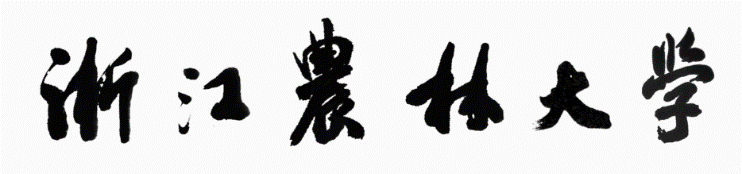
**** 

**《操作系统》期末课程实验报告**

**（2020-2021学年第一学期）**

**Name（姓名）：周炜翔、张金浩**

**Student ID（学号）：201805120532、201805120629**

**Department（学院）：信息工程学院**

**Major（专业）：计算机科学与技术**

**Class（班级）：183班**

**Date（日期）：2021/01/05**

目录

[一、课题设计的目的 3](#_Toc60841653)

[二、课程设计的问题描述 3](#_Toc60841654)

[2.1 获得进程数和块数 3](#_Toc60841655)

[2.2 获取每个块的大小、每个进程请求的大小 3](#_Toc60841656)

[2.3 为了与大小的空闲块由小变大 3](#_Toc60841657)

[2.4 选择可以使用以上定义分配的第一个空闲存储块 3](#_Toc60841658)

[三、详细设计 4](#_Toc60841659)

[3.1 算法流程图 4](#_Toc60841660)

[3.2 操作步骤 4](#_Toc60841661)

[四、源代码 5](#_Toc60841662)

[五、测试结果 11](#_Toc60841663)

[5.1 测试样例表格一览 11](#_Toc60841664)

[5.2 测试样例对应截图 12](#_Toc60841665)

[六、总结 26](#_Toc60841666)

# 一、课题设计的目的

了解、体会、运用最佳适应算法。

# 二、课程设计的问题描述

## 2.1 获得进程数和块数

obtain the number of Processes and number of blocks

## 2.2 获取每个块的大小、每个进程请求的大小

get the size of each block；get the size each process requests

## 2.3 为了与大小的空闲块由小变大

order the free block with size from small to big

## 2.4 选择可以使用以上定义分配的第一个空闲存储块

Then select the first free memory block that can be allocated using the above definition

# 三、详细设计

## 3.1 算法流程图

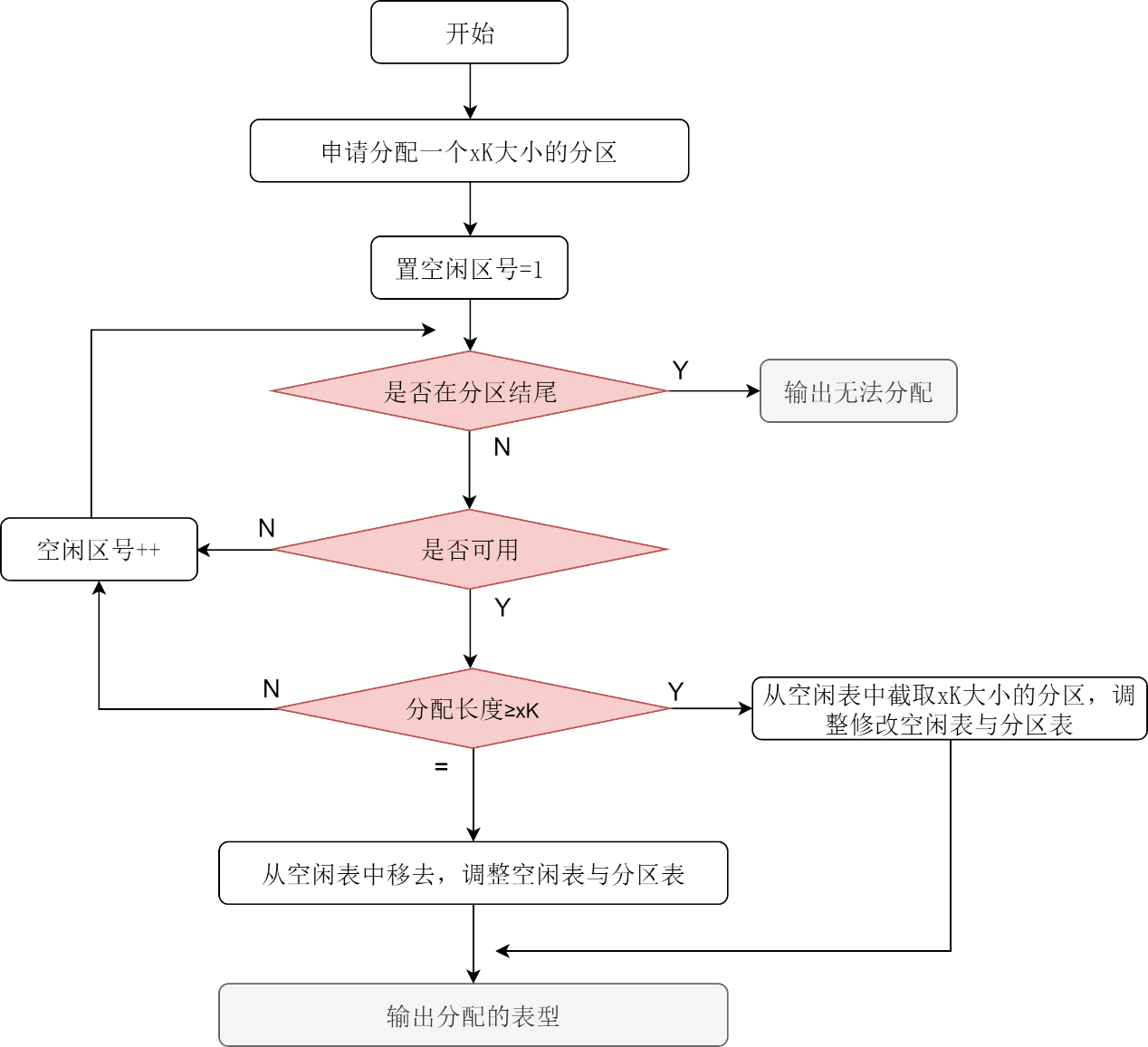


图3-1 算法流程图

## 3.2 操作步骤

1. 进入Best Fit（最佳适应算法）
2. 输入内存块总大小（单位：KB）；
3. 选择操作
   1. 分配内存
   2. 回收内存
   3. 显示内存
   4. 退出
4. 若选择分配内存
   1. 输入分配的内存块数
   2. 输入需要分配的主存大小（单位：KB）
   3. 成功则显示分配完毕
   4. 失败则显示“内存不足，分配失败”
5. 若选择回收内存
   1. 输入要释放的分区号
   2. 选择“已分配”内存后释放的分区号后显示为“空闲”
   3. 选择“未分配”内存后释放的分区号后显示为“错误”
6. 若选择显示内存
   1. 则显示“当前分配情况”
7. 若选择退出
   1. 则程序退出，按任意键继续

# 四、源代码

1. #include <iostream>
2. #include <stdlib.h>
3. **using** **namespace** std;
4. #define Free 0        //空闲状态
5. #define Busy 1        //已用状态
6. #define OK 1          //完成
7. #define ERROR 0       //出错
8. **int** MAX\_length = 640; //定义最大主存信息640KB
9. **typedef** **int** Status;
10. **int** flag; //标志位  0为空闲区     1为已分配的工作区
12. **typedef** **struct** FreAarea //定义一个空闲区说明表结构
13. {
14. **long** size;    //分区大小
15. **long** address; //分区地址
16. **int** state;    //状态
17. } ElemType;
19. **typedef** **struct** DuLNode // 线性表的双向链表存储结构
20. {
21. ElemType data;
22. **struct** DuLNode \*prior; //前趋指针
23. **struct** DuLNode \*next;  //后继指针
24. }
26. DuLNode,
27. \*DuLinkList;
28. DuLinkList block\_first; //头结点
29. DuLinkList block\_last;  //尾结点
30. Status Alloc(**int**);      //内存分配
31. Status free(**int**);       //内存回收
32. Status First\_fit(**int**);  //首次适应算法
33. Status Best\_fit(**int**);   //最佳适应算法
34. **void** show();            //查看分配
35. Status Initblock();     //开创空间表
37. Status Initblock() //开创带头结点的内存空间链表
38. {
39. block\_first = (DuLinkList)malloc(**sizeof**(DuLNode));
40. block\_last = (DuLinkList)malloc(**sizeof**(DuLNode));
41. // fist的前驱和后去
42. block\_first->prior = NULL;
43. block\_first->next = block\_last;
45. // last 的前驱和后续
46. block\_last->prior = block\_first;
47. block\_last->next = NULL;
49. block\_last->data.address = 0;
50. block\_last->data.size = MAX\_length;
51. block\_last->data.state = Free;
53. **return** OK;
54. }
56. Status Alloc(**int** i) //分配主存
57. {
58. **int** request = 0;
59. cout << "操作编号 " << i;
60. cout << " 请输入需要分配的主存大小(单位:KB)：" << endl;
61. cin >> request;
62. **if** (request < 0 || request == 0)
63. {
64. cout << "分配大小不合适，请重试！" << endl;
65. **return** ERROR;
66. }
68. **if** (Best\_fit(request) == OK)
69. cout << "分配成功！" << endl;
70. **else**
71. cout << "内存不足，分配失败！" << endl;
72. **return** OK;
73. }
75. Status Best\_fit(**int** request) //最佳适应算法
76. {
77. **int** ch; //记录最小剩余空间
78. DuLinkList temp = (DuLinkList)malloc(**sizeof**(DuLNode));
79. temp->data.size = request;
80. temp->data.state = Busy;
81. DuLNode \*p = block\_first->next;
82. DuLNode \*q = NULL; //记录最佳插入位置
84. **while** (p) //初始化最小空间和最佳位置
85. {
86. **if** (p->data.state == Free && (p->data.size >= request))
87. {
88. **if** (q == NULL)
89. {
90. q = p;
91. ch = p->data.size - request;
92. }
93. **else** **if** (q->data.size > p->data.size)
94. {
95. q = p;
96. ch = p->data.size - request;
97. }
98. }
99. p = p->next;
100. }
102. // q 为预期的目标位置
103. **if** (q == NULL)
104. **return** ERROR;                 //没有找到空闲块
105. **else** **if** (q->data.size == request) // 大小刚刚好
106. {
107. q->data.state = Busy;
108. **return** OK;
109. }
110. **else** /// 可分配内存块相对大
111. {
112. temp->prior = q->prior;
113. temp->next = q;
114. temp->data.address = q->data.address;
115. q->prior->next = temp;
116. q->prior = temp;
117. q->data.address += request;
118. q->data.size = ch;
119. **return** OK;
120. }
121. **return** OK;
122. }
124. Status free(**int** flag) //主存回收
125. {
126. DuLNode \*p = block\_first;
127. **for** (**int** i = 0; i <= flag; i++)
128. **if** (p != NULL)
129. p = p->next;
130. **else**
131. **return** ERROR;
133. p->data.state = Free;
134. **if** (p->prior != block\_first && p->prior->data.state == Free) //与前面的空闲块相连
135. {
136. p->prior->data.size += p->data.size; //空间扩充,合并为一个
137. p->prior->next = p->next;            //去掉原来被合并的p
138. p->next->prior = p->prior;
139. p = p->prior;
140. }
142. **if** (p->next != NULL && p->next != block\_last && p->next->data.state == Free) //与后面的空闲块相连
143. {
144. p->data.size += p->next->data.size; //空间扩充,合并为一个
145. p->next->next->prior = p;
146. p->next = p->next->next;
147. }
149. **if** (p->next == block\_last && p->next->data.state == Free) //与最后的空闲块相连
150. {
151. p->data.size += p->next->data.size;
152. p->next = NULL;
153. }
155. **return** OK;
156. }
158. **void** showNow() //显示主存分配情况
159. {
160. **int** flag = 0;
161. cout << "\n              当前存分配情况:\n";
162. cout << "----------------------------------------------\n\n";
163. DuLNode \*p = block\_first->next;
164. cout << "分区号\t起始地址\t分区大小\t状态\n\n";
165. **while** (p)
166. {
167. cout << "  " << flag++ << "\t";
168. cout << "  " << p->data.address << "\t\t";
169. cout << " " << p->data.size << "KB\t\t";
170. **if** (p->data.state == Free)
171. cout << "空闲\n\n";
172. **else**
173. cout << "已分配\n\n";
174. p = p->next;
175. }
176. cout << "----------------------------------------------\n\n";
177. }
179. **void** main() //主函数
180. {
181. cout << "进入Best Fit\n";
183. **while** (1)
184. {
185. cout << "\n请输内存块范围: ";
186. cin >> MAX\_length;
187. cout << "\n";
188. **if** (MAX\_length <= 0)
189. {
190. cout << "内存块分配不能为0!" << endl;
191. }
192. **else**
193. {
194. **break**;
195. }
196. }
198. Initblock(); //开创空间表
199. **int** choice;  //操作选择标记
201. **while** (1)
202. {
203. cout << "请输入您的操作：";
204. cout << "\n1: 分配内存 2: 回收内存  3: 显示内存 0: 退出\n";
206. cin >> choice;
207. **if** (choice == 1)
208. {
209. showNow();
210. **int** flag\_count = 0;
211. cout << "输入你要分配的内存块数: ";
212. cin >> flag\_count;
213. **for** (**int** i = 1; i <= flag\_count; i++)
214. {
215. Alloc(i); // 分配内存
216. }
217. }
218. **else** **if** (choice == 2) // 内存回收
219. {
220. showNow();
221. **int** flag;
222. cout << "请输入您要释放的分区号：" << endl;
223. cin >> flag;
224. free(flag);
225. }
226. **else** **if** (choice == 0)
227. **break**; //退出
228. **else** **if** (choice == 3)
229. showNow();
230. **else** //输入操作有误
231. {
232. cout << "输入有误，请重试！" << endl;
233. **continue**;
234. }
235. }
236. }

# 五、测试结果

## 5.1 测试样例表格一览

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 实验操作 | 理想结果 | 实验结果 | 是否正确 | 对应图片 |
| 01 | 总内存块大小为-1KB | 报错并退出 | 报错并退出 | 是 | 5-2-01 |
| 02 | 总内存块总大小0KB | 报错并退出 | 报错并退出 | 是 | 5-2-02 |
| 03 | 总内存块总大小1700KB | 分配成功，跳转至操作选择 | 分配成功，跳转至操作选择 | 是 | 5-2-03 |
| 04 | 选择分配内存 | 选择成功，输入块数 | 选择成功，输入块数 | 是 | 5-2-04 |
| 05 | 输入5块内存块 | 操作成功，分配每块内存块 | 操作成功，分配每块内存块 | 是 | 5-2-05 |
| 06 | 100,200,300,500,600 | 操作成功，没有内存剩余 | 操作成功，没有内存剩余 | 是 | 5-2-06 |
| 07 | 继续选择分配1KB内存 | 报错并返回 | 报错并返回 | 是 | 5-2-07 |
| 08 | 选择回收内存 | 操作成功，输入回收的分区号 | 操作成功，输入回收的分区号 | 是 | 5-2-08 |
| 09 | 选择回收2号分区内存 | 回收成功，2号分区显示空闲 | 回收成功，2号分区显示空闲 | 是 | 5-2-09 |
| 10 | 选择分配内存 | 选择成功，输入块数 | 选择成功，输入块数 | 是 | 5-2-10 |
| 11 | 分配1块212KB内存 | 分区2分配，并有88KB剩余 | 分区2分配，并有88KB剩余 | 是 | 5-2-11 |
| 12 | 选择回收4号分区内存 | 成功，3、4分区合并空闲 | 成功，3、4分区合并空闲 | 是 | 5-2-12 |
| 13 | 选择分配内存 | 选择成功，输入块数 | 选择成功，输入块数 | 是 | 5-2-13 |
| 14 | 分配1块417KB内存 | 分区3分配，剩余171KB | 分区3分配，剩余171KB | 是 | 5-2-14 |
| 15 | 选择回收1号分区内存 | 回收成功，1号分区显示空闲 | 回收成功，1号分区显示空闲 | 是 | 5-2-15 |
| 16 | 分配1块112KB内存 | 分区4分配，剩余59KB | 分区4分配，剩余59KB | 是 | 5-2-16 |
| 17 | 选择回收6号分区内存 | 成功，5、6分区合并空闲 | 成功，5、6分区合并空闲 | 是 | 5-2-17 |
| 18 | 分配1块112KB内存 | 分区5分配，剩余233KB | 分区5分配，剩余233KB | 是 | 5-2-18 |
| 19 | 选择显示内存 | 当前内存显示 | 当前内存显示 | 是 | 5-2-19 |
| 20 | 操作数除0~3之外 | 显示错误，跳转下一个操作 | 显示错误，跳转下一个操作 | 是 | 5-2-20 |
| 21 | 操作数0 | 退出 | 退出 | 是 | 5-2-21 |
| 22 | 总内存块总大小100KB | 分配成功，跳转至操作选择 | 分配成功，跳转至操作选择 | 是 | 5-2-22 |
| 23 | 输入分配3块内存 | 操作成功，输入内存大小 | 操作成功，输入内存大小 | 是 | 5-2-23 |
| 24 | 输入第1块50KB | 操作成功，分配完成 | 操作成功，分配完成 | 是 | 5-2-24 |
| 25 | 输入第2块20KB | 操作成功，分配完成 | 操作成功，分配完成 | 是 | 5-2-25 |
| 26 | 输入第3块60KB | 内存不足，分配失败 | 内存不足，分配失败 | 是 | 5-2-26 |
| 27 | 选择回收0号分区内存 | 回收成功，0号分区显示空闲 | 回收成功，0号分区显示空闲 | 是 | 5-2-27 |
| 28 | 再次回收0号分区内存 | 回收成功，0号分区显示空闲 | 回收成功，0号分区显示空闲 | 是 | 5-2-28 |

## 5.2 测试样例对应截图

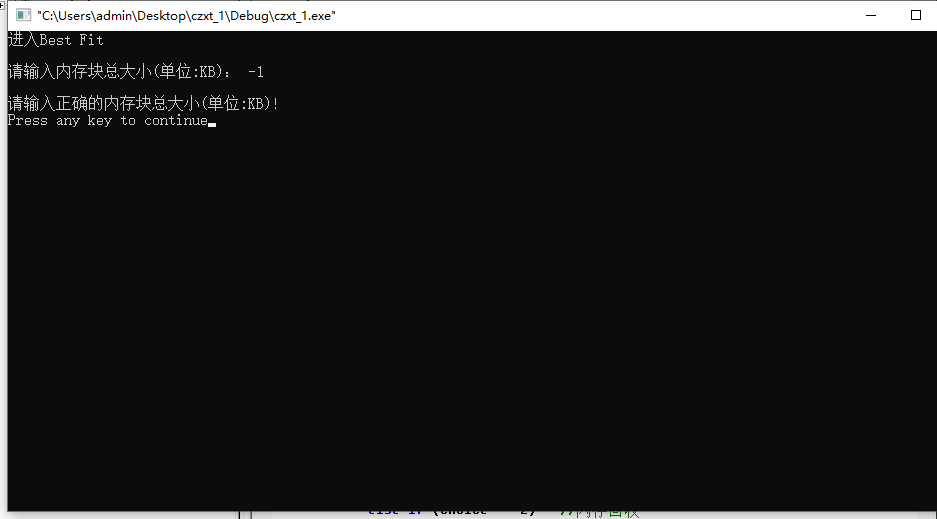


图 5-2-01

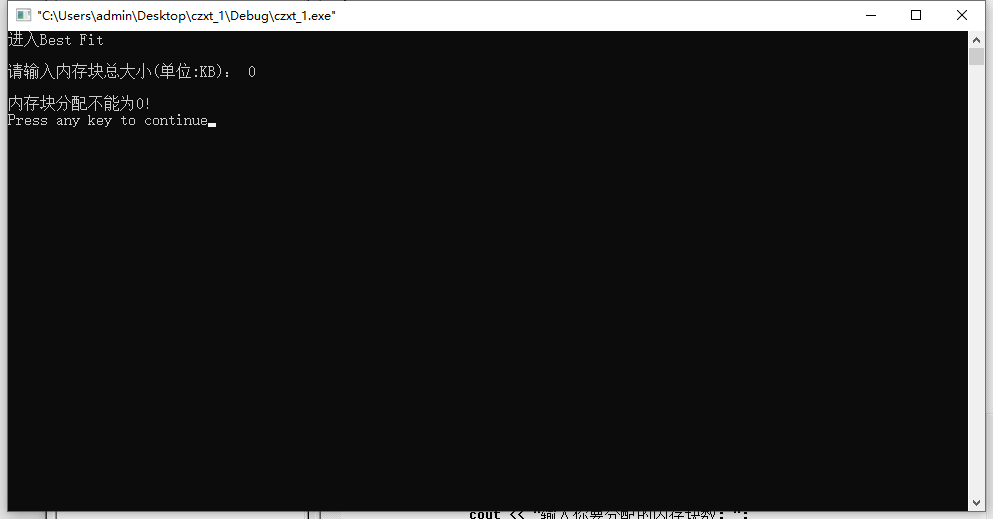


图 5-2-02



图 5-2-03



图 5-2-04



图 5-2-05

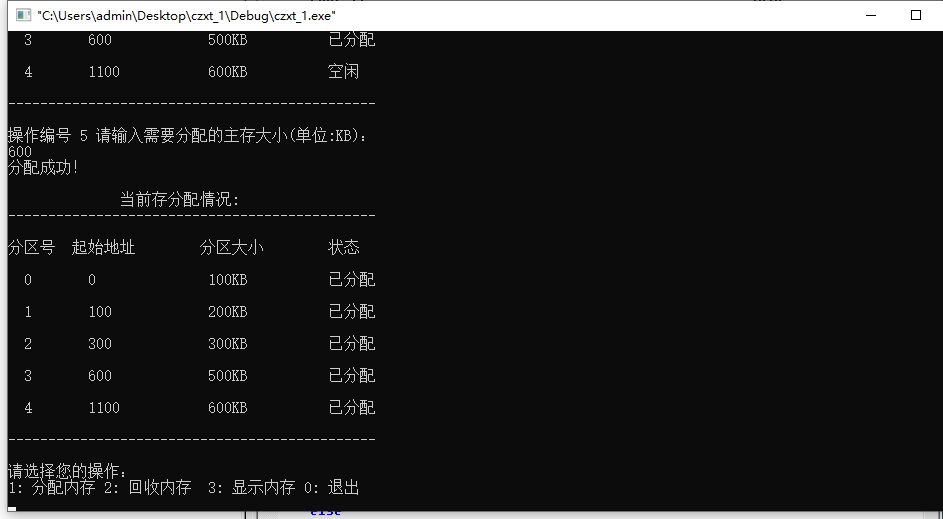


图 5-2-06

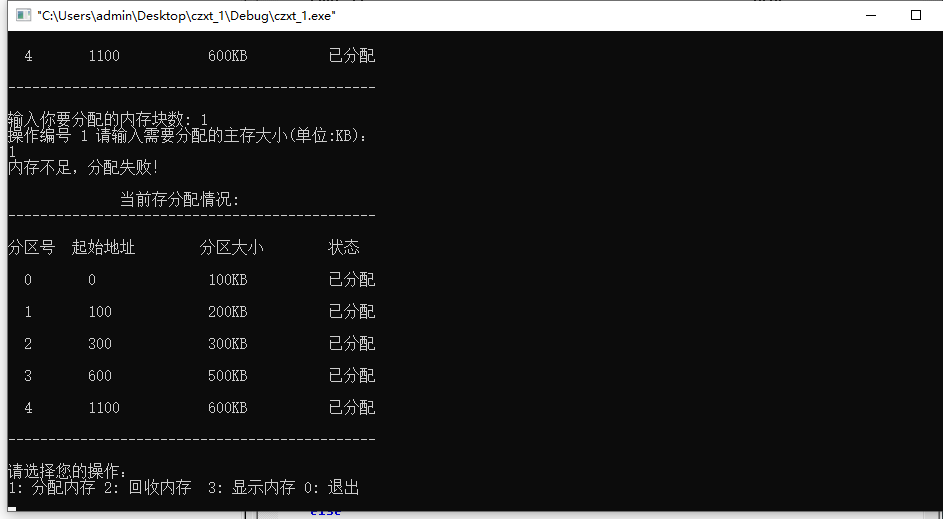


图 5-2-07

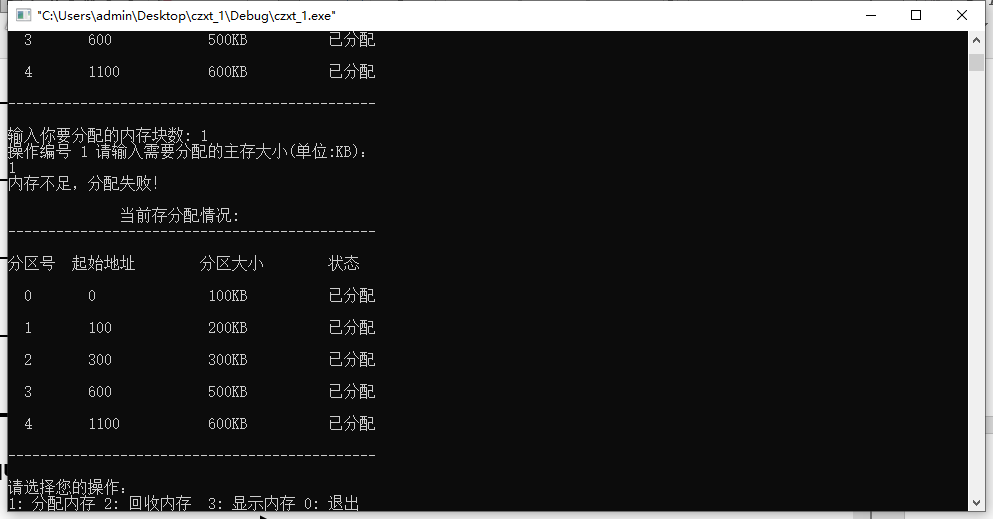


图 5-2-08



图 5-2-09

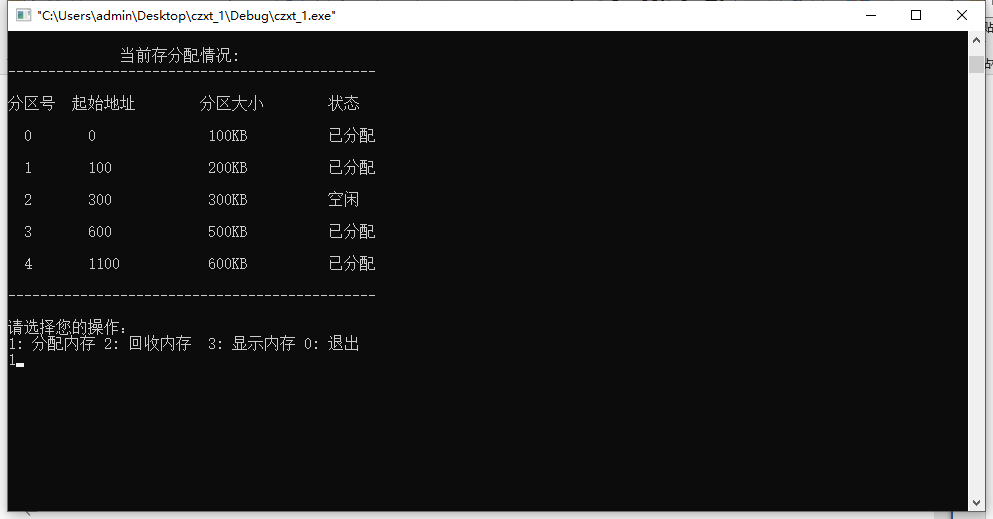


图 5-2-10



图 5-2-11



图 5-2-12



图 5-2-13



图 5-2-14



图 5-2-15



图 5-2-16



图 5-2-17



图 5-2-18



图 5-2-19



图 5-2-20



图 5-2-21

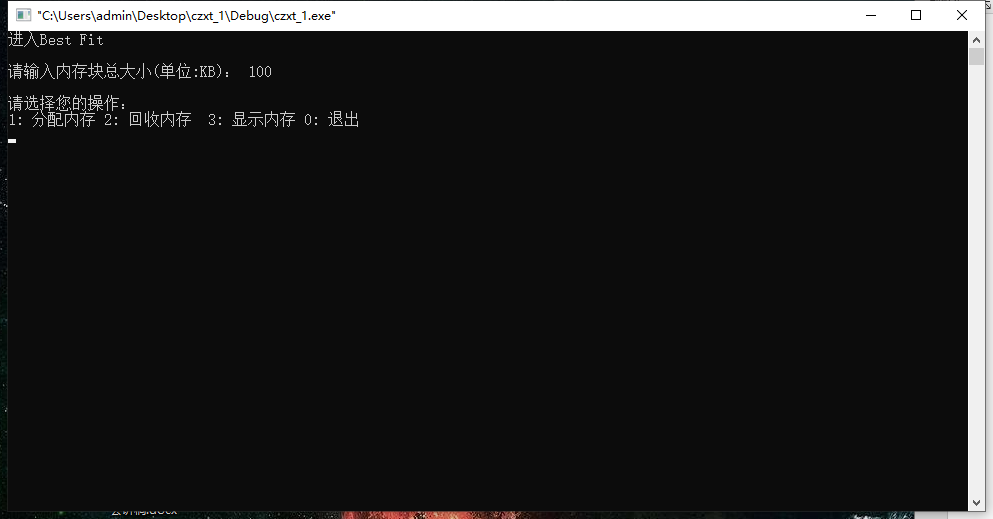


图 5-2-22



图 5-2-23



图 5-2-24



图 5-2-25



图 5-2-26



图 5-2-27



图 5-2-28

# 六、总结

本次的操作系统实验课题我们通过不断地尝试，完成了这次实验课题。感觉还是要多学习算法知识，多实践，不能单纯靠书上理论，书上的理论一般只能告诉你，他是怎么实现的，实现的效果如何，若真的要去模拟，去实现这个最佳适应算法，还是要在网络上查找资料，向老师请教、与同学们交流。在做课题的过程中，我们对最佳适应算法有了深入的学习与了解，也让我们对操作系统这么课程有了新的体悟。

计算机操作系统的最佳适应算法是动态内存分区分配算法的一种。它能够从全部空闲区找出满足作业要求并且最小的空闲分区，这种算法能够让产生的碎片尽量缩小。为了提高寻找速度，这种算法要求将所有的空闲区按其内容以从小到大的顺序形成一个空闲分区链。这样，第一次找到的能满足要求的空闲区，必然是最佳的。最佳适应算法利用的思想就是将地址相邻近的自由区与回收区进行有效地合并，通过初始化空闲区、分配空闲区、回收空闲区实现模拟的内存管理，从而尽量减少碎片的产生，并尽可能的利用内存空间。

用C++语言模拟内存分配管理的最佳适应算法提高了程序设计的灵活性，能有效的解决系统内存分区产生的碎片以及防止了内存分配后未被释放所产生的内存遗留问题，使内存的利用率得到提高，可以提高系统的稳定性以及学生的编程能力。同时最佳适应算法在降低系统中由于内存碎片而带来的系统风险等方面起到了巨大的作用，在一定程度上提高了系统的实时性、可靠性，具有其他算法无法比拟的优越性。[[1]](#footnote-1)

1. 文献参考《使用最佳适应算法对内存实现模拟动态分区管理\_王传俊》 [↑](#footnote-ref-1)