实验四 内存分配算法

1. 实验目的

利用教师提供的图形包代码，编写程序，实现内存的分配算法（最佳适配和邻近适配）。

1. 实验内容

* 在所提供的程序基础之上实现最佳适配和邻近适配算法。
* 增加测试案例，验证程序的正确性

删除掉一个分配的进程（进程3）

重新分配一个进程，假设进程6大小为100

* 利用VC++6.0实现上述程序设计和调试操作。
* 通过阅读和分析实验程序，熟悉内存分配算法。

1. 实验步骤

本实验要求利用事先已经编写好的图形包软件模拟内存的分配。

实现动态分区的分配算法。最佳适配算法选择内存空闲块中最适合进程大小的块分配。邻近适配算法从上一次分配的地址开始查找符合要求的块，所查找到的第一个满足要求的空闲块就分配给进程。

1. 主要程序结构（附注释）

本次实验将待分配的进程和内存空间视作整体进行操作，则应构造两个结构体分别用来表示。各结构体的参数以及注解如下所示：

//内存块结构 表示主存空间占用情况(链表形式)

typedef struct MemoryBlock{

long StartAddr;//内存块始址/KB

long BlockLength;//内存块长度/KB

int JobIndex;//属于哪个作业 nBelongToJob=0表示"空闲"

MemoryBlock \*nextPointer;//指向下一个MemoryBlock

MemoryBlock \*prePointer;//指向上一个MemoryBlock

};

//作业控制块信息结构

typedef struct JCBInfo{

char \*JobName;//作业名

int JobIndex;//作业序号

int JobLength;//作业大小

int \*JobPageTable;//页表首地址

JBCInfo \*nextPointer;//指向下一个JBCInfo

JBCInfo \*prePointer;//指向上一个JBCInfo

};

1. 主要算法
2. 首次适配算法

首次适配算法是指在分配给进程内存的时候，始终将进程分配给第一个可以装下该进程的空间。算法的实质是遍历整个内存，找到当前未被分配的空间并与内存的大小进行比较，如空间大小大于进程的大小则直接装入。

分配时，创建一个进程大小的新空间tmpMemBlock（指针），新空间的信息由进程的的信息以及被分配的空间的信息构成。然后将新空间装入被分配的空间中，prePointer指向被分配的空间prePointer指针指向的模块，nextPointer指针指向分配的空间;同时改变被分配的空间的初始地址和长度，并将其prePointer指针指向新空间。当然还要考虑被分配的进程是不是被放在內存的第一个空间块这一情况。

1. 最佳适配算法

最佳适配算法与首次适配算法的不同是，系统选择所有能容纳进程的空间中最小的那个来存放该进程。因此，需定义变量BestArea\_Size和Wantted\_Position，分别用来保存当前最适配空间的大小和保存当前最适配空间的她址，在遍历整个內存的同时，不断刷新两个变量的值以找到可以存放进程的最小空间。

分配时，与首次适配算法分配时的描述相同。

1. 下次适配算法

下次适配算法类似于首次适配算法，也是将进程分配给第一个可以装下该进程的空间。不同的是下次适配算法每次是从上一次分配的地址开始遍历内存，因此需要设置一个全局变量MarkPoint，用来标记当前分配后指定的位置，注意当MarkPoint的值变成null时，要重新赋值使其跳转到内存的链表头部。除此之外的步骤与首次适配算法相同。

分配时，与首次适配算法分配时的描述相同。

1. 内存释放算法

因为内存的大小只有1024KB，所以如果不对进程进行释放，很快就会空间不足。对于内存释放算法，最基本的操作是将该内存块的JobIndex改为0(空闲状态)，除此之外还需要考虑多种的情况来进一步合并空闲空间，假定被释放块的指针为p：

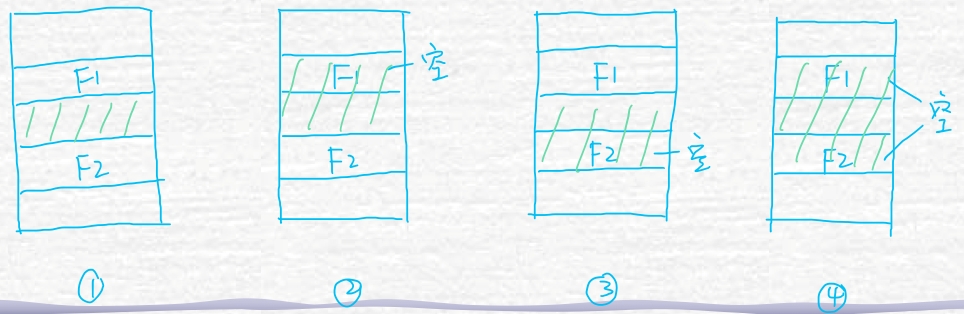
1. 被释放的位置在内存的顶部

这种情况的只需考虑下一个内存块是否处于空闲状态。如果不处于被占用的状态则释放完成；否则，需要改变p的BlockLength值（空间合并）和nextPointer值（指向下一个内存块的nextPointer所指向的模块），其他的值不变。

1. 被释放的位置在内存的底部

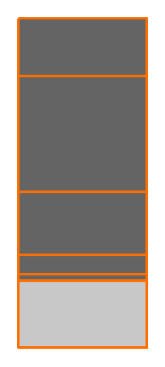
这种情况只需考虑上一个内存块是否处于空闲状态。如果不处于被占用的状态则释放完成；否则，需要改变p的BlockLength值（空间合并）和prePointer值（指向下一个内存块的prePointer所指向的模块），还有p的StartAddr(由于向上合并指针上移），其他的值不变。

1. 被释放的位置在内存的中间

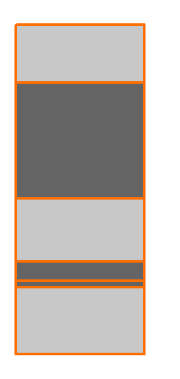


这种情况最为复杂，需要考虑的是上下两个内存是否处于空闲状态。如果不处于被占用的状态则释放完成，否则，若上一块处在占用状态，需要改变p的BlockLength值（空间合并）和prePointer值（指向下一个内存块的prePointer所指向的模块），还有p的StartAddr(由于向上合并指针上移）；若下一块处在占用状态，需要改变p的BlockLength值（空间合并）和nextPointer值（指向下一个内存块的nextPointer所指向的模块），其他的值不变。

1. 执行结果
2. 初始界面(已用首次适配算法)



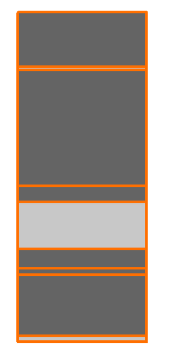
1. 删除作业1、3(测试)



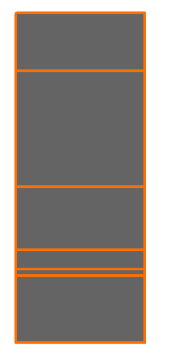
1. 测试最佳适配算法(插入大小为20、186、20的作业)



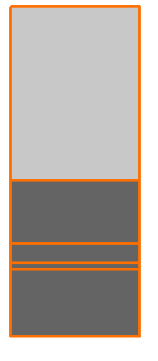
1. 测试下次适配算法(插入大小为190、170、50的作业)



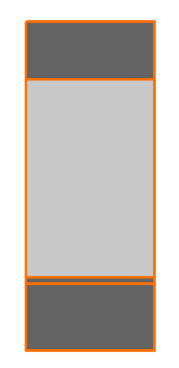
1. 测试内存释放算法(先变成全满状态)



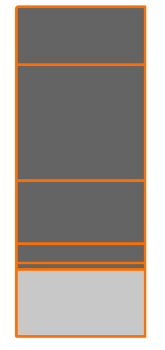
1. 先删作业2再删作业1



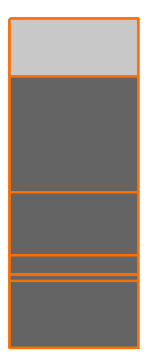
1. 先删作业2、4再删作业3



1. 删最后一个作业



1. 删作业1



1. 问题及解决方法
2. 对MFC内容不了解，在显示结果方面程序比较难执行。上网查阅资料才得以进一步掌握。
3. 其实内存分配算法在本质上来说是对指针的操作，只需要注意指针与指针之间的关系以及不要出现空指针即可。
4. 相比之下释放内存的算法中要考虑比较多的可能性，不过细心一点并与同学讨论一下也可以基本上很顺利的写出来，程序中共给出了五种可能的结果并附有详细的注解。
5. 在下次适配算法的实现中定义了一个MarkPoint指针来指向最后一次分配的地址位置。这里就要注意如果再执行释放算法时，要先判断释放的位置是不是刚好处在MarkPoint指针所在的位置，如果是，则释放完内存后紧接着运行下次适配算法则会出现问题。要在每次的释放算法中向上合并的过程中加上对MarkPoint指针值的修改。