**操作系统课程设计报告**

**题目 虚拟存储管理**

**院 、 系 计算机与控制工程学院**

**专 业 计算机科学与技术**

**班 级 计201-1**

**姓 名 边明豪**

**学 号 201970504109**

**指导教师 翟一鸣**

2022年 6月 1日**目录**

1 项目描述 1

2 系统结构分析 1

3 系统详细设计 2

4 系统主要算法 3

5程序实现结果 4

6 课程设计总结 6

附录：源程序 6

# 1 项目描述

为了使大的作业(其地址空间超过内存可用空间)或多个作业的地址空间之和超过实际主存空间时，仍能运行，引入了“虚拟存储器”的概念。使作业的一部分地址空间在主存，另一部分在辅存，由操作系统实现多级存储的自动管理，实现主存间的自动覆盖。

模拟请求分页虚拟存储管理技术中的硬件地址变换、缺页中断及页式淘汰算法（5种）处理缺页中断。要求通过本实验，对请求分页管理有一个清楚的概念。

# 2 系统结构分析

内存块数、逻辑地址位数、页面大小、选择的何种算法、工作序列大小以及工作序列指令，都是由用户自己输入。以上由菜单循环控制。

**页表的格式**



说明：

状态位：0代表不在内存中 1代表在内存中

访问位：0代表未访问 1代表访问过

**内存块格式**



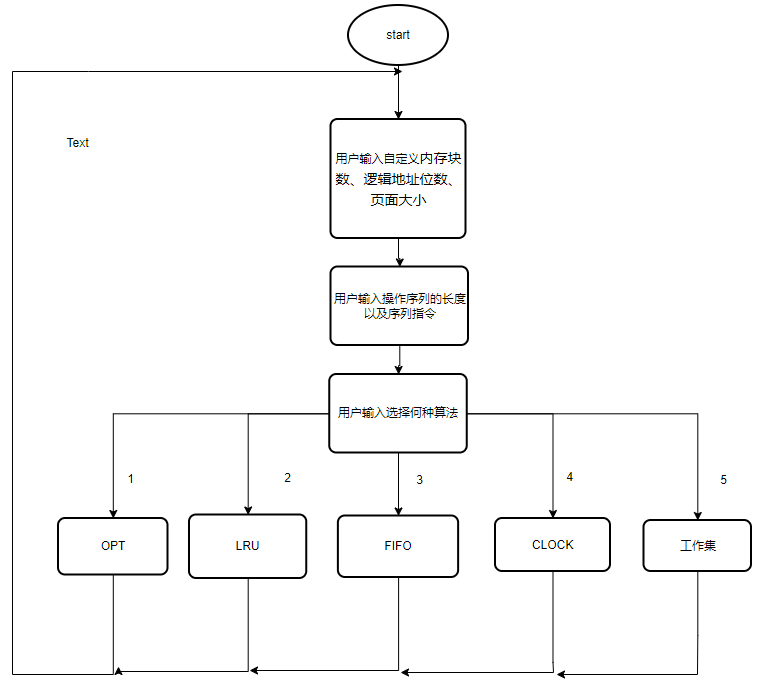
**地址转换**：页号=逻辑地址/页面大小 偏移地址=逻辑地址%页面大小

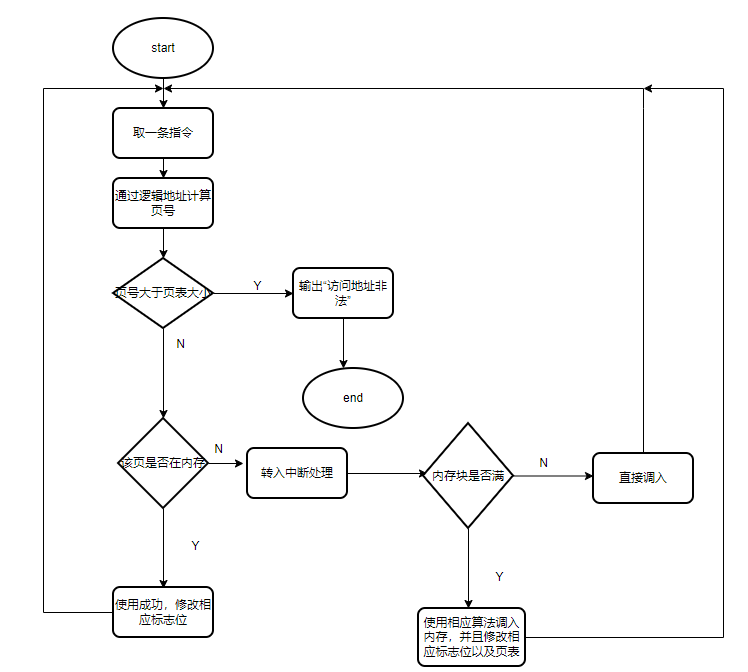
通过页表的页号与块号映射得到块号

物理地址由块号和偏移地址组成

**缺页中断：**当要使用的作业不在内存中时，发生缺页中断，使用相应算法调入。

# 3 系统详细设计





# 4 系统主要算法

1. **逻辑地址**：通过随机数生成偏移地址，通过页号\*页面大小+偏移地址算出逻辑地址。
2. **OPT 二维vector+逆序预处理 页面替换时间复杂度 O(内存块大小) 空间复杂度不变 O（N）**

1创建stl动态数组vector[n] 逆序遍历push\_back()进每个指令的位置（要先push进去一个哨兵n+1，以此来记录最远距离）

2 正序依次遍历工作指令，pop\_back当前指令vector的最后一个元素

3 当发生缺页时，遍历内存块，找出内存块里最远使用的内存块（即为每个内存块vector的最后一个元素 O（1）直接判断）

1. **LRU 维护每个工作指令最后出现的时间 时间复杂度O(内存块大小)**

用last数组记录每个工作指令最后出现的时间。（在内存替换和新放入的时候更新维护）

每次缺页时，只需要遍历内存块，O（1）比较找出最近最久未使用的内存块

1. **FIFO 队列+pair（O（1）取出） 时间复杂度O(1)**

使用stl的pair数据结构，每次将一组pair（工作号，在内存中的地址）放入队列中，取出时可同时找到在内存块中的位置，方便用于输出更新的状态。

并且设置全局变量检验belady现象。

1. **CLOCK 模拟 每次CLOCK不会超过两次 时间复杂度O(N)**

每当插入一个工作时，将其标为1，指针后移。

当指针位置>=块号时 指针置为1 相当于一个循环队列

当发生缺页时，指针一直后移，直到找到某个物理课的标记为为0结束。如果标志位为1则置为0.

1. **工作集算法 双指针（滑动窗口）+set（底层红黑树）去重+set（O（1）查询） 页面替换时间复杂度O（内存块大小）**

双指针l,r 维护当前窗口大小，并且将数据从set中插入删除。

Set自动进行去重，并且可以查询某个元素是否在set中

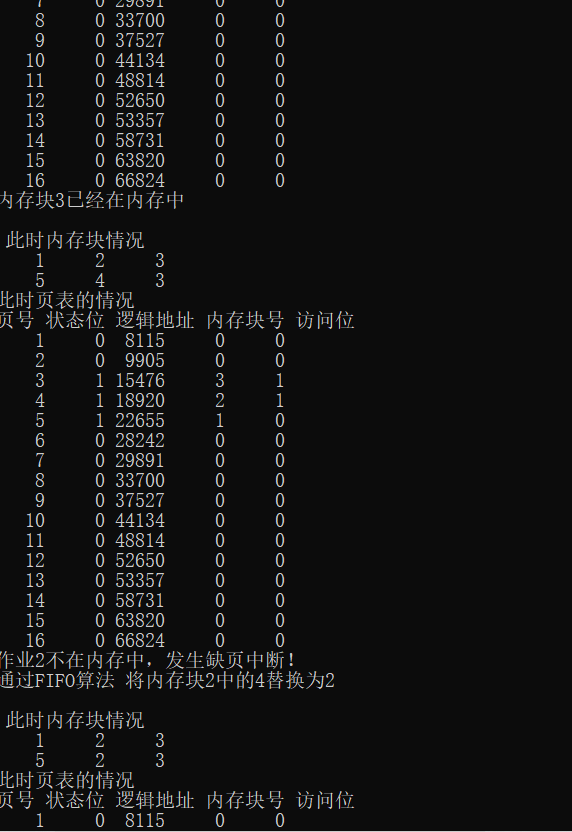
# 5程序实现结果

由于算法过多，这里的程序实现就先展示fifo的belay现象和工作集算法。

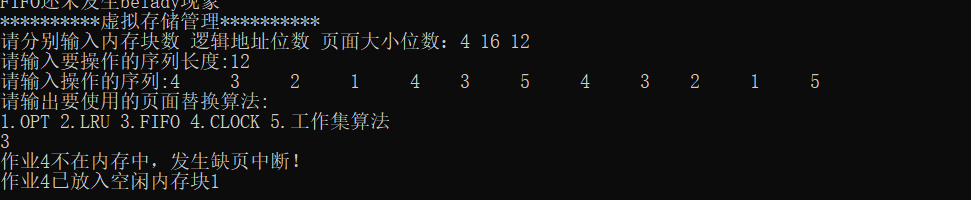
FIFO 输入



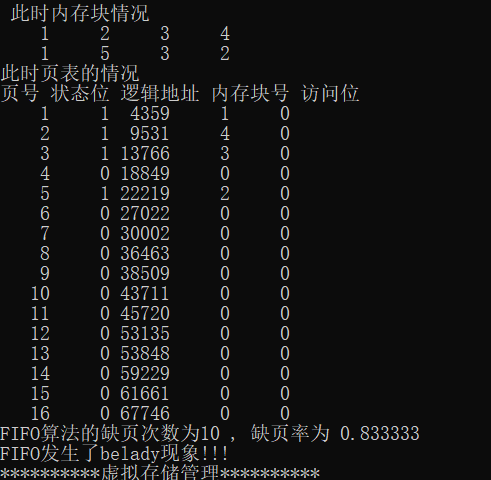
过程中的一些细节



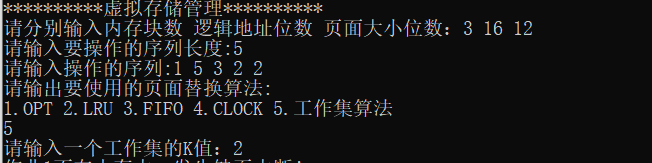
输入第二组FIFO数据 内存块变为4 但缺页率增加了

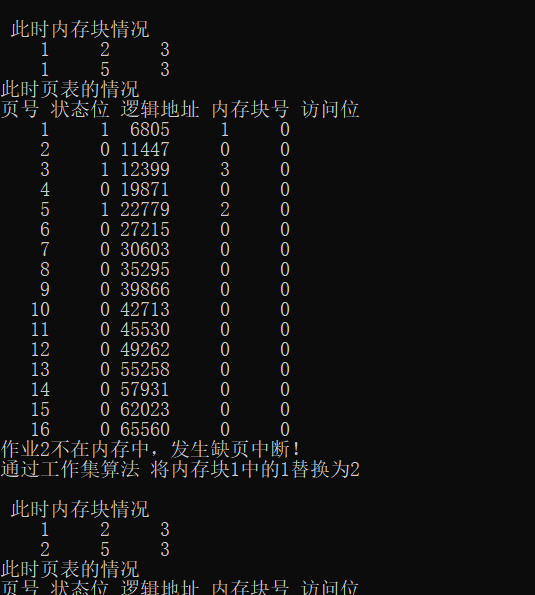


输出出现了belady现象



工作集算法







# 6 课程设计总结

本次课程设计完全自己手写独立完成了虚拟内存的管理，感触也是非常大。首先，理论知识得到了扎实的提升，在开始之前因为对于整个虚拟内存的流程不够熟悉而导致无从下手，很多细节无法下手，所以必须得加深完理论知识才能开始。在课设之前自认为理论扎实，实则理解深度还差一大截。其次，本次课设除了基础数据结构的设计花费时间外，最主要的时间都花在设计五个算法的算法上，这也是最大的乐趣之一，将平常训练acm题目的思维用到了这个地方。本次在设计第一个OPT的时候，想最优算法就卡住了，先用模拟实现了，更优的算法在两天后想出来了，用stl 的Vector 在空间不增加的情况下，将时间从O（n2）降了一个维度到O(内存块大小),这个思路也是很妙。在后面的LRU算法中，用到了stl 的pair数据结构优化了时间，不用再循环一次查找索引下标了，直接可以O（1）查出来。除了课本的三个算法，我又加了两个算法，CLOCK算法和工作集算法，因为我看这个以后考研等之中也是要考的。工作集算法我用了双指针+set去重+O(1)查询实现的。本次课设很有成就感，再次体会到了算法的灵魂。

# 附录：源程序

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 版权所有 (C)2022，BianMingHao

\*

\* 文件名称： main.cpp

\* 文件标识：无

\* 内容摘要：虚拟存储管理的主菜单

\* 其它说明：无

\* 当前版本： V1.0

\* 作 者：边明豪

\* 完成日期： 2022.5.31

\* 修改内容：创建

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. #include "algorithm.cpp"
2. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
3. \* 功能描述：菜单主页面控制函数
4. \* 输入参数： 选择器
5. \* 输出参数： 输出相应功能
6. \* 返回值：  无
7. \* 其它说明：消息字段之间用分号(;)分隔
8. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
9. int main() {
10. srand(0);
11. while(1) {
12. printf("\*\*\*\*\*\*\*\*\*\*虚拟存储管理\*\*\*\*\*\*\*\*\*\*\n");
13. printf("请分别输入内存块数 逻辑地址位数 页面大小位数：");
14. scanf("%d %d %d",&MemoryBlock,&LogicalAddress,&PageSize);
15. PageNumberSize = LogicalAddress - PageSize ;
16. PageNumberSize = (1<<PageNumberSize);
17. printf("请输入要操作的序列长度:");
18. scanf("%d",&n);
19. printf("请输入操作的序列:");
20. init();
21. for(int i = 1 ; i<=n  ; i++) {
22. scanf("%d",&work[i]);
23. }
24. printf("请输出要使用的页面替换算法:\n");
25. printf("1.OPT 2.LRU 3.FIFO 4.CLOCK 5.工作集算法\n");
26. int op;
27. cin>>op;
28. switch (op) {
29. case 1:
30. OPT();
31. break;
32. case 2 :
33. LRU();
34. break;
35. case 3:
36. FIFO();
37. break;
38. case 4:
39. CLOCK();
40. break;
41. case 5:
42. printf("请输入一个工作集的K值：");
43. scanf("%d",&k);
44. WORK();
45. }
46. }
47. return 0;
48. }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 版权所有 (C)2022，BianMingHao

\*

\* 文件名称： algorithm.cpp

\* 文件标识：无

\* 内容摘要：地址转换、缺页中断、页面替换算法

\* 其它说明：无

\* 当前版本： V1.0

\* 作 者：边明豪

\* 完成日期： 2022.5.31

\* 修改内容：创建

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. #include<bits/stdc++.h>
2. using namespace std;
3. //#include "algorithm.h";
4. int MemoryBlock,LogicalAddress,PageSize,PageNumberSize;
5. int n,i*\_cnt=0 , o\_*cnt=0;
6. priority*\_queue<int , vector<int> , greater<int> > belady\_*q;
7. queue<pair<int,int>> q;
8. set<int> work\_set;
9. struct  table {
10. int PageNumber;
11. bool stand;
12. int MemoryNumber;
13. int ExternalMemoryAddress;
14. bool vis;
15. }\*PageTable;
16. int *\*InnerMemory , \**ExternalMemory,\*work;
17. int b\_cnt=0,k=0;
18. struct node
19. {
20. int id;
21. double lv;
22. }belady[100005];
23. void OPT();
24. void LRU();
25. void FIFO();
26. void CLOCK();
27. void WORK();
28. void init();
29. int AddressTranslation(int x);
30. void showmemory();
31. void showtable();
32. /**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*
33. \* 功能描述：OPT算法的实现 二维vector 空间O(N)+时间O(内存大小);
34. \* 输入参数： 无
35. \* 输出参数： 输出OPT算法相关信息
36. \* 返回值： 0-成功   其他-失败
37. \* 其它说明：消息字段之间用分号(;)分隔
38. **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/
39. void OPT() {
40. int \*last;
41. last = new int(n);
42. int ci=0;
43. double lv=0;
44. //初始化各个值最先出现的时间为最大
45. vector<int> v[(1<<PageSize)];
46. for(int i = 1 ;i <= (1<<PageSize) ; i++)
47. {
48. v[i].push\_back(n+1);
49. }
50. for(int i = n ; i>=1 ; i--)
51. {
52. v[work[i]].push\_back(i);
53. }
54. for(int i =1 ; i<=n ; i++) {
55. v[work[i]].pop\_back();
56. if(PageTable[work[i]].PageNumber==0)
57. {
58. printf("错误：页号%d超过页表大小\n",work[i]);
59. return ;
60. }
62. if(PageTable[work[i]].stand==0) {
63. printf("作业%d不在内存中，发生缺页中断！\n",work[i]);
64. ci++;
65. if(i\_cnt<MemoryBlock) {
66. InnerMemory[++i\_cnt]=work[i];
67. printf("作业%d已放入空闲内存块%d\n",work[i],i\_cnt);
68. PageTable[work[i]].stand=1;
69. PageTable[work[i]].MemoryNumber=i\_cnt;
70. } else {
71. int maxx=0,maxxloc=0;
72. for(int j = 1; j<=MemoryBlock ; j++) {
73. //     for(int k  = i+1  ; k<=n ; k++) {
74. //      if(work[i]==work[k]) {
75. //       last[work[i]]=k;
76. //       continue;
77. //      }
78. //     }
79. //     if(last[work[j]]>maxx) {
80. //      maxx=last[work[j]];
81. //      maxxloc=j;
82. //     }
83. int size=v[work[InnerMemory[j]]].size();
84. //printf("maxx %d",v[work[InnerMemory[j]]][size-1]);
85. if(v[work[InnerMemory[j]]][size-1]>maxx)
86. {
87. maxx=v[work[InnerMemory[j]]][size-1];
88. maxxloc=j;
89. //printf("maxxloc%d\n",maxxloc);
90. }
91. }
92. printf("通过opt算法 将内存块%d中的%d替换为%d\n",maxxloc,InnerMemory[maxxloc],work[i]);

95. //cout<< PageTable[work[i]].MemoryNumber<<" PageTable[work[i]].MemoryNumber";
97. PageTable[InnerMemory[maxxloc]].stand=0;
98. PageTable[InnerMemory[maxxloc]].vis=0;
99. PageTable[InnerMemory[maxxloc]].MemoryNumber=0;
100. PageTable[work[i]].vis=0;
101. PageTable[work[i]].stand=1;
102. PageTable[work[i]].MemoryNumber=maxxloc;
103. InnerMemory[maxxloc]=work[i];
104. //cout<< PageTable[InnerMemory[maxxloc]].MemoryNumber<<" PageTable[InnerMemory[maxxloc]].MemoryNumber";
105. }
106. } else {
107. printf("内存块%d已经在内存中\n",work[i]);
108. PageTable[work[i]].vis=1;
109. }
110. showmemory();
111. showtable();
113. }
114. lv=1.0\*ci/n;
115. printf("opt算法的缺页次数为%d , 缺页率为 %lf\n",ci,lv);
116. };
117. /**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*
118. \* 功能描述：LRU算法的实现 普通模拟 时间复杂度O(内存块大小)
119. \* 输入参数： 无
120. \* 输出参数： 输出LRU算法相关信息
121. \* 返回值： 0-成功   其他-失败
122. \* 其它说明：消息字段之间用分号(;)分隔
123. **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/
124. void LRU() {
125. int \*last;
126. int ci=0;
127. double lv=0;
128. last = new int(n);
130. for(int i =1 ; i<=n ; i++) {
131. if(PageTable[work[i]].PageNumber==0)
132. {
133. printf("错误：页号%d超过页表大小\n",work[i]);
134. return ;
135. }
136. if(PageTable[work[i]].stand==0) {
137. printf("作业%d不在内存中，发生缺页中断！\n",work[i]);
138. ci++;
139. if(i\_cnt<MemoryBlock) {
140. InnerMemory[++i\_cnt]=work[i];
141. printf("作业%d已放入空闲内存块%d\n",work[i],i\_cnt);
142. PageTable[work[i]].stand=1;
143. PageTable[work[i]].MemoryNumber=i\_cnt;
144. last[work[i]]=i;
145. } else {
146. int minn=INT\_MIN,minnloc=0;
147. for(int j = 1; j<=MemoryBlock ; j++) {
148. if(last[work[j]]<minn) {
149. minn=last[work[j]];
150. minnloc=j;
151. }
152. }
153. printf("通过LRU算法 将内存块%d中的%d替换为%d\n",minnloc,InnerMemory[minnloc],work[i]);
154. PageTable[InnerMemory[minnloc]].stand=0;
155. PageTable[InnerMemory[minnloc]].vis=0;
156. PageTable[InnerMemory[minnloc]].MemoryNumber=0;
157. InnerMemory[minnloc]=work[i];
158. PageTable[work[i]].stand=1;
159. PageTable[work[i]].MemoryNumber=minnloc;
160. PageTable[work[i]].vis=0;
161. last[work[i]]=i;
162. }
163. } else {
164. printf("内存块%d已经在内存中\n",work[i]);
165. PageTable[work[i]].vis=1;
166. last[work[i]]=i;
167. }
168. showmemory();
169. showtable();
170. }
171. lv=1.0\*ci/n;
172. printf("LRU算法的缺页次数为%d , 缺页率为 %lf\n",ci,lv);
173. };
174. /**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*
175. \* 功能描述：FIFO算法的实现  队列+pair直接取出 时间复杂度O(1)
176. \* 输入参数： 无
177. \* 输出参数： 输出FIFO算法相关信息
178. \* 返回值： 0-成功   其他-失败
179. \* 其它说明：消息字段之间用分号(;)分隔
180. **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/
181. void FIFO() {
182. int \*last;
183. last = new int(n);
184. //初始化各个值最先出现的时间为最大
185. int ci=0;
186. double lv=0.0;
187. for(int i =1 ; i<=n ; i++) {
188. if(PageTable[work[i]].PageNumber==0)
189. {
190. printf("错误：页号%d超过页表大小\n",work[i]);
191. return ;
192. }
193. if(PageTable[work[i]].stand==0) {
194. printf("作业%d不在内存中，发生缺页中断！\n",work[i]);
195. ci++;
196. if(i\_cnt<MemoryBlock) {
197. InnerMemory[++i\_cnt]=work[i];
198. printf("作业%d已放入空闲内存块%d\n",work[i],i\_cnt);
199. PageTable[work[i]].stand=1;
200. PageTable[work[i]].MemoryNumber=i\_cnt;
201. q.push({work[i],i\_cnt});
202. } else {
204. int nowwork=q.front().first;
205. int nowloc=q.front().second;
206. q.pop();
207. printf("通过FIFO算法 将内存块%d中的%d替换为%d\n",nowloc,InnerMemory[nowloc],work[i]);
208. PageTable[InnerMemory[nowloc]].stand=0;
209. PageTable[InnerMemory[nowloc]].vis=0;
210. PageTable[InnerMemory[nowloc]].MemoryNumber=0;
211. InnerMemory[nowloc]=work[i];
212. q.push({work[i],nowloc});
213. PageTable[work[i]].stand=1;
214. PageTable[work[i]].MemoryNumber=nowloc;
215. PageTable[work[i]].vis=0;
216. }
217. } else {
218. printf("内存块%d已经在内存中\n",work[i]);
219. PageTable[work[i]].vis=1;
221. }
222. showmemory();
223. showtable();
224. }
225. lv=1.0\*ci/n;
226. printf("FIFO算法的缺页次数为%d , 缺页率为 %lf\n",ci,lv);
227. belady[++b\_cnt].lv=lv;
228. belady[b\_cnt].id=MemoryBlock;
229. int f=0;
230. for(int i = 1 ; i<b\_cnt ;i++)
231. {
232. if(lv>belady[i].lv&&MemoryBlock>belady[i].id)
233. {
234. f=1;
235. printf("FIFO发生了belady现象!!!\n");
236. break;
237. }
238. }
239. if(f==0)
240. printf("FIFO还未发生belady现象\n");
241. };
242. /**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*
243. \* 功能描述：CLOCK算法的实现
244. \* 输入参数： 无
245. \* 输出参数： 输出CLOCK算法相关信息
246. \* 返回值： 0-成功   其他-失败
247. \* 其它说明：消息字段之间用分号(;)分隔
248. **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/
249. void CLOCK() {
250. int \*last;
251. int ci=0;
252. double lv=0;
253. int \*vis;
254. vis = new int(n);
255. int zhi=1,loc;
256. for(int i =1 ; i<=n ; i++) {
257. if(PageTable[work[i]].PageNumber==0)
258. {
259. printf("错误：页号%d超过页表大小\n",work[i]);
260. return ;
261. }
262. if(PageTable[work[i]].stand==0) {
263. printf("作业%d不在内存中，发生缺页中断！\n",work[i]);
264. ci++;
265. if(i\_cnt<MemoryBlock) {
266. InnerMemory[++i\_cnt]=work[i];
267. printf("作业%d已放入空闲内存块%d\n",work[i],i\_cnt);
268. PageTable[work[i]].stand=1;
269. PageTable[work[i]].MemoryNumber=i\_cnt;
271. vis[InnerMemory[zhi]]=1;
272. zhi++;
273. if(zhi>=MemoryBlock+1) zhi=1;
274. } else {
276. while(1)
277. {
278. if(vis[InnerMemory[zhi]]==1)
279. {
280. vis[InnerMemory[zhi]]=0;
281. }
282. else
283. {
284. vis[InnerMemory[zhi]]=1;
285. loc=zhi;
286. break;
287. }
288. zhi++;
289. if(zhi>=MemoryBlock+1) zhi=1;
290. }
291. printf("通过CLOCK算法 将内存块%d中的%d替换为%d\n",loc,InnerMemory[loc],work[i]);
292. PageTable[InnerMemory[loc]].stand=0;
293. PageTable[InnerMemory[loc]].vis=0;
294. PageTable[InnerMemory[loc]].MemoryNumber=0;
295. InnerMemory[loc]=work[i];
296. PageTable[work[i]].stand=1;
297. PageTable[work[i]].MemoryNumber=loc;
298. PageTable[work[i]].vis=0;
299. }
300. } else {
301. printf("内存块%d已经在内存中\n",work[i]);
302. PageTable[work[i]].vis=1;
303. }
304. showmemory();
305. showtable();
306. }
307. lv=1.0\*ci/n;
308. printf("CLOCK算法的缺页次数为%d , 缺页率为 %lf\n",ci,lv);
309. };
310. /**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*
311. \* 功能描述：工作集算法的实现 双指针+set set O(1)查询  时间复杂度O(内存块大小)
312. \* 输入参数： 无
313. \* 输出参数： 输出工作集算法相关信息
314. \* 返回值： 0-成功   其他-失败
315. \* 其它说明：消息字段之间用分号(;)分隔
316. **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/
317. void WORK()
318. {
319. int \*last;
320. int ci=0;
321. double lv=0;
322. last = new int(n);
323. int l = 0,r=0,loc;
325. set<int>::iterator it;
326. for(int i =1 ; i<=n ; i++) {
327. if(PageTable[work[i]].PageNumber==0)
328. {
329. printf("错误：页号%d超过页表大小\n",work[i]);
330. return ;
331. }
332. if(PageTable[work[i]].stand==0) {
333. printf("作业%d不在内存中，发生缺页中断！\n",work[i]);
334. ci++;
335. if(i\_cnt<MemoryBlock) {
336. InnerMemory[++i\_cnt]=work[i];
337. printf("作业%d已放入空闲内存块%d\n",work[i],i\_cnt);
338. PageTable[work[i]].stand=1;
339. PageTable[work[i]].MemoryNumber=i\_cnt;
340. if(r+1<=n)r++;
341. if(r-l>k)
342. {
343. work\_set.erase(work[l]);
344. l++;
345. }
346. work\_set.insert(work[r]);
347. } else {
348. if(r+1<=n)r++;
349. if(r-l>k)
350. {
351. work\_set.erase(work[l]);
352. l++;
353. }
354. for(int j = 1 ; j <= MemoryBlock ; j++)
355. {
356. if((it=work\_set.find(InnerMemory[j]))==work\_set.end())
357. {
358. loc=j;
359. break;
360. }
361. }
362. printf("通过工作集算法 将内存块%d中的%d替换为%d\n",loc,InnerMemory[loc],work[i]);
363. PageTable[InnerMemory[loc]].stand=0;
364. PageTable[InnerMemory[loc]].vis=0;
365. PageTable[InnerMemory[loc]].MemoryNumber=0;
366. InnerMemory[loc]=work[i];
367. PageTable[work[i]].stand=1;
368. PageTable[work[i]].MemoryNumber=loc;
369. PageTable[work[i]].vis=0;
370. work\_set.insert(work[i]);
371. }
372. } else {
373. if(r+1<=n)r++;
374. if(r-l>k)
375. {
376. work\_set.erase(work[l]);
377. l++;
378. }
379. printf("内存块%d已经在内存中\n",work[i]);
380. PageTable[work[i]].vis=1;
381. }
382. showmemory();
383. showtable();
384. }
385. lv=1.0\*ci/n;
386. printf("工作集算法的缺页次数为%d , 缺页率为 %lf\n",ci,lv);
387. };
388. /**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*
389. \* 功能描述：页表 内存 外存等参数初始化
390. \* 输入参数： 无
391. \* 输出参数： 无
392. \* 返回值： 无
393. \* 其它说明：消息字段之间用分号(;)分隔
394. **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/
395. void init() {
396. srand((unsigned)time(NULL));
397. PageTable = new table[PageNumberSize+5]();
398. InnerMemory = new int[MemoryBlock+5]();
399. ExternalMemory = new int[PageNumberSize+5]();
400. work = new int[PageNumberSize+5]();
401. i\_cnt=0;
402. for(int i =1 ; i<=PageNumberSize ; i++) {
403. int OffsetAddress = rand()%(1<<PageSize)+1;
404. PageTable[i].PageNumber=i;
405. PageTable[i].stand=0;
406. PageTable[i].ExternalMemoryAddress=i\*(1<<PageSize)+OffsetAddress;
407. PageTable[i].MemoryNumber=0;
408. PageTable[i].vis=0;
409. }
410. while(!q.empty())
411. {
412. q.pop();
413. }
414. work\_set.clear();
415. }
416. /**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*
417. \* 功能描述：地址转换 找到页号和偏移地址
418. \* 输入参数： 逻辑地址x
419. \* 输出参数： 页号
420. \* 返回值： 无
421. \* 其它说明：消息字段之间用分号(;)分隔
422. **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/\*\*
423. \* 功能描述：用于展示页表的使用情况
424. \* 输入参数： 无
425. \* 输出参数： 输出页表当前状态
426. \* 返回值： 无
427. \* 其它说明：消息字段之间用分号(;)分隔
428. **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/
429. void showtable() {
430. printf("此时页表的情况\n");
431. printf("页号 状态位 逻辑地址 内存块号 访问位\n");
432. for(int j = 1 ; j <=PageNumberSize ; j++) {
433. printf("%5d %5d %5d %5d %5d",j,PageTable[j].stand,PageTable[j].ExternalMemoryAddress,PageTable[j].MemoryNumber,PageTable[j].vis);
434. printf("\n");
435. }
436. }
437. /**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*
438. \* 功能描述：用于展示内存使用情况
439. \* 输入参数： 无
440. \* 输出参数：输出内存情况
441. \* 返回值：无
442. \* 其它说明：消息字段之间用分号(;)分隔
443. **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/
444. void showmemory() {
445. printf("\n 此时内存块情况\n");
446. printf("内存块号 对应的工作号 物理地址\n");
447. for(int j = 1 ; j<=MemoryBlock ; j++) {
448. printf("%5d %5d %10d\n",j,InnerMemory[j],AddressTranslation(PageTable[InnerMemory[j]].ExternalMemoryAddress));
449. }
450. printf("\n");
451. // for(int j = 1 ; j<=MemoryBlock ; j++) {
452. // printf("%5d ",InnerMemory[j]);
453. // }
454. // printf("\n");
455. }/**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*
456. \* 功能描述：用于释放内存
457. \* 输入参数： 无
458. \* 输出参数：无
459. \* 返回值：无
460. \* 其它说明：消息字段之间用分号(;)分隔
461. **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/
462. void free()
463. {
464. delete[] PageTable;
465. PageTable=nullptr;
466. delete[] InnerMemory;
467. InnerMemory=nullptr;
468. delete[] ExternalMemory;
469. ExternalMemory = nullptr;
470. delete[] work ;
471. work = nullptr;
472. }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 版权所有 (C)2022，BianMingHao

\*

\* 文件名称： algorithm.h

\* 文件标识：无

\* 内容摘要：项目的头文件

\* 其它说明：无

\* 当前版本： V1.0

\* 作 者：边明豪

\* 完成日期： 2022.5.31

\* 修改内容：创建

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. #pragma once
2. #ifndef ALGORITHM\_H
3. #define ALGORITHM\_H
4. #include<bits/stdc++.h>
5. using namespace std;
6. extern int MemoryBlock,LogicalAddress,PageSize,PageNumberSize;
7. extern int n,i\_cnt=0 , o\_cnt=0;
8. extern priority\_queue<int , vector<int> , greater<int> > belady\_q;
9. extern queue<pair<int,int>> q;
10. extern set<int> work\_set;
11. struct  table {
12. int PageNumber;
13. bool stand;
14. int MemoryNumber;
15. int ExternalMemoryAddress;
16. bool vis;
17. } ;
18. extern table \*PageTable;
19. extern int \*InnerMemory , \*ExternalMemory,\*work;
20. extern int b\_cnt=0,k=0;
21. struct node
22. {
23. int id;
24. double lv;
25. };
26. extern node belady[100005];
27. extern void OPT();
28. extern void LRU();
29. extern void FIFO();
30. extern void CLOCK();
31. extern void WORK();
32. extern void init();
33. extern int AddressTranslation(int x);
34. extern void showmemory();
35. extern void showtable();
36. #endif

**操作系统课程设计成绩评价表**

注：请在验收答辩前，对除“答辩”以外的其他项目完成自评。

| **课程目标** | **评价依据** | **评价指标及标准** | **满分** | **学生自评** | **教师确认** |
| --- | --- | --- | --- | --- | --- |
| 课程目标1：针对相对复杂的实际的应用问题，能综合运用操作系统、数据结构、程序设计等多种知识，在对问题进行分析、建模基础上，选择建构合适的数据结构，设计较优的调度与管理算法，并通过编程与调试，完成仿真系统的构建。 | 源程序和课程设计报告 | 源程序：程序排版规范，一句一行，缩格排放；程序中的注释适当，每一个函数前，通过注释给出了关于功能、接口的说明。 | 5 | 4 |  |
| 结构：程序模块功能划分合理，接口清晰、高效。 | 5 | 5 |  |
| 功能：完整、正确实现系统管理功能，功能完善。 | 5 | 5 |  |
| 界面：采用命令行或菜单操作或图形用户界面，操作流程清晰便利。 | 5 | 4 |  |
| 创新：实现系统有什么功能特色 | 5 | 4 |  |
| 报告：按模板要求写作，（手写）字迹清楚，书写工整，或（打印）版面在边距、行距等方面处理恰当，整体可读性好。 | 5 | 5 |  |
| 验收答辩：操作熟练，程序运行正确，能正确回答相关问题。对程序出现的问题能做出相应处理和改进。 | 10 | / |  |
| 课程目标2：理解设计技术参数的要求及实现方式，能识别系统中对处理时间、时间复杂度和存储空间的要求，将之用于系统模型、存储结构和管理算法的设计中，并能分析算法的性能和效率。 | 课程设计报告“3 项目的主要算法”和“6 项目的实现结果”部分 | 用伪代码或流程图等形式规范、清晰地描述了主要算法 | 5 | 5 |  |
| 正确分析和讨论主要算法的性能和效率。 | 5 | 5 |  |
| 答辩：正确回答相关问题。连续两个问题回答错误此项不得分。 | 10 | / |  |
| 课程目标3：选择恰当的操作系统平台和程序开发环境，用于对进程管理、存储管理和文件管理中复杂的管理问题进行模拟仿真和开发 | 源程序与项目验收 | 系统平台：是否采用Linux平台，选用Windows平台只能得3分。 | 5 | 5 |  |
| 程序组织：程序采用多文件组织，采用单文件方案此项不得分。 | 5 | 4 |  |
| 答辩：正确回答相关问题。连续两个问题回答错误此项不得分。 | 10 | / |  |
| 课程目标4：能理解诚信守则是基本的道德底线，在课程设计环节中态度认真，不抄袭代码，独立或小组合作完成，自觉遵守纪律和规范。 | 源程序与项目验收 | 代码是否涉嫌抄袭，重复率超过50%此项不得分。 | 5 | 5 |  |
| 答辩：正确回答相关问题。连续两个问题回答错误此项不得分。 | 5 |  |  |
| 课程目标5：能认识到课程设计是一个小型软件项目，可以预测设计开发期间所遇到的困难和风险，能协调自己（或团队）开发时间与成本关系，能对课程设计的成果进行评价和分析。 | 课程设计报告“1 项目描述”和“2系统结构分析”部分 | 能否体现出对于整个设计开发流程各个环节的思考和预测，以及开发程序期间是否与其他同学团队合作。 | 10 | 8 |  |
| 合 计 | | | 100 |  |  |