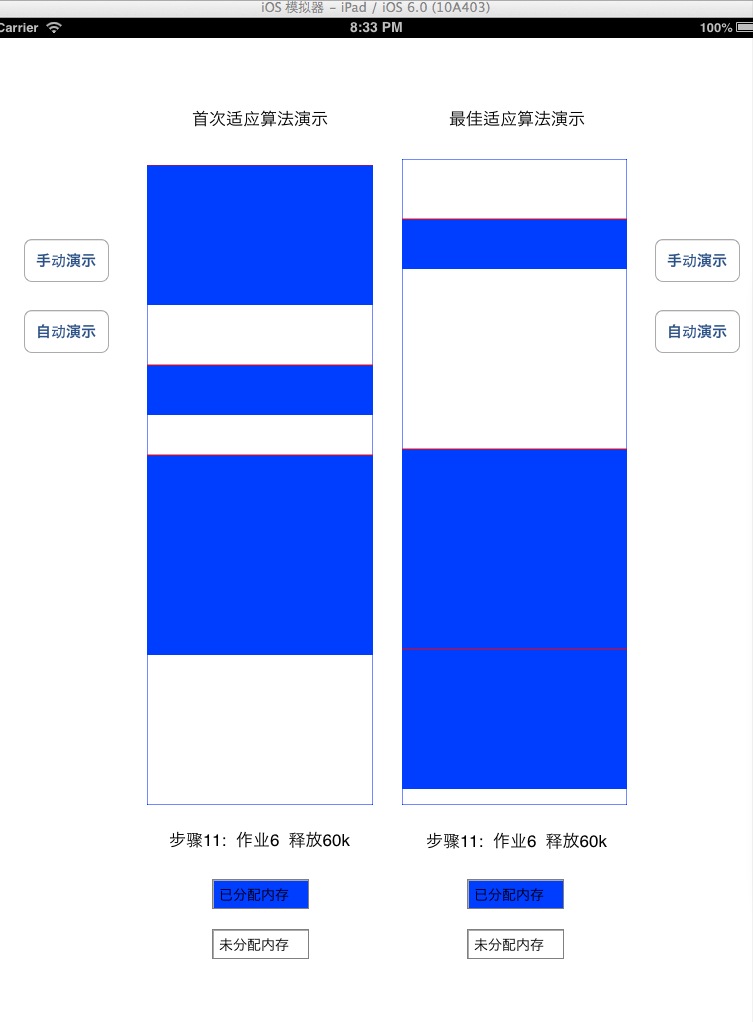
**内存分配设计报告**

1152745 邱峰

1. **程序使用说明**



上图为最先适配算法和最佳适应算法都执行完11条任务后的结果。

自动演示按钮会关闭手动演示按钮，等到自动演示结束后，手动演示按钮才可以使用。

手动演示按钮每点一次，会进行一次步骤演示。

1. **程序设计图**

主要的类有Memory类

改类共有4个对外主要功能的函数

applyMemory(int memorySize)

像内存申请一段大小为memorySize的空间,成功返回申请到的首地址，失败返回-1

addMemory(int idNumber,int memorySize,int start)

在内存地址为start的地方加入编号为idNumber，且大小为memorySize的任务

freeMemroy(int idNumber)

释放编号为idNumber的内存

clearMemory()

释放当前所有的任务，清空内存。

Memroy有2个NsMutableArray（可变数组，这边被我改造成链表）的属性，freeMmeory和useMemory分别记录空闲的内存和已经被使用的内存。

useMemory和freeMemory中的元素为一个struct，包括以下变量

int start 每一块空闲内存的起始位置

int memorySize 每一块空闲内存的大小

int idNumber 若是被使用，则idNumber为多少。

1. **实现思想**

applyMemory(int memorySize)

根据whoAmI确定申请内存的算法

如果是最先适配算法，则从链表的头开始，找到一个freeMemroy[i].memorySize>=memorySize的地方，记录freeMemory[i].start，同时，对freeMemroy[i]做出相应修改，修改分为2种情况：

1. 如果是freeMemroy[i].memorySize==memorySize，直接将freeMemroy[i]从链表中删除
2. 如果freeMemroy[i].memorySize>memorySize，修改freeMemroy[i]的start和memroySize

如果是最佳适配算法，则在所有freeMemroy[i].memorySize>=memorySize中，找到一个freeMemroy[i].memorySize最小的freeMemroy[i]，并且按上述方法修改freeMemroy[i]。

addMemory(int idNumber,int memorySize,int start)

这里要做的是一个useMemroy链表的维护操作，按照start的顺序，对链表进行维护，维护的算法略。

freeMemroy(int idNumber)

这里是我觉得整个项目最难的地方

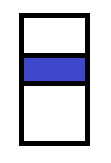
首先需要找到useMemroy[indexUse].idNumber==idNumber的indexUse。

第二部，将useMemroy[indexUse]从useMemory链表中删除。

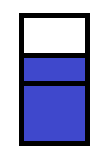
最后要做的是维护freeMemory链表，由于释放了新的内存出来，所以，freeMemroy链表的维护成为了一个新的问题。

链表的维护分为4总情况：

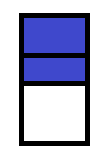
1. 可进行上中下合并（关于这种情况，题目中给出的数据并不能够显示出来，而这种情况也是容易被忽略的情况）

如图：，当释放中间蓝色的区域时，需要将上面那块内存和下面那块内存，以及释放的内存合并为一块。具体操作属于链表的基础操作,笔者在这里不做阐述。

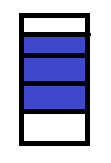
1. 只能进行向上的合并

如图，当释放掉中间那个内存块时，需要和上面的那个内存块合并，因此，需要修改上面那个freeMemory内存块的memorySize

1. 只能进行向下的合并

如图，当释放中间蓝色的区域时，需要修改下面那个内存块的memorySize和start。

1. 上下都不能合并

如图，当释放最中间那个内存块时，需要在freeMemory中增添一个新的memory，并且需要注意维护好链表。

上述3个函数基本就是本次项目的核心函数了。有了这3个函数，每次向内存加入一个任务，只需要先applyMemory(memorySize),如果返回不是-1，则addMemory(idNumber, memorySize,start)，其中start为applyMemroy返回的值。

删除一个任务，直接执行freeMemroy( idNumber)即可

**四、可以改进的地方**

这个项目有一个缺点，就是在applyMemory的时候，我直接修改了freeMemory，但是，实际中这一步并没有把任务加入到内存，所以我觉得这里没有必要修改freeMemory，而修改的目的主要是防止在没有addMemroy的时候，这块内存被其他任务抢占了。所以，最终我还是在applayMemory时修改了freeMemroy。

这样，造成的一个问题就是，如果一个任务，applyMemory，但是没有执行addMemroy，造成这块内存一直被它霸占着，别人都没办法用了，之后也无法释放。

1. **心得体会**

这个项目是我在完成页面调度之后完成的，个人感觉这个项目难度其实

比页面调度的那个项目要难。难度的增加在于链表的维护上面出现了4种情况的分歧，其中，第一种情况题目中给的数据更是无法看出来，需要自己经过认真的思考才能发现。而这四种情况的增加，也增大了维护链表的难度，尤其是第一种情况。它需要修改一个链表节点，删除一个链表节点，很容易就遗漏。

Hint：由于源代码为objective-c的代码，所以，本文档函数接口与源代码略有差异，写过oc代码的可以看出文档与源代码的差异所在