设计文档

1.概要

1.1. 动态内存管理

存储管理——每一种数据结构都必须研究该结构的存储结构，但它是借助于某一高级语言中的变量说明来加以描述的，并没有涉及到具体的存储分配。

实际上，结构中的每个数据元素都占有一定的内存位置，在程序执行的过程中，数据元素的存取是通过对应的存储单元来进行的。研究数据存储与内存单元对应问题，就是存储管理问题。

1.1.2.动态存储管理的基本问题

- 1、如何根据用户提出的“请求”来分配内存。

根据申请内存大小利用相应分配策略分配给用户所需空间；若分配块大小与申请大小相差不多，则将此块全部分给用户；否则，将分配块分为两部分，一部分给用户使用，另一部分继续留在可利用空闲表中。

- 2、如何收回被用户“释放”的内存，以备新的“请求”产生时重新进行分配。

测试回收块前后相邻内存块是否空闲；若是则需将回收块与相邻空闲块合并成较大的空闲块，再链入可利用空闲表中。

1.1.3.各种内存分配方法优势及不足

- **1、首次拟合法**

方案：按分区的先后次序，从头查找，找到符合要求的第一个分区

优点：该算法的分配和释放的时间性能较好，较大的空闲分区可以被保留在内存高端。操作方便，查找快捷；

缺点：但随着低端分区不断划分而产生较多小分区，每次分配时查找时间开销会增大。

- **2、最佳拟合法**

方案：分配不小于 n 且最接近 n 的空闲块的一部分。

优势：尽可能将大的空闲块留给大程序使用；

缺点：从个别来看，外碎片较小，但从整体来看，会形成较多外碎片。较大的空闲分区可以被保留。

- **3、最坏拟合法**

方案：分配不小于 n 且是最大的空闲块的一部分。

优势：尽可能减少分配后无用碎片；

缺点：基本不留下小空闲分区，但较大的空闲分区不被保留。

- **4、循环首次拟合法（又称下次拟合法）**

方案：按分区的先后次序，从上次分配的分区起查找（到最后分区时再回到开头），找到符合要求的第一个分区

优势：该算法的分配和释放的时间性能较好，使空闲分区分布得更均匀

缺点：较大的空闲分区不易保留

1.2. 边界标识法

边界标识法（boundary tag method）是操作系统中用以进行动态分区分配的一种存储管理方法。系统将所有的空闲块链接在一个双重循环链表结构的可利用空间表中；分配可按首次拟合进行，也可按最佳拟合进行。

采用边界标识法系统的特点：在每个内存区的头部和底部两个边界上分别设有标识，以标识该区域为占用块或空闲块，使得在回收用户释放的空闲块时易于判别在物理位置上与其相邻的内存区域是否为空闲块，以便将所有地址连续的空闲存储区组合成一个尽可能大的空闲块。

1.2.1.结点结构（C 语言描述）

下图是 C 语言描述的边界标识法节点结构，结点由 3 部分组成：头部 head 域、space 域和底部 foot 域。

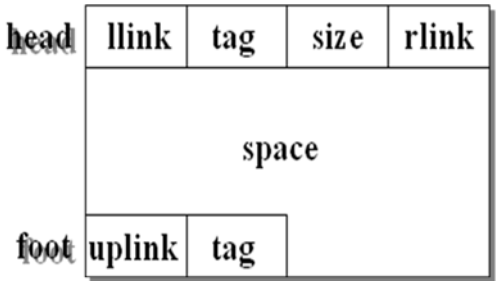

图. 节点结构

space：为一组地址连续的存储单元，是可以分配给用户使用的内存区域。其大小有 head 中的 size 域指示,并以头部 head 和底部 foot 作为它的两个边界。

tag：标志域,在 head 和 foot 中都设有该域,值为“0”表示空闲块,值为“1”表示占用块。

size：整个结点的大小，包括头部 head 和底部 foot 所占空间。

llink：链域，位于头部 head 域，指向前驱结点。

rlink：链域，位于头部 head 域，指向后继结点。

foot：位于结点底部,它的地址是随结点中 space 空间的大小而变的.

uplink：链域，位于尾部 foot 域，指向本结点头部，其值即为该空闲块的首地址。

1.2.2.分配算法

找到一个空闲的内存块，作为起始搜索内存块，搜索算法可以采用首次拟合法和最佳拟合法。下面针对首次拟合的分配算法进行一下介绍。设申请空间的大小为 size_n，搜索内存块的指针为 pointer_pav。首先获取当前内存块头部信息中内存块的大小 size，如果内存块的大 size 大于或者等于申请空间的大小 size_n，则将此内存块的一部分或整个分配给用户。如果内存块的大小 size 小于申请空间的大小 size_n，则将搜索内存块的指针 pointer_pav 指向下一个空闲的内存块。继续比较，直到找到或搜索完所有的空闲内存块为止。 为了避免产生无法使用的内存碎片，我们定义一个常量 size_e。如果分配给用户后内存块剩余空间的大小小于常量 size_e，则将整个内存块全部分配给用户。否则，只分配申请空间的大小。

另外，如果搜索指针的位置固定从某一个内存块开始，势必产生内存分布不均，一些地方密集，而一些地方稀疏。为了解决这个问题，我们将搜索指针指向每次释放后的内存块。

时间复杂度的讨论：

该算法的时间主要花费在查找第一个满足条件的内存块（简称命中块）上。而查找时间的长短又由链表的长度和命中块在链表中的位置决定。设链表的长度为 n，命中块在链表中的位置为 i：

1、当 i=0 时，其时间复杂度为 O(1)，这是最好情况 ；

2、当 i=n-1 时，其时间复杂度为 O(n)， 这是最差情况 ；

因为命中块的位置是任意的，所以我们分析一下平均时间复杂度。假设每一个内存块满足条件的概率是均等的，都为 1/n，那么 n 的值越大花费的时间越长。因此查找算法的平均时间复杂度为 O(n)。

1.2.3.回收算法

回收算法是将用户不用的内存块回收，以便再次分配。 根据内存块物理上左右邻区的空闲占用情况分为四种情况：

1、左右邻区都为空闲的情况；

2、左右邻区都为占用的情况；

3、左邻区为空闲，右邻区为占用的情况；

4、左邻区为占用，右邻区为空闲的情况。

每一种情况对应一种回收算法。下面对每一种回收算法分别进行一下介绍：

第一种：左右邻区都为空闲的情况

为了让三个空闲块合并成一个空闲块，可以修改左邻块的空间大小，然后，将右邻块从链表中删除，再修改合并后内存块的底部信息。

第二种：左右邻区都为占用的情况

这种情况比较简单，只要将释放块插入到空闲块链表中即可。因为链表是无序的，所以插入位置是任意的。

第三种：左邻区为空闲，右邻区为占用的情况

修改左邻块的空间大小，然后修改底部信息，将此释放块和左邻块合并成一个内存块。

第四种：左邻区为占用，右邻区为空闲的请况

修改释放块的头部信息，然后修改释放块的底部信息。

时间复杂度的讨论：

由于在每个内存块的头部和底部都设立标识，所以不需要进行查找来找到与它毗邻的空闲块进行合并；其次，链表中内存块的位置是无序的，所以不需要查找链表进行插入。总之，不管哪种情况释放算法的时间复杂度都为常量。

2.设计思想

本程序所实现的内存分配器采用了上面所描述的边界标识算法，采用边界标识法内存系统的特点：在每个内存区的头部和底部两个边界上分别设有标识，以标识该区域为占用块或空闲块，使得在回收用户释放的空闲块时易于判别在物理位置上与其相邻的内存区域是否为空闲块，以便将所有地址连续的空闲存储区组合成一个尽可能大的空闲块。

2.1.分配方案

2.2.回收方案

3.宏定义及数据结构描述

3.1.内存节点定义

3.2.常见的宏定义

4.程序流程

5.程序性能分析

5.1.实验平台

硬件平台：

处理器：Intel(R) Core ™2 Duo CPU T5250 @1.5GHZ 1.5GHZ

内存：2GB　DRAM

软件平台：

操作系统：Ubuntu 10.04LTS

Linux Kernel：2.6.32-24+gcc4.4

5.2.性能分析