

**实 验 报 告**

**（ 2024/2025 学年 第 一 学期 ）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 计算机操作系统 | | | | | |
| 实验名称 | 页面置换算法模拟 | | | | | |
| 实验时间 | 2024 | 年 | 12 | 月 | 11 | 日 |
| 指导单位 | 物联网学院 | | | | | |
| 指导教师 |  | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 学生姓名 |  | 班级学号 |  |
| 学院(系) | 物联网学院 | 专 业 | 网络工程 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **实验名称** | 页面置换算法模拟 | | | | | | | | | **指导教师** | |  |
| **实验类型** | | 验证性实验 | | **实验学时** | | 2 | | | **实验时间** | | | 2022.12.11 |
| 1. **实验目的和要求** | | | | | | | | | | | | |
| **实验目的：**学习虚拟存储管理中页面调度算法，通过编程模拟实验FIFO算法、LRU算法、OPT算法，比较各算法的性能。  **实验要求：**  深入理解虚拟存储机制，掌握虚拟存储中页面置换算法实现方法。 | | | | | | | | | | | | |
| **二、实验环境(实验设备)** | | | | | | | | | | | | |
| Linux/windows | | | | | | | | | | | | |
| 1. **实验原理及内容**   1. 先进先出的算法(FIFO)  2. 最近最久未使用算法(LRU)  3. 最佳置换算法(OPT)  1(先进先出(FIFO)置换算法的思路  该算法总是淘汰最先进入内存的页面，即选择在内存中驻留时间最久的页面予以淘汰。该算法实现简单，只需把一个进程已调入内存的页面，按照先后次序连接成一个队列，并设置一个替换指针，使它总指向最老的页面。  2(最近久未使用(LRU)置换算法的思路  最近久未使用置换算法的替换规则，是根据页面调入内存后的使用情况来进行决策的。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所经历的时间，当需淘汰一个页面的时候选择现有页面中其时间值最大的进行淘汰。  3.最佳(OPT)置换算法的思路  其所选择的被淘汰的页面，奖是以后不使用的，或者是在未来时间内不再被访问的页面，采用最佳算法，通常可保证获得最低的缺页率。  4(FIFO页面置换算法  当需要访问一个新的页面时，首先调用findExist(i)函数来查看物理块中是否就有这个页面，若要查看的页面物理块中就有，则调用display函数直接显示，不需要替换页面;如果要查看的页面物理块中没有，就需要寻找空闲物理块放入，若存在有空闲物理块，则将页面放入;若没有空闲物理块，则调用findReplace函数替换页面。并将物理块中所有页面timer++。  5(LRU页面置换算法  当需要访问一个新的页面，首先调用findExist(i)函数查看物理块中是否就有这个页面。  6. OPT页面置换算法  当需要访问一个新的页面，首先调用findExist(i)函数来查看物理块中是否有这个页面。  7(寻找置换页面函数findReplace比较三个物理块中的时间标记timer，找到时间最久的。  **1. 先进先出的算法(FIFO)**  #include <deque>  #include <cstdio>  #include <algorithm>  using namespace std;  int main()  {  deque<int> dq;  deque<int>::iterator pos;  int numyk, numqueye = 0;  printf("请输入物理页框块数:");  scanf("%d", &numyk);  int n;  printf("\n请输入页面走向个数：");  scanf("%d", &n);  for (int i = 0; i < n; i++)  {  int in;  scanf("%d", &in);  if (dq.size() < numyk) // 存在空余页框  {  int flag = 0;  for (pos = dq.begin(); pos != dq.end(); pos++) // 遍历队列  if ((\*pos) == in)  {  flag = 1;  break;  }  if (!flag) // 不存在此元素  {  numqueye++;  dq.push\_back(in); // 放入队列  }  }  else // 不存在多余页框  {  int flag = 0;  for (pos = dq.begin(); pos != dq.end(); pos++)  if ((\*pos) == in)  {  flag = 1;  break;  } // 存在该元素  if (!flag) // 不存在此元素 则置换最先进入的项  {  numqueye++; // 缺页数+1  dq.pop\_front(); // 最先进入的出队列  dq.push\_back(in); // 进队列  }  }  }  printf("fifo缺页次数为:%d\n", numqueye);  printf("fifo缺页中断率为：%lf\n", (double)numqueye \* 1.0 / n);  }    **2. 最佳置换算法(OPT)**  #include <deque>  #include <cstdio>  #include <algorithm>  using namespace std;  struct opt  {      int value;      int time;  };  const int maxn = 105;  int a[maxn];  int main()  {      deque<opt> dq;      deque<opt>::iterator pos;      int numyk, numqueye = 0;      printf("请输入物理页框块数:");      scanf("%d", &numyk);      int n;      printf("\n请输入页面走向个数：");      scanf("%d", &n);      for (int i = 0; i < n; i++)          scanf("%d", &a[i]);      for (int i = 0; i < n; i++)      {          printf("第%d个\n", i);          int in;          in = a[i];          if (dq.size() < numyk) // 存在多余页框          {              int flag = 0;              for (pos = dq.begin(); pos != dq.end(); pos++)                  if ((\*pos).value == in) // 存在元素和它相同                  {                      flag = 1;                      break;                  } // 存在该元素              if (!flag) // 不存在此元素              {                  numqueye++;                  opt temp;                  temp.value = in;                  int f = 0;                  for (int j = i + 1; j < n; j++)                      if (a[j] == in)                      {                          f = 1;                          temp.time = j - i;                          break;                      }                  if (!f)                      temp.time = n;                  dq.push\_back(temp);              }          }          else // 不存在多余页框          {              int flag = 0;              for (pos = dq.begin(); pos != dq.end(); pos++)                  if ((\*pos).value == in)                  {                      flag = 1;                      break;                  } // 存在该元素              if (!flag) // 不存在此元素 则置换time最大的项              {                  numqueye++; // 缺页数+1                  int m = dq.front().time;                  printf("m初始值为%d\n", m);                  deque<opt>::iterator mp = dq.begin(); // 注意此处千万注意初始化 否则有可能erase找不到对象崩溃                  for (pos = dq.begin(); pos != dq.end(); pos++)                  {                      printf("%d %d\n", (\*pos).value, (\*pos).time);                      if ((\*pos).time > m)                      {                          printf("迭代");                          mp = pos; // 时间最大的元素的位置                          m = (\*pos).time;                      }                  }                  opt temp;                  temp.value = in;                  int f = 0;                  dq.erase(mp);                  for (int j = i + 1; j < n; j++)                      if (a[j] == in)                      {                          f = 1;                          temp.time = j - i;                          break;                      }                  if (!f)                      temp.time = n;                  dq.push\_back(temp);              }          }          // 每次之后重置          for (pos = dq.begin(); pos != dq.end(); pos++)          {              printf("队列中的元素为 %d\n", (\*pos).value);              int f = 0;              for (int j = i + 1; j < n; j++)                  if (a[j] == (\*pos).value)                  {                      f = 1;                      (\*pos).time = j - i;                      break;                  }              if (!f)                  (\*pos).time = n;          }      }      printf("opt缺页次数为:%d\n", numqueye);      printf("opt缺页中断率为：%lf\n", (double)numqueye \* 1.0 / n);  }    **2. 最近最久未使用算法(LRU)**  #include <deque>  #include <cstdio>  #include <algorithm>  using namespace std;  struct opt  {      int value; // 值      int time;  // 时间  };  const int maxn = 105;  int a[maxn];  int main()  {      deque<opt> dq;      deque<opt>::iterator pos;      int numyk, numqueye = 0;      printf("请输入物理页框块数:");      scanf("%d", &numyk);      int n;      printf("请输入页面走向个数：");      scanf("%d", &n);      for (int i = 0; i < n; i++)          scanf("%d", &a[i]);      for (int i = 0; i < n; i++)      {          int in;          in = a[i];          if (dq.size() < numyk) // 存在多余页框          {              int flag = 0;              for (pos = dq.begin(); pos != dq.end(); pos++)                  if ((\*pos).value == in) // 存在元素和它相同                  {                      flag = 1;                      break;                  } // 存在该元素              if (!flag) // 不存在此元素              {                  numqueye++;                  opt temp;                  temp.value = in;                  temp.time = 0;                  dq.push\_back(temp);              }          }          else // 不存在多余页框          {              int flag = 0;              for (pos = dq.begin(); pos != dq.end(); pos++)                  if ((\*pos).value == in)                  {                      flag = 1;                      break;                  } // 存在该元素              if (!flag) // 不存在此元素              {                  numqueye++; // 缺页数+1                  int m = dq.front().time;                  deque<opt>::iterator mp = dq.begin(); // 注意此处千万注意初始化 否则有可能erase找不到对象崩溃                  for (pos = dq.begin(); pos != dq.end(); pos++)                  {                      printf("%d %d\n", (\*pos).value, (\*pos).time);                      if ((\*pos).time > m)                      {                          printf("迭代");                          mp = pos; // 时间最大的元素的位置                          m = (\*pos).time;                      }                  }                  printf("此时队列中所剩时间最长的元素为%d\n", (\*mp).value);                  opt temp;                  temp.value = in;                  int f = 0;                  dq.erase(mp);                  temp.time = 0; // 加进来之后                  dq.push\_back(temp);              }          }          // 每次之后各对象的time          for (pos = dq.begin(); pos != dq.end(); pos++)          {              printf("队列中的元素为 %d\n", (\*pos).value);              int f = 0;              for (int j = i; j >= 0; j--)                  if (a[j] == (\*pos).value)                  {                      (\*pos).time = i - j;                      break;                  }              printf("距离上次时间为%d\n", (\*pos).time);          }      }      printf("lru缺页次数为:%d\n", numqueye);      printf("lru缺页中断率为：%lf\n", (double)numqueye \* 1.0 / n);  } | | | | | | | | | | | | |
| **四、实验小结（包括问题和解决方法、心得体会、意见与建议等）** | | | | | | | | | | | | |
| 在本次实验中，我们通过编程实现了虚拟存储管理中的三种经典页面置换算法：先进先出（FIFO）、最近最久未使用（LRU）、和最佳置换（OPT）算法。通过模拟实验，我们深入理解了每种算法的工作原理，并对比了它们的性能和适用场景。  在实验过程中，我们遇到了以下几个问题。首先，在实现FIFO算法时，由于对队列操作的理解不够深入，初始版本的程序未能正确更新缺页中断率。在代码调试过程中，我们通过详细跟踪队列的入队和出队逻辑，成功修复了问题，确保程序能够正确计算缺页次数和中断率。其次，在实现OPT算法时，如何预测未来的页面访问顺序是一个挑战。为此，我们在代码中引入了一个数组来记录页面的未来访问情况，从而精确找到需要替换的页面。最后，LRU算法的实现中，对时间标记的更新逻辑需要特别小心，以确保每次访问后时间值的正确性。通过在代码中加入调试打印，我们成功解决了这一问题。  通过本次实验，我深刻体会到不同页面置换算法在实现和性能上的差异。例如，FIFO算法的实现相对简单，但在实际场景中容易出现缺页率反常增加的异常；LRU算法更符合现实使用情况，但需要额外的时间和空间来维护页面的最近使用时间；而OPT算法虽然理论上缺页率最低，但在实际中因无法准确预测未来而难以直接应用。这让我认识到，算法选择需要根据实际场景的需求和限制进行权衡。  本次实验还培养了我解决问题的能力和对细节的关注。在编程过程中，调试和优化代码让我更加理解虚拟存储管理的核心思想，同时也提升了代码编写的规范性和逻辑性。这些经验将为今后的学习和工作打下坚实的基础。 | | | | | | | | | | | | |
| **五、指导教师评语** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **成 绩** | | |  | | **批阅人** | |  | **日 期** | | |  | |