1. 5 实验问题

动态分区管理中，分配算法采用最佳适应法，实现内存的分配和回收。

1. 测试用例

内存空间：16

|  |  |
| --- | --- |
| 进程名 | 进程大小 |
| P1 | 10 |
| P2 | 2 |
| P3 | 4 |
| P4 | 5 |
| P5 | 6 |
| P6 | 3 |
| P7 | 1 |

1. 程序设计

数据结构：

物理结构：

struct free\_sect//空闲分区结构定义

{

int base\_address;//空闲分区的起始地址

int sect\_len;//空闲分区的长度

};

struct used\_sect//已分配的内存分区结构定义

{

string jobname;//作业名

int base\_address; //作业所占分区的起始地址

int sect\_len;//作业所占分区的长度

};

逻辑结构：

list<free\_sect>freetable;//空闲分区表

list<used\_sect>usedtable;//已分配分区表

1. 算法设计

分配：

S0：检测是否有足够大的空闲区可供分配

是，则s1；否，则等待回收

S1：选取最优节点分配：

S1.1：把空闲区表头设为初始最优节点，找到更小的能满足的区间则替换。

S1.2：当初始节点不能满足进程大小时，最优节点（随迭代器）后移一个块。

S2：更新已分配区表和空闲区表：创建新的已分配节点，写入进程名和计算出的进程起始地址，插入已分配区表

S2.1：当空闲区分配后没有碎片，删除当前空闲区

S2.1：当空闲区分配后有碎片，更新空闲区信息为剩下的碎片区

回收：

S0：检测当前回收的进程是否在内存中

是，则s1；否，则退出

S1：把选中的块写入新的free\_sect类型的节点，算出首地址，再从已分配区表删去当前选中的块，并判断回收的块的位置关系状态

S2.1：下界是空闲区

当前节点合并入空闲区。

S2.1.1:上下界都是空闲区

上邻空闲区与当前节点合并入下邻空闲区。

S2.2：下界不是空闲区而上界是空闲区

当前节点合并入上邻空闲区。

S2.3：上下界都不是空闲区

不合并空闲区，并把新节点插入空闲区表。

主函数（操作界面）：

Case 0: exit

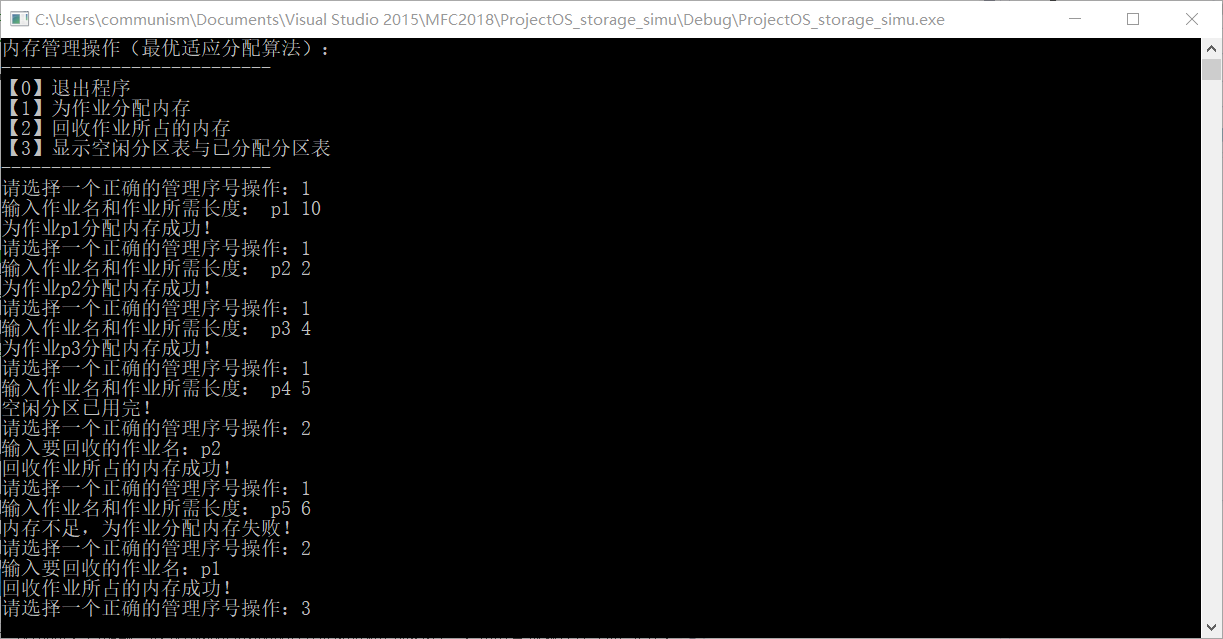
Case 1: allocate

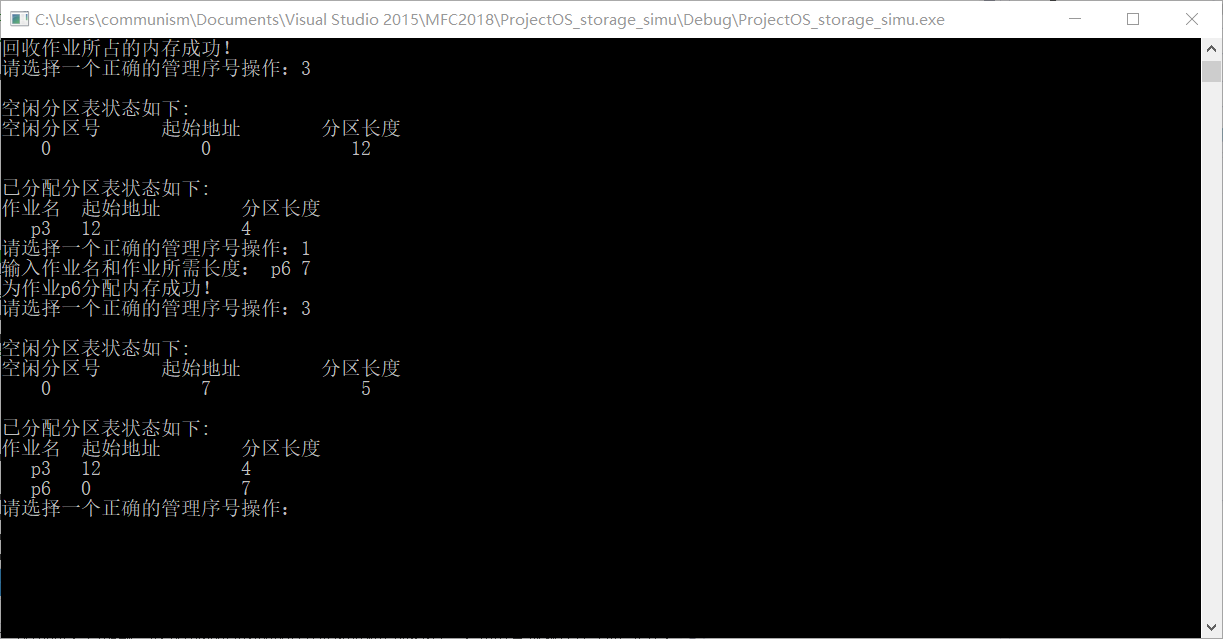
Case 2: reclaim

Case 3: display

1. 运行与测试

程序运行截图：





正确性分析：

进程运行模拟序列：p1—p2—p3（分配成功）

P4（内存不足）空闲区为空，无法分配

P2进程终止，回收内存，出现空闲区，但是空闲区不足够分配p5进程

P1进程终止，回收内存，展示空闲区与已分配区，当前空闲区已经合并（因为相邻）

P6进程进入内存，展示分配后的空闲区与已分配区

两次展示与p4，p5的测试结果都符合期望结果。