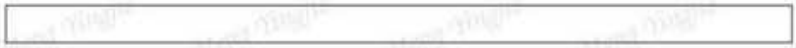
二. 概念

1. 可利用空间

系统运行初始，整个存储区是一整块，但随着一系列的分配和释放之后，原来的一整块存储区在很大程度上被四分五裂了，形成了空闲块和被占用块相间的局面。

系统运行初始：



系统运行中的某瞬间时刻存储空间的布局状态可能如下：

可利用空间(available space)：在存储区域中，当前还没有被使用(占用)的空间。

§11.1 存储管理问题概述 Data Structures

二. 概念

1. 可利用空间

系统运行初始，整个存储区是一整块，但随着一系列的分配和释放之后，原来的一整块存储区在很大程度上被四分五裂了，形成了空闲块和被占用块相间的局面。

系统运行初始：

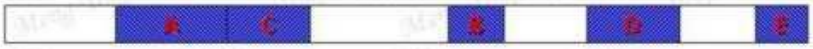
系统运行中的某瞬间时刻存储空间的布局状态可能如下：

可利用空间(available space)：在存储区域中，当前还没有被使用(占用)的空间。

编写：蒙应杰 第2页

2. 可利用空间表

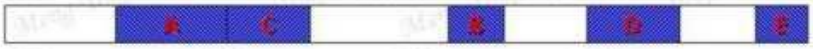
系统运行中的某瞬间时刻存储空间的布局状态可能如下：



对于这些可利用空间(即空闲的存储块组成的一个数据对象)，由于各自由块都可以基于每块的起始地址顺序看成一个逻辑上的顺序关系。

2. 可利用空间表

系统运行中的某瞬间时刻存储空间的布局状态可能如下：

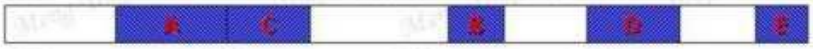


对于这些可利用空间(即空闲的存储块组成的一个数据对象)，由于各自由块都可以基于每块的起始地址顺序看成一个逻辑上的顺序关系。

因此，我们可以采用表结构的形式进行组织和管理可利用空间，此时，这个表就称为可利用空间表(available space list)。

2. 可利用空间表

系统运行中的某瞬间时刻存储空间的布局状态可能如下：



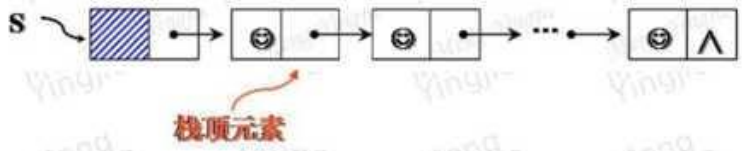
对于这些可利用空间(即空闲的存储块组成的一个数据对象)，由于各自由块都可以基于每块的起始地址顺序看成一个逻辑上的顺序关系。

因此，我们可以采用表结构的形式进行组织和管理可利用空间，此时，这个表就称为可利用空间表(available space list) 。

要注意的是落实到实际应用中，存储管理不仅涉及自由块的管理，也涉及到使用块的管理。此处仅讨论了自由块的管理讨论。

由于可利用空间表本身属线性结构，因此可以采用链接或者向量来进行具体组织。

用链表时：一般多用一个向前链表来记载空闲存储块的情况，把空闲块链接在一起,这个表称为空闲存储块链表。类似的可以看成一个链接方式存储的栈。



由于可利用空间表本身属线性结构，因此可以采用链接或者向量来进行具体组织。

用链表时：一般多用一个向前链表来记载空闲存储块的情况，把空闲块链接在一起,这个表称为空闲存储块链表。类似的可以看成一个链接方式存储的栈。

也可以利用向量来组织空闲存储块。

三. 存储问题概述

由于数据结构对于存储问题的讨论，主要是寻找一般性存储管理问题的普遍性解决方案，就涉及到的范畴来说，可以将存储问题问题归纳概括为四个方面加以研究。

三. 存储问题概述

1. 可利用空间表的有效组织问题

数据结构中的可利用空间表中的一个元素实际上就是一个空白的存储块，它涉及到块的起始地址、块的容量大小等信息。特别是块的容量大小直接关系到存储的申请满足性。

最长见的组织方式有3种：

- 等长结点组织
- 最大公约数
- 混合长度组织

(1).等长结点组织

既在一个可利用空间表中所有结点的空间大小一致。

针对需求的不同，对于不同需求长度的结点，采用建立不同的可利用空间表的方法。这种方法管理可利用空间表的代价比较高。

(1).等长结点组织

既在一个可利用空间表中所有结点的空间大小一致。

针对需求的不同，对于不同需求长度的结点，采用建立不同的可利用空间表的方法。这种方法管理可利用空间表的代价比较高。

为适应不等长的需求，处理变长情况可以采用：

统一按照较长的结点组织可利用空间表，把变长结点按等长结点进行分配。结点长度差别不大时可采用，但长度差别很大时，会造成很大浪费。

(2).最大公约数：

按照所有需求结点长度的最大公约数为长度组织可利用空间表。

分配时每个结点按大小一次从可利用空间表中分配若干表目，表目之间再用指针链接。

例如，将最大公约数看作一套房子的话，有些家庭可能须要将几套房子联结起来才能满足需求。

(2).最大公约数:

按照所有需求结点长度的最大公约数为长度组织可利用空间表。

分配时每个结点按大小一次从可利用空间表中分配若干表目，表目之间再用指针链接。

例如，将最大公约数看作一套房子的话，有些家庭可能须要将几套房子联结起来才能满足需求。

这种方式各表目间使用了较多的指针，除增加了一定的存储开销外，给按结点为单位进行的各种处理带来了困难。

(3).混合长度组织:

把各种大小的结点（即可利用存储块）混合组织在一个可利用空间表中。

分配时需要按结点长度在可利用空间表中检索，找到其长度大于等于申请长度的结点，从中裁剪截取合适的长度。

回收时不能简单把结点放入可利用空间表，必须考虑到相邻地址空间的结点的合并问题。

(3).混合长度组织：

把各种大小的结点（即可利用存储块）混合组织在一个可利用空间表中。

分配时需要按结点长度在可利用空间表中检索，找到其长度大于等于申请长度的结点，从中裁剪截取合适的长度。

回收时不能简单把结点放入可利用空间表，必须考虑到相邻地址空间的结点的合并问题。

这种方式需求的满足最容易实现，也比较有利于解决存储共享问题，缺陷：分配及回收算法复杂，易出现碎片问题 (storage fragmentation) 。

在长期动态运行过程中，该方法可能导致整个空间被分割成许多大小不等的碎片，某些碎片由于太小而长期得不到使用，将这种现象称为碎片问题。

2. 如何解决碎片问题？

为了解决此问题，可以在申请分配时取大于等于它的最小块给予分配。

根本的彻底解决方法就是要进行存储压缩，即将整个存储空间的自由存储块向存储空间的一端进行集中，这种集中的结果就是产生连续地址的大面空间。但该工作相当费时，也十分危险。

2. 如何解决碎片问题？

为了解决此问题，可以在申请分配时取大于等于它的最小块给予分配。

根本的彻底解决方法就是要进行存储压缩，即将整个存储空间的自由存储块向存储空间的一端进行集中，这种集中的结果就是产生连续地址的大面空间。但该工作相当费时，也十分危险。

存储压缩的基本步骤：

(1). 给占用结点分配新地址；

(2). 修改占用结点中的指针值；

(3). 将占用结点内容搬到新分配位置。

3. 无用结点的收集

所谓无用结点是指可以回收但还没有回收的结点。也称为存储废料的收集。

这些无用结点的存在会影响空间使用的效率，因此存储管理系统要有能力把这些无用结点找出，并送回到可利用空间表中。

3. 无用结点的收集

所谓无用结点是指可以回收但还没有回收的结点。也称为存储废料的收集。

这些无用结点的存在会影响空间使用的效率，因此存储管理系统要有能力把这些无用结点找出，并送回到可利用空间表中。

通常的做法是：首先普查一次存储空间(相当于广义表的遍历)，把那些已经不属于任何链上的结点做标记，然后将它们收集到可利用空间表中，回收过程也可以与压缩一起进行。

4. 存储空间溢出处理

此处不再讨论。

本节结束

一. 背景

这是一种比较实用的动态存储管理方案。

能在不做检索工作的前提下，发现和回收空闲存储块，以达到既不影响处理速度又可以回收空闲块的目的。

伙伴系统中为了避免许多明显的链域关系，利用二进制地址编码之间的内在数学关系，使用隐含的寻址规则捕获相邻存储块的地址，这是伙伴系统可用性的基础。

二. 概念及原理

1.伙伴及伙伴系统

所谓伙伴，是指存储区中有一定地址关系且容量相等的一对存储块。

二. 概念及原理

1.伙伴及伙伴系统

伙伴系统：可利用空间以伙伴为基础而建立的动态存储管理系统。

二. 概念及原理

1.伙伴及伙伴系统

伙伴系统：可利用空间以伙伴为基础而建立的动态存储管理系统。

该系统主要是利用计算机的二进制编址的特点来划分存储空间。使得每块长度都限制在2^k(k=1,2,...).

当申请长度不等于2的幂次时，选择最接近的大于申请容量的2^k个单元进行分配，而把多余的作为结点或结构内部的空余进行处理。

例如，申请13个单元，则分配16(= 2^4)个单元。

二. 概念及原理

2.伙伴的隐含寻址基础

因为伙伴系统中约定所有存储块容量都必须是2的幂，同时由于所有存储块的地址都在0和存储区最大地址2^m-1之内。

易证明长度为2^k的任何存储块的起始地址是2^k的倍数，简而言之，这个2^k容量的存储块的起始地址从右边数至少有k个0。因此伙伴存储块起始地址有以下关系：

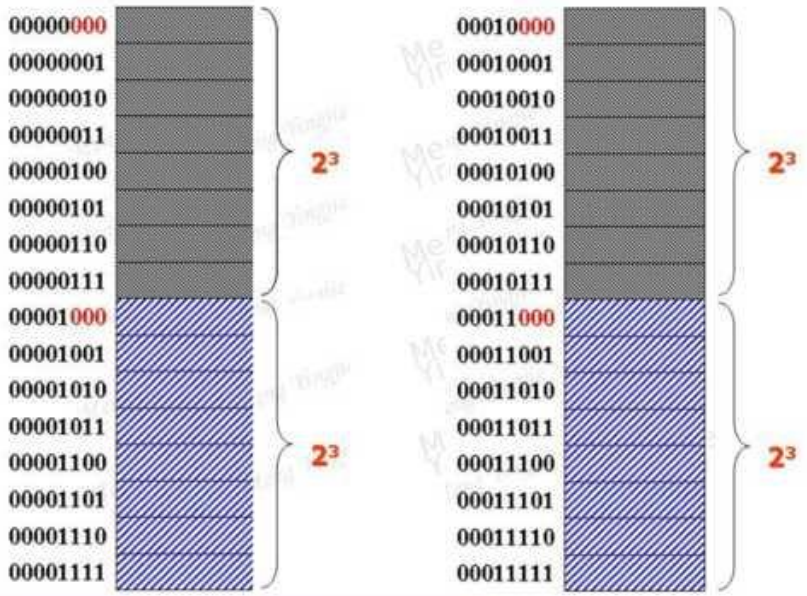
2^k容量的存储块起始地址：

$$\times\times\ldots\times\ \underbrace{\times 0\ldots 0}_{k个}\ ，其中\times\in\{0,1\}$$

则，其伙伴起始地址：

$$\times\times\ldots\times\ \underbrace{\bar{\times} 0\ldots 0}_{k个}\ ，其中:\bar{\times}为\times的反码$$

直观的图示：



一般地，令$buddy_k(z)$是容量为2^k，起始地址为Z的存储块的伙伴存储块起始地址，则：

$$Buddy(k,z)=\begin{cases} Z+2^k, & \text{若 } Z \bmod (2^{k+1})=0 \\ Z-2^k, & \text{若 } Z \bmod (2^{k+1})=2^k \end{cases}$$

一般地，令$buddy_k(z)$是容量为2^k，起始地址为Z的存储块的伙伴存储块起始地址，则：

$$Buddy(k,z)=\begin{cases} Z+2^k, & \text{若 } Z \bmod (2^{k+1})=0 \\ Z-2^k, & \text{若 } Z \bmod (2^{k+1})=2^k \end{cases}$$

若起始地址的编址从a开始(而不是0)，则：

$$Buddy(k,z)=\begin{cases} Z+2^k, & \text{若 } (z-a) \bmod (2^{k+1})=0 \\ Z-2^k, & \text{若 } (z-a) \bmod (2^{k+1})=2^k \end{cases}$$

而该函数在计算机上利用“按位加”,或“选择求补”指令是很容易计算的。

3.伙伴系统的基本约定：

在伙伴系统的使用中要注意有4个基本约定前提：

3.伙伴系统的基本约定：

在伙伴系统的使用中要注意有4个基本约定前提：

(1).分配存储块时，其容量一律按照2的幂次给予满足。

例：若请求n个单元，则系统给予容量为2^k大小的空间，它满足关系$2^{k-1}<n\leqslant 2^k$，而已经分配的2^k-n这个余数称为内部碎片。

3.伙伴系统的基本约定：

在伙伴系统的使用中要注意有4个基本约定前提：

(2). 每个存储块至少开辟一位作为伙伴系统控制用户互斥使用该存储块的标志（即该块是否已分配）。

可以用该存储块的第一个单位的高位字节或整个字节来做标志。

$$\text{tag} = \begin{cases} 1, & \text{若本块已分配} \\ 0, & \text{若本块为可利用块} \end{cases}$$

3.伙伴系统的基本约定：

在伙伴系统的使用中要注意有4个基本约定前提：

(3).为了提高效率，相同大小(都为2^k)的可利用块都链接在一个链表中。

这样系统中有m+1个链表，因为系统中存储块的大小可以是2^0、2^1、2^2、…、2^m, 其中2^m为存储区最大的可利用存储单位。

因此，对于这m+1个链表需要使用一个表头向量进行管理。

3.伙伴系统的基本约定：

在伙伴系统的使用中要注意有4个基本约定前提：

(4).假设存储块初态是容量2^m的单一可利用块,其地址是0..2^m-1。

实际这样在地址编址和计算中是很容易实现的。利用前面给的函数关系“按位加”,或“选择求补”指令是容易计算的。

三. 分配与回收

1.存储的分配

该过程可通过两步完成：

(1).确定满足请求分配容量为n的相应容量2^k的 k；

(2).直接分配所确定的k的这个链表的第一个存储块，并标记为分配。

三. 分配与回收

1.存储的分配

该过程可通过两步完成:

(1).确定满足请求分配容量为n的相应容量2^k的 k;

(2).直接分配所确定的k的这个链表的第一个存储块，并标记为分配。

若链表 k 为空,即没有容量为2^k的可利用块，则需要从链表 k+1获得容量为2^{k+1}的可利用块，将它分解为两半, 第一块给予分配请求，第二块链接在链表k中,称第二块为已分配的那个第一块的伙伴。

若链表k+1也空时,还可以将2^{k+2}的更大可利用块进行分，依次类推(除非没有更大的可利用块)，最终就会出现容量为2^k的可利用块来满足请求分配。

2.存储的回收

(1).确定释放容量为n的相应容量2^k的k;

(2).根据伙伴关系函数直接检测该存储块的伙伴存储块的状态标记是否为可利用块。

如果其伙伴是被分配状态，则只需将被释放存储块回收到可利用块链表k中;

如果其伙伴是未分配状态,则将两者结合为容量为2^{k+1}的存储块。

2.存储的回收

(1).确定释放容量为n的相应容量2^k的k;

(2).根据伙伴关系函数直接检测该存储块的伙伴存储块的状态标记是否为可利用块。

如果其伙伴是被分配状态，则只需将被释放存储块回收到可利用块链表k中;

如果其伙伴是未分配状态,则将两者结合为容量为2^{k+1}的存储块。

然后再计算容量为2^{k+1}这个存储块的伙伴存储块，重复做上述工作，以便结合成更大的可利用存储块，依次类推，一直结合成可能的最大存储块为止，并把它回收到相应的可利用块链表中。

本章结束