[**加载可执行二进制文件之2.change\_ldt**](https://www.cnblogs.com/joey-hua/p/5638481.html)

前面分析完了copy\_strings函数，这里来分析另一个注意的函数change\_ldt。

先来看调用处：

|  |
| --- |
| // 根据a\_text 修改局部表中描述符基址和段限长，并将参数和环境空间页面放置在数据段末端。  // 执行下面语句之后，p 此时是以数据段起始处为原点的偏移值，仍指向参数和环境空间数据开始处，  // 也即转换成为堆栈的指针。    p += change\_ldt (ex.a\_text, page) - MAX\_ARG\_PAGES \* PAGE\_SIZE; |

解释的很清楚，也就是说p指向的是相当于在图9-23的左方添加了64M-MAX\_ARG\_PAGES \* PAGE\_SIZE的大小容量。总容量为64M。

|  |
| --- |
| struct exec  {    unsigned long a\_magic;    /\* 执行文件魔数。使用N\_MAGIC 等宏访问。 \*/    unsigned a\_text;      /\* 代码长度，字节数。 \*/    unsigned a\_data;      /\* 数据长度，字节数。 \*/    unsigned a\_bss;       /\* 文件中的未初始化数据区长度，字节数。 \*/    unsigned a\_syms;      /\* 文件中的符号表长度，字节数。 \*/    unsigned a\_entry;     /\* 执行开始地址。 \*/    unsigned a\_trsize;        /\* 代码重定位信息长度，字节数。 \*/    unsigned a\_drsize;        /\* 数据重定位信息长度，字节数。 \*/  };    // 下面对执行文件的头结构数据进行处理，首先让ex 指向执行头部分的数据结构。    ex = \*((struct exec \*) bh->b\_data);    /\* read exec-header \*//\* 读取执行头部分 \*/ |

这里的ex为读取的可执行二进制文件头部分。下面进入change\_ldt函数：

|  |
| --- |
| //// 修改局部描述符表中的描述符基址和段限长，并将参数和环境空间页面放置在数据段末端。  // 参数：text\_size - 执行文件头部中a\_text 字段给出的代码段长度值；  // page - 参数和环境空间页面指针数组。  // 返回：数据段限长值(64MB)。  static unsigned long  change\_ldt (unsigned long text\_size, unsigned long \*page)  {    unsigned long code\_limit, data\_limit, code\_base, data\_base;    int i;    // 根据执行文件头部a\_text 值，计算以页面长度为边界的代码段限长。并设置数据段长度为64MB。    code\_limit = text\_size + PAGE\_SIZE - 1;    code\_limit &= 0xFFFFF000;    data\_limit = 0x4000000;  // 取当前进程中局部描述符表代码段描述符中代码段基址，代码段基址与数据段基址相同。    code\_base = get\_base (current->ldt[1]);    data\_base = code\_base;  // 重新设置局部表中代码段和数据段描述符的基址和段限长。    set\_base (current->ldt[1], code\_base);    set\_limit (current->ldt[1], code\_limit);    set\_base (current->ldt[2], data\_base);    set\_limit (current->ldt[2], data\_limit);  /\* make sure fs points to the NEW data segment \*/  /\* 要确信fs 段寄存器已指向新的数据段 \*/  // fs 段寄存器中放入局部表数据段描述符的选择符(0x17)。    \_\_asm\_\_ ("pushl $0x17\n\tpop %%fs"::);  // 将参数和环境空间已存放数据的页面（共可有MAX\_ARG\_PAGES 页，128kB）放到数据段线性地址的  // 末端。是调用函数put\_page()进行操作的（mm/memory.c, 197）。    data\_base += data\_limit;    for (i = MAX\_ARG\_PAGES - 1; i >= 0; i--)      {        data\_base -= PAGE\_SIZE;        if (page[i])      // 如果该页面存在，      put\_page (page[i], data\_base);  // 就放置该页面。      }    return data\_limit;        // 最后返回数据段限长(64MB)。  } |

第一二行的意思是code\_limit最少也要有一内存页的长度。

data\_limit赋值为64M。

紧接着设置当前进程的LDT的代码段和数据段的基址和段限长。

注意最后一段比较关键，从数据段末尾data\_base开始放置参数和环境空间已存放数据的页面。page是参数环境空间所有页面地址的数组。如果page[i]该页面存在，就调用put\_page函数：

|  |
| --- |
| /\*  \* 下面函数将一内存页面放置在指定地址处。它返回页面的物理地址，如果  \* 内存不够(在访问页表或页面时)，则返回0。  \*/  //// 把一物理内存页面映射到指定的线性地址处。  // 主要工作是在页目录和页表中设置指定页面的信息。若成功则返回页面地址。  // 在缺页异常的C 函数do\_no\_page()中会调用此函数。对于缺页引起的异常，由于任何缺页缘故而  // 对页表作修改时，并不需要刷新CPU 的页变换缓冲（或称Translation Lookaside Buffer，TLB），  // 即使页表项中标志P 被从0 修改成1。因为无效页项不会被缓冲，因此当修改了一个无效的页表项  // 时不需要刷新。在此就表现为不用调用Invalidate()函数。  unsigned long  put\_page (unsigned long page, unsigned long address)  {    unsigned long tmp, \*page\_table;    /\* NOTE !!! This uses the fact that \_pg\_dir=0 \*/  /\* 注意!!!这里使用了页目录基址\_pg\_dir=0 的条件 \*/    // 如果申请的页面位置低于LOW\_MEM(1Mb)或超出系统实际含有内存高端HIGH\_MEMORY，则发出警告。    if (page < LOW\_MEM || page >= HIGH\_MEMORY)      printk ("Trying to put page %p at %p\n", page, address);  // 如果申请的页面在内存页面映射字节图中没有置位，则显示警告信息。    if (mem\_map[(page - LOW\_MEM) >> 12] != 1)      printk ("mem\_map disagrees with %p at %p\n", page, address);  // 计算指定地址在页目录表中对应的目录项指针。    page\_table = (unsigned long \*) ((address >> 20) & 0xffc);  // 如果该目录项有效(P=1)(也即指定的页表在内存中)，则从中取得指定页表的地址??page\_table。    if ((\*page\_table) & 1)      page\_table = (unsigned long \*) (0xfffff000 & \*page\_table);    else      {  // 否则，申请空闲页面给页表使用，并在对应目录项中置相应标志7（User, U/S, R/W）。然后将  // 该页表的地址??page\_table。        if (!(tmp = get\_free\_page ()))      return 0;        \*page\_table = tmp | 7;        page\_table = (unsigned long \*) tmp;      }  // 在页表中设置指定地址的物理内存页面的页表项内容。每个页表共可有1024 项(0x3ff)。    page\_table[(address >> 12) & 0x3ff] = page | 7;  /\* no need for invalidate \*/  /\* 不需要刷新页变换高速缓冲 \*/    return page;          // 返回页面地址。  } |

首先通过data\_base获取到相对应的页目录项指针，如果有效则获取页表地址。比较关键的是最后一句，address>>12表示data\_base/4k，再与0x3ff（1024个项），因为这里不是字节而是索引，所以不是0xffc。然后用这个索引和page（参数和环境空间已存放数据的页面地址）绑定。

最后返回64M。