山东大学 软件 学院

**操作系统** 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202100300157 | 姓名：孙荣骏 | | 班级：21.3 |
| 实验编号：实验6 | | | |
| 实验题目：进程互斥实验 | | | |
| 实验学时：2 | | 实验日期：2022.5.19 | |
| 实验目的：  加深对于存储管理的了解，掌握虚拟存储器的实现原理；观察和了解重要的页面  置换算法和置换过程。练习模拟算法的编程技巧，锻炼分析试验数据的能力。 | | | |
| 硬件环境：  宿主机  机型：联想拯救者r7000p2021  CPU：AMD R7 5800H  内存：16G  虚拟机  RAM：4GB | | | |
| 软件环境：  虚拟机：Ubuntu 16.04  宿主机：win10 | | | |
| 实验步骤与内容：  **独立实验：**                              主要实验代码及注释如下：  #include "vm.h"  using namespace std;  Replace::Replace() {  int i;  //设定总得访问页数,并分配相应的引用页号和淘汰页号记录数组空间  cout << "Please input page numbers :";  cin >> PageNumber;  ReferencePage = new int[sizeof(int) \* PageNumber];  EliminatePage = new int[sizeof(int) \* PageNumber];  op = new int[sizeof(int) \* PageNumber];  //输入引用页号序列(页面走向),初始化引用页数组  cout << "Please input reference page string :";  for (i = 0; i < PageNumber; i++)  cin >> ReferencePage[i]; //引用页暂存引用数组  //设定内存实页数(帧数),并分配相应的实页号记录数组空间(页号栈)  cout << "Please input page frames :";  cin >> FrameNumber;  PageFrames = new int[sizeof(int) \* FrameNumber];  ReferenceBit = new int[sizeof(int) \* FrameNumber];  count = new int[sizeof(int) \* FrameNumber];  ModifyBit = new int[sizeof(int) \* FrameNumber];  }  Replace::~Replace() {}  void Replace::InitSpace(char \*MethodName) {  int i;  cout << endl << MethodName << endl;  FaultNumber = 0;  //引用还未开始,-1 表示无引用页  for (i = 0; i < PageNumber; i++) EliminatePage[i] = -1;  for (i = 0; i < FrameNumber; i++) {  PageFrames[i] = -1;  ReferenceBit[i] = 0;  count[i] = 0;  ModifyBit[i] = 0;  }  }  //分析统计选择的算法对于当前输入的页面走向的性能  void Replace::Report(void) {  //报告淘汰页顺序  cout << endl << "Eliminate page:";  for (int i = 0; EliminatePage[i] != -1; i++)  cout << EliminatePage[i] << " ";  //报告缺页数和缺页率  cout << endl << "Number of page faults = " << FaultNumber << endl;  cout << setw(6) << setprecision(3);  cout << "Rate of page faults = "  << 100 \* (float)FaultNumber / (float)PageNumber << "%" << endl;  }  //最近最旧未用置换算法  void Replace::Lru(void) {  int i, j, k, l, next;  InitSpace("LRU");  //循环装入引用页  for (k = 0, l = 0; k < PageNumber; k++) {  next = ReferencePage[k];  //检测引用页当前是否已在实存  for (i = 0; i < FrameNumber; i++) {  if (next == PageFrames[i]) {  //引用页已在实存将其调整到页记录栈顶  next = PageFrames[i];  for (j = i; j > 0; j--) PageFrames[j] = PageFrames[j - 1];  PageFrames[0] = next;  break;  }  }  if (PageFrames[0] == next) {  //如果引用页已放栈顶，则为不缺页，报告当前内存页号  for (j = 0; j < FrameNumber; j++)  if (PageFrames[j] >= 0) cout << PageFrames[j] << " ";  cout << endl;  continue; //继续装入下一页  } else  // 如果引用页还未放栈顶，则为缺页，缺页数加 1  FaultNumber++;  //栈底页号记入淘汰页数组中  EliminatePage[l] = PageFrames[FrameNumber - 1];  //向下压栈  for (j = FrameNumber - 1; j > 0; j--) PageFrames[j] = PageFrames[j - 1];  PageFrames[0] = next; //引用页放栈顶  //报告当前实存中页号  for (j = 0; j < FrameNumber; j++)  if (PageFrames[j] >= 0) cout << PageFrames[j] << " ";  //报告当前淘汰的页号  if (EliminatePage[l] >= 0)  cout << "->" << EliminatePage[l++] << endl;  else  cout << endl;  }  //分析统计选择的算法对于当前引用的页面走向的性能  Report();  }  //先进先出置换算法  void Replace::Fifo(void) {  int i, j, k, l, next;  InitSpace("FIFO");  //循环装入引用页  for (k = 0, j = l = 0; k < PageNumber; k++) {  next = ReferencePage[k];  //如果引用页已在实存中，报告实存页号  for (i = 0; i < FrameNumber; i++)  if (next == PageFrames[i]) break;  if (i < FrameNumber) {  for (i = 0; i < FrameNumber; i++) cout << PageFrames[i] << " ";  cout << endl;  continue; // 继续引用下一页  }  //引用页不在实存中，缺页数加 1  FaultNumber++;  EliminatePage[l] = PageFrames[j]; //最先入页号记入淘汰页数组  PageFrames[j] = next; //引用页号放最先入页号处  j = (j + 1) % FrameNumber; //最先入页号循环下移  //报告当前实存页号和淘汰页号  for (i = 0; i < FrameNumber; i++)  if (PageFrames[i] >= 0) cout << PageFrames[i] << " ";  if (EliminatePage[l] >= 0)  cout << "->" << EliminatePage[l++] << endl;  else  cout << endl;  }  //分析统计选择的算法对于当前引用的页面走向的性能  Report();  }  //时钟(二次机会）置换算法  void Replace::Clock(void) {  int j, i, k, l, next;  InitSpace("Clock");  for (k = 0, j = l = 0; k < PageNumber; k++) {  next = ReferencePage[k];  for (i = 0; i < FrameNumber; i++)  if (next == PageFrames[i]) {  ReferenceBit[i] = 1;  break;  }  if (i < FrameNumber) {  for (i = 0; i < FrameNumber; i++)  if (PageFrames[i] != -1)  cout << PageFrames[i] << " ";  else  break;  cout << endl;  continue;  }  while (true) {  if (ReferenceBit[j] == 1) {  ReferenceBit[j] = 0;  j = (j + 1) % FrameNumber;  } else {  EliminatePage[l] = PageFrames[j];  PageFrames[j] = next;  FaultNumber++;  for (i = 0; i < FrameNumber; i++)  if (PageFrames[i] >= 0) cout << PageFrames[i] << " ";  if (EliminatePage[l] >= 0)  cout << "->" << EliminatePage[l++] << endl;  else  cout << endl;  j = (j + 1) % FrameNumber;  break;  }  }  }  Report();  }  //增强二次机会置换算法  void Replace::Eclock(void) {  int j, i, k, l, next;  InitSpace("EClock");  srand(time(0));  for (i = 0; i < PageNumber; i++) op[i] = rand() % 2;  for (k = 0, j = l = 0; k < PageNumber; k++) {  next = ReferencePage[k];  for (i = 0; i < FrameNumber; i++)  if (next == PageFrames[i]) {  ReferenceBit[i] = 1;  if (op[k]) ModifyBit[i] = 1;  break;  }  if (i < FrameNumber) {  for (i = 0; i < FrameNumber; i++)  if (PageFrames[i] != -1)  cout << PageFrames[i] << " ";  else  break;  cout << endl;  continue;  }  int min = 10 \* ReferenceBit[j] + ModifyBit[j];  int index = j;  int cnt = 0;  i = (index + 1) % FrameNumber;  while (cnt++ < FrameNumber) {  if (10 \* ReferenceBit[i] + ModifyBit[i] < min) {  min = 10 \* ReferenceBit[i] + ModifyBit[i];  index = i;  }  }  EliminatePage[l] = PageFrames[index];  PageFrames[index] = next;  ReferenceBit[index] = 0;  ModifyBit[index] = 0;  FaultNumber++;  j = (index + 1) % FrameNumber;  for (i = 0; i < FrameNumber; i++)  if (PageFrames[i] >= 0) cout << PageFrames[i] << " ";  if (EliminatePage[l] >= 0)  cout << "->" << EliminatePage[l++] << endl;  else  cout << endl;  }  Report();  }  //最不经常使用置换算法  void Replace::Lfu(void) {  int j, i, k, l, next;  InitSpace("Lfu");  for (k = 0, j = l = 0; k < PageNumber; k++) {  next = ReferencePage[k];  for (i = 0; i < FrameNumber; i++)  if (next == PageFrames[i]) {  count[i]++;  break;  }  if (i < FrameNumber) {  for (i = 0; i < FrameNumber; i++)  if (PageFrames[i] != -1)  cout << PageFrames[i] << " ";  else  break;  cout << endl;  continue;  }  FaultNumber++;  int min = count[0];  int index = 0;  for (i = 0; i < FrameNumber; i++) {  if (count[i] < min) {  min = count[i];  index = i;  }  }  EliminatePage[l] = PageFrames[index];  PageFrames[index] = next;  count[index] = 1;  for (i = 0; i < FrameNumber; i++)  if (PageFrames[i] >= 0) cout << PageFrames[i] << " ";  if (EliminatePage[l] >= 0)  cout << "->" << EliminatePage[l++] << endl;  else  cout << endl;  }  Report();  }  //最经常使用置换算法  void Replace::Mfu(void) {  int j, i, k, l, next;  InitSpace("Mfu");  for (i = 0; i < FrameNumber; i++) count[i] = 1;  for (k = 0, j = l = 0; k < PageNumber; k++) {  next = ReferencePage[k];  for (i = 0; i < FrameNumber; i++)  if (next == PageFrames[i]) {  count[i]++;  break;  }  if (i < FrameNumber) {  for (i = 0; i < FrameNumber; i++)  if (PageFrames[i] != -1)  cout << PageFrames[i] << " ";  else  break;  cout << endl;  continue;  }  FaultNumber++;  int max = count[0];  int index = 0;  for (i = 0; i < FrameNumber; i++) {  if (count[i] > max) {  max = count[i];  index = i;  }  }  EliminatePage[l] = PageFrames[index];  PageFrames[index] = next;  count[index] = 0;  for (i = 0; i < FrameNumber; i++)  if (PageFrames[i] >= 0) cout << PageFrames[i] << " ";  if (EliminatePage[l] >= 0)  cout << "->" << EliminatePage[l++] << endl;  else  cout << endl;  }  Report();  }  int main(int argc, char \*argv[]) {  Replace \*vmpr = new Replace();  vmpr->Lru();  vmpr->Fifo();  vmpr->Clock();  vmpr->Eclock();  vmpr->Lfu();  vmpr->Mfu();  return 0;  }  #include <malloc.h>  #include <stdlib.h>  #include <iomanip>  #include <iostream>  class Replace {  public:  Replace();  ~Replace();  void InitSpace(char\* MethodName); //初始化页号记录  void Report(void); // 报告算法执行情况  void Fifo(void); //先进先出算法  void Lru(void); //最近最旧未用算法  void Clock(void); //时钟(二次机会）置换算法  void Eclock(void); //增强二次机会置换算法  void Lfu(void); //最不经常使用置换算法  void Mfu(void); //最经常使用置换算法  private:  int\* ReferencePage; //存放要访问到的页号  int\* EliminatePage; //存放淘汰页号  int\* PageFrames; //存放当前正在实存中的页号  int PageNumber; //访问页数  int FrameNumber; //实存帧数  int FaultNumber; //失败页数  int\* ReferenceBit;  int\* ModifyBit;  int\* count;  int\* op;  };  head = vm.h  srcs = vm.cc  objs = vm.o  opts = -w -g -c  all: vm  vm: $(objs)  g++ $(objs) -o vm  vm.o: $(head) $(srcs)  g++ $(opts) $(srcs)  clean:  rm vm \*.o | | | |
| 结论分析与体会：  我加深了对于两种置换算法：LRU算法和FIFO 算法的理解，能对两种算法给定任意序列不同的页面引用串和任意帧实内存块数的组合测试，显示页置换的过程。  我也学会了采用C++语言用Replace类描述了置换算法及其属性。  我加深了对于存储管理的了解，掌握虚拟存储器的实现原理；观察和了解重要的页面置换算法和置换过程。练习模拟算法的编程技巧，也锻炼了分析试验数据的能力。 | | | |