****

**计算机体系结构实验报告**

|  |  |
| --- | --- |
| 学生姓名： | 谭哲文 |
| 学 号： | 8202191123 |
| 专业班级： | 计科2105班 |
| 指导教师： | 余腊生 |
| 学 院： | 计算机学院 |
| 完成时间： | 2023/12/31 |

**实验2 使用LRU方法更新Cache**

**一、实验目的**

了解和掌握寄存器分配和内存分配的有关技术。

**二、实验内容**

结合数据结构的相关知识，使用LRU的策略，对一组访问序列进行内部的Cache更新。

LRU置换算法是选择最近最久未使用的页面予以置换。该算法赋予每个页面一个访问字段，用来记录一个页面自上次被访问以来经历的时间T，当须淘汰一个页面时，选择现有页面中T值最大的，即最近最久没有访问的页面。这是一个比较合理的置换算法。

举例说明此问题，例如：

有一个CACHE采用组相连映象方式。每组有四块，为了实现LRU置换算法，在快表中为每块设置一个2位计数器。我们假设访问序列为“1,1,2,4,3,5,2,1,6,7,1,3”。在访问CACHE的过程中，块的装入，置换及命中时，具体情况如下表所示：

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 1 | 2 | 4 | 3 | 5 | 2 | 1 | 6 | 7 | 1 | 3 |
| Cache块0 | 1 | 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 7 | 7 | 7 |
| Cache块1 |  |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 |
| Cache块2 |  |  |  | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 1 | 1 |
| Cache块3 |  |  |  |  | 3 | 3 | 3 | 3 | 6 | 6 | 6 | 6 |
|  | 装入 | 命中 | 装入 | 装入 | 装入 | 置换 | 命中 | 置换 | 置换 | 置换 | 命中 | 置换 |

**三、设计描述**

设计要求是使用LRU的策略，对一组访问序列进行内部的Cache更新。实现LRU算法可以在每次访问Cache时，把当前Cache块的count值置为0。如果Cache有空闲块，就直接将数据加载到Cache。如果Cache没有空闲块，就从Cache中查找数据是否命中。如果命中，就将该Cache块的count值置为0，并且更新其它Cache块的count值。如果Cache中的数据没有命中，就从Cache块中找count值最大的Cache块进行置换。

**具体分析：**

代码需要实现一个LRU置换算法，使用了一个Cache类型的数组来表示缓存，每个Cache块有三个属性：state，表示这个块是否被使用；value，表示这个块存储的值；count，表示这个块没被使用的次数。

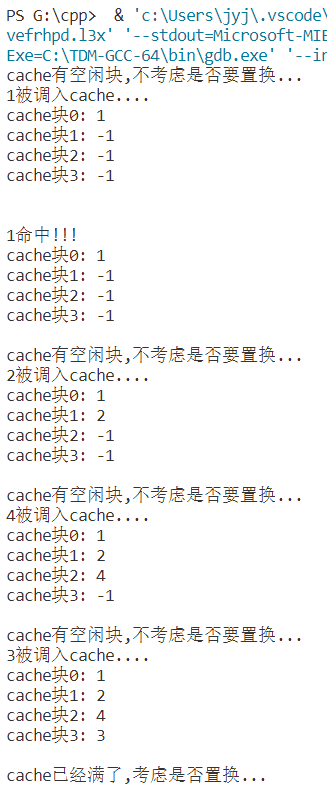
在主函数中调用了up\_cache函数，在up\_cache函数中，使用一个循环遍历测试数据，对于每一个数据，首先检查Cache是否有空闲块，如果有，则将当前数据调入缓存，并更新Cache块的信息，如果Cache没有空闲块，则检查Cache中是否有值与当前数据相等的块，如果有，则标记命中，并更新Cache块的信息，如果Cache中没有值与当前数据相等的块，则进行置换，将count值最大的Cache块置换掉，将当前数据调入缓存，并更新Cache块的信息。

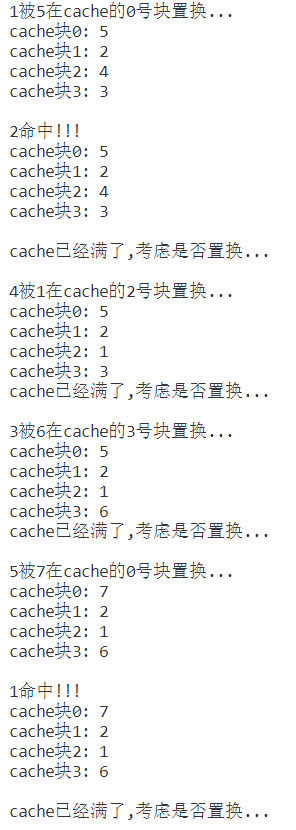
**四、程序清单**

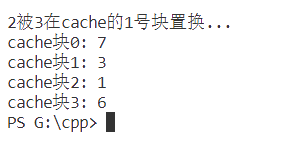
|  |
| --- |
| #include <iostream>  using namespace std;  // Cache类型表示缓存块  class Cache  {  public:      // state表示这个块是否被使用      bool state = false;      // value表示这个块存储的值      int value = -1;      // count表示这个块没被使用的时间      int count = 0;  };  // 缓存块数  const int M = 4;  // 声明Cache类型的数组表示缓存  Cache cache[M];  // 测试页面数  const int N = 12;  // 测试数据  int walk\_sort[] = {1, 1, 2, 4, 3, 5, 2, 1, 6, 7, 1, 3};  void up\_cache();  int main()  {      up\_cache();      return 0;  }  void up\_cache()  {      int i = 0;      while (i < N)      {          int j = 0;          // 满么？          while (j < M)          {              if ((cache[j].state == false) && (walk\_sort[i] != cache[j].value))              {                  cout << "cache有空闲块,不考虑是否要置换..." << endl;                  cout << walk\_sort[i] << "被调入cache...." << endl;                  cache[j].value = walk\_sort[i++];                  cache[j].state = true;                  cache[j].count = 0;                  int kk = 0;                  for (int x = 0; x < M; x++)                  {                      cout << "cache块" << x << ": " << cache[x].value << endl;                  }                  cout << endl;                  // 更新其它cache块没使用时间                  while (kk < M)                  {                      if (kk != j && cache[kk].value != -1)                      {                          cache[kk].count++;                      }                      kk++;                  }                  break;              }              if (cache[j].value == walk\_sort[i])              {                  cout << endl;                  cout << walk\_sort[i] << "命中!!!" << endl;                  for (int x = 0; x < M; x++)                  {                      cout << "cache块" << x << ": " << cache[x].value << endl;                  }                  cout << endl;                  int kk = 0;                  i++;                  cache[j].count = 0;                  // 更新其它cache块没使用时间                  while (kk < M)                  {                      if (kk != j && cache[kk].value != -1)                      {                          cache[kk].count++;                      }                      kk++;                  }              }              j++;          }          if (j == M)          {              cout << "cache已经满了,考虑是否置换..." << endl;              cout << endl;              int k = 0;              while (k < M)              {                  if (cache[k].value == walk\_sort[i])                  {                      cout << endl;                      cout << walk\_sort[i] << "命中!!!" << endl;                      for (int x = 0; x < M; x++)                      {                          cout << "cache块" << x << ": " << cache[x].value << endl;                      }                      i++;                      cache[k].count = 0;                      int kk = 0;                      // 更新其它cache块没使用时间                      while (kk < M)                      {                          if (kk != k)                          {                              cache[kk].count++;                          }                          kk++;                      }                      break;                  }                  k++;              }              // 考虑置换那一块.              if (k == M)              {                  int ii = 0;                  int t = 0; // 要替换的cache块号.                  int max = cache[ii].count;                  ii++;                  while (ii < M)                  {                      if (cache[ii].count > max)                      {                          max = cache[ii].count;                          t = ii;                      }                      ii++;                  }                  // 置换                  cout << cache[t].value << "被" << walk\_sort[i] << "在cache的" << t << "号块置换..." << endl;                  cache[t].value = walk\_sort[i++];                  cache[t].count = 0;                  for (int x = 0; x < M; x++)                  {                      cout << "cache块" << x << ": " << cache[x].value << endl;                  }                  int kk = 0;                  // 更新其它cache块没使用时间                  while (kk < M)                  {                      if (kk != t)                      {                          cache[kk].count++;                      }                      kk++;                  }              }          }      }  } |

**五、结果分析**

**输出结果：**







**调试分析：**

根据运行结果分析符合预期的结果，证明LRU算法正确。

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 1 | 2 | 4 | 3 | 5 | 2 | 1 | 6 | 7 | 1 | 3 |
| Cache块0 | 1 | 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 7 | 7 | 7 |
| Cache块1 |  |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 |
| Cache块2 |  |  |  | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 1 | 1 |
| Cache块3 |  |  |  |  | 3 | 3 | 3 | 3 | 6 | 6 | 6 | 6 |
|  | 装入 | 命中 | 装入 | 装入 | 装入 | 置换 | 命中 | 置换 | 置换 | 置换 | 命中 | 置换 |

**六、实验心得**

在这次实验中，我们学习了LRU（Least Recently Used）算法，并将它应用到了我们的Cache程序中。实验发现，通过使用LRU算法， Cache可以更有效地维护最近使用的数据，并将最久未使用的数据淘汰出去。这样， Cache不仅可以保存更多的数据，而且可以更快地访问最近使用的数据，这对于提高程序的效率是非常有帮助的。

通过这次实验，我对LRU算法有了更深的理解，并且学会了如何使用LRU算法来更新Cache。这对于我今后开发程序时，优化程序的效率会有很大的帮助。我们可以使用LRU算法来维护Cache中的数据，从而让Cache更有效地工作，提高程序的效率。我很高兴能够在这次实验中学到这么多知识，并期待未来能够运用所学的知识来开发更加优秀的程序。