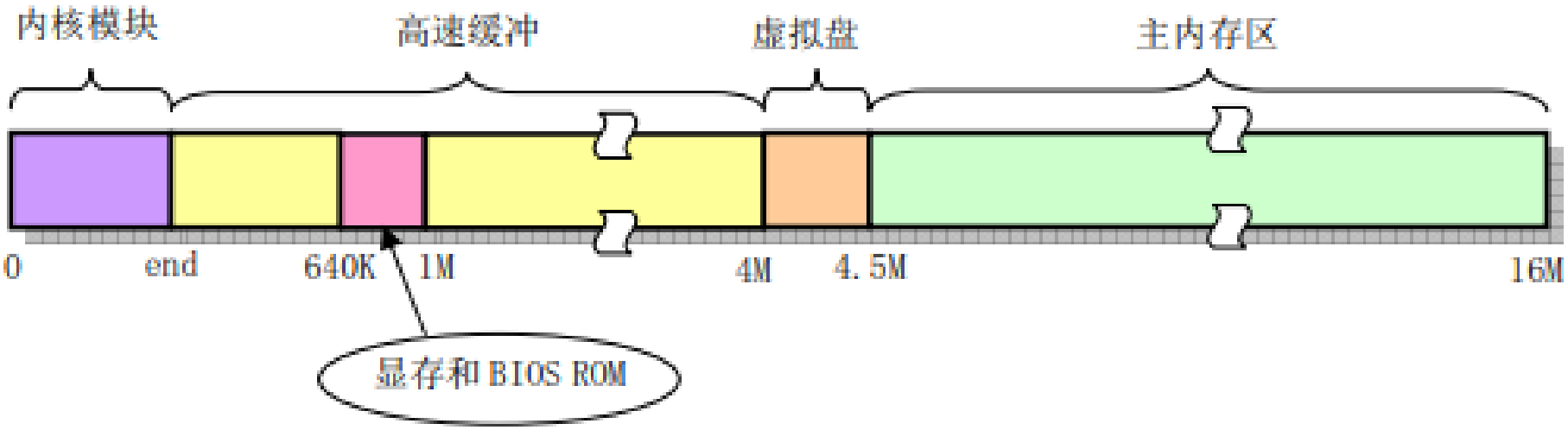
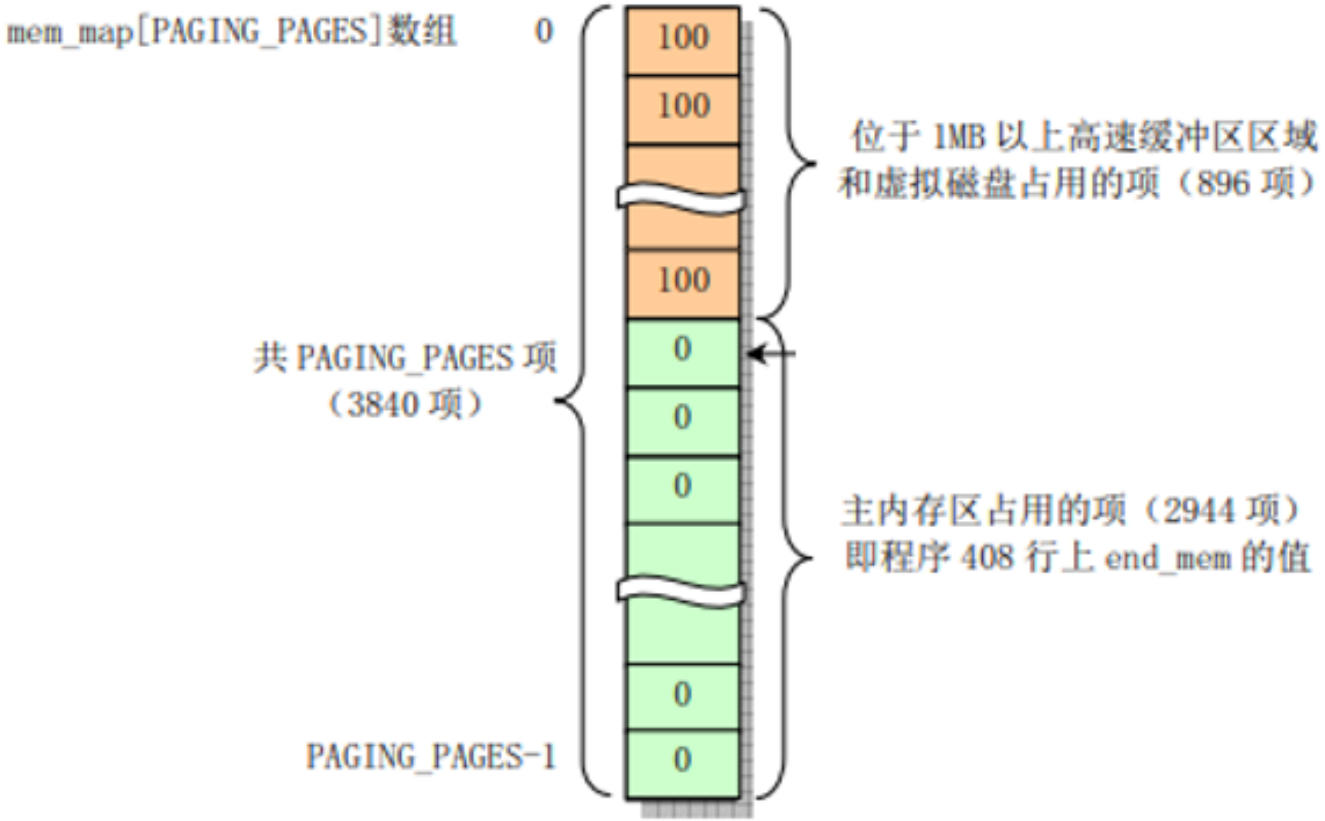
\mm\memory.c
\mm\page.s
\lib\malloc.c


内核模块
高速缓冲
虚拟盘
主内存区
0
end
640K
1M
4M
4.5M
16M
显存和 BIOS ROM


mem_map[PAGING_PAGES]数组
0
100
100
100
0
0
0
0
0
PAGING_PAGES-1
共 PAGING_PAGES 项
（3840 项）
位于 1MB 以上高速缓冲区区域
和虚拟磁盘占用的项（896 项）
主内存区占用的项（2944 项）
即程序 408 行上 end_mem 的值

```
算法：get_free_page
输入：无
输出：空闲页面物理地址
{
    从最后一项开始查找mem_map 空闲项;
    if( 没有空闲项 )
        renturn  0;
    将空闲项内容置 1，表示已经被占用;
    将空闲项对应的下标转换为对应的物理页面的物理地址＝
        ( 数组下标 <<12 ＋ LOW_MEM)
    将该物理页内容清零
    return  对应的物理地址;
}
void free_page(unsigned long addr){
    出错检查;
    mem_map[(addr>>12)--];
}
```

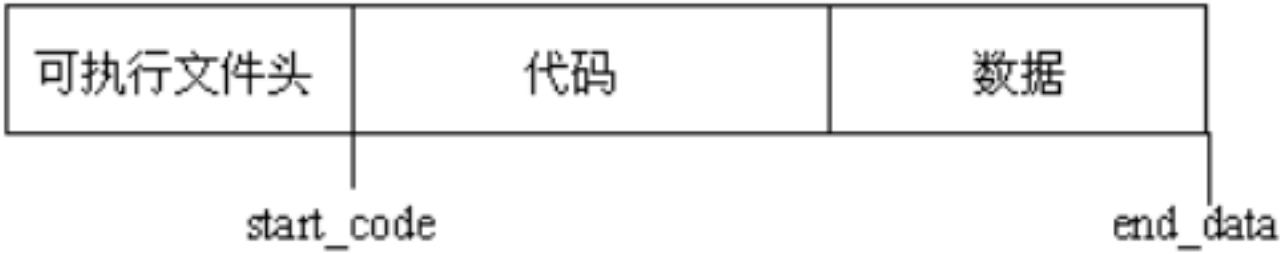
```
算法：put_page
输入：物理页面地址 page
      线性地址 address
输出：如果成功，返回 page；如果失败，返回 0
{
      根据线性地址高 10 位找到对应的页目录表项;
      if ( 页目录表项对应的页表在内存中 )
          根据页目录表项的到页表的物理地址;
        else{
          分配新的物理页面作为新的页表;
          初始化页目录表项，使它指向新的页表;
          根据页目录表项的到页表的物理地址;
           }
      根据线性地址中间 10 位找到对应的页表项;
      if( 对应的页表项已经被使用 )
          显示出错信息，返回 0;
       设置对应的页表项，使它指向物理页面;
      return 物理页面地址;
}
```

```
算法：copy_mem
输入：子进程进程号 nr
      子进程进程控制块 p
输出：如果成功，返回 0
{
      取得父进程的数据段、代码段的段限长和基地址；
      if（数据段和代码段段限长和基地址不合法）
            显示出错信息，死循环；
      设置子进程的数据段、代码段的段限长和基地址；
      共享代码段和数据段内存空间（copy_page_tables）
      if（共享失败）{
            释放子进程共享内存空间时申请的页面；
            return 共享失败；
       }
       return 0；
}
```

```
算法：copy_page_tables
输入：共享源页面起始地址 from
      共享目的空间页面起始地址 to
      被共享空间的大小 size
输出：如果成功，返回 0
{
      for( 每个要共享空间的页表项 ){
          复制页表项;
          if( 对应页不存在 )
            continue;
          if( 被共享页在主内存块映射表映射范围内 ){
              将两个页表项都置为只读;
              对应页面映射项内容加 1;
          }
          else
              只将复制的页表项置为只读;
      }
   }
  刷新页变换高速缓冲;
}
```

算法：share_page
输入：共享地址 address
输出：如果成功，返回 1

```
{
        if ( 进程 A 对应的页表项已经存在 )
            显示错误信息，死循环;
        将进程 P 对应的页表项属性设为只读;
        设置进程 A 对应地址的页表项;
        物理页引用数加 1;
        刷新页变换高速缓冲。
        return 1;
      }
    return 0;
}
```



算法：free_page_tables
输入：要释放空间起始线性地址 from
　　要释放空间大小 size
输出：如果成功，返回 0；如果失败，使调用对象进入死循环

```
{
    计算要释放的空间所占的页表数;
    for( 每个要释放的页表 ){
        for( 每个页表项 )
            if( 页表项映射有物理页面 )
                释放物理页面 free_page();
            将该页表项设为空闲;
        }
        释放页表使用的物理页;
        将该页表对应的页目录项设为空闲;
    }
    刷新页变换高速缓冲;
    return 0;
}
```

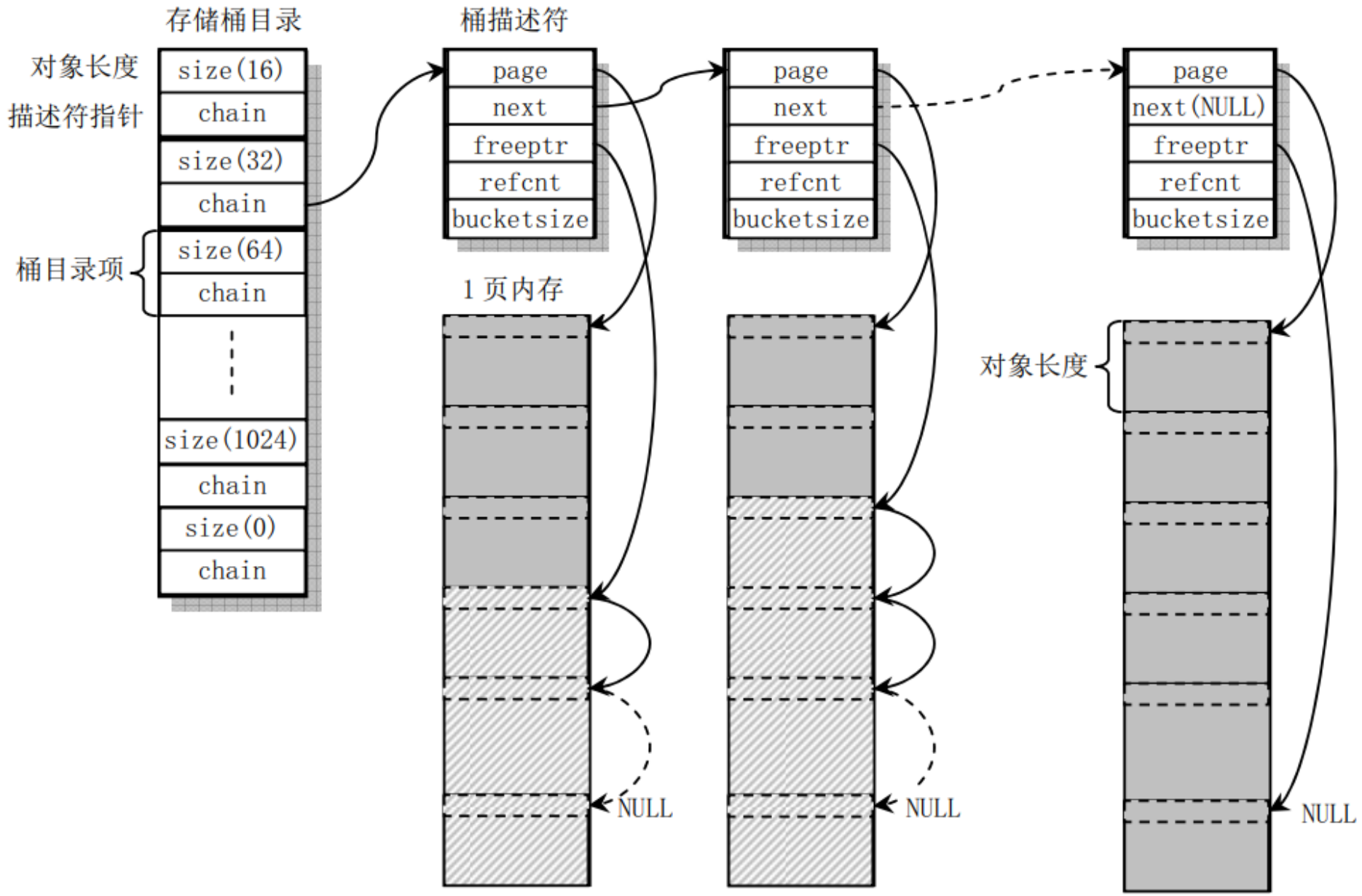
\mm\page.s

```
算法：page_fault
输入：出错码 error_code
      出错线性地址 address
输出：无
{
    保存现场;
    根据出错码判断出错原因;
    if ( 缺页 )
        作缺页处理do_no_page(error_code, address);
    else
        作写保护出错处理 do_wp_page(error_code, address);
    恢复现场;
    return;
}
```

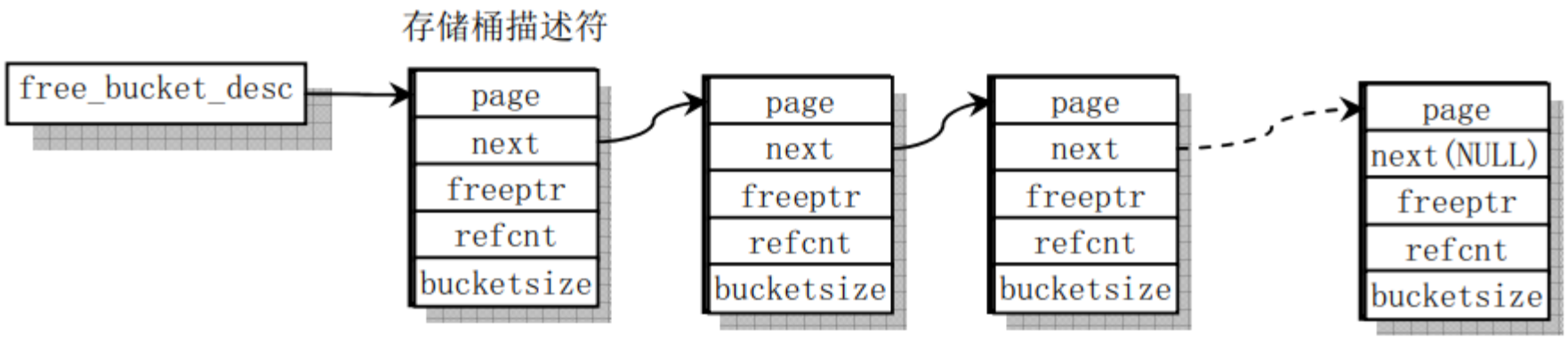
```
算法：do_no_page
输入：出错码 error_code
      出错线性地址 address
输出：无
{
    if ( 出错进程没有对应的可执行文件
       || 出错地址不在代码和数据段 )
        {
          分配物理页面并映射到出错线性地址 ( 使用 get_empty_page());
          return;
        }
    试图共享页面 ( 使用 share_page());
    if ( 共享页面成功 )
          return ;
    分配新的物理页面 (get_free_page());
    从可执行文件中将页面对应的内容读入内存;
    将页面中不属于代码段和数据段的内容清零;
    将新的物理页面映射到出错线性地址 (put_page());
}
```

```
算法：do_wp_page
输入：出错码 error_code
      出错线性地址 address
输出：无
{
    if ( 出错地址属于进程的代码段 )
        将进程终止；
    if ( 出错页面属于主内存块且共享计数为 1)
     {
        取消写保护；
        刷新页变换高速缓冲；
        return；
     }
    申请一个新的物理页；
    if ( 出错页面属于主内存块 )
        共享计数减 1；
    使出错时的页表项指向新的物理页；
    刷新页变换高速缓冲；
    复制共享页的内容到新的物理页；
    return；
}
```

\lib\malloc.c


存储桶目录
桶描述符
对象长度
描述符指针
size(16)
chain
size(32)
chain
桶目录项
size(64)
chain
size(1024)
chain
size(0)
chain
page
next
freeptr
refcnt
bucketsize
page
next
freeptr
refcnt
bucketsize
page
next(NULL)
freeptr
refcnt
bucketsize
1 页内存
NULL
NULL
对象长度
NULL

算法： init_bucket_desc
输入：无
输出：无
{
    申请一个空闲页（get_free_page）;
    for（每一个 bucket_desc 结构）
    {
        if（不是最后一个 bucket_desc 结构）
            next 指向下一个 bucket_desc 结构;
    }
    使最后一项的 next 指针指向 free_bucket_desc 指向的内容;
    使 free_bucket_desc 指向第一个 bucket_desc 结构;
    return ;



```
算法：malloc
输入：申请内存块大小 len
输出：如果成功，返回内存块指针；失败则返回 NULL；
{
    查找一个桶链表，链表中 桶的内存块是能够满足要求的最小块；
   if（没有搜索到符合要求的链）
      {
          打印出错信息：请求块过大；
          进入死循环；
       }
    关中断；
    在链表中查询还有空闲内存块的桶；
    if（链表中所有桶都没有空闲的内存块）
      {
          if（没有空闲桶描述符）
              初始化一个页面用作桶描述符（init_bucket_desc）；
          从空闲桶描述符链表中分配一个桶描述符；
          分配一个新物理页面作为桶；
          初始化桶页面；
          设置桶描述符指针；
          将新桶连入对应的链表；
      }
    从桶中分配一个空闲内存块；
    开中断；
    return 空闲内存块指针；
}
```

```
输入：释放对象指针obj
      释放对象大小 size（如果是 0，表示没有指定大小）
输出：无
{
    for（每一个桶链表）{
        if（链表中桶的内存块大小＜释放对象大小）
            continue;
        for（每一个桶）
            if（是释放对象对应的桶
                退出搜索;
    }
    if（搜索桶失败）
        显示出错信息，死循环;
    关中断;
    将要释放的内存块链入桶的空闲链表;
    修改桶的相关信息;
    if（桶中所有的内存块都是空闲的）
    {
        释放桶对应的内存块;
        释放桶对应的描述符;
    }
    开中断;
    return;
}
```