# 实验三：动态分区式存储管理

## 实验目的

1. 深入了解动态分区存储管理的实现原理。
2. 掌握动态分区式存储管理方式的内存分配和回收的实现。

## 实验预备知识

动态分区管理方式

## 实验内容

编写程序模拟完成动态分区存储管理方式的内存分配和回收。实验具体包括：首先确定内存空闲分配表；然后采用最佳适应算法完成内存空间的分配和回收；最后编写主函数对所做工作进行测试。

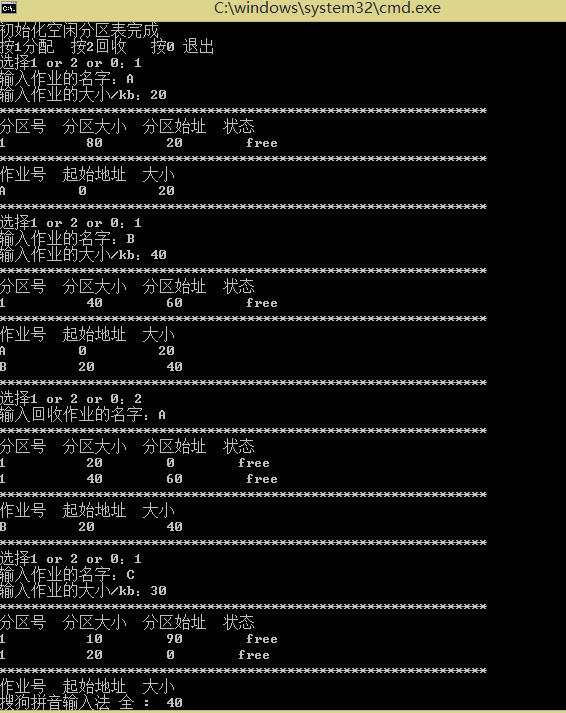
## 实验要求

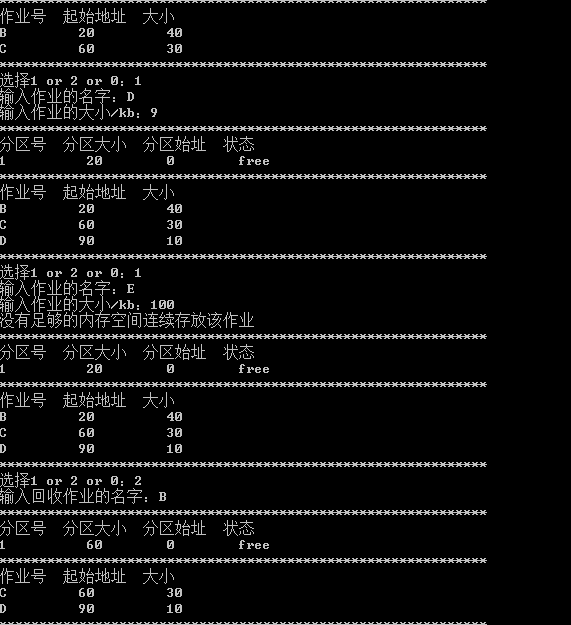
* 1. 实验题目
  2. 给出程序流程和源程序（附有详细注释）

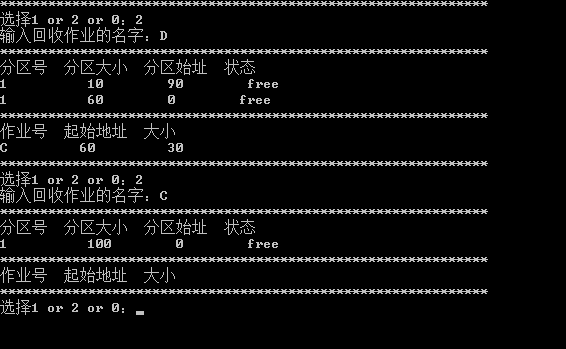
1. /\*
2. 程序的流程在代码中已经体现的很具体了，表的数据结构，初始化空闲分区表结构，以及已占分区表，分配内存，回收内存，表内容的打印等等，每一部分的内容都以模块化的函数封装，结构比较清晰；算法的内容按照书本p128-p130的分配回收内容，以及最佳适应算法来书写；
3. 实验：动态分区式存储管理
4. 语言：c++
5. \*/
6. #include<iostream>
7. #include<string>
8. using namespace std;
9. #define SIZE 1
10. static int freen=0;
11. static int busyn=0;
12. //空闲分区表数据结构
13. struct freeTable{
14. int number;//分区号
15. int size;//大小
16. int startAddress;//起始地址
17. string zhuangtai;//状态
18. };
19. //已占分区表数据结构
20. struct busyTable{
21. string job;//作业名字
22. int size;//大小
23. int startAddress;//起始地址

26. };
27. //初始化空闲分区表
28. void Initialize(freeTable \*free,int memorySize,int osSize){
29. free[freen].number=1;
30. free[freen].size=memorySize-osSize;
31. free[freen].startAddress=0;
32. free[freen].zhuangtai="free";
33. freen++;
34. cout<<"初始化空闲分区表完成"<<endl;
35. }
36. //重置空闲分区表，按从小到大
37. void reset(freeTable\*free){
38. for(int i=0;i<freen-1;i++){
39. for(int j=0;j<freen-1-i;j++){
40. if(free[j].size>free[j+1].size){
41. int n=free[j].number;
42. free[j].number=free[j+1].number;
43. free[j+1].number=n;
44. freeTable temp=free[j];
45. free[j]=free[j+1];
46. free[j+1]=temp;
47. }
48. }
49. }
50. }
51. void delbusy(busyTable\*busy,int n){
52. for(int i=n;i<busyn-1;i++){
53. busy[i].job=busy[i+1].job;
54. busy[i].size=busy[i+1].size;
55. busy[i].startAddress=busy[i+1].startAddress;
56. }
57. busyn--;
58. }
59. void delfree(freeTable\*free,int n){
60. for(int i=n;i<freen-1;i++){
61. free[i].number=free[i+1].number;
62. free[i].size=free[i+1].size;
63. free[i].startAddress=free[i+1].startAddress;
64. free[i].zhuangtai=free[i+1].zhuangtai;
65. }
66. freen--;
67. }
68. //分区分配
69. void distribution(freeTable \*free,busyTable\*busy,string name,int jobSize){
70. for(int i=0;i<freen;i++){
72. if(free[i].size>=jobSize){
73. //已占分区表增加一项
74. busy[busyn].job=name;
75. busy[busyn].startAddress=free[i].startAddress;
76. busy[busyn].size=jobSize;
78. busyn++;
79. //修改空闲分区表
80. if(free[i].size>jobSize){
81. if(free[i].size-jobSize<=SIZE){
82. busy[busyn-1].size=free[i].size;
83. delfree(free,i);
84. return;
85. }
86. else{
87. free[i].size=free[i].size-jobSize;
88. free[i].startAddress=free[i].startAddress+jobSize;
89. reset(free);
90. return;
91. }
92. }
93. else{
94. delfree(free,i);
95. return;
96. }
98. }
99. }
100. cout<<"没有足够的内存空间连续存放该作业"<<endl;

103. }
104. //回收内存
105. void recycling(freeTable\*free,busyTable\*busy,string name){
106. int i;
107. for(i=0;i<busyn;i++){
108. if(busy[i].job==name){
110. break;
111. }
112. }
113. for(int j=0;j<freen;j++){
114. if(free[j].startAddress+free[i].size==busy[i].startAddress){
115. for(int k=0;k<freen;k++){
116. if(free[k].startAddress==busy[i].startAddress+busy[i].size){
117. free[j].size=free[j].size+free[k].size+busy[i].size;
118. delfree(free,k);
119. reset(free);
120. delbusy(busy,i);
121. return;
122. }
123. }
124. free[j].size=free[j].size+busy[i].size;
125. reset(free);
126. delbusy(busy,i);
127. return;
128. }
130. }
131. for(int j=0;j<freen;j++){
132. if(free[j].startAddress==busy[i].startAddress+busy[i].size){
133. for(int k=0;k<freen;k++){
134. if(free[k].startAddress+free[k].size==busy[i].startAddress){
135. free[k].size=free[k].size+free[j].size+busy[i].size;
136. delfree(free,j);
137. reset(free);
138. delbusy(busy,i);
139. return;
140. }
141. }
142. free[j].startAddress=busy[i].startAddress;
143. free[j].size+=busy[i].size;
144. reset(free);
145. delbusy(busy,i);
146. return;
147. }
149. }
150. free[freen].number=freen;
151. free[freen].size=busy[i].size;
152. free[freen].startAddress=busy[i].startAddress;
153. free[freen].zhuangtai="free";
154. freen++;
155. reset(free);
156. delbusy(busy,i);
157. return;
158. }
159. //打印
160. void func(freeTable\*free,busyTable\*busy){
161. cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
162. cout<<"分区号"<<" "<<"分区大小"<<" "<<"分区始址"<<" "<<"状态"<<endl;
163. for(int i=0;i<freen;i++){
164. cout<<free[i].number<<" "<<free[i].size<<" "<<free[i].startAddress<<" "<<free[i].zhuangtai<<endl;
165. }
166. cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
167. cout<<"作业号"<<" "<<"起始地址"<<" "<<"大小"<<endl;
168. for(int j=0;j<busyn;j++){
169. cout<<busy[j].job<<" "<<busy[j].startAddress<<" "<<busy[j].size<<endl;
170. }
171. cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
172. }
173. int main(void){
174. freeTable free[20];
175. busyTable busy[20];
176. Initialize(free,110,10);
177. cout<<"按1分配"<<" "<<"按2回收"<<" "<<" 按0 退出"<<endl;
178. while(true){
179. int n;
180. cout<<"选择1 or 2 or 0：";
181. cin>>n;
182. if(n==0){
183. return 0;
184. }
185. if(n==1){
186. cout<<"输入作业的名字：";
187. string name;
188. cin>>name;
189. cout<<"输入作业的大小/kb：";
190. int size;
191. cin>>size;
192. distribution(free,busy, name,size);
193. func(free,busy);
194. }
195. if(n==2){
196. cout<<"输入回收作业的名字：";
197. string name;
198. cin>>name;
199. recycling(free,busy,name);
200. func(free,busy);
201. }
202. }
203. return 0;
204. }
     1. 程序运行截图







* 1. 收获、体会及对该实验的改进意见和见解

1.更加了解了动态分区存储管理的实现原理。

2.更加掌握了动态分区式存储管理方式的内存分配和回收的实现。

将它们打包，取名为：姓名\_学号\_实验X，其中X为实验号。

## 实验提示

由于是实验，没有真正的内存分配。所以在实验中首先应建立一张空闲区表，初始状态只有一个空闲登记项（假定的内存空闲区）和一张所有状态都为“空”的已分配区表。假定内存空间110KB，OS占用10KB，其余为空闲区。然后可以选择进行内存分配或回收：若是分配，要求输入作业名和所需内存空间大小；若是回收，输入回收作业的作业名。程序循环进行内存分配和回收，直到用户选择退出系统。在每次作业提交（内存分配）及作业结束（内存回收）时显示两张表的内容，以检查内存的分配和回收是否正确。