

**课程设计(综合实验)报告**

( 2019 – 2020 年度第 2 学期)

名 称： 操作系统综合实验

题 目： 主存储器分配回收

院 系： 计算机系

班 级： 软件1802

学 号： 201809000121

学生姓名： 王毅涛

指导教师： 王德文

设计周数： 分散进行

成 绩：

日期： 年 月 日

**一、 综合实验的目的与要求**

一个好的计算机系统不仅要有一个足够容量的、存取速度高的、稳定可靠的主存储器， 而且要能合理地分配和使用这些存储空间。当用户提出申请存储器空间时，存储管理必须根 据申请者的要求，按一定的策略分析主存空间的使用情况，找出足够的空闲区域分配给申请 者。当作业撤离或主动归还主存资源时，则存储管理要收回作业占用的主存空间或归还部分 主存空间。主存的分配和回收的实现与主存储器的管理方式有关的，通过本实验帮助学生理 解在不同的存储管理方式下应怎样实现主存空间的分配和回收。

**二、实验正文**

实验分为两个小题，第一个是研究可变分区管理时，不同大小的作业装入主存时，主存如何分配空间，释放空间，第二个是模拟页式虚拟存储器的页面置换算法，随机产生若干条指令，为它们分配页号和存储块。

第一个小题，设定Space类，表示空闲块，设定Process类，表示需要服务或者已经服务完成的进程，一个Allocator类用来组织安排上面的两样东西，最后一个Client类启动Allocator类的主方法。

进程类里面，存放大小，名称，起始时间，结束时间，以及占用的主存空间的起始地址。空闲块类中存放起始地址，长度，状态标志位。分配类主程序会指挥用户输入主存大小，系统大小，进程数目和各个进程的名字，起始终止时间，占据空间。

先排序，遍历,然后检查时间变量是否符合开始执行，如果是的话，“执行”该进程，不是则跳过，然后遍历检查有没有进程正在运行且结束时间已经到了，如果有，则“扫除”该进程。然后打印基本信息。时间到就停止。

第二个小题：设定全局时间变量，每次设定好四个进程号，依次变换成页然后送物理块，发生冲突按照FIFO算法替换。

**三、综合实验总结或结论**

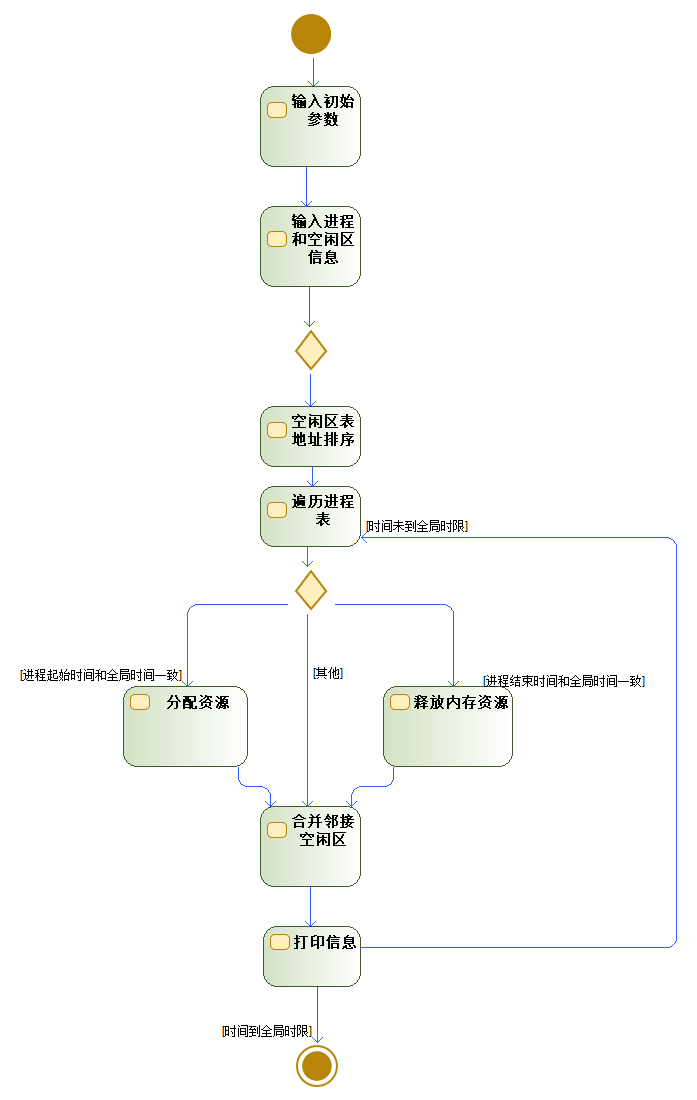
实验结果基本符合实验要求，本次实验编码量较大，且第二小题题目要求较为晦涩难懂，需要提前规划好方法，数据结构，以免编码时手忙脚乱漫无目的。

**四、参考文献**

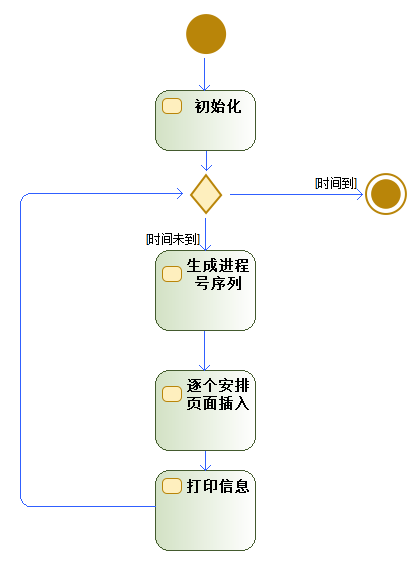
[1] 汤小丹, 梁红兵.《计算机操作系统》. 西安电子科技大学出版社, 第四版. 2014-05

**附录（设计流程图、程序、表格、数据等）**

第一题流程图：



第二题流程图：



第一题代码：

分配：

1. **package** threeAllocation;
3. **import** java.util.ArrayList;
4. **import** java.util.Arrays;
5. **import** java.util.List;
6. **import** java.util.Scanner;
8. **public** **class** Allocater {
9. Scanner in =**new** Scanner(System.in);
10. Space [] spaceChart;
11. Process [] processChart;
12. **int** systemSize;
13. **int** maxSize;
14. **int** ramSize;
15. **int** time;
16. //int chartTop;//用来表明最大的索引值
17. **public** Allocater() {
18. **super**();
19. **this**.processChart=**new** Process[10];
20. **this**.spaceChart=**new** Space [10];
21. //this.chartTop=0;
23. }
24. **public** **void** FillProcess() {
25. System.out.println("输入主存容量，整数。");
26. **this**.maxSize=in.nextInt();
27. System.out.println("输入系统大小，整数。");
28. **this**.systemSize=in.nextInt();
29. **this**.ramSize=**this**.maxSize-**this**.systemSize;
30. **this**.spaceChart[0]=**new** Space(**this**.systemSize,**this**.ramSize);
31. System.out.println("有几个需要服务的进程？");
32. **int** count=in.nextInt();
33. **for** (**int** i=0;i<count;i++) {
34. System.out.println("请输入进程大小，进程名称，进程起始时间，进程结束时间，空格或回车分割。");
35. **this**.processChart[i]=**new** Process(in.nextInt(), in.next(), in.nextInt(), in.nextInt());
36. }
37. }
38. **public** **void** Sort() {
39. **int** max=0;
40. **for** (**int** i=0;i<**this**.spaceChart.length;i++) {
41. **if** (**this**.spaceChart[i]==**null**) {
42. max=i-1;
43. **break**;
44. }
45. **else** {;}
46. }
47. **for** (**int** i=0;i<max;i++) {
48. **for** (**int** j=0;j<max-i;j++) {
49. **if** (spaceChart[j].getStart()>spaceChart[j+1].getStart()) {
50. Space temp=spaceChart[j];
51. spaceChart[j]=spaceChart[j+1];
52. spaceChart[j+1]=temp;
53. }
54. **else**;
55. }
56. }
57. }
58. **public** **void** Main() {
59. **this**.FillProcess();
60. **for** (time=0;time<30;time++) {//限定30s时间
61. **this**.Sort();//先排序
62. **if** (time==27) {;}**else** {;}
63. **for** (**int** i=0;i<**this**.processChart.length;i++) {//遍历进程
64. **this**.Sort();
65. **if** (**this**.processChart[i]==**null**) {//遍历到NULL直接跳过
66. **continue**;
67. }
68. **else** {
69. Process tmp=**this**.processChart[i];
70. **if** (tmp.startTime==time) {
71. **this**.Allocate(tmp);
72. **this**.Join();
73. }
74. **else** **if** (tmp.endTime==time) {
75. **this**.Sweep(tmp);
76. **this**.Join();
77. }
78. **else** {
79. **this**.Join();
80. **continue**;}
81. }//else
83. }//for
84. System.out.println("-----------Time Is "+time);
85. **this**.Printme();
86. }//for
87. }//main
88. **private** **void** Sweep(Process tmp) {//进程退出时归还资源
89. **int** max=0;
90. **for** (**int** i=0;i<**this**.spaceChart.length;i++) {
91. **if** (**this**.spaceChart[i]==**null**) {
92. max=i-1;
93. **break**;
94. }
95. **else** {;}
96. }
97. max++;
98. **if** (max>=**this**.spaceChart.length) {
99. System.out.println("返还失败");
100. **return**;
101. }
102. **else** {;}
103. **this**.spaceChart[max]=**new** Space(tmp.getSpaceStart(),tmp.size);
104. **this**.spaceChart[max].setStatus(1);
105. }
106. **public** **void** Allocate(Process pro) {//进程进入时给他分配
107. **int** max=0;
108. **for** (**int** i=0;i<**this**.spaceChart.length;i++) {
109. **if** (**this**.spaceChart[i]==**null**) {
110. max=i-1;
111. **break**;
112. }
113. **else** {;}
114. }
115. **if** (max==0) {
116. **if** (**this**.spaceChart[0].len<pro.size) {
117. System.out.println("空间不足，拒绝分配");
118. **return**;
119. }
120. **else** {
121. pro.setSpaceStart(**this**.spaceChart[0].start);
122. **this**.spaceChart[0].len-=pro.size;
123. **this**.spaceChart[0].start+=pro.size;
125. **return**;
126. }
127. }
128. **else** {
129. **for** (**int** i=0;i<max;i++) {
130. **if** (**this**.spaceChart[i].len<pro.size) {
131. **continue**;
132. }
133. **else** {
134. pro.setSpaceStart(**this**.spaceChart[i].start);
135. **this**.spaceChart[i].len-=pro.size;
136. **this**.spaceChart[i].start+=pro.size;
137. **continue**;
138. }
139. }
140. }
141. }
142. **public** **void** Printme() {//打印基本信息
143. **int** maxSpace=0;
144. **for** (**int** i=0;i<**this**.spaceChart.length;i++) {
145. **if** (**this**.spaceChart[i]==**null**) {
146. maxSpace=i-1;
147. **break**;
148. }
149. **else** {;}
150. }
151. **int** maxProc=0;
152. **for** (**int** i=0;i<**this**.processChart.length;i++) {
153. **if** (**this**.processChart[i]==**null**) {
154. maxProc=i-1;
155. **break**;
156. }
157. **else** {;}
158. }
159. System.out.println("下面开始打印信息\n空闲区：");
160. System.out.println("起始地址 长度 状态");
161. **for** (**int** i=0;i<=maxSpace;i++) {
162. Space temp=**this**.spaceChart[i];
163. System.out.println(temp.start+"#"+temp.len+"#"+temp.status);
164. }
165. System.out.println("进程区：");
166. System.out.println("名称 大小 起始时间 终止时间 ");
167. **for** (**int** i=0;i<=maxProc;i++) {
168. Process temp=**this**.processChart[i];
169. System.out.println(temp.name+"#"+temp.size+"#"+temp.startTime+"#"+temp.endTime);
170. }
171. }

174. **public** **int** getTime() {
175. **return** time;
176. }
177. **public** **void** setTime(**int** time) {
178. **this**.time = time;
179. }
180. **public** **void** Join() {//把空间上连续或完全重叠的多个空闲区合在一起
181. **this**.Sort();
182. List<Space> list=Arrays.asList(**this**.spaceChart);//将数组转换为list集合
183. List<Space> arr=**new** ArrayList<Space>(list);
184. **for** (**int** i=0;i<arr.size();i++) {
185. **if** (arr.get(i)==**null**) {**continue**;}
186. **else** {;}
187. **if** (arr.get(i).getLen()==0) {//零一概剔除
188. arr.remove(i);
189. i--;
190. }
191. **else** **if** (arr.get(i+1)!=**null**&&(arr.get(i).getLen()!=0)) {//邻接的焊接起来
192. **if** (arr.get(i).getStart()+arr.get(i).getLen()==arr.get(i+1).start) {
193. arr.get(i).setLen(arr.get(i).len + arr.get(i+1).getLen());
194. arr.remove(i+1);
195. }
196. **else** {;}
197. }
198. **else** {;}
199. }
200. **for** (**int** i=0;i<**this**.spaceChart.length;i++) {
201. **this**.spaceChart[i]=**null**;
202. }
203. **for** (**int** i=0;i<arr.size();i++) {
204. **this**.spaceChart[i]=arr.get(i);
205. }
206. }

209. }

空闲区：

1. **package** threeAllocation;
3. **public** **class** Space {
4. **int** start;//起始地址
5. **int** len;//长度
6. **int** status;//0表示未分配，1表示空闲
7. **public** Space(**int** start, **int** len) {
8. **super**();
9. **this**.start = start;
10. **this**.len = len;
11. **this**.status = 0;
12. }
13. **public** **int** getStart() {
14. **return** start;
15. }
16. **public** **int** getLen() {
17. **return** len;
18. }
19. **public** **int** getStatus() {
20. **return** status;
21. }
22. **public** **void** setStart(**int** start) {
23. **this**.start = start;
24. }
25. **public** **void** setLen(**int** len) {
26. **this**.len = len;
27. }
28. **public** **void** setStatus(**int** status) {
29. **this**.status = status;
30. }
31. **public** **int** compareTo(Space oneSpace) {
32. **return** Double.compare(**this**.getStart(),oneSpace.getStart());
33. }
35. }

进程：

1. **package** threeAllocation;
3. **public** **class** Process {
4. **int** size;
5. String name;
6. **int** startTime;
7. **int** endTime;
8. **int** spaceStart;//占用的SPACE开始地址
9. //int spaceLen;//占据的SPACE大小
10. //int spaceDivis;//space占据后，切分为两块，这就是分界地址,PROCESS总在前一块
11. **public** Process(**int** size, String name, **int** startTime, **int** endTime) {
12. **super**();
13. **this**.size = size;
14. **this**.name = name;
15. **this**.startTime = startTime;
16. **this**.endTime = endTime;
17. }
18. **public** **boolean** Finished(**int** curTime) {
19. **if** (curTime>=**this**.endTime) {
20. **return** **true**;
21. }
22. **else** {**return** **false**;}
23. }
24. **public** **boolean** Started(**int** curTime) {
25. **if** (curTime<**this**.startTime) {
26. **return** **true**;
27. }
28. **else** {**return** **false**;}
29. }
30. **public** **int** getSpaceStart() {
31. **return** spaceStart;
32. }
34. **public** **void** setSpaceStart(**int** spaceStart) {
35. **this**.spaceStart = spaceStart;
36. }




42. }

第二题代码：

进程：

1. **package** threePage;
3. **public** **class** Instruction {
4. String info;//用于区分
5. **boolean** visited;//false初始或未进入执行 true已经执行过，应该撤走（页表）或者跳过（主程序的随机选择）
6. **int** addr;//0~319
7. **int** destPage;
8. **int** offset;
9. **public** Instruction(String info, **int** addr) {
10. **super**();
11. **this**.info = info;
12. **this**.addr = addr;
13. **this**.visited=**false**;
14. **this**.destPage=addr/10;
15. **this**.offset=addr%10;
16. }
17. /////////////get&&set
18. **public** **void** recalcPageOffset() {
19. **this**.destPage=addr/10;
20. **this**.offset=addr%10;
21. }
23. **public** String getInfo() {
24. **return** info;
25. }
26. **public** **int** getDestPage() {
27. **return** destPage;
28. }
30. **public** **int** getOffset() {
31. **return** offset;
32. }
33. **public** **boolean** isVisited() {
34. **return** visited;
35. }
36. **public** **int** getAddr() {
37. **return** addr;
38. }
39. **public** **void** setInfo(String info) {
40. **this**.info = info;
41. }
42. **public** **void** setVisited(**boolean** visited) {
43. **this**.visited = visited;
44. }
45. **public** **void** setAddr(**int** addr) {
46. **this**.addr = addr;
47. **this**.destPage=addr/10;
48. **this**.offset=addr%10;
49. }

52. }

页：

1. **package** threePage;
3. **public** **class** Page {
5. Instruction ins;//暂存的指令.-114表示初始状态
6. **int** pageNum;//编号
7. **int** destBlock;
8. **int** offset;
9. **public** Page() {
10. **super**();
11. // TODO Auto-generated constructor stub
12. }
14. **public** Page(**int** pageNum) {
15. **super**();
16. **this**.pageNum = pageNum;
17. **this**.destBlock=pageNum/8;
18. **this**.offset=pageNum%8;
19. }
21. **public** Instruction getIns() {
22. **return** ins;
23. }
24. **public** **int** getPageNum() {
25. **return** pageNum;
26. }
27. **public** **int** getDestBlock() {
28. **return** destBlock;
29. }
30. **public** **int** getOffset() {
31. **return** offset;
32. }
33. **public** **void** setDestBlock(**int** destBlock) {
34. **this**.destBlock = destBlock;
35. }
36. **public** **void** setOffset(**int** offset) {
37. **this**.offset = offset;
38. }
39. **public** **void** setIns(Instruction ins) {
40. **this**.ins = ins;
41. }
42. **public** **void** setPageNum(**int** pageNum) {
43. **this**.pageNum = pageNum;
44. **this**.destBlock=pageNum/10;
45. **this**.offset=pageNum%10;
46. }
47. **public** **void** reCalcDestOffset() {
48. **this**.destBlock=pageNum/10;
49. **this**.offset=pageNum%10;
50. }
51. }

物理块：

1. **package** threePage;
3. **public** **class** MemBlock {
4. **int** blockNum;//编号
5. **int** lastIOTime;//最近写入的时刻
6. Page pageSave;//暂存的页
7. **public** MemBlock(**int** blockNum) {
8. **super**();
9. **this**.blockNum = blockNum;
10. **this**.lastIOTime=-114;//特殊初始值
11. }
12. **public** **int** getBlockNum() {
13. **return** blockNum;
14. }
15. **public** **int** getLastIOTime() {
16. **return** lastIOTime;
17. }
18. **public** Page getPageSave() {
19. **return** pageSave;
20. }
21. **public** **void** setBlockNum(**int** blockNum) {
22. **this**.blockNum = blockNum;
23. }
24. **public** **void** setLastIOTime(**int** lastIOTime) {
25. **this**.lastIOTime = lastIOTime;
26. }
27. **public** **void** setPageSave(Page pageSave) {
28. **this**.pageSave = pageSave;
29. }

32. }

主程序：

1. **package** threePage;
3. //import java.util.ArrayList;
4. **import** java.util.Random;
6. **public** **class** Conductor {
7. Random rGenerator = **new** Random();
8. **public** **static** **final** **int** instruNum=320;//320个地址指令，
9. **public** **static** **final** **int** pages=32; //一共32页
10. **public** **static** **final** **int** memBlock=4; //一共4物理块，i页对应物理块号是i/8,产生置换则移除
11. **int** time;
12. Page [] pageChart;//存放页面信息
13. Instruction [] insChart;//存放进程
14. MemBlock [] blockChart;//存放物理块信息
15. Instruction i1;
16. Instruction i2;
17. Instruction i3;
18. Instruction i4;
20. **public** Conductor() {
21. **super**();
22. // TODO Auto-generated constructor stub
23. **this**.pageChart=**new** Page[**this**.pages];
24. **this**.insChart=**new** Instruction[**this**.instruNum];
25. **this**.blockChart=**new** MemBlock[**this**.memBlock];
26. }
27. **public** **void** Main() {
28. **this**.initialize();
29. System.out.println("页面 物理0 物理1 物理2 物理3");
30. **for** (time=0;time<10;time++) {
31. **this**.setInsSequel();//安排好I1~I4都是什么块
32. **this**.insertPageChart(i1);
33. **this**.insertPageChart(i2);
34. **this**.insertPageChart(i3);
35. **this**.insertPageChart(i4);
37. }
38. }
39. **public** **void** initialize() {//初始化三个表
40. **for** (**int** i=0;i<**this**.instruNum;i++) {
41. **this**.insChart[i]=**new** Instruction("Instruction "+i,i);
42. }
43. **for** (**int** i=0;i<**this**.pages;i++) {
44. **this**.pageChart[i]=**new** Page(i);
45. }
46. **for** (**int** i=0;i<**this**.memBlock;i++) {
47. **this**.blockChart[i]=**new** MemBlock(i);
48. }
49. }//initia()
51. **public** **void** setInsSequel() {//设定I1 I2 I3 I4
52. **int** m1=-1;
53. **int** m2=-1;
54. **int** m3=-1;
55. **int** m4=-1;
56. **int** times=0;
57. **while** (times<=300) {
58. **int** m=**this**.rGenerator.nextInt(320);
59. m1=m+1;
60. m2=**this**.rGenerator.nextInt(m+2);
61. m3=m2+1;
62. m4=**this**.rGenerator.nextInt(318-m2)+2+m2;
63. **if** (**this**.setInsSequelSecure(m1)&&**this**.setInsSequelSecure(m2)&&
64. **this**.setInsSequelSecure(m3)&&**this**.setInsSequelSecure(m4))
65. {
66. **break**;
67. }
68. **else** {times++;**continue**;}
69. }
71. **if** (times>300) {System.out.println("Attempt tims out of limit");**return**;}
72. **else** {
73. **this**.i1=**this**.insChart[m1];
74. **this**.i2=**this**.insChart[m2];
75. **this**.i3=**this**.insChart[m3];
76. **this**.i4=**this**.insChart[m4];
77. **return**;}
78. }
80. **public** Boolean setInsSequelSecure(**int** index) {
81. **if** (index>=**this**.insChart.length||index<0) {
82. **return** **false**;
83. }
84. **else** **if** (**this**.insChart[index].isVisited()) {
85. **return** **false**;
87. }
88. **else** {**return** **true**;}
89. }
91. **public** Instruction fetchIns(**int** index) {//超限者排除，在数组内的，如果已经执行过也排除
92. **if** (index>=**this**.insChart.length||index<0||**this**.insChart[index].visited==**true**) {
93. **return** **null**;
94. }
95. **else** {
96. **return** **this**.insChart[index];
97. }
98. }//fetchIns
100. **public** **void** insertPageChart(Instruction inst) {//插入替换页表里的INS数据项
101. **if** (inst==**null**) {
102. System.out.println("Null Error");
103. **return**;}
104. **else** {;}
105. inst.recalcPageOffset();//计算应该分入哪一个页块
106. **if** (inst.getDestPage()>=0&&inst.getDestPage()<**this**.pageChart.length) {
107. inst.setVisited(**true**);
108. **this**.pageChart[inst.getDestPage()].setIns(inst);//该页的INS置为最新的INS
109. **this**.insertBlockChart(**this**.pageChart[inst.getDestPage()]);//该页块插入物理块
110. **return**;
111. }
112. **else** {
113. System.out.println("\n\n\n\n\nError at insertPageChart\n\n\n\n\n");//越界问题
114. **return**;
115. }
117. }//出来了无缝衔接insertBlockChart
118. **public** **void** insertBlockChart(Page pg) {
119. pg.reCalcDestOffset();//重新计算理论块表号
120. **int** tmpNum=pg.getDestBlock();//理论物理块号，引导到块表里的某一块
121. **if** (**this**.blockChart[tmpNum].getLastIOTime()==-114) {//要是空就直接放入
122. **this**.blockChart[tmpNum].setPageSave(pg);//存入页
123. **this**.blockChart[tmpNum].setLastIOTime(time);//更新时间（全局时钟）
124. **this**.printme(pg);
125. **return**;
126. }**else** {
127. MemBlock tmp = **null**;
128. **int** minTime=1919;//找最小的，或者另外找到空就直接置入
129. **for** (MemBlock mb:**this**.blockChart) {
130. **if** (mb.getLastIOTime()==-114) {
131. mb.setPageSave(pg);
132. mb.setLastIOTime(time);
133. **this**.printme(pg);
134. **return**;
135. }
136. **else** {
137. **if** (mb.getLastIOTime()<minTime) {
138. tmp=mb;
139. minTime=mb.getLastIOTime();
140. }
141. **else** {
142. ;
143. }
144. }
145. }//第一次遍历
146. **if** (tmp==**null**) {
147. System.out.println("\n\n\nUnknown Error At insertBlockChart\n\n\n");
148. **return** ;
149. }
150. **else** {
151. tmp.setLastIOTime(time);
152. tmp.setPageSave(pg);
153. **this**.printme(pg);
154. **return**;}
155. }
156. }//insertMemBlockChart
157. **public** **void** printme(Page pg) {
158. System.out.print(pg.getPageNum()+"#");
159. **if** (**this**.blockChart[0].getPageSave()!=**null**) {
160. System.out.print(**this**.blockChart[0].getPageSave().getPageNum()+"#");
161. }**else** {System.out.print(" "+"#");}
162. **if** (**this**.blockChart[1].getPageSave()!=**null**) {
163. System.out.print(**this**.blockChart[1].getPageSave().getPageNum()+"#");
164. }**else** {System.out.print(" "+"#");}
165. **if** (**this**.blockChart[2].getPageSave()!=**null**) {
166. System.out.print(**this**.blockChart[2].getPageSave().getPageNum()+"#");
167. }**else** {System.out.print(" "+"#");}
168. **if** (**this**.blockChart[3].getPageSave()!=**null**) {
169. System.out.print(**this**.blockChart[3].getPageSave().getPageNum()+"#");
170. }**else** {System.out.print(" "+"#");}
172. System.out.println();
173. //      System.out.println(
174. //              pg.getPageNum()+"#"+
175. //              this.blockChart[0].getPageSave().getPageNum()+"#"+
176. //              this.blockChart[1].getPageSave().getPageNum()+"#"+
177. //              this.blockChart[2].getPageSave().getPageNum()+"#"+
178. //              this.blockChart[3].getPageSave().getPageNum()+"#");
179. }//printme
180. }