[**内存管理之1.初始化**](https://www.cnblogs.com/joey-hua/p/5597705.html)

【版权所有，转载请注明出处。出处：<http://www.cnblogs.com/joey-hua/p/5597705.html> 】

Linux内核因为使用了内存分页机制，所以相对来说好理解些。因为内存分页就是为了方便管理内存。

说到内存分页，最根部的要属页目录表了，head.h中：

[?](https://www.cnblogs.com/joey-hua/p/5597705.html)

|  |  |
| --- | --- |
| 1 | extern unsigned long pg\_dir[1024];  // 内存页目录数组。每个目录项为4 字节。从物理地址0 开始。 |

然后再看head.s：

[?](https://www.cnblogs.com/joey-hua/p/5597705.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | /\*  \* head.s 含有32 位启动代码。  \* 注意!!! 32 位启动代码是从绝对地址0x00000000 开始的，这里也同样是页目录将存在的地方，  \* 因此这里的启动代码将被页目录覆盖掉。  \*/  .text  .globl \_idt,\_gdt,\_pg\_dir,\_tmp\_floppy\_area  \_pg\_dir:                            # 页目录将会存放在这里。  ... |

页目录存放的地方是从绝对地址0开始的，接下来分配页表：

[?](https://www.cnblogs.com/joey-hua/p/5597705.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | /\* Linus 将内核的内存页表直接放在页目录之后，使用了4 个表来寻址16 Mb 的物理内存。  \* 如果你有多于16 Mb 的内存，就需要在这里进行扩充修改。  \*/  # 每个页表长为4 Kb 字节（1 页内存页面），而每个页表项需要4 个字节，因此一个页表共可以存放  # 1024 个表项。如果一个页表项寻址4 Kb 的地址空间，则一个页表就可以寻址4 Mb 的物理内存。  # 页表项的格式为：项的前0-11 位存放一些标志，例如是否在内存中(P 位0)、读写许可(R/W 位1)、  # 普通用户还是超级用户使用(U/S 位2)、是否修改过(是否脏了)(D 位6)等；表项的位12-31 是  # 页框地址，用于指出一页内存的物理起始地址。  .org 0x1000             # 从偏移0x1000 处开始是第1 个页表（偏移0 开始处将存放页表目录）。  pg0:    .org 0x2000  pg1:    .org 0x3000  pg2:    .org 0x4000  pg3:    ... |

分配了4个页表空间，因为一个页表项对应的是一个页也就是4K，所以一个页表就是4K\*1024=4M，这里有4个页表所以就是16M。并且注意这里是从.org 0x1000开始的，前面4K的空间是留给整个页目录的。

下面是设置四个页目录项的属性，因为只有4个页表，所以相应的就是4个页目录项：

[?](https://www.cnblogs.com/joey-hua/p/5597705.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | # 下面4 句设置页目录表中的项，因为我们（内核）共有4 个页表所以只需设置4 项。  # 页目录项的结构与页表中项的结构一样，4 个字节为1 项。参见上面113 行下的说明。  # "$pg0+7"表示：0x00001007，是页目录表中的第1 项。  # 则第1 个页表所在的地址 = 0x00001007 & 0xfffff000 = 0x1000；  # 第1 个页表的属性标志 = 0x00001007 & 0x00000fff = 0x07，表示该页存在、用户可读写。      movl $pg0+7,\_pg\_dir             /\* set present bit/user r/w \*/      movl $pg1+7,\_pg\_dir+4           /\*  --------- " " --------- \*/      movl $pg2+7,\_pg\_dir+8           /\*  --------- " " --------- \*/      movl $pg3+7,\_pg\_dir+12      /\*  --------- " " --------- \*/ |

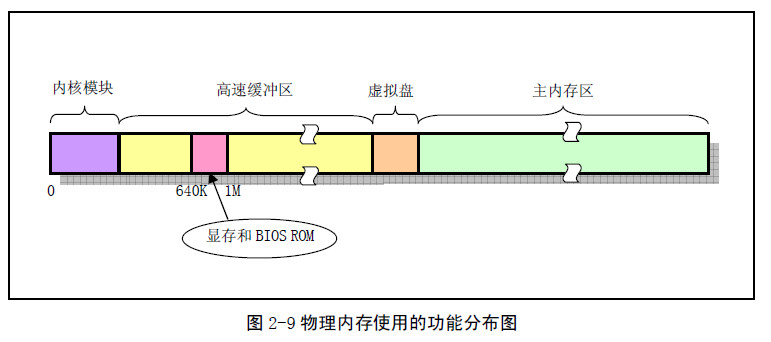
注意这里+7的意思是，先看PDE [**页目录项格式**](http://www.cnblogs.com/joey-hua/p/5382690.html) 可知，每个页目录项的0-11位是属性，12-31位才是基址，7对应的二进制是111，也就是P、R/W、U/S位都为1.

好，现在页目录和4个页表都分配好了。接下来进入初始化程序mem\_init(main\_memory\_start, memory\_end);在main.c和memory.c中：

[?](https://www.cnblogs.com/joey-hua/p/5597705.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | Static long memory\_end = 0; // 机器具有的物理内存（字节数）。  static long buffer\_memory\_end = 0;  // 高速缓冲区末端地址。  static long main\_memory\_start = 0;  // 主内存（将用于分页）开始的位置。      memory\_end = (1 << 20) + (EXT\_MEM\_K << 10);   // 内存大小=1Mb 字节+扩展内存(k)\*1024 字节。    memory\_end &= 0xfffff000; // 忽略不到4Kb（1 页）的内存数。    if (memory\_end > 16 \* 1024 \* 1024) // 如果内存超过16Mb，则按16Mb 计。      memory\_end = 16 \* 1024 \* 1024;    if (memory\_end > 12 \* 1024 \* 1024) // 如果内存>12Mb，则设置缓冲区末端=4Mb      buffer\_memory\_end = 4 \* 1024 \* 1024;    else if (memory\_end > 6 \* 1024 \* 1024) // 否则如果内存>6Mb，则设置缓冲区末端=2Mb      buffer\_memory\_end = 2 \* 1024 \* 1024;    else      buffer\_memory\_end = 1 \* 1024 \* 1024;    // 否则则设置缓冲区末端=1Mb    main\_memory\_start = buffer\_memory\_end;    // 主内存起始位置=缓冲区末端；      // 如果定义了内存虚拟盘，则初始化虚拟盘。此时主内存将减少。参见kernel/blk\_drv/ramdisk.c。  #ifdef RAMDISK          // 如果定义了内存虚拟盘，则主内存将减少。    main\_memory\_start += rd\_init (main\_memory\_start, RAMDISK \* 1024);  #endif      mem\_init (main\_memory\_start, memory\_end); |

这里结合下图参考：



如果定义了内存虚拟盘，主内存区开始位置main\_memory\_start就从虚拟盘末端开始，否则就从高速缓冲区末端开始；而主内存区结尾memory\_end则为16M。

[?](https://www.cnblogs.com/joey-hua/p/5597705.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | /\* 下面定义若需要改动，则需要与head.s 等文件中的相关信息一起改变 \*/  // linux 0.11 内核默认支持的最大内存容量是16M，可以修改这些定义以适合更多的内存。  #define LOW\_MEM 0x100000                                            // 内存低端（1MB）。  #define PAGING\_MEMORY (15\*1024\*1024)                // 分页内存15MB。主内存区最多15M。  #define PAGING\_PAGES (PAGING\_MEMORY>>12)  // 分页后的物理内存页数。相当于除以4096  #define MAP\_NR(addr) (((addr)-LOW\_MEM)>>12)   // 指定内存地址映射为页号。  #define USED 100                                                                    // 页面被占用标志，参见405 行。    static long HIGH\_MEMORY = 0;    // 全局变量，存放实际物理内存最高端地址。    // 内存映射字节图(1 字节代表1 页内存)，每个页面对应的字节用于标志页面当前被引用（占用）次数。  static unsigned char mem\_map[PAGING\_PAGES] = { 0, };      //// 物理内存初始化。  // 参数：start\_mem - 可用作分页处理的物理内存起始位置（已去除RAMDISK 所占内存空间等）。  // end\_mem - 实际物理内存最大地址。  // 在该版的linux 内核中，最多能使用16Mb 的内存，大于16Mb 的内存将不于考虑，弃置不用。  // 0 - 1Mb 内存空间用于内核系统（其实是0-640Kb）。  void  mem\_init (long start\_mem, long end\_mem)  {    int i;      HIGH\_MEMORY = end\_mem;                // 设置内存最高端。    for (i = 0; i < PAGING\_PAGES; i++)     // 首先置所有页面为已占用(USED=100)状态，      mem\_map[i] = USED;                              // 即将页面映射数组全置成USED。    i = MAP\_NR (start\_mem);                       // 然后计算可使用起始内存的页面号。    end\_mem -= start\_mem;                         // 再计算可分页处理的内存块大小。    end\_mem >>= 12;                                     // 从而计算出可用于分页处理的页面数。    while (end\_mem-- > 0)                          // 最后将这些可用页面对应的页面映射数组清零。      mem\_map[i++] = 0;  } |

首先注意PAGING\_MEMORY，因为1M以下的内存空间是内核所用，不参与分页，所以主内存大小最多为15M。所以PAGING\_PAGES就是主内存大小除以4096（一页就是4096字节）就是内存页面总数。

所以mem\_map的作用就是内存页面映射。

这里注意一下MAP\_NR(start\_mem)这个宏，用主内存区开始地址减去不参与分页的1M空间，得到从可用于分页的内存地址开始的容量，再除以4096（一页就是4096字节）就可以得到页号。

然后计算主内存区的大小，并把mem\_map中从主内存区start\_mem开始的页号置为0.

到这里为止，内存管理的初始化就算结束了！