

实验报告目录

01 实验概述

1.1 实验目的

1.2 实验要求

02 实验设计与实现

2.1 实验设计概述

2.2 数据结构与全局定义

2.3 首次适应算法实现

2.3.1 内存分配

2.3.2 内存回收

2.4 最佳适应算法实现

2.4.1 内存分配

2.4.2 内存回收

2.5 内存分配与回收封装

2.5.1 内存分配封装

2.5.2 内存回收封装

2.6 打印内存状态

2.7 主函数

03 实验测试与结果

3.1 首次适应算法测试

3.2 最佳适应算法测试

04 实验总结

01 实验概述

1.1 实验目的

通过编写模拟程序以加深对动态分区分配方式及其实现过程的理解。

1.2 实验要求

编写C程序，模拟实现动态分区存储管理方案。要求实现**首次/最佳**适应算法的内存块分配和回收，假设初始状态下，可用内存空间是一块连续区域，大小为102400B。要求实现的功能有：

1. 分配内存
2. 回收内存
3. 显示内存（显示空闲分区和已分配分区的情况）

02 实验设计与实现

2.1 实验设计概述

- 首次适应算法
 - 空闲分区链表按地址升序排列；
 - 分配：从头开始查找，找到第一个满足大小的空闲分区，分配给进程；
 - 回收：将回收的分区插入空闲分区链表中，若与前后空闲分区相邻，则合并；回收时需保持空闲分区链表仍按地址升序排列；
- 最佳适应算法
 - 空闲分区链表按分区大小升序排列；
 - 分配：从头开始查找，找到第一个满足大小的空闲分区，分配给进程；
 - 回收：将回收的分区插入空闲分区链表中，若与前后空闲分区相邻，则合并；回收时需保持空闲分区链表仍按分区大小升序排列；

2.2 数据结构与全局定义

由于动态内存分配需要对空闲内存块表、已分配内存块表频繁进行插入和删除操作，因而使用可 $O(1)$ 增删的链表组织信息。

这里使用C++中的STL list实现本实验所需的链表功能，并定义如下固定长度内存分区结构体：

```
1 // 固定大小内存分区结构
2 struct Block
3 {
4     string name;    // 进程名
5     int startAddr; // 起始地址
6     int size;       // 大小
7     Block(string name, int startAddr, int size) :
8         name(name), startAddr(startAddr), size(size) {}
9 };
```

定义本实验全局变量：

- 内存总空间大小 MAX_SIZE ；
- 空闲分区链表 freeList ；
- 已分配分区连边 allocList ；

```
1  const int MAX_SIZE = 102400;
2
3  list<Block> freeList; // 空闲分区链表
4  list<Block> allocList; // 已分配分区链表
```

对链表进行初始化，将可用内存空间作为一整块空闲分区：

```
1  // 初始化，将可用内存空间作为一整块空闲分区
2  void Init()
3  {
4      freeList.clear();
5      allocList.clear();
6      freeList.push_back(Block("1", 0, MAX_SIZE));
7  }
```

2.3 首次适应算法实现

2.3.1 内存分配

首次适应算法的内存分配实现如下：

```
1  // 首次适应算法分配内存（空闲分区链表按照起始地址排序）
2  // 输入：申请空间名称和大小
3  // 输出：分配是否成功
4  bool FirstFitAlloc(string name, int size)
5  {
6      // 遍历空闲分区链表，找到第一个满足要求的分区
7      for (auto it = freeList.begin(); it != freeList.end(); it++)
8      {
9          if (it->size >= size)
10         {
11             // 创建分配分区
```

```

12         Block allocBlock(name, it->startAddr, size);
13
14         // 插入已分配分区链表, 按照起始地址排序
15         auto it2 = allocList.begin();
16         while (it2 != allocList.end() && it2->startAddr < allocBlock.startAddr)
17             it2++;
18         allocList.insert(it2, allocBlock);
19
20         // 更新空闲分区链表
21         if (it->size == size)
22         {
23             // 分区大小相等, 删除当前空闲分区
24             freeList.erase(it);
25         }
26         else
27         {
28             // 分区大小不等, 更新空闲分区, 起始地址后移, 大小减小
29             it->startAddr += size;
30             it->size -= size;
31         }
32         return true;
33     }
34 }
35
36 return false;
37 }

```

2.3.2 内存回收

首次适应算法的内存回收实现如下:

```

1 // 首次适应算法回收内存
2 // 输入: 回收空间名称
3 // 输出: 与回收分区合并的空闲分区数, -1表示回收失败(未找到)
4 int FirstFitFree(string name)
5 {
6     // 遍历已分配分区链表, 找到要回收的分区
7     for (auto it = allocList.begin(); it != allocList.end(); it++)
8     {
9         if (it->name == name)
10        {
11            // 创建空闲分区
12            Block freeBlock(name, it->startAddr, it->size);

```

```

13
14 // 插入空闲分区链表，按照起始地址排序
15 auto it2 = freeList.begin();
16 while (it2 != freeList.end() && it2->startAddr < freeBlock.startAddr)
17     it2++;
18
19 freeList.insert(it2, freeBlock);
20 it2--; // 将指针移动到新插入分区位置
21
22 // 更新已分配分区链表
23 allocList.erase(it);
24
25 // 合并空闲分区
26 // 仅可能与前后两个分区合并，
27 // 检查是否存在且地址是否连续
28 int mergeCount = 0;
29 auto itBefore = it2;
30 itBefore--; // 前一个分区
31 if (it2 != freeList.begin() &&
32     itBefore->startAddr + itBefore->size == it2->startAddr)
33 {
34     // 与前一个分区合并
35     // 保留当前分区，删除前一个分区
36     it2->size += itBefore->size;
37     it2->startAddr = itBefore->startAddr;
38     freeList.erase(itBefore);
39     mergeCount++;
40 }
41
42 auto itAfter = it2;
43 itAfter++; // 后一个分区
44 if (itAfter != freeList.end() &&
45     it2->startAddr + it2->size == itAfter->startAddr)
46 {
47     // 与后一个分区合并
48     // 保留当前分区，删除后一个分区
49     it2->size += itAfter->size;
50     freeList.erase(itAfter);
51     mergeCount++;
52 }
53 return mergeCount;
54 }
55 }
56 return -1; // 未找到name对应的分区
57 }

```

2.4 最佳适应算法实现

2.4.1 内存分配

最佳适应算法的内存分配实现如下：

```
1 // 最佳适应算法分配内存（空闲分区链表按照大小排序）
2 // 输入：申请空间名称和大小
3 // 输出：分配是否成功
4 bool BestFitAlloc(string name, int size)
5 {
6     // 遍历空闲分区链表，找到最小的满足要求的分区
7     // 由于空闲分区链表按照大小排序，因此只需找到首个满足大小的分区即可
8     auto it = freeList.begin();
9     while (it != freeList.end() && it->size < size)
10         it++;
11
12     // 如果找到最佳分区
13     if (it != freeList.end())
14     {
15         // 创建分配分区
16         Block allocBlock(name, it->startAddr, size);
17
18         // 插入已分配分区链表，按照起始地址排序
19         // 即插入到第一个大小大于等于当前分区的分区之前
20         auto it2 = allocList.begin();
21         while (it2 != allocList.end() && it2->startAddr < allocBlock.startAddr)
22             it2++;
23         allocList.insert(it2, allocBlock); // 注意：insert插入it位置之前
24
25         // 更新空闲分区链表
26         if (it->size == size)
27         {
28             // 分区大小相等，删除当前空闲分区
29             freeList.erase(it);
30         }
31         else
32         {
33             // 分区大小不等，更新空闲分区，起始地址后移，大小减小
34             it->startAddr += size;
35             it->size -= size;
```

```

36         }
37         return true;
38     }
39
40     return false;
41 }
42

```

2.4.2 内存回收

最佳适应算法的内存回收实现如下：

```

1 // 最佳适应算法回收内存
2 // 回收分区按照分区大小排序，合并后仍按照分区大小排序
3 // 输入：回收空间名称
4 // 输出：与回收分区合并的空闲分区数，-1表示回收失败(未找到)
5 int BestFitFree(string name)
6 {
7     // 遍历已分配分区链表，找到要回收的分区
8     for (auto it = allocList.begin(); it != allocList.end(); it++)
9     {
10         if (it->name == name)
11         {
12
13             // 创建空闲分区
14             Block freeBlock(name, it->startAddr, it->size);
15
16             // 删除已分配分区
17             allocList.erase(it);
18
19             int mergeCount = 0;
20             // 首先检查是否可与前后两个分区合并
21             for (auto it2 = freeList.begin(); it2 != freeList.end(); it2++)
22             {
23                 // 如果it2是freeBlock的前一个分区
24                 if (it2->startAddr + it2->size == freeBlock.startAddr)
25                 {
26                     // 将it2合并到freeBlock，删除it2
27                     freeBlock.startAddr = it2->startAddr;
28                     freeBlock.size += it2->size;
29                     freeList.erase(it2);
30                     mergeCount++;
31                 }

```



```

32
33         // 如果it2是freeBlock的后一个分区
34         if (it2->startAddr == freeBlock.startAddr + freeBlock.size)
35         {
36             // 将it2合并到freeBlock, 删除it2
37             freeBlock.size += it2->size;
38             freeList.erase(it2);
39             mergeCount++;
40         }
41
42         // 如果已经合并了两个分区, 退出循环
43         if (mergeCount == 2)
44             break;
45     }
46
47     // 插入空闲分区链表, 保持大小升序排序
48     auto it2 = freeList.begin();
49     while (it2 != freeList.end() && it2->size < freeBlock.size)
50         it2++;
51     freeList.insert(it2, freeBlock);
52
53     return mergeCount;
54 }
55 }
56 return -1; // 未找到name对应的分区
57 }

```

2.5 内存分配与回收封装

实验需实现两种内存分配算法, 为方便调用, 将内存分配和回收封装为函数, 使得根据输入选择, 传入不同算法的函数指针即可实现不同算法的内存分配和回收。

2.5.1 内存分配封装

```

1 // 读入申请空间名称和大小, 根据传入的分配函数指针分配内存
2 // (分配函数指针可为FirstFitAlloc或BestFitAlloc)
3 void Alloc(bool (*AllocFunc)(string, int))
4 {
5     string name;
6     int size;
7     cout << "请输入申请空间名称 > ";

```

```

8      cin >> name;
9      cout << "请输入申请空间大小 > ";
10     cin >> size;
11
12     if (AllocFunc(name, size))
13         cout << "分配成功!" << endl;
14     else
15         cout << "分配失败!" << endl;
16 }
17

```

2.5.2 内存回收封装

```

1  // 读入回收空间名称，根据传入的回收函数指针回收内存
2  // （回收函数指针可为FirstFitFree或BestFitFree）
3  void Free(int (*FreeFunc)(string))
4  {
5      string name;
6      cout << "请输入回收空间名称 > ";
7      cin >> name;
8
9      int mergeCount = FreeFunc(name);
10     if (mergeCount == -1)
11     {
12         cout << "回收失败!" << endl;
13     }
14     else
15     {
16         cout << "回收成功! 合并空闲分区数: " << mergeCount << endl;
17     }
18 }

```

2.6 打印内存状态

打印内存空闲分区和已分配分区的状态，实现如下：

```

1  void Show()
2  {
3      // 打印空闲分区
4      cout << "\n空闲分区: " << endl;
5      cout << "编号  起始地址  大小(B)\n";

```

```

6      cout.setf(ios::right);
7
8      int idx = 0;
9      for (auto it = freeList.begin(); it != freeList.end(); it++)
10     {
11         cout << setw(3) << ++idx
12             << setw(9) << it->startAddr
13             << setw(10) << it->size
14             << endl;
15     }
16     cout << endl;
17
18     // 打印已分配分区
19     cout << "已分配分区: " << endl;
20     cout << "名称  起始地址  大小(B)\n";
21     for (auto it = allocList.begin(); it != allocList.end(); it++)
22     {
23         cout << setw(3) << it->name
24             << setw(9) << it->startAddr
25             << setw(10) << it->size
26             << endl;
27     }
28 }

```

2.7 主函数

初始化内存空间，根据用户选择设置Alloc和Free对应算法，循环读入用户输入，根据输入选择执行不同操作，实现如下：

```

1      int main()
2      {
3          Init();
4
5          string op; // string 避免异常输入
6          do
7          {
8              cout << "===== 动态分区模拟实验 =====\n";
9              cout << "选择分配算法(1.首次适应算法 2.最佳适应算法),\n";
10             cout << "输入算法编号 > ";
11             cin >> op;
12         } while (op != "1" && op != "2");
13

```

```

14 // 根据用户选择的分配算法，设置分配函数指针
15 auto AllocFunc = (op == "1") ? FirstFitAlloc : BestFitAlloc;
16 auto FreeFunc = (op == "1") ? FirstFitFree : BestFitFree;
17
18 while (true)
19 {
20     cout << endl;
21     cout << "=====\n";
22     cout << "选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),\n";
23     cout << "请输入操作编号 > "; cin >> op;
24
25     if (op == "1")
26         Alloc(AllocFunc);
27     else if (op == "2")
28         Free(FreeFunc);
29     else if (op == "3")
30         Show();
31     else if (op == "4")
32         break;
33     else
34         cout << "输入错误，请重新输入!" << endl;
35 }
36
37 return 0;
38 }

```

03 实验测试与结果

3.1 首次适应算法测试

```
1 PS D:\files\Course\2023Fall-操作系统\lab\lab6>.\dynamic.exe
2 ===== 动态分区模拟实验 =====
3 选择分配算法(1.首次适应算法 2.最佳适应算法),
4 输入算法编号 > 1
5
6 =====
7 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
8 请输入操作编号 > 3
9
10 空闲分区:
11 编号 起始地址 大小(B)
12     1         0   102400
13
14 已分配分区:
15 名称 起始地址 大小(B)
16
17 =====
18 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
19 请输入操作编号 > 1
20 请输入申请空间名称 > s1
21 请输入申请空间大小 > 100
22 分配成功!
23
24 =====
25 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
26 请输入操作编号 > 3
27
28 空闲分区:
29 编号 起始地址 大小(B)
30     1     100   102300
31
32 已分配分区:
33 名称 起始地址 大小(B)
34     s1         0    100
```

```

35
36 =====
37 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
38 请输入操作编号 > 1
39 请输入申请空间名称 > s2
40 请输入申请空间大小 > 200
41 分配成功!
42
43 =====
44 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
45 请输入操作编号 > 1
46 请输入申请空间名称 > s3
47 请输入申请空间大小 > 300
48 分配成功!
49
50 =====
51 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
52 请输入操作编号 > 1
53 请输入申请空间名称 > s4
54 请输入申请空间大小 > 400
55 分配成功!
56
57 =====
58 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
59 请输入操作编号 > 3
60
61 空闲分区:
62 编号 起始地址 大小(B)
63 1 1000 101400
64
65 已分配分区:
66 名称 起始地址 大小(B)
67 s1 0 100
68 s2 100 200
69 s3 300 300
70 s4 600 400
71
72 =====
73 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
74 请输入操作编号 > 2
75 请输入回收空间名称 > s1
76 回收成功! 合并空闲分区数: 0
77
78 =====
79 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),

```

```

80      请输入操作编号 > 3
81
82      空闲分区:
83      编号  起始地址  大小(B)
84          1         0      100
85          2      1000     101400
86
87      已分配分区:
88      名称  起始地址  大小(B)
89      s2     100      200
90      s3     300      300
91      s4     600      400
92
93      =====
94      选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
95      请输入操作编号 > 2
96      请输入回收空间名称 > s3
97      回收成功! 合并空闲分区数: 0
98
99      =====
100     选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
101     请输入操作编号 > 3
102
103     空闲分区:
104     编号  起始地址  大小(B)
105         1         0      100
106         2      300      300
107         3      1000     101400
108
109     已分配分区:
110     名称  起始地址  大小(B)
111     s2     100      200
112     s4     600      400
113
114     =====
115     选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
116     请输入操作编号 > 2
117     请输入回收空间名称 > s2
118     回收成功! 合并空闲分区数: 2
119
120     =====
121     选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
122     请输入操作编号 > 3
123
124     空闲分区:

```

```

125  编号  起始地址  大小(B)
126      1      0      600
127      2    1000    101400
128
129  已分配分区:
130  名称  起始地址  大小(B)
131      s4      600      400
132
133  =====
134  选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
135  请输入操作编号 > 1
136  请输入申请空间名称 > s5
137  请输入申请空间大小 > 500
138  分配成功!
139
140  =====
141  选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
142  请输入操作编号 > 3
143
144  空闲分区:
145  编号  起始地址  大小(B)
146      1      500      100
147      2    1000    101400
148
149  已分配分区:
150  名称  起始地址  大小(B)
151      s5      0      500
152      s4      600      400
153
154  =====
155  选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
156  请输入操作编号 > 1
157  请输入申请空间名称 > s6
158  请输入申请空间大小 > 6000000
159  分配失败!
160
161  =====
162  选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
163  请输入操作编号 > 2
164  请输入回收空间名称 > s4
165  回收成功! 合并空闲分区数: 2
166
167  =====
168  选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
169  请输入操作编号 > 3

```



```

170
171 空闲分区:
172 编号 起始地址 大小(B)
173     1      500   101900
174
175 已分配分区:
176 名称 起始地址 大小(B)
177    s5        0    500
178
179 =====
180 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
181 请输入操作编号 > 2
182 请输入回收空间名称 > s5
183 回收成功! 合并空闲分区数:  1
184
185 =====
186 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
187 请输入操作编号 > 3
188
189 空闲分区:
190 编号 起始地址 大小(B)
191     1        0   102400
192
193 已分配分区:
194 名称 起始地址 大小(B)
195
196 =====
197 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
198 请输入操作编号 > 4
199 PS D:\files\Course\2023Fal-操作系统\lab\lab6>

```

3.2 最佳适应算法测试

```

1 PS D:\files\Course\2023Fal-操作系统\lab\lab6>.\dynamic.exe
2 ===== 动态分区模拟实验 =====
3 选择分配算法(1.首次适应算法 2.最佳适应算法),
4 输入算法编号 > 2
5

```

```

6 =====
7 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
8 请输入操作编号 > 3
9
10 空闲分区:
11 编号 起始地址 大小(B)
12     1         0    102400
13
14 已分配分区:
15 名称 起始地址 大小(B)
16
17 =====
18 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
19 请输入操作编号 > 1
20 请输入申请空间名称 > s1
21 请输入申请空间大小 > 100
22 分配成功!
23
24 =====
25 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
26 请输入操作编号 > 3
27
28 空闲分区:
29 编号 起始地址 大小(B)
30     1     100    102300
31
32 已分配分区:
33 名称 起始地址 大小(B)
34     s1         0     100
35
36 =====
37 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
38 请输入操作编号 > 1
39 请输入申请空间名称 > s2
40 请输入申请空间大小 > 200
41 分配成功!
42
43 =====
44 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
45 请输入操作编号 > 1
46 请输入申请空间名称 > s3
47 请输入申请空间大小 > 300
48 分配成功!
49
50 =====

```

```

51 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
52 请输入操作编号 > 1
53 请输入申请空间名称 > s4
54 请输入申请空间大小 > 400
55 分配成功!
56
57 =====
58 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
59 请输入操作编号 > 3
60
61 空闲分区:
62 编号 起始地址 大小(B)
63     1      1000    101400
64
65 已分配分区:
66 名称 起始地址 大小(B)
67     s1         0      100
68     s2      100      200
69     s3      300      300
70     s4      600      400
71
72 =====
73 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
74 请输入操作编号 > 2
75 请输入回收空间名称 > s1
76 回收成功! 合并空闲分区数: 0
77
78 =====
79 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
80 请输入操作编号 > 3
81
82 空闲分区:
83 编号 起始地址 大小(B)
84     1         0      100
85     2      1000    101400
86
87 已分配分区:
88 名称 起始地址 大小(B)
89     s2      100      200
90     s3      300      300
91     s4      600      400
92
93 =====
94 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
95 请输入操作编号 > 2

```

```

96      请输入回收空间名称 > s3
97      回收成功! 合并空闲分区数:  0
98
99      =====
100     选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
101     请输入操作编号 > 3
102
103     空闲分区:
104     编号  起始地址  大小(B)
105         1         0      100
106         2       300      300
107         3      1000     101400
108
109     已分配分区:
110     名称  起始地址  大小(B)
111     s2     100      200
112     s4     600      400
113
114     =====
115     选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
116     请输入操作编号 > 1
117     请输入申请空间名称 > s5
118     请输入申请空间大小 > 50
119     分配成功!
120
121     =====
122     选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
123     请输入操作编号 > 3
124
125     空闲分区:
126     编号  起始地址  大小(B)
127         1         50       50
128         2       300      300
129         3      1000     101400
130
131     已分配分区:
132     名称  起始地址  大小(B)
133     s5         0       50
134     s2     100      200
135     s4     600      400
136
137     =====
138     选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
139     请输入操作编号 > 2
140     请输入回收空间名称 > s2

```

```

141 回收成功! 合并空闲分区数:  2
142
143 =====
144 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
145 请输入操作编号 > 3
146
147 空闲分区:
148 编号  起始地址  大小(B)
149     1      50      550
150     2     1000     101400
151
152 已分配分区:
153 名称  起始地址  大小(B)
154    s5      0      50
155    s4     600     400
156
157 =====
158 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
159 请输入操作编号 > 1
160 请输入申请空间名称 > s6
161 请输入申请空间大小 > 6000000
162 分配失败!
163
164 =====
165 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
166 请输入操作编号 > 2
167 请输入回收空间名称 > s4
168 回收成功! 合并空闲分区数:  2
169
170 =====
171 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
172 请输入操作编号 > 3
173
174 空闲分区:
175 编号  起始地址  大小(B)
176     1      50     102350
177
178 已分配分区:
179 名称  起始地址  大小(B)
180    s5      0      50
181
182 =====
183 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
184 请输入操作编号 > 2
185 请输入回收空间名称 > s5

```

```
186 回收成功! 合并空闲分区数:  1
187
188 =====
189 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
190 请输入操作编号 > 3
191
192 空闲分区:
193 编号  起始地址  大小(B)
194    1      0    102400
195
196 已分配分区:
197 名称  起始地址  大小(B)
198
199 =====
200 选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
201 请输入操作编号 > 4
```

04 实验总结

本次实验编写动态分区模拟程序，分别实现了适应算法和最佳适应算法的内存分配和回收功能，对两种算法的特点和区别有了更深入的理解。在本实验的实现过程中，使用了C++中的STL list实现链表，使得链表的插入和删除操作可在 $O(1)$ 完成，提高了程序的效率。