

# LRU实验报告

57119101 王晨阳

2021年6月8日

实验目的

实验步骤

程序

运行结果

算法分析

实验体会

## 实验目的

通过实验，理解LRU页面置换算法的算法思想及其实现方法，比较各种实现算法的复杂度和实现难度，体会LRU算法与各种近似算法间的区别，并进而加深对虚拟内存概念的理解

## 实验步骤

### 程序

```
1  #include <cstdlib>
2  #include <ctime>
3  #include <iostream>
4  #include <vector>
5  using namespace std;
6
7  int Cache_Cap; //capacity
8  int Number_of_Pages;
9  int Test_Time;
10
11 //counter
12 struct Unit1
13 {
14     Unit1(int v, int c)
15     {
16         val = v;
17         cnt = c;
18     }
19     int val;
20     int cnt;
21 };
22
23 vector<Unit1> Cache1;
24
25 void Vis1(int Val, int &Pos, bool &Find)
```

```

26 {
27     int MaxPos = -1;
28     int MaxCnt = -1;
29     Find = false;
30     for (int i = 0; i < Cache1.size(); i++)
31     {
32         if (Cache1[i].cnt > MaxCnt)
33         {
34             MaxCnt = Cache1[i].cnt;
35             MaxPos = i;
36         }
37         Cache1[i].cnt++;
38         if (Cache1[i].val == Val)
39         {
40             Find = true;
41             Pos = i;
42             Cache1[Pos].cnt = 0;
43         }
44     }
45     if (!Find)
46     {
47         if (Cache1.size() < Cache_Cap)
48         {
49             Cache1.push_back(Unit1(Val, 0));
50             Pos = Cache1.size() - 1;
51             Cache1[Pos].cnt = 0;
52         }
53         else
54         {
55             Cache1[MaxPos].val = Val;
56             Cache1[MaxPos].cnt = 0;
57             Pos = MaxPos;
58         }
59     }
60 }
61
62 //stack
63 struct Unit2
64 {
65     Unit2(int v)
66     {
67         val = v;
68     }
69     int val;
70     int pre, nxt;
71 };
72
73 vector<Unit2> Cache2;
74
75 int top = 0, bot = 0;
76
77 void Vis2(int Val, int &Pos, bool &Find)
78 {
79     Find = false;
80     for (int i = 0; i < Cache2.size(); i++)
81     {
82         if (Cache2[i].val == Val)
83         {

```

```

84         Find = true;
85         Pos = i;
86         if (bot == Pos)
87         {
88             bot = Cache2[Pos].pre;
89             Cache2[top].pre = Pos;
90             Cache2[Pos].nxt = top;
91             top = Pos;
92         }
93         else if (top != Pos)
94         {
95             Cache2[Cache2[Pos].pre].nxt = Cache2[Pos].nxt;
96             Cache2[Cache2[Pos].nxt].pre = Cache2[Pos].pre;
97             Cache2[top].pre = Pos;
98             Cache2[Pos].nxt = top;
99             top = Pos;
100        }
101    }
102 }
103 if (!Find)
104 {
105     if (Cache2.size() < Cache_Cap)
106     {
107         Cache2.push_back(Unit2(Val));
108         Pos = Cache2.size() - 1;
109         Cache2[top].pre = Pos;
110         Cache2[Pos].nxt = top;
111         top = Pos;
112     }
113     else
114     {
115         Pos = 1;
116         bot = Cache2[Pos].pre;
117         Cache2[top].pre = Pos;
118         Cache2[Pos].nxt = top;
119         top = Pos;
120         Cache2[Pos].val = Val;
121     }
122 }
123 }
124
125 //Additional-Reference-Bits
126 struct Unit3
127 {
128     Unit3(int v, int b)
129     {
130         val = v;
131         bit = b;
132     }
133     int val;
134     unsigned char bit;
135 };
136
137 vector<Unit3> Cache3;
138
139 void Vis3(int Val, int &Pos, bool &Find)
140 {
141     int MinPos = -1;

```

```

142     unsigned char MinBit = 0xFF;
143     Find = false;
144     for (int i = 0; i < Cache3.size(); i++)
145     {
146         Cache3[Pos].bit <= 1;
147         if (Cache3[i].val == Val)
148         {
149             Find = true;
150             Pos = i;
151             Cache3[Pos].bit += (unsigned char)(1);
152         }
153         else if (Cache3[i].bit < MinBit)
154         {
155             MinBit = Cache3[i].bit;
156             MinPos = i;
157         }
158     }
159     if (!Find)
160     {
161         if (Cache3.size() < Cache_Cap)
162         {
163             Cache3.push_back(Unit3(Val, 0xFF));
164             Pos = Cache3.size() - 1;
165             Cache3[Pos].bit = 0xFF;
166         }
167         else
168         {
169             Cache3[MinPos].val = Val;
170             Cache3[MinPos].bit = 0xFF;
171             Pos = MinPos;
172         }
173     }
174 }
175
176 //Second chance
177 struct Unit4
178 {
179     Unit4(int v, int r)
180     {
181         val = v;
182         ref = r;
183     }
184     int val;
185     bool ref;
186 };
187
188 vector<Unit4> Cache4;
189
190 int ptr = 0;
191
192 void Vis4(int Val, int &Pos, bool &Find)
193 {
194     Find = false;
195     for (int i = 0; i < Cache4.size(); i++)
196     {
197         if (Cache4[i].val == Val)
198         {
199             Find = true;

```

```

200         Pos = i;
201         return;
202     }
203 }
204 if (ptr >= Cache4.size())
205 {
206     Cache4.push_back(Unit4(Val, 1));
207     ptr = (ptr + 1) % Cache_Cap;
208 }
209 else
210 {
211     while (Cache4[ptr].ref == 1)
212     {
213         Cache4[ptr].ref = 0;
214         ptr = (ptr + 1) % Cache_Cap;
215     }
216     Cache4[ptr].val = Val;
217     Cache4[ptr].ref = 1;
218 }
219 }
220
221 int main()
222 {
223     cout << "Input the capacity of the cache: ";
224     cin >> Cache_Cap;
225     cout << "Input the total number of different pages: ";
226     cin >> Number_of_Pages;
227     cout << "Input the time of tests: ";
228     cin >> Test_Time;
229
230     int WrongPage1 = 0, WrongPage2 = 0, WrongPage3 = 0, WrongPage4 = 0;
231
232     srand(time(0));
233     for (int i = 1; i <= Test_Time; i++)
234     {
235         int Pos;
236         bool Find;
237         int Num = rand() % Number_of_Pages;
238         Vis1(Num, Pos, Find);
239         if (!Find)
240             WrongPage1++;
241         Vis2(Num, Pos, Find);
242         if (!Find)
243             WrongPage2++;
244         Vis3(Num, Pos, Find);
245         if (!Find)
246             WrongPage3++;
247         Vis4(Num, Pos, Find);
248         if (!Find)
249             WrongPage4++;
250     }
251     cout<<endl;
252     cout << "Counter Implementation: " << WrongPage1 << " pages not
found." << endl;
253     cout << "Stack Implementation: " << WrongPage2 << " pages not
found." << endl;
254     cout << "Additional-Reference-Bits Algorithm: " << WrongPage3 << " pages not
found." << endl;

```

```
255     cout << "Second chance Algorithm: " << WrongPage4 << " pages not  
      found." << endl;  
256     cout<<endl;  
257 }
```

## 运行结果

### EXAMPLE

```
1   Input the capacity of the cache: 4  
2   Input the total number of different pages: 5  
3   Input the time of tests: 1000000  
4  
5   Counter Implementation:          199380 pages not found.  
6   Stack Implementation:           199651 pages not found.  
7   Additional-Reference-Bits Algorithm: 199573 pages not found.  
8   Second chance Algorithm:         199651 pages not found.
```

多次尝试后得出结论：若 Cache 大小为  $n$ ，共有  $p$  个不同的页，则在 *完全随机* 的状况下命中略高于  $\frac{c}{p}$ ，且四种算法命中率相近。考虑到现实情况会有大量的反复使用，命中率会远高于随机状况。

## 算法分析

若 Cache 大小为  $n$ ，在不使用哈希表优化的情况下，四种算法的时间复杂度均为  $O(n)$ （因为查找复杂度为  $O(n)$ ）。

若使用哈希表优化查找（使查找时间复杂度为  $O(1)$ ），则算法时间复杂度如下

算法	时间复杂度
Counter Implementation	$O(n)$
Stack Implementation	$O(1)$
Additional-Reference-Bits Algorithm	$O(n)$
Second chance Algorithm	最优状况下 $O(1)$ ，最坏状况下 $O(n)$

综上，若要求效率最高，应当使用哈希表优化的栈实现 LRU。

但考虑到硬件不足以支持较高的空间复杂度和复杂的数据结构，Counter Implementation 和 Stack Implementation 较难实现；

而 Additional-Reference-Bits Algorithm 和 Second chance Algorithm 区别不大。

## 实验体会

通过本次实验，掌握了LRU的四种算法及其实现，了解了四种算法的优劣，加深了对操作系统概念的理解。提高了动手能力，解决问题的能力得到强化。

