*郑宇飞 计科1602 16281089*

*北京交通大学 | 计算机与信息技术学院*

*操作系统实验4*

Memory allocation

操作系统实验4： 地址分配

**实验目的：**

本实验旨在通过用C语言编程实现一系列的地址分配方法以及对紧凑的实现，来加强对于此部分知识的掌握程度。

**实验环境：**

搭载C语言编译器的个人电脑

**实验设计与步骤：**

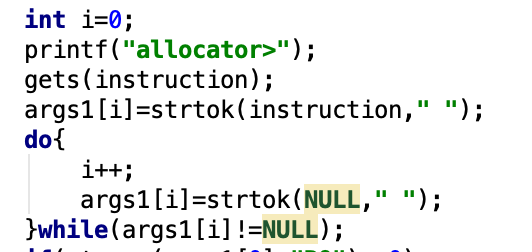
首先，此实验的要求可以分为对于三种地址分配模式算法的实现，compaction的实现以及移除某一进程占用空间的实现，以及一些实验细节要求的实现。

我们按照实验的顺序来进行设计说明，第一步应该是设计实验“界面”的大体要求，也就是detail里面的内容：

可以看到有4种指令，RQ 进程名 分配空间 分配方式，RL 进程名，C，STAT指令，那么问题来了，我们如何准确的识别指令呢？一种思路就是将输入的字符串用空格分隔开，然后对于分割后的第一个字符进行识别，如果是RQ，那么进入RQ相应识别语句；如果是RL，进入RL相应的识别语句……

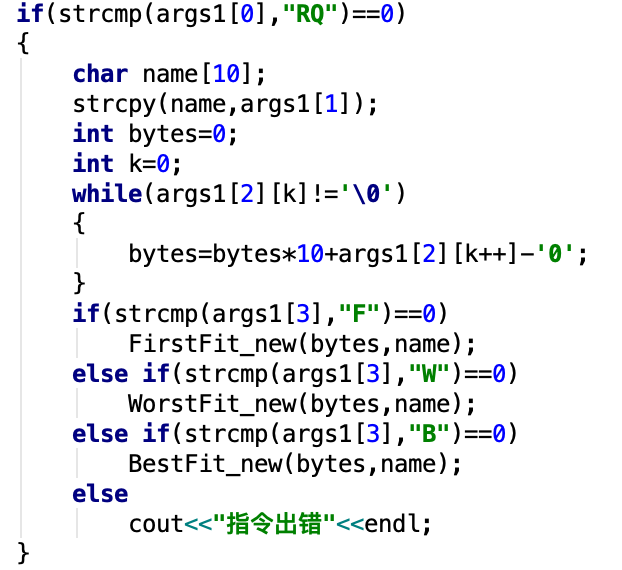
大体就如下所示：

提取空格分隔的字符串：

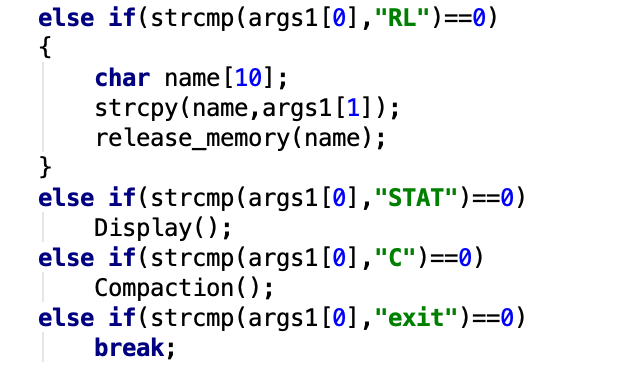


还是用到了这个比较万金油的函数。

接下来进入各个识别模式：



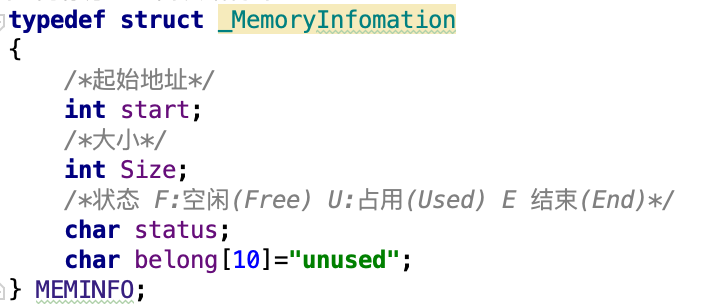
以上为RQ的识别命令



以上为RL，STAT，C，exit的识别命令。

接下来，就是具体的指令设计：

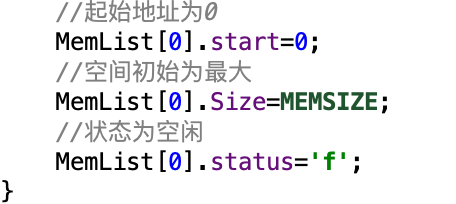
在对于三种地址的分配算法说明之前，非常有必要说明我们所用的结构是什么？



我们用这样的一个结构来代表内存空间，具体的内容上述注释已经提到，这个belong就是分配给某个进程的标志，因为后面的STAT需要使用，加入将这一空间分配给了P0，那么这个belong就是P0.

基于这个结构，我们可以建立一个数组，代表最多有多少个内存块（这里注意，不一定每个内存块都要有size，这里的数组上限只限定一个最大的值，具体到哪结束其实是有status里的‘e’来决定）

下图初始化程序中的一部分可以更好的说明括号里的内容



接下来就是三种算法的设计：

Firstfit：

这种算法就是从最低的地址开始往上查找，如果找到了一个符合大小的工作区，就直接把目标内存分配出去。所以具体实现我们分为以下步骤：

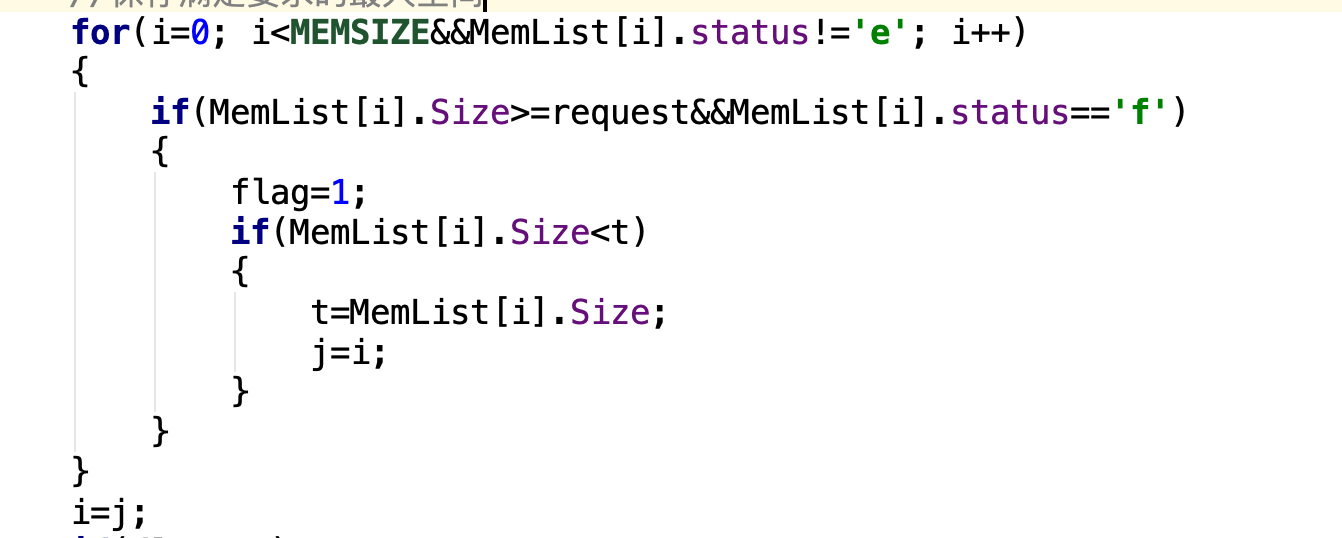
1. 遍历空间区数组，找到符合的内存区，找到就分配
2. 找到了以后将这个内存区的元素统统后移，然后把这个内存区分成两个内存区，一个status是使用，一个status是未使用（因为不可能找到的内存区的大小恰好等于这个需要分配的内存的大小）。

第二部分的具体实现如下：



接下来是Bestfit算法：

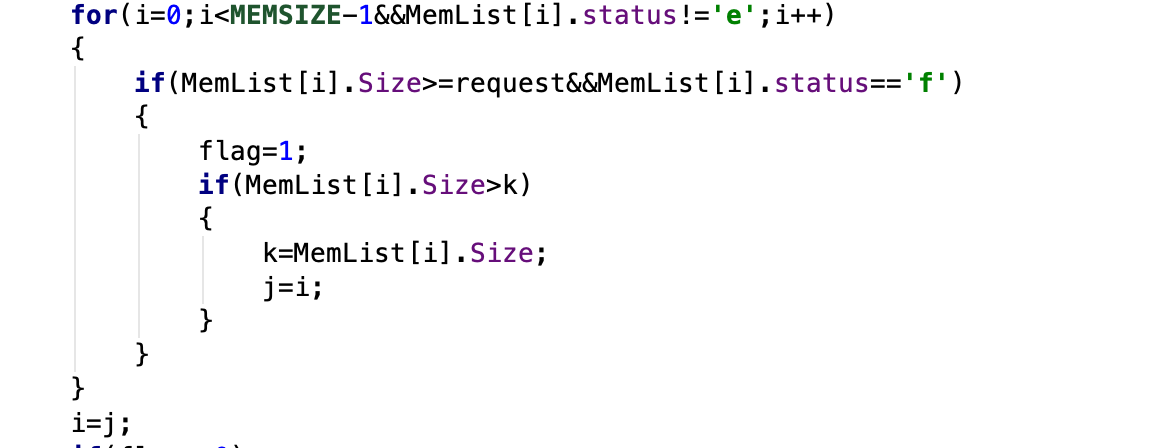
这一算法要求找到一个最适合需要分配内存大小的内存块，分配出去。这就要求我们在上一个算法的基础上加入一个比较的步骤，也就是比较出一个比所需要内存区大但是却最小的内存区，然后分配，具体如下所示：



Worstfit算法：

这一算法比较简单粗暴，挑选最大的空闲分区来给所需分配内存的进程使用，所以我们只需要直接找到最大的空间，然后和需要分配内存的空间进行比较，如果发现内存空间不够没那么其他内存空间一定也就都不够用了，因为我们找的是最大的一块内存空间。

下图为寻找最大内存空间的算法：



对于内存空间分配算法的设计完成后，接下来对RL，释放内存空间的算法进行设计：

首先这个函数传进去的参数一定是某个进程的名字，在这个函数里我们需要遍历所有的数组来找到目标内存块，然后将其释放（使用状态设置为f），最后一定要进行合并：思考这样一种情况， 0-100 空，101-200占用，201-300空，释放完毕100-200后应该是什么结果呢？是0-300空还是0-100空，101-200空，201-300空呢？显然是前者，所以我们需要在置位完成后进行内存相邻判断，然后进行合并操作：



最后是Compaction函数的设计：

这一函数通俗一点的讲，我们在不断的分配删除的过程中，总有一些小的碎片存在，这些碎片可能过小，导致什么都无法分配，试想，如果我们有1万片 几个KB的碎片，如果compaction了后，他们和成一个整体就又可以进行内存的分配了，但是如果不进行紧凑算法，可能就会全部浪费掉了。

我采用的思路是把这样的碎片的总共大小统计一遍，然后建立一个新的temp数组用来存放所有已经占用的进程，然后直接调用初始化进程函数，再按地址从小到大顺序装入进程所占用的内存区，最后装入一个内存大小为所有碎片大小和的空闲内存区，达到目的



大致的实验设计到此基本就完成了。

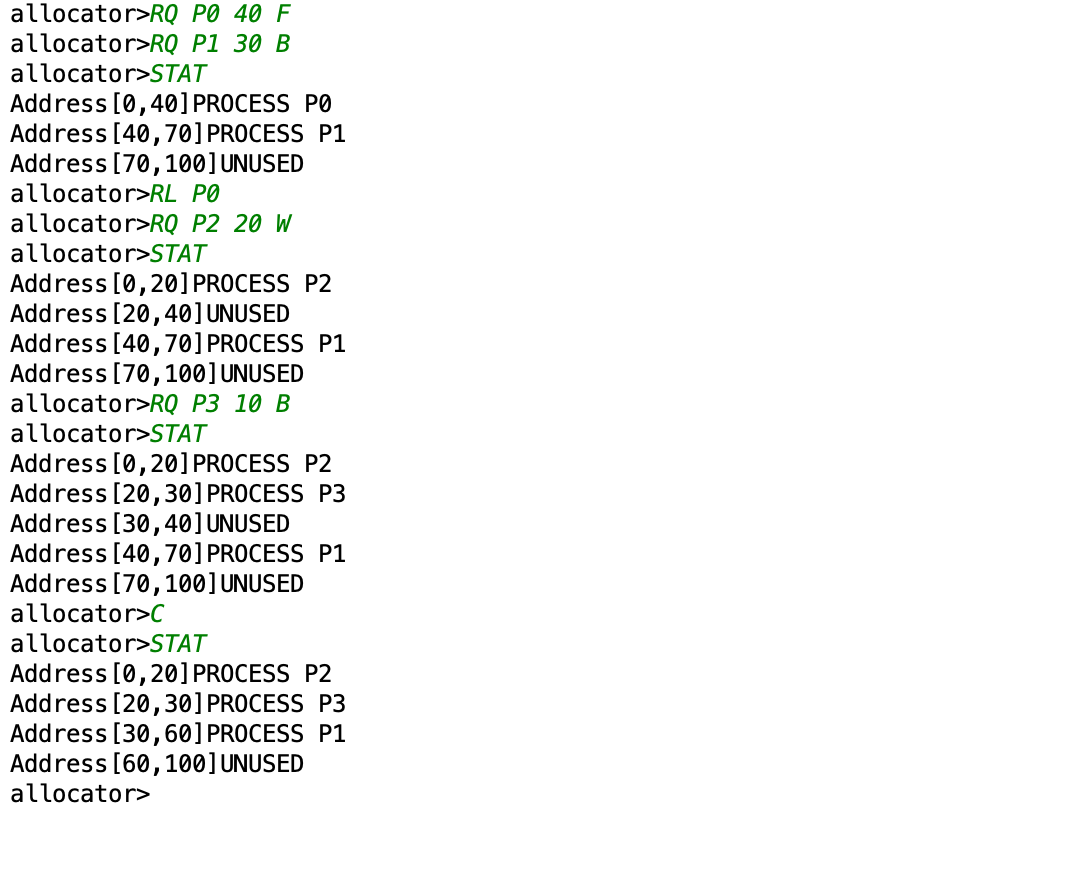
实验结果以及分析：

在验证算法可靠性时我们应该设计一种场景，让这种场景包含算法的所有内容，并且能够验证其的真实有效性。

我们设计如下场景：

先firstfit 以及bestfit两片内存区，删除第一个firstfit 的区域，然后使用worstfit，观察它到底分配在了哪里，然后使用bestfit再验证bestfit的有效性。这时候一定会有碎片产生，我们使用compaction来观察紧凑后的结果如何。

具体结果如下所示：



由第一二个RQ可以通过STAT看到具体的内存分配情况，移除第一个内存区后，使用worstfit，面临两个选择，选择大小为40的还是大小为30的更为worst？经过验证，选择了前者，说明worstfit成立，那么这时候就剩下两片20和30的内存区未被分配，使用bestfit分配一个空间为10的内存区，是选择20还是选择30的区域进行分配呢？经过实验验证，选择了前者，说明bestfit是成立的。然后我们通过指令C进行紧凑操作，可以看到，两片大小为10与30的内存区最后被成功合成了一个大小为40的内存区，且其他的进程内存区的位置也被重新分配了。

经过初步验证，实验结果有效。

**实验遇到的问题：**

遇到的主要问题可能就是内存区域的表示，参考了许多博客，大部分都是用链表等比较复杂的结构来实现的，后来看到了一篇博客，借鉴了一下，用这种结构实现，我觉得很通俗易懂。

另外就是算法方面，刚开始由于对这种内存表示结构没有彻底的理解，忘记了后移操作，造成了很严重的错误。

最后就是对于消除内存空间的算法，忘记了合并左右空闲的区域，但是后来调试的过程中，很快就发现了问题并且解决。

**内存分配方式：**

1. 最先适应算法：  
   算法原理分析：  
   将空闲的内存区按其在储存空间中的起始地址递增的顺序排列，为作业分配储存空间时，从空闲区链的始端开始查找，选择第一个满足要求的空闲区，而不管它究竟有多大  
   优点:  
   1.在释放内存分区的时候，如果有相邻的空白区就进行合并，使其成为一个较大的空白区  
   2.此算法的实质是尽可能的利用储存器的低地址部分，在高地址部分则保留多的或较大的空白区，以后如果需要较大的空白区，就容易满足  
   缺点：  
   1.在低地址部分很快集中了许多非常小的空白区，因而在空白区分配时，搜索次数增加，影响工作效率。
2. 最坏适应算法  
   算法原理分析：  
   扫描整个空闲分区或者链表，总是挑选一个最大的空闲分区分割给作业使用  
   优点：可以使得剩下的空闲分区不至于太小，产生碎片的几率最小，对中小作业有利，同时该算法的查找效率很高  
   缺点：会使得储存器中缺乏大的空闲分区
3. 最佳适应算法  
   算法原理分析：  
   从全部空闲区中找出满足作业要求的，且大小最小的空闲分区的一种计算方法，这种方法能使得碎片尽量小，为适应此算法，空闲分区表中的空闲分区要按从小到大进行排序，自表头开始查找第一个满足要求的自由分区分配  
   优点：能使得碎片尽量的小,保留了最大空闲区  
   缺点：造成了许多小的空闲区

**参考资料：**

《操作系统概念》

**建议：**

这个实验内容非常好，充分结合了课本上的教学内容，也同时锻炼了编程技巧，非常好。