**Linux Memory Management**

**فهرست مندرجات :**

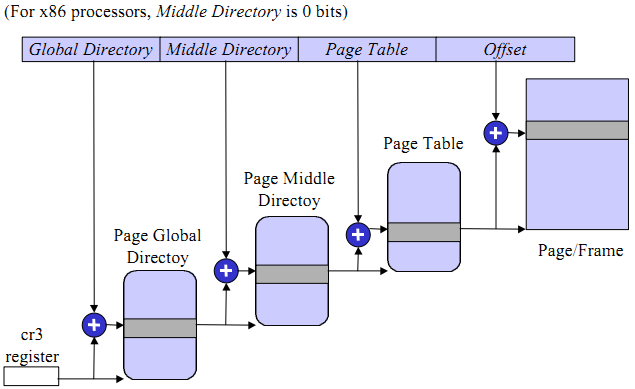
1. مکانیزم Paging
2. فضای آدرس فیزیکی و منطقی حافظه
3. مدیریت Page Frame در لینوکس
4. انواع مدیریت حافظه های استفاده شده در لینوکس
   1. تخصیص پیوسته
      1. الگوریتم Buddy
      2. Slab Allcovator
   2. تخصیص ناپیوسته
5. فضای آدرس دهی پردازه ها(Processes)
6. رسیدگی کننده(Handler) Page Fault
7. نحوه اختصاص حافظه به پردازه جدید
8. مدیریت حافظه Heap
9. منابع تحقیقاتی

**مکانیزم Paging**

* پردازنده هایی از قبیل Intel X86 توانایی پشتیبانی از شیوه Segment را دارا می باشند.
* ولی در لینوکس از شیوه Segment استفاده نمیشود، چراکه :
  1. استفاده از مقدار ثبات Segment یکسان برای تمام پردازه ها(Process)، مدیریت حافظه را آسان میکند.
  2. حفظ قابلیلت استفاده بر روی سایر پردازنده ها (Portability).
* در شیوه Paging استفاده شده در لینوکس :
  1. اندازه هر Page File برابر 4K.
  2. یک Page Table سه لایه برای مدیریت 64 بیت خط آدرس.
  3. در پردازنده هایی از قبیل X86 :
     + تنها به دو لایه Page Table نیاز میباشد.
     + Page Table توسط سخت افزار پشتیبانی میشود.
     + امکان بهره گیری از TLB(Translation lookaside buffer) فراهم آمده است.

**فضای آدرس (Addresses Space) منطقی/ فیزیکی**

* شمای آدرس های منطقی در لینوکس :



* حدود 2 مگابایت اول **حافظه فیزیکی** بمنظور معماری کامپیوتر جاری و اطلاعات و ... سیستم عامل رزرو شده است و مابقی برای انجام Paging در دسترس میباشد.
* **فضای منطقی** آدرس های یک پردازه به دو قسمت تقسیم شده است :
  1. که شامل 0X00000000 تا PAGE\_OFFSET-1 میشود و میتوان چه در User mode و چه در Kernel mode آن را آدرس دهی کرد
  2. شامل PAGE\_OFFSET تا 0Xffffffff میشود و تنها میتوان در Kernel mode آن را آدرس دهی کرد
* PAGE\_OFFSET معمولا 0Xc0000000میباشد

**مدیریت Page Frame در لینوکس**

* کرنل وضعیت کنونی هر Page Frame را در آرایه ای از struct page ها ، بنام mem\_map ، نگه میدارد.
* همچنین کرنل تعداد استفاده از یک Page Frame را نگه میدارد . مقدار 0 بمعنای آزاد بودن Page Frame و عدد بزرگتر از 0 بمعنای تعداد استفاده از آن Page Farme میباشد.
* همچنین پرچم(Flag) هایی از قبیل Dirty، Locked، Referenced و غیره نیز برای Page Frame ها وجود دارد.
* کرنل Page Frame ها را بوسیله دستورات زیر گرفته و یا آزاد میکند :

\_\_get\_free\_pages ( gfp \_mask, order )

free\_pages ( addr, order )

* در تئوری، شیوه Paging نیاز به تخصیص حافظه پیوسته را از بین برده است - بعضی از عملیات مانند DMA در روند Paging خلل ایجاد میکنند .
* از طرف دیگر باید توجه داشت که شیوه تخصیص حافظه پیوسته Page Table کرنل را تغییر نمیدهد که در نتیجه باعث حفظ TLB میگردد و زمان دسترسی را کاهش میدهد !
* در نتیجه لینوکس از شیوه ای بمنظور تخصیص پیوسته Page Frame (allocating contiguous page frames) استفاده میکند .

در این روش مشکل External Fragmentation نیز وجود نخواهد داشت.

**انواع مدیریت حافظه های استفاده شده در لینوکس :**

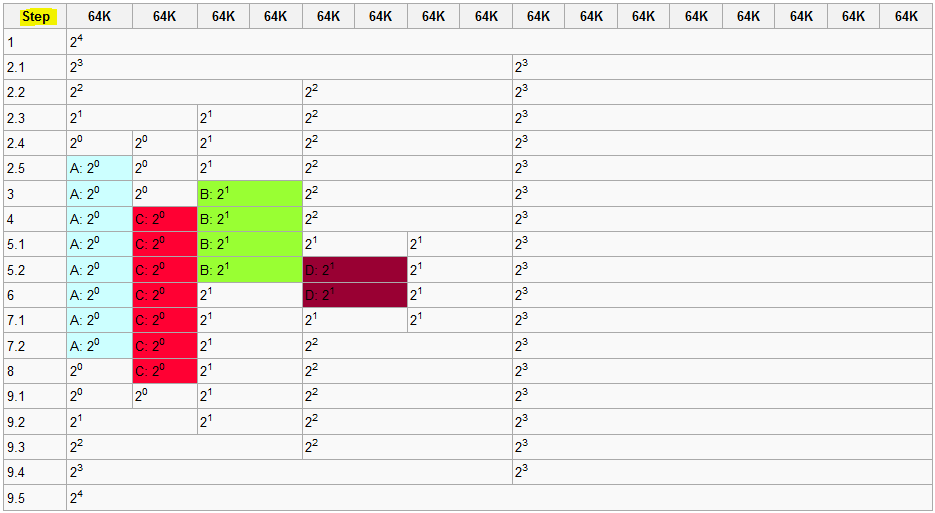
1. تخصیص حافظه پیوسته (contiguous memory allocation)
2. تخصیص حافظه ناپیوسته (noncontiguous memory allocation)

تخصیص حافظه پیوسته (contiguous memory allocation) ]4و5و6[

* در این شیوه مدیریت حافظه که یک شیوه سنتی محسوب میشود، تخصیص حافظه به پردازه ها از بلاک های متوالی حافظه صورت میگیرد.
* اشکال این شیوه : مشکل External Fragmentation .
* ویژگی این روش : در حافظه هایی با حجم بالا مشکل های مرتبط با کش(Cache) و تاخیر دسترسی (Access Latency) وجود ندارد.

الگوریتم Buddy ]7و8[

* تمامی page frame های موجود به 10 لیست از بلاک ها تقسیم‌بندی میشوند که شامل لیست هایی با 1، 2، 4، 8، 16، 32، 64، 128، 256، 512 page frame پیوسته میشود.
  + آدرس اولین page frame در یک بلاک ضریبی از سایز آن دسته میباشد. بعنوان مثال در یک بلاک 16تایی، آدرس اولین page frame برابر 16x212 خواهد بود.
* الگوریتم برای تخصیص مثلا یک بلاک 128 تایی از page frame های پیوسته :
  + ابتدا به دنبال یک بلاک خالی در لیست 128 تایی بگرد
  + اگر بلاک خالی پیدا نشد، در لیست 256 تایی بدنبال یک بلاک خالی بگرد
  + اگر بلاکی پیدا کرد، کرنل 128 تا از 256 page frame موجود را تخصیص میدهد و 128 تای باقیمانده را در لیست 128تایی ها قرار میدهد
  + اگر در لیست های بزرگتر پیدا شد، انقدر بلاک را خرد کن ( بصورت باینری) تا حافظه مورد نیاز را اختصاص بدهی و بلاک های بوجود آمده را در لیست مناسب قرار بده
  + اگر هیچ بلاکی قابل تخصیص نیست یک پیغام خطا بازگردان
* یک مثال از نحوه تخصیص حافظه (Buddy System) :



This allocation could have occurred in the following manner

1. The initial situation.
2. Program A requests memory 34K, order 0.
   1. No order 0 blocks are available, so an order 4 block is split, creating two order 3 blocks.
   2. Still no order 0 blocks available, so the first order 3 block is split, creating two order 2 blocks.
   3. Still no order 0 blocks available, so the first order 2 block is split, creating two order 1 blocks.
   4. Still no order 0 blocks available, so the first order 1 block is split, creating two order 0 blocks.
   5. Now an order 0 block is available, so it is allocated to A.
3. Program B requests memory 66K, order 1. An order 1 block is available, so it is allocated to B.
4. Program C requests memory 35K, order 0. An order 0 block is available, so it is allocated to C.
5. Program D requests memory 67K, order 1.
   1. No order 1 blocks are available, so an order 2 block is split, creating two order 1 blocks.
   2. Now an order 1 block is available, so it is allocated to D.
6. Program B releases its memory, freeing one order 1 block.
7. Program D releases its memory.
   1. One order 1 block is freed.
   2. Since the buddy block of the newly freed block is also free, the two are merged into one order 2 block.
8. Program A releases its memory, freeing one order 0 block.
9. Program C releases its memory.
   1. One order 0 block is freed.
   2. Since the buddy block of the newly freed block is also free, the two are merged into one order 1 block.
   3. Since the buddy block of the newly formed order 1 block is also free, the two are merged into one order 2 block.
   4. Since the buddy block of the newly formed order 2 block is also free, the two are merged into one order 3 block.
   5. Since the buddy block of the newly formed order 3 block is also free, the two are merged into one order 4 block.

As you can see, what happens when a memory request is made is as follows:

* If memory is to be **allocated**

1. Look for a memory slot of a suitable size (the minimal 2k block that is larger or equal to that of the requested memory)
   1. If it is found, it is allocated to the program
   2. If not, it tries to make a suitable memory slot. The system does so by trying the following:
      1. Split a free memory slot larger than the requested memory size into half
      2. If the lower limit is reached, then allocate that amount of memory
      3. Go back to step 1 (look for a memory slot of a suitable size)
      4. Repeat this process until a suitable memory slot is found

* If memory is to be **freed**

1. Free the block of memory
2. Look at the neighboring block - is it free too?
3. If it is, combine the two, and go back to step 2 and repeat this process until either the upper limit is reached (all memory is freed), or until a non-free neighbour block is encountered

* وقتی یک بلاک آزاد میگردد، کرنل سعی بر ادغام جفت های مشابه(free Buddy Block) با سایز b به یک 2b دارد.
* دو بلاک buddy تلقی میشوند اگر :

1. اگر هم سایز باشند
2. هر دو در آدرس فیزیکی متوالی واقع شده باشند
3. آدرس فیزیکی page اول از بلاک اول ضریبی از 2bx212 باشد

* عملیات ادغام متوالیا تکرار میشود.
* لینوکس از دو نوع Buddy System متفاوت برای مدیریت حافظه استفاده میکند.
  + یکی برای page frame های مناسب برای DMA (بعنوان مثال page frame هایی با آدرس هایی کمتر از 16MB).
  + دیگری برای سایر.
* هر Buddy System مبتنی است بر ..
  + آرایه توصیفگر frame ها ، mem\_map
  + آرایه 10 تایی از free\_area\_struct، هر درایه برای یک سایز(1، 2، ... ، 512 )( هر درایه یک لینک لیست دوطرفه از بلاک های page frame با سایز مربوطه خود میباشد) .
  + ده بیت‌مپ (Bitmap) برای نگهداری وضعیت تخصیص بلاک های موجود.
* الگوریتم Buddy برای مدیریت حجم های بالای حافظه بخوبی عمل میکند. اما در رابطه با بخش های کوچک حافظه با مشکل Internal Fragmentation مواجه است !
* برای حل این مشکل از نسخه 2.2 لینوکس slab allocator معرفی گردید

مکانیزم Slab Allocator ]9و10[

* ایده اصلی : نگهداری اشیا(objects) بتازگی استفاده شده در کش
* در این شیوه :
  + قسمت های مختلف حافظه بعنوان یک اشیا همراه با اطلاعات و متدها ( مانند سازنده و مخرب و ... ) دیده میشوند.
  + اشیا بصورت دسته بندی شده به کش فرستاده میشوند
  + یک مجموعه ویژه در کش برای عملیات های سیستم عامل در نظر گرفته میشود
  + Slab cache شامل هیچ یا چند slab میشود. slab یک یا چند page frame پیوسته از Buddy System میباشد.
  + اشیا بوسیله دستور kmem\_cache\_alloc(cachep) تخصیص می یابند ( cachep یک اشاره‎گر به کش میباشد).
  + اشیا بکمک دستور kmem\_cache\_free(cachep, objp) آزاد میشوند.
  + یک گروه از کش های عمومی وجود دارد که با سایز های گسترده ای بین 32 تا 131072 بایت توزیع شده است.
  + برای گرفتن شی ای از این کش ها از دستور kmalloc(size, flags) استفاده میشود.
  + برای آزاد کردن شی ای از این کش ها عمومی از دستور kfree(objp) استفاده میشود.

تخصیص حافظه ناپیوسته (noncontiguous memory allocation) ]6[

* در این شیوه از یک حافظه مجازی(Virtual Memory) برای پیاده سازی شیوه سنتی " تخصیص حافظه ناپیوسته " استفاده میگردد.
* یک فضا در فضای آدرس مجازی بین VMALLOC\_START و VMALLOC\_END رزرو شده است. مکان VMALLOC\_START به میزان حافظه فیزیکی بستگی دارد ولی محدوده همیشه حداقل VMALLOC\_RESERVE میباشد که بر روی یک X86 برابر 128 مگابایت میباشد.
* لینوکس از تخصیص حافظه ناپیوسته عموما پرهیز میکند. اما گاها برای اختصاص حافظه به گونه ای از درخواست های محدود استفاده از این روش کارامدتر میشود.
* لینوکس بیشتر آدرس های بالاتر از PAGE\_OFFSET را بمنظور تخصیص ناپیوسته استفاده میکند.
* برای تخصیص و آزاد کردن حافظه به روش ناپیوسته از دستورات vmalloc(size) و vfree(addr) استفاده میگردد.

مرور : انواع تخصیص حافظه در کرنل لینوکس

* کرنل حافظه پویا خود را به یکی از 3 روش زیر میگیرد :

1. استفاده از \_\_get\_free\_pages() برای گرفتن pages از Buddy System
2. kmem\_cache\_alloc() یا kmalloc() برای استفاده از تخصیص‌گر slab برای گرفتن اشیا خاص یا عمومی
3. vmalloc() برای گرفتن حافظه ناپیوسته

**فضای آدرس دهی پردازه ها**

* فضای آدرس(Address Space) یک پردازه، شامل تمام آدرس های منطقی قابل دسترس توسط پردازه میشود.
* فضای آدرس هر پردازه مستقل از دیگری است (مگر اینکه مشترک باشد) .
* کرنل عملیات تخصیص آدرس منطقی برای پردازه ها را در یک بازه بنام "مناطق حافظه"(Memory Reigns) انجام میدهد. این ناحیه یک مقدار اولیه و یک طول دارد که مضربی از 4096 می باشد.
* برای کرنال، دو دسته درخواست حافظه اتفاق می افتد :

1. درخواست های غیر ضروری :
   1. نیازی به تمام Page ها همیشه نیست
   2. تخصیص حافظه ممکن است برای لحظه ای امکان پذیر نباشد
2. درخواست های نامطمئن
   1. کرنل باید آمادگی روبرو شدن با خطا را داشته باشد

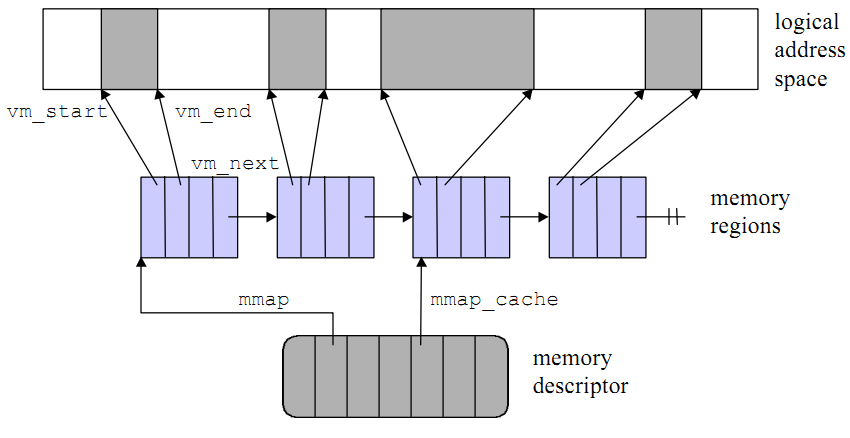
* در نتیجه کرنل لینوکس با توجه به پردازه، عملیات تخصیص حافظه را گوناگون انجام میدهد.
* شرایطی که پروسه ها حافظه جدید میگیرند :

1. ایجاد یک پروسه جدید (fork())
2. بارگزاری یک برنامه جدید (execve())
3. نگاشت حافظه به فایل (mmap())
4. رشد پشته(Stack)
5. ایجاد حافظه اشتراکی (shmat())
6. گسترش heap (malloc())

* تمامی اطلاعات مرتبط با فضای آدرس یک پردازه در بخش توصیفگر حافظه (memory descriptor) (mm\_struct) قرار گرفته است که از طریق فیلد mm از توصیفگر پردازه(process descriptor) قابل دسترس میباشد. بعضی از اطلاعاتی که قابل دسترس میباشد :

1. یک اشاره گر به بالاترین مرحله Page Table یا همان Page Global Directory از طریق pgd
2. تعداد page frameهای تخصیص یافته از طریق rss
3. ...

* توصیفگر حافظه از کش اختصاص‌گر slab (slab allocator cache) و با دستور mm\_alloc() تخصیص می یابد.
* شمایی انتزاعی از توصیفگر حافظه، مناطق حافظه و فضای آدرس منطقی



* برای تخصیص و آزاد سازی یک آدرس مجازی، کرنل بترتیب از دستورات Do\_mmap() و do\_munmap() استفاده میکند.

**رسیدگی کننده Page Fault**

* زمانی که یک پردازه از کرنل تقاضای حافظه بیشتر میکند، پردازه تنها فضای آدرس منطقی بیشتر میگیرد و نه حافظه ی فیزیکی بیشتر .
* زمانی که پردازه قصد استفاده و دسترسی به فضای آدرس منطقی جدید خود را میکند، یک Page Fault رخ میدهد تا به کرنل بگوید که واقعا به این حافظه نیاز دارد.
* رسیدگی کننده Page Fault ، آدرس منطقی تقاضا شده را با ناحیه ای از حافظه که در اختیار پردازه هست مقایسه میکند تا یا متوجه خطا در تقاضا شود (عدم امکان دسترسی پردازه به آن آدرس مجازی) و یا متوجه نیاز پردازه به اختصاص از حافظه فیزیکی شود. همچنین آدرس تقاضا شده، ممکن است بدلایلی از روی حافظه فیزیکی به روی دیسک سخت منتقل شده باشد.

نحوه اختصاص حافظه به پردازه جدید

* زمانی که کرنل یک پردازه جدید ایجاد میکند، به آن یک فضای آدرس کاملا جدید نمیدهد و در ابتدا پردازه جدید از فضای آدرس پردازه پدر خود بصورت اشتراکی استفاده میکند. کرنل در این زمان از page farme های مشترک محافظت میکند.
  + سپس اولین مرتبه ای که پردازه پدر یا پردازه جدید قصد نوشتن بر روی page frame های مشترک را داشته باشند یک استثنا (exception) رخ میدهد.
  + کرنل این استثنا را دریافت کرده و یک نسخه کپی از آن farme ، برای پردازه ای که قصد نوشتن داشت، ایجاد میشود.

مدیریت حافظه Heap

* هر پردازه میتوانند بر روی heap خود، حافظه پویا جدید بدست بیاورد .
* دستورات زبان C بمنظور انجام تغییرات در Heap : malloc(), calloc() وfree().
* تمامی این دستورات از دستور اصلی brk() مشتق شده اند.
  + این تنها دستوری میباشد که یک فراخوانی سیستمی (System Call) را انجام میدهد.
  + این دستور مستقیبا میزان حافظه Heap را تغییر میدهد.
  + این دستور در حقیقت عمل تخصیص و رها سازی فضای آدرس منطقی را برعهده دارد.
* زمانی که پردازه یک page frame میگیرد، عمل تخصیص قطعه حافظه های کوچک (مانند: malloc ( sizeof(char)\*50) در user space صورت میگیرد.

منابع تحقیقاتی

منبع اصلی:

www.inf.fu-berlin.de/lehre/SS01/OS/Lectures/Lecture14.pdf

سایر منابع:

[1] http://tldp.org/LDP/tlk/mm/memory.html

[2] http://en.wikipedia.org/wiki/Translation\_lookaside\_buffer

[3] http://www.cs.umd.edu/~meesh/cmsc411/website/saltz/cs412/lect3.html

[4] http://www.techopedia.com/definition/3769/contiguous-memory-allocation

[5] http://siber.cankaya.edu.tr/ozdogan/OperatingSystems/ceng328/node180.html

[6] http://www.kernel.org/doc/gorman/html/understand/understand010.html

[7]http://en.wikipedia.org/wiki/Buddy\_memory\_allocation

[8]www.cs.purdue.edu/homes/hosking/690M/p421-peterson.pdf

[9]en.wikipedia.org/wiki/Slab\_allocation

[10] http://www.kernel.org/doc/gorman/html/understand/understand011.html