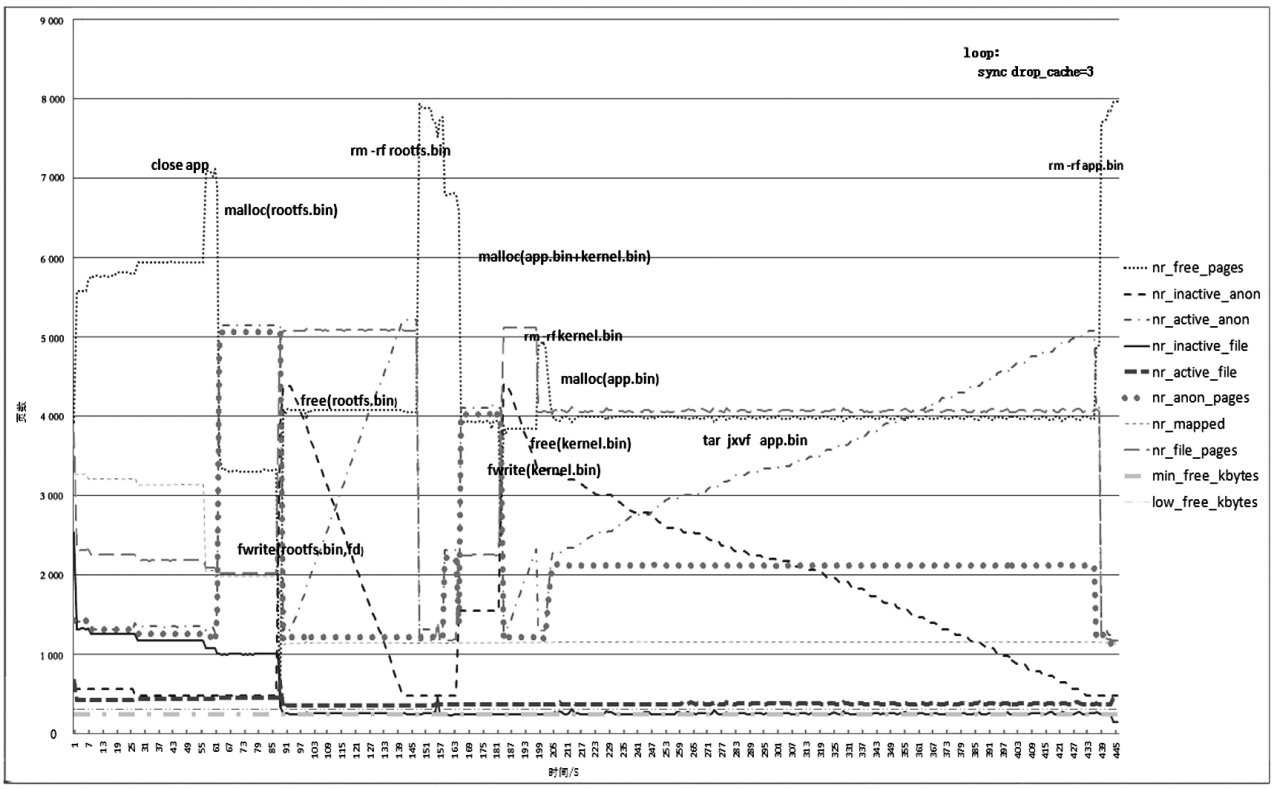

图 4　内存变化图

3.2　解决方案

在图 4 中，执行 malloc（rootfs.bin）（在内存的动态存储区中分配一段连续空间）后，nr_free_pages 会急剧减少，匿名内存页急剧增多，正在使用的匿名内存也跟着增多。执行 fwrite 时，nr_file_pages 明显增加，说明内存中正在写入文件，执行 free 时剩余内存回升。可见，当 Linux 可使用的内存较小时，不能一次性 malloc（分配）较大内存，以避免在剩余内存急剧减小到回收水位线时引发 OOM 事件。正确的做法应是多次 malloc 小内存，分批次执行 fwrite，先将 buffer 写入/tmp 目录下，再执行 free，以使内存变化比较平滑。

在 1 Gb 内存环境中，通过不断调整水位线（min_free_kbytes）的方式测试出剩余可用内存为 29 M 左右，而在一次性读入文件到更新文件的过程中，29 M 内存是无法满足要求的，需要更改在线升级方案。更改的方案有两种：第一种是定时清理内存，在 daemon_server 中加入定时清理内存线程，并在需要升级时开启定时清理内存命令；第二种是更改升级时接收文件的机制。第二种方案因为在 tcp 接收更新文件时，不仅需要 malloc 与文件大小相同的内存，还需要执行 fwrite 将 malloc 出来的 buffer 写入/tmp 目录下，而这一步也需要内存支持，增加了内存开销。图 5 为 TCP 接收更新文件时内存的变化图。在图 5 中，上述过程显示为在 31 s 处出现一个突降。

由图 5 可知，第一次执行 fwrite 时，在 15 s 处出现大突降，而在后面解压 app.bin 的过程中，内存一直维持在水位线上，因为总内存较小，所以设定水位线原始默认值为 971 K。据此，本文将内存 malloc 成小块，采用单个小块接收，接收完之后进行 MD5 校验，如果校

验无误，再对每一小块都执行 fwrite，将它们都写入到/tmp 目录下,每成功写入一块,就 free 相应内存,这样不会造成内存突降,而且可增加系统运行的稳定性。具体实施方案如图 6 所示。

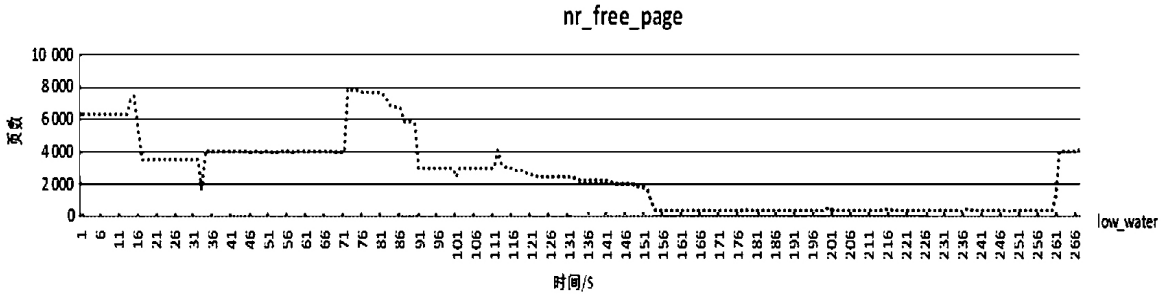

图 5　tcp 接收更新文件内存变化图

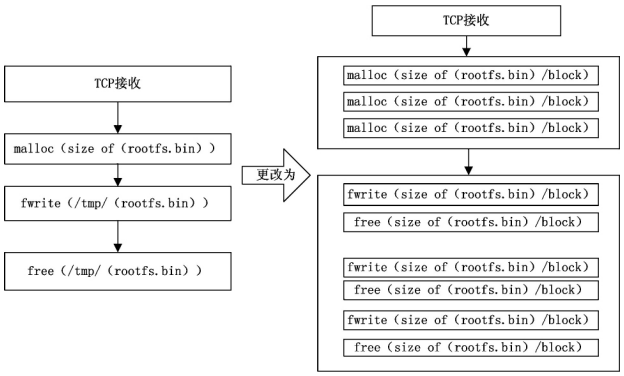

图 6　更新 tcp 接收文件机制框图

由以上分析可知，结合自刷新内存 cache 和改变 malloc 方式的方案可以使内存变化更为平缓，同时避免 OOM 事件发生。最终内存优化结果如图 7 所示。

4　结束语

本文讨论了一种 Linux 内存管理机制的方法,并给出优化结果。通过打 patch 补丁,修改内存管理器和设低内存水位标记,调整了内存消耗,避免了 OOM 事件发生。结合定时清理内存和改变内存分配,优化了内存使用,使内存变化更加缓和,提高了系统的稳定性。

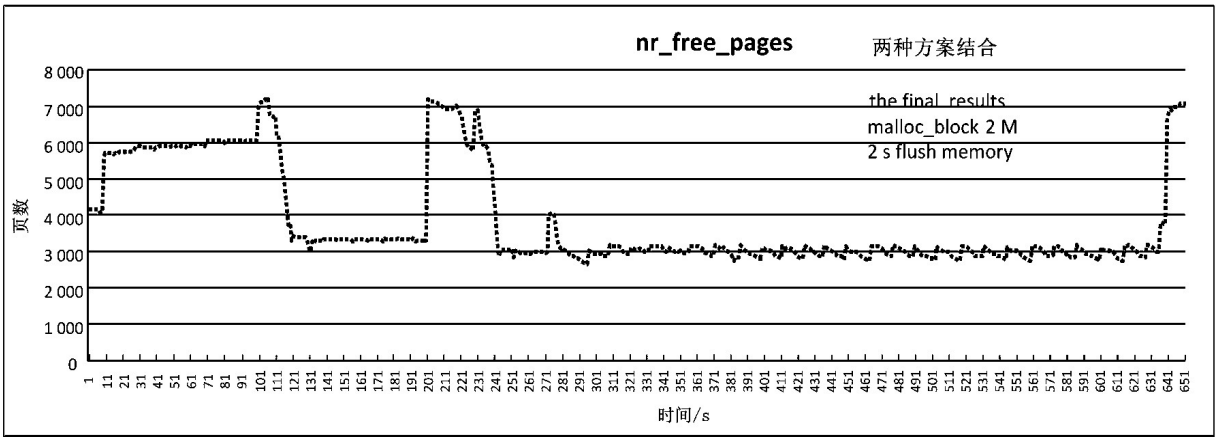

图 7　最终内存使用优化结果图

参考文献:

[1] 田泉，艾丽蓉，陈杰.Linux 实时内存的研究与实现[J].微电子学与计算机,2014(8):45-48.

[2] 肖竟华,陈岚.Linux 内存管理实现的分析与研究[J].计算机技术与发展,2007(2):66-67.

[3] CORBET J, RUBINI A, KROAH-HARTMA G. Linux Device Drivers[M]. 3rd Editon. American: Oreilly and Associates Inc, 2005: 85-88.

[4] BOVET D P, CESATI M.深入 Linux 内核[M].2 版.陈莉君,冯锐,牛欣源,译.北京:中国电力出版社,2003:101-103.

[5] 于国龙,崔忠伟,左羽.嵌入式 Linux 内存管理的优化[J].四川理工学院学报(自然科学版),2015(8):33-36.

[6] SADAMOO.Linux 页面回收浅析[EB/OL].(2013-07-30)[2016-05-13].http://blog.csdn.net/sadamoo/article/details/9625963.

【责任编辑　梅欣丽】

A Research Method for Memory Management Mechanism in Linux

XIE Wendi, LENG Xianjin, JIN Jian

(College of Electronics and Communication Engineering, Anhui Xinhua University, Hefei 230088)

Abstract: Based on the research of Linux memory management, this paper elaborated the memory management model, slab allocation process of buddy system and memory recovery strategy. It put forward a method of avoiding the occurrence of OOM (out of memory) event and the problem of running out of memory by modifying the memory manager and adjusting water level line in Linux. It also analyzed the process of the memory in detail.

Keywords: memory management; memory recovery strategy; water level line; OOM

(上接第 4 页)

[11] 张恭庆,林源渠.泛函分析讲义[M].北京:北京大学出版社,2012:48.

【责任编辑　王云鹏】

The Permanence for a Ratio-dependent Nonautonomous System with Machaelis-Menten Functional Response and Diffusion

LIANG Guizhen[1], SONG Ge[1,2]

(1. College of Mathematics and Information Science, Xinxiang University, Xinxiang 453003, China;
2. College of Mathematics and Statistics, Zhengzhou University, Zhengzhou 450001, China)

Abstract: A kind of ratio-dependent nonautonomous diffusion predator-prey system with Machaelis-Menten functional response was established in this paper, and the sufficient conditions for the persistence of this system were obtained by comparison theorem. When the system was periodic, sufficient conditions that ensure the existence, uniqueness, global asymptotic stability and positive periodic solution of this system were obtained by constructing a Liapunov function.

Keywords: diffusion; Machaelis-Menten functional response; prey-predator system; persistence; global asymptotic stability