阅读：5,666　　作者：解学武

伙伴系统管理动态内存

上一节　　下一节

前面介绍了系统在分配与回收存储空间时采取的边界标识法。本节再介绍一种管理存储空间的方法——伙伴系统。

伙伴系统本身是一种动态管理内存的方法，和边界标识法的区别是：使用伙伴系统管理的存储空间，无论是空闲块还是占用块，大小都是 2 的 n 次幂（n 为正整数）。

例如，系统中整个存储空间为 2^m 个字。那么在进行若干次分配与回收后，可利用空间表中只可能包含空间大小为：2^0、2^1、2^2、...、2^m 的空闲块。

> 字是一种计量单位，由若干个字节构成，不同位数的机器，字所包含的字节数不同。例如，8 位机中一个字由 1 个字节组成；16 位机器一个字由 2 个字节组成。

可利用空间表中结点构成

伙伴系统中可利用空间表中的结点构成如图 1 所示：

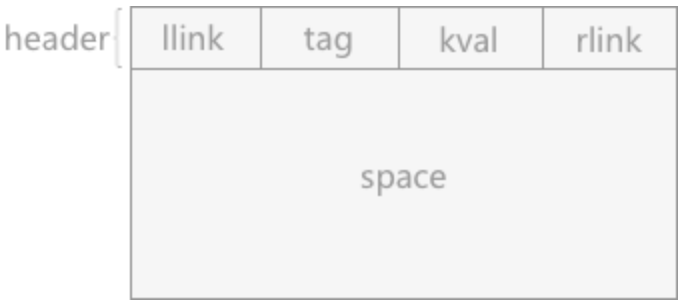


图 1 结点构成

header 域表示为头部结点，由 4 部分构成：

- llink 和 rlink 为结点类型的指针域，分别用于指向直接前驱和直接后继结点。
- tag 值：用于标记内存块的状态，是占用块（用 1 表示）还是空闲块（用 0 表示）
- kval ： 记录该存储块的容量。由于系统中各存储块都是 2 的 m 幂次方，所以 kval 记录 m 的值。

代码表示为：

```
01.  typedef struct WORD_b{
02.      struct WORD_b *llink;//指向直接前驱
```

```
    int tag;//记录该块是占用块还是空闲块
    int kval;//记录该存储块容量大小为2的多少次幂
    struct WORD_b *rlink;//指向直接后继
    OtherType other;//记录结点的其它信息
}WORD_b,head;
```

在伙伴系统中，由于系统会不断地接受用户的内存申请的请求，所以会产生很多大小不同但是都是容量为 2^m 的内存块，所以为了在分配的时候查找方便，系统采用将大小相同的各自建立一个链表。对于初始容量为 2^m 的一整块存储空间来说，形成的链表就有可能有 m+1 个，为了更好的对这些链表进行管理，系统将这 m+1 个链表的表头存储在数组中，就类似于邻接表的结构，如图 2。

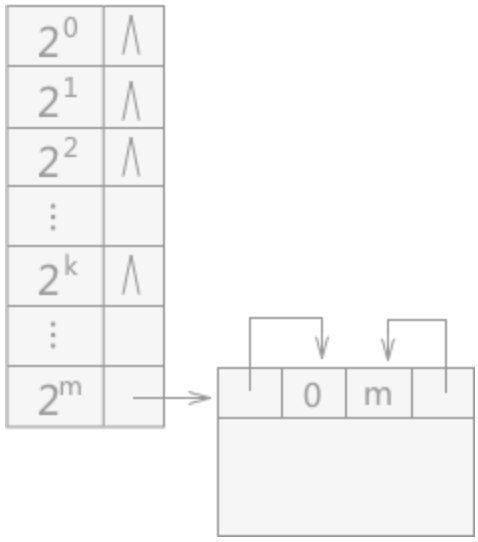


图 2 伙伴系统的初始状态

可利用空间表的代码表示为：

```
#define m 16//设定m的初始值
typedef struct HeadNode {
    int nodesize;//记录该链表中存储的空闲块的大小
    WORD_b * first;//相当于链表中的next指针的作用
}FreeList[m+1];//一维数组
```

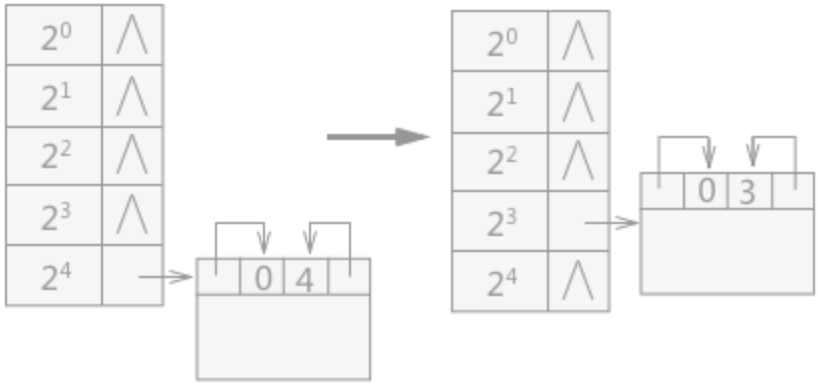
分配算法

伙伴系统的分配算法很简单。假设用户向系统申请大小为 n 的存储空间，若 $2^{k-1} < n <= 2^k$，此时就需要查看可利用空间表中大小为 2^k 的链表中有没有可利用的空间结点：

- 如果该链表不为 NULL，可以直接采用头插法从头部取出一个结点，提供给用户使用；
- 如果大小为 2^k 的链表为 NULL，就需要依次查看比 2^k 大的链表，找到后从链表中删除，截取相应大小的空间给用户使用，剩余的空间，根据大小插入到相应的链表中。

例如，用户向系统申请一块大小为 7 个字的空间，而系统总的内存为 2^4 个字，则此时按照伙伴系统的分配算法得出：$2^2 < 7 < 2^3$，所以此时应查看可利用空间表中大小为 2^3 的链表中是否有空闲结点：

- 如果有，则从该链表中摘除一个结点，直接分配给用户使用；
- 如果没有，则需依次查看比 2^3 大的各个链表中是否有空闲结点。假设，在大小 2^4 的链表中有空闲块，则摘除该空闲块，分配给用户 2^3 个字的空间，剩余 2^3 个字，该剩余的空闲块添加到大小为 2^3 的链表中。



（A）分配前　　（B）分配后

图 3 伙伴系统分配过程

回收算法

无论使用什么内存管理机制，在内存回收的问题上都会面临一个共同的问题：如何把回收的内存进行有效地整合，伙伴系统也不例外。

当用户申请的内存块不再使用时，系统需要将这部分存储块回收，回收时需要判断是否可以和其它的空闲块进行合并。

在寻找合并对象时，伙伴系统和边界标识法不同，在伙伴系统中每一个存储块都有各自的“伙伴”，当用户释放存储块时只需要判断该内存块的伙伴是否为空闲块，如果是则将其合并，然后合并的新的空闲块还需要同其伙伴进行判断整合。反之直接将存储块根据大小插入到可利用空间表中即可。

判断一个存储块的伙伴的位置时，采用的方法为：如果该存储块的起始地址为 p，大小为 2^k，则其伙伴所在的起始地址为：

$$\begin{cases} p+2^k\ (\text{若 } p \text{ MOD } 2^{k+1}=0) \\ p-2^k\ (\text{若 } p \text{ MOD } 2^{k+1}=2^k) \end{cases}$$

例如，当大小为 2^8，起始地址为 512 的伙伴块的起始地址的计算方式为：

由于 512 MOD 2^9=0，所以，512+2^8=768，及如果该存储块回收时，只需要查看起始地址为 768 的存储块的状态，如果是空闲块则两者合并，反之直接将回收的释放块链接到大小为 2^8 的链表中。

总结

使用伙伴系统进行存储空间的管理过程中，在用户申请空间时，由于大小不同的空闲块处于不同的链表中，所以分配完成的速度会更快，算法相对简单。

回收存储空间时，对于空闲块的合并，不是取决于该空闲块的相邻位置的块的状态；而是完全取决于其伙伴块。所以即使其相邻位置的存储块时空闲块，但是由于两者不是伙伴的关系，所以也不会合并。这也就是该系统的缺点之一：由于在合并时只考虑伙伴，所以容易产生存储的碎片。

上一节 下一节

联系方式　购买教程（带答疑）