|  |
| --- |
| **《操作系统实验报告》** |
|  |
| 院 系 电子与信息工程学院 |
| 专 业 计算机科学与技术 |
| 姓 名 庄镇华 |
| 学 号 1853790 |
| 题 目 UNIX V6++存储管理部分 |
| 指导老师 方 钰 |
| 联系方式 17721295617 |

# 实验目的

结合课程所学知识，通过阅读和分析 UNIX V6++中关于内存管理部分的源代码，进一步了解操作系统内存管理模块的设计与实现细节。

# 预备知识

* 掌握进程执行过程中，堆栈段的变化和造成这种变化的原因。
* 连续存储管理方式中可变分区分配的首次适应分配算法与相应的回收过程；
* UNIX V6++进程用户态与核心态的虚实地址构成；UNIX V6++中页表机制的构成及地址转换过程。

# 实验内容

## 按照实验指导的要求仔细阅读相关代码，并完成表 1~表 5（所有斜体字部分为选作部分）。

### 表1：MemoryDescriptor 类

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 静态数据成员： | | | | | |
| 名称 | | 值 | 含义 | | |
| *USER\_SPACE\_SIZE* | | 0x800000 | 用户空间大小 | | |
| *USER\_SPACE\_PAGE\_TABLE\_CNT* | | 0x2 | 用户页表个数 | | |
| *USER\_SPACE\_START\_ADDRESS* | | 0x0 | 用户空间的起始地址 | | |
| 成员函数 | | | | | |
| 名称 | 输入参数 | | | 返回值 | 主要功能 |
| MemoryDescriptor  ~MemoryDescriptor | 无 | | | 无 | 构造函数  析构函数 |
| Initialize | 无 | | | 无 | 申请并初始化 PageDirectory |
| Release | 无 | | | 无 | 释放进程时，需调用该操作释放被占用的页表 |
| MapEntry | virtualAddress(虚拟地址)  size(需要映射的虚拟地址大小)  phyPageIdx(实物理页索引号)  isReadWrite(页属性true读写) | | | unsigned int  (结束物理页索引号) | 填充虚实映表，并返回结束页物理地址索引号 |
| MapTextEntrys | textStartAddress  (Text段虚拟地址)  textSize(Text 段大小)  textPageIdxInPhyMemory  (Text段物理地址) | | | 无 | 填充当前进程虚实映射页表中的 Text段，读写属性设置为 false |
| MapDataEntrys | dataStartAddress  (data段虚拟地址)  dataSize(data 段大小)  dataPageIdxInPhyMemory  (data 段物理地址) | | | 无 | 填充当前进程虚实映射页表中的 data段，读写属性设置为 true |
| MapStackEntrys | stackSize(stack段大小)  stackPageIdxInPhyMemory  (stack 段物理地址) | | | 无 | 填充当前进程虚实映射页表中的 stack段，读写属性设置为 true |
| 数据成员 | | | | | |
| 名称 | | 类型 | 含义 | | |
| m\_UserPageTableArray | | PageTable\* | 指向相对虚实地址映射表 | | |
| m\_TextStartAddress | | unsigned long | 代码段起始地址 | | |
| m\_TextSize | | unsigned long | 代码段长度 | | |
| m\_DataStartAddress | | unsigned long | 数据段起始地址 | | |
| m\_DataSize | | unsigned long | 数据段长度 | | |
| m\_StackSize | | unsigned long | 栈段长度 | | |

### 表2：MapNode 结构体

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 含义 |
| m\_Size | unsigned long | 分配空间的大小 |
| m\_AddressIdx | unsigned long | 分配空间的起始地址 |

### 表3：Allocator 类

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 静态数据成员： | | | | | | |
| 名称 | | 类型 | | 含义 | | |
| *m\_Instance* | | Allocator | | Allocator 的实例 | | |
| 静态成员函数 | | | | | | |
| 名称 | 输入参数 | | 返回值 | | 主要功能 |
| GetInstance | 无 | | 无 | | 返回 Allocator 的实例 |
| 成员函数 | | | | | |
| 名称 | 输入参数 | | 返回值 | | 主要功能 |
| Alloc | map  (空闲空间索引表)  size  (需要空间的大小) | | unsigned long  (为其分配空间的起始地址) | | 使用首次适应算法进行空间的分配，并返回为其分配空间的起始地址 |
| Free | map  (空闲空间索引表)  size  (回收空间大小)  addrIdx  (需回收空间的起始地址) | | unsigned long  成功则返回 0 | | 回收空间，并处理回收时可能的多种情况 |

### 表 4：KernelAllocator 类

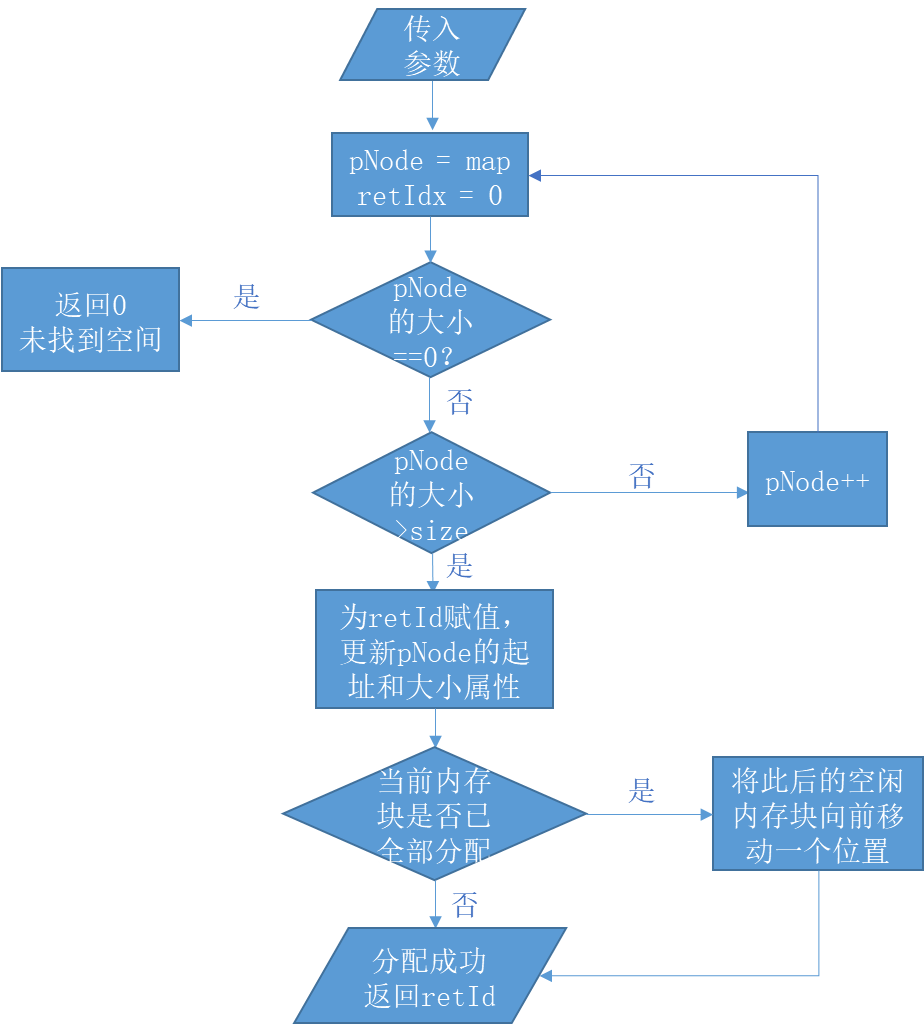
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 静态数据成员： | | | | | |
| 名称 | | 值 | | 含义 | |
| *MEMORY\_MAP\_ARRAY\_SIZE* | | 0x200 | | 最多可以分配的对象个数为 512 个 | |
| *KERNEL\_HEAP\_START\_ADDR* | | 0x180000 + 0xC0000000 | | 内核堆的起始地址 | |
| *KERNEL\_HEAP\_SIZE* | | 0x80000 | | 内核堆的大小 | |
| 成员函数 | | | | | |
| 名称 | 输入参数 | | 返回值 | | 主要功能 |
| KernelAllocator | Allocator\* allocator | | 无 | | 构造函数 |
| ~KernelAllocator | 无 | | 无 | | 析构函数 |
| Initialize | 无 | | int  成功则返回0 | | 对 map 进行初始化，将整个 512K 内存堆赋给 map[0]，空闲空间索引表其他部分全部赋值为0 |
| AllocMemory | size(需要分配空间的大小) | | unsigned long  (分配空间的首地址) | | 分配 size 大小的空间，并返回分配空间的首地址 |
| FreeMemeory | size  (需要回收空间的大小)  memoryStartAddress  (需要回收空间的首地址) | | unsigned long  成功则返回 0 | | 回收位于memoryStartAddress 的size 大小的空间，并处理回收时可能的多种情况 |
| 数据成员 | | | | | |
| 名称 | 类型 | | 含义 | | |
| map | mapNode数组 | | 内核空闲空间索引表 | | |

### 表 5：PageManager 类及两个子类

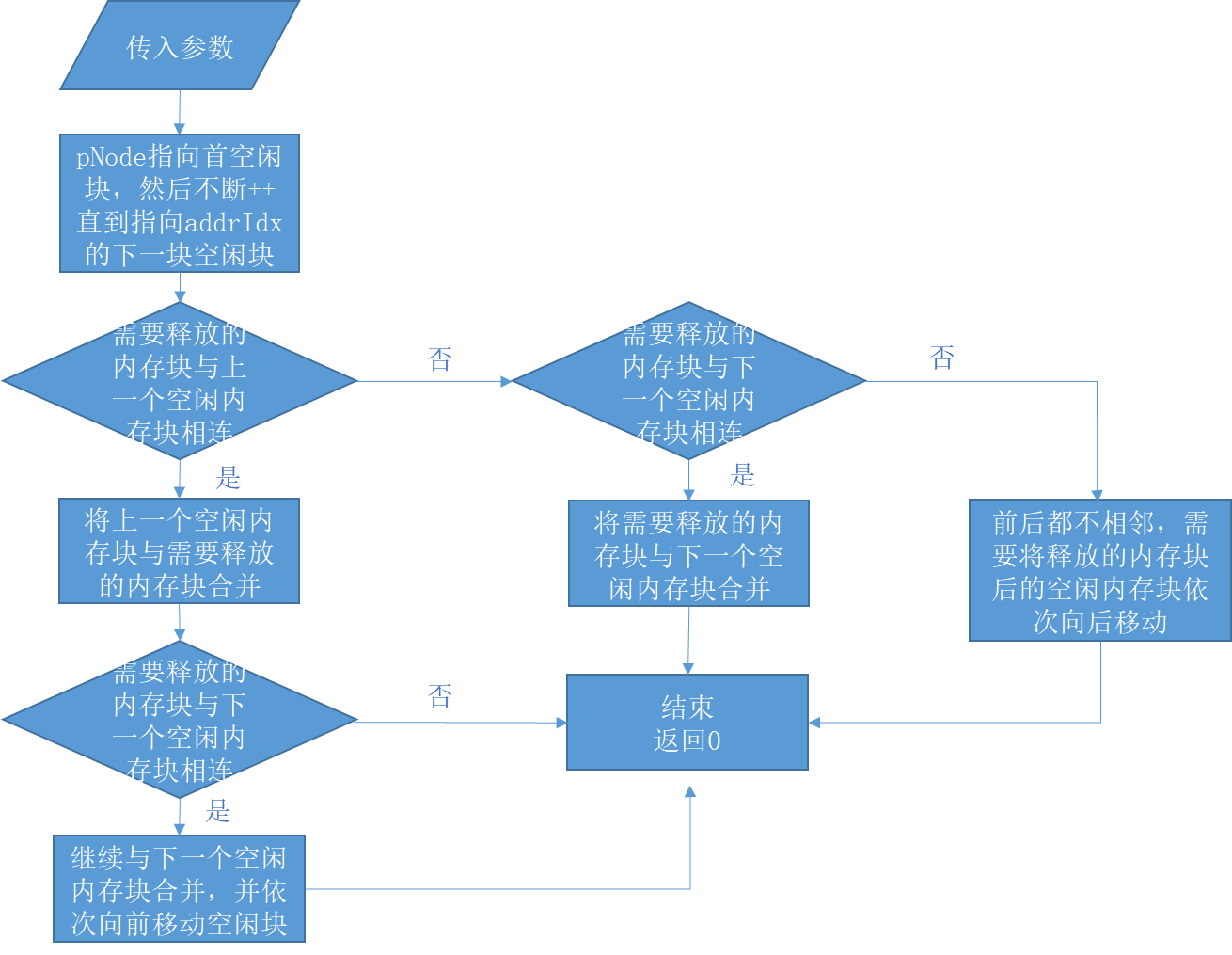
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 静态数据成员： | | | | | | | | |
| 名称 | | | 值 | | 含义 | | | |
| *PHY\_MEM\_SIZE* | | | 系统启动时根据物理内存大小设置 | | 物理内存大小 | | | |
| *PAGE\_SIZE* | | | 0x1000 | | 物理内存页大小 | | | |
| *MEMORY\_MAP\_ARRAY\_SIZE* | | | 0x200 | | 最多可分配512个对象 | | | |
| *KERNEL\_MEM\_START\_ADDR* | | | 0x100000 | | 内核映像从1M物理内存开始 | | | |
| *KERNEL\_SIZE* | | | 0x80000 | | 内核映像大小限制 | | | |
| 成员函数 | | | | | | | | |
| 名称 | 输入参数 | | | | 返回值 | | | 主要功能 |
| PageManager | Allocator\* allocator | | | | 无 | | | 构造函数 |
| ~PageManager | 无 | | | | 无 | | | 析构函数 |
| Initialize | 无 | | | | int  (成功则返回 0) | | | 完成对 MapNode map数组的初始化清零 |
| AllocMemory | size(需分配内存大小) | | | | unsigned long  (成功分配的物理内存区起始地址，返回 0表示分配失败) | | | 物理内存分配 |
| FreeMemeory | size  (需释放内存大小)  memoryStartAddress  (需释放内存起址) | | | | unsigned long  (成功则返回 0) | | | 物理内存释放 |
| 数据成员 | | | | | | | | |
| 名称 | 类型 | | | | 含义 | | | |
| map | MapNode数组 | | | | 内核空闲空间索引表 | | | |
| m\_pAllocator | Allocator\* | | | | 进行空间分配操作的 Allocator 指针 | | | |
| 子类 1：KernelPageManager | | | | | | | | |
| 静态数据成员： | | | | | | | | |
| 名称 | | | | 值 | | | | 含义 |
| *KERNEL\_PAGE\_POOL\_START\_ADDR* | | | | 0x200000 + 0x2000  + 0x2000 | | | | 内核物理内存/虚  实映射表的起始空  间 (0x20400) |
| *KERNEL\_PAGE\_POOL\_SIZE* | | | | 0x200000 - 0x4000 | | | | 内核物理内存/虚实映射表占有的空间 |
| 成员函数： | | | | | | | | |
| 名称 | | 输入参数 | | | | 返回值 | | 主要功能 |
| KernelPageManager | | Allocator\* allocator | | | | 无 | | 构造函数 |
| Initialize | | 无 | | | | int  (成功则返回0) | | 初始化MapNode map[0]为内核物理页区起始地址、大小 |
| 子类 2：UserPageManager | | | | | | | | |
| 静态数据成员： | | | | | | | | |
| 名称 | | | | 值 | | | 含义 | |
| *USER\_PAGE\_POOL\_START\_ADDR* | | | | 0x400000 | | | 用户物理内存区域起址 | |
| *USER\_PAGE\_POOL\_SIZE* | | | | 0x200000 - 0x4000 | | | 用户物理内存区域大小 | |
| 成员函数： | | | | | | | | |
| 名称 | | 输入参数 | | | | 返回值 | | 主要功能 |
| UserPageManager | | Allocator\* allocator | | | | 无 | | 构造函数 |
| Initialize | | 无 | | | | int  (成功返回0) | | 初始化MapNode map[0]为用户物理页区起始地址、大小 |

## 绘制两个函数的流程图。

### Alloc 函数流程图

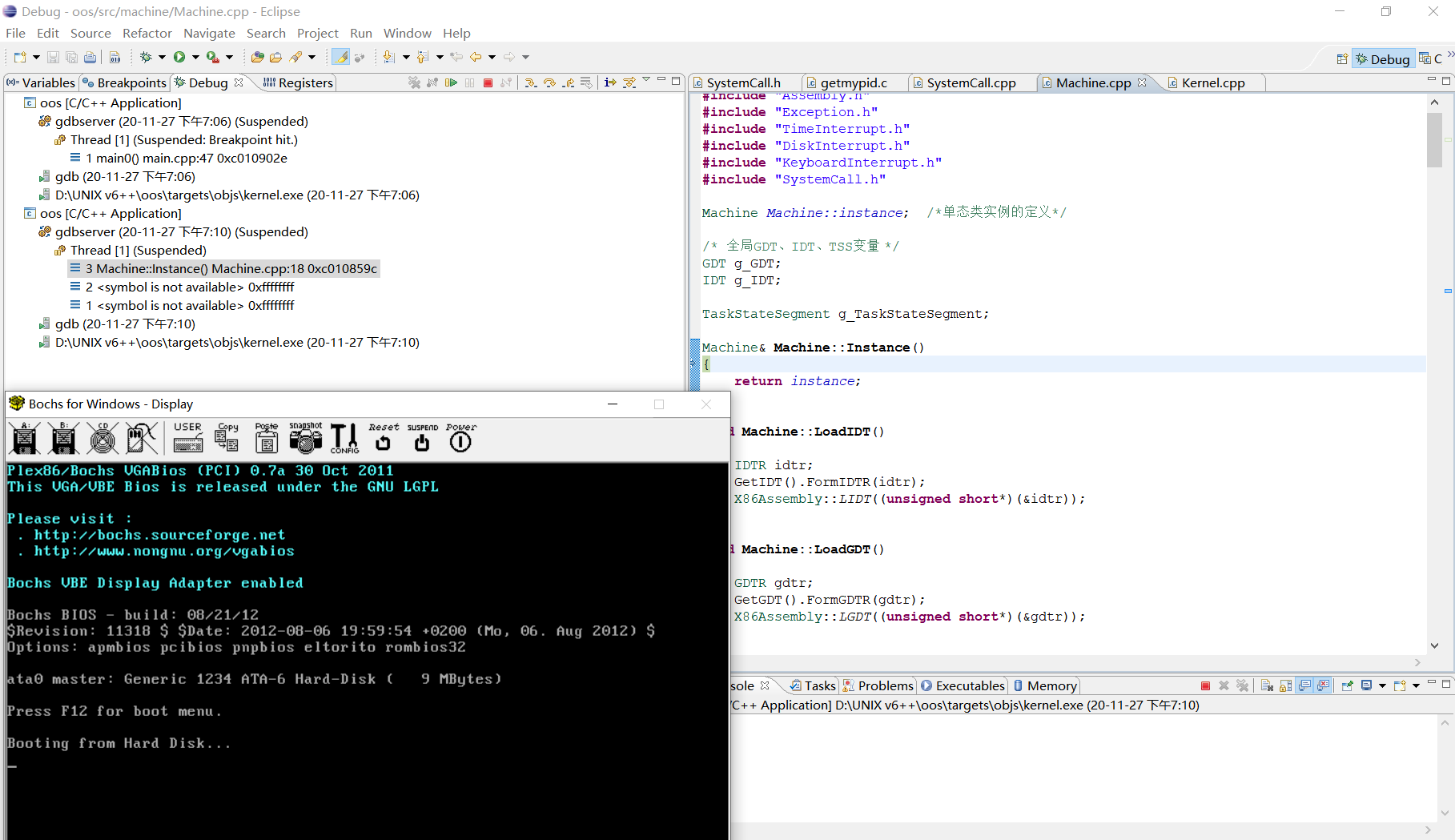


### Free 函数流程图



## 尝试将 UNIX V6++的首次适应算法修改为循环首次适应算法，并重新编译内核。给出修改过的分配和回收函数的流程图与代码，及相应的说明。

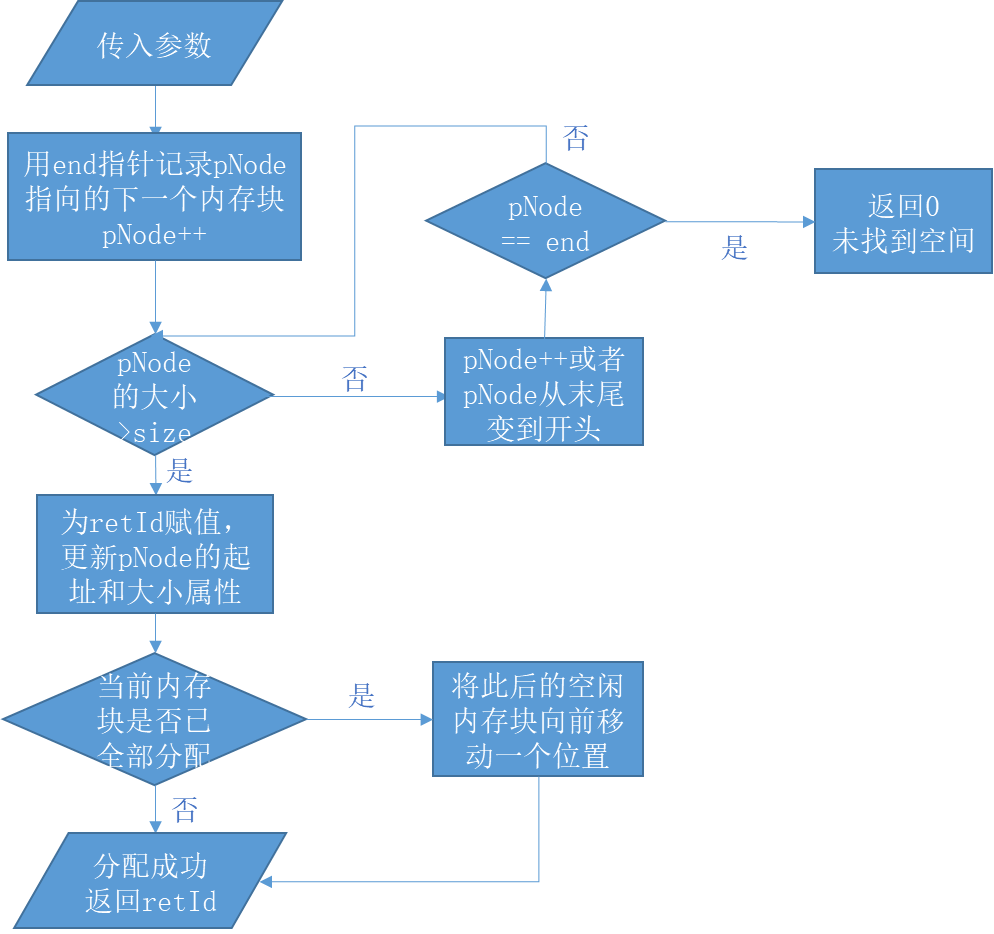
### 结果



### 说明

只需要修改分配函数代码，代码修改如下：需要注意两点，一个是保留上次的内存块位置，方便下次从这里查找；另一个是查到表末尾时要注意回到开头，直到遇到上次位置才能结束。

### 流程图



### 代码

|  |
| --- |
| unsigned long Allocator**::**Alloc**(**MapNode map**[],** unsigned long size**)**  **{**  static MapNode**\*** pNode**;**  unsigned long retIdx **=** 0**;**  //循环从下一个pNode开始寻找  **if** **(**pNode −**>** m\_Size **==** 0**)**  pNode **=** map**;**  **else**  pNode**++;**    MapNode**\*** end **=** pNode**;** //用于保存上一次的MapNode  **while** **(**1**)**  **{**  **if** **(**pNode −**>** m\_Size **>=** size**)**  **{**  retIdx **=** pNode −**>** m\_AddressIdx**;**  pNode −**>** m\_AddressIdx **+=** size**;**  pNode −**>** m\_Size −**=** size**;**  /\* 当前内存正好分配完成，将该MapNode所在位置后面的MapNode都向前移动一个位置 \*/  **if** **(**pNode −**>** m\_Size **==** 0**)**  **{**  MapNode**\*** pNextNode **=** **(**pNode **+** 1**)** **;**  **for** **(** **;** pNextNode −**>** m\_Size**;** **++** pNode**,** **++**pNextNode**)**  **{**  pNode −**>** m\_AddressIdx **=** pNextNode −**>** m\_AddressIdx **;**  pNode −**>** m\_Size **=** pNextNode −**>** m\_Size **;**  **}**  pNode −**>** m\_AddressIdx **=** pNode −**>** m\_Size **=** 0**;**  **}**  **return** retIdx **;**  **}**    /\* 遍历到结尾，返回到头部 \*/  **if** **(**pNode −**>** m\_Size **==** 0**)**  pNode **=** map**;**  **else**  pNode**++;**  **if** **(**pNode **==** end**)**  **break;**  **}**  /\* no match found \*/  **return** 0**;**  **}** |