内存管理——动态分区分配方式的模拟

设计方案

1851201 周子龙

1. 问题背景

在计算机的运行过程中，进程在执行前需要先行装载进入内存中，而内存的大小是有限的，各个进程需要占用的内存大小也不一，如果不加限制地随意载入内存，则会导致内存利用率低下。

动态分区指的是不预先对内存进行划分，在每次进程发出申请内存请求的时候根据具体的需要“量体裁衣”，按需分配。以期望尽量减小内碎片，提高内存的利用效率。

但是在程序不断申请和释放内存的过程中，不当的管理和分配会导致产生大量的外碎片，这也会导致内存的利用率降低。

本项目将模拟内存调度的中算法：首次适应算法和最佳适应算法对内存块进行分配和回收，并显示出每次分配后的空前区链的情况，让用户直观的感受到外碎片的产生，以及在执行过程中的具体分配和回收的过程。

1. 问题分析

项目要求显示出每次分配以及回收后的空闲分区链，故应当维护一份空闲分区表。同时，以图形化界面的方式展示每次分配后的内存占有情况，我们也应当维护一份内存使用表。下面对具体的两种分配算法进行分析：

* 1. 首次适应算法

该算法要求在为进程分配第一个合适大小的内存空间。为了对内存进行分配和回收，程序中应当分别存储当前空闲内存情况（分配）、当前内存中进程的占用情况（回收），其中这两个数据结构应当分别满足如下几点功能：

* + 1. 存储空闲区：
       1. 展示：展示当前的空闲区分布情况
       2. 整理：在适当时机合并首尾相连的空闲区（如将0-100与100-200合并为0-200一个区域）
       3. 更改：更改某一区域的分区情况（如将0-100更改为50-100）
       4. 删除：若某一进程大小刚好对应一空闲区大小，则需要将该区域删除。
       5. 插入：当进程释放资源时，将该区域插入到空闲分区中。
       6. 查找：当进程请求内存空间的时候，能够对区域进行查找，返回是否能够分配的信息。
    2. 存储分配区：
       1. 插入：当为进程分配内存后将新的分配信息插入到该数据结构中
       2. 删除：当进程释放内存资源后将该进程相关信息删除，并与与存储空闲区进行交互，更新存储空闲区的内存闲置情况。

由于使用的是首次适应算法，该算法要求对空闲区能够按照地址顺序查找，因此，存放数据时应当考虑按照空闲区起始地址的大小进行顺序排放。为了避免由于数据更新不及时而认为造成外碎片的情况（如进程1需要200k资源，但是空闲区表记录的信息为1-100 100-200 此时系统会认为没有足够的空间）应当在每一次空闲区进行插入操作时进行一定规则的整理，合并这样的连续空间。考虑到这是一项基本操作，我们应当经可能的为其设计复杂度低的算法和数据结构。

基于以上设计分析，我们使用按照空闲区域起始地址大小进行排列的最小堆来存放空闲区域，这样可以将每次的插入和整理操作进行合并，且只需要的时间复杂度。存储分配区的操作较少，以简单的链表实现即可。每次的插入操作时间复杂度为，删除操作需要以进程号（PID）进行关键字查找，故时间复杂度为

但是考虑到进程平凡的申请和释放操作，如果考虑这一对操作，则时间复杂度下降为，最小堆实际上是没有必要的。故均采用链表实现即可。

* 1. 最佳适应算法

最佳适应算法每次为进程分配最合适的内存空间，即该空间大小是所有能满足进程需求的空闲区中最小的一个。

该算法的存储分配区和存储空闲区的要求与首次适应算法要求相同，而查询空闲区的方式略有不同：

该算法要求每次分配最佳内存空闲区大小，故再分配时需要涉及到查询操作，考虑到这是一个基本操作，应当为其设计合适的算法和数据结构以进行优化。

我们如果沿用最小堆的设计不变，但应该按照空间大小进行存放，这就造成了整理空间的困难。具体的插入操作逻辑如下：

1. 遍历所有表项，查看是否可以合并，若可以则合并
2. 若没有，则找到合适的插入位置，并插入

这样，在一次完整的申请释放环节后，实际的复杂度为，与连标的操作无异，故而进采用链表结构实现即可。

1. 设计约束

通过上面的分析，我们需要维护两个表：idel（空闲区表），busy（已分配表），均由简单的连标实现即可。

1. 内存管理逻辑

1. 实现
2. 开发环境

系统环境：Mac OS 10.14.5 Visual Studio Code

语言：JavaScript

图形化界面：html5

执行环境：Google chrome 83.0.4103.61 （64位）

1. 运行