Министерство науки и высшего образования Российской Федерации

Новосибирский Государственный технический университет

Кафедра автоматизированных систем управления



**Отчет по лабораторной работе № 1**

**по дисциплине «Технологии и методы программирования»**

**«**Моделирование системы управления виртуальной памятью**»**

Выполнили

студенты группы АП-026

Давыдов М.С.

Косточкин М.

г. Новосибирск

2021 г

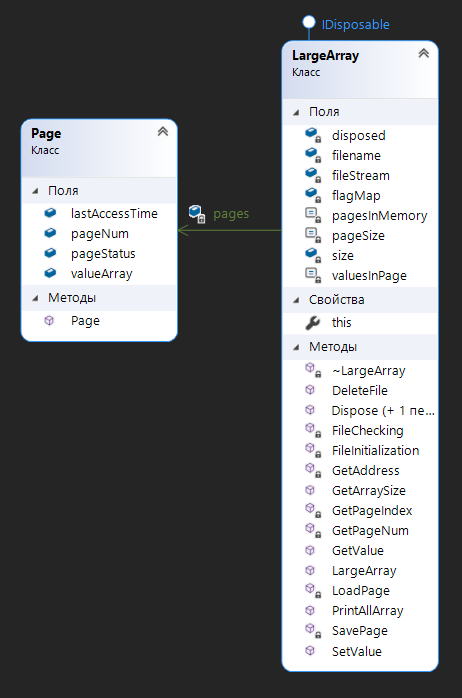
**1. ЗАДАНИЕ**

Реализовать класс для управления виртуальной памятью. Объект этого класса моделирует массив целого типа очень большой размерности (>10000 элементов).

При создании объекта инициализируется файл подкачки (двоичный файл прямого доступа). Файл содержит сигнатуру - два байта, представляющие символы ‘ВМ’, и блоки (страницы) размерности 512 байт, Каждая страница состоит из элементов, соответствующих ячейкам моделируемого массива, и содержащих байты значений целого типа. Каждой странице предшествует битовая карта (массив байтов), в которой каждый бит соответствует ячейке моделируемого массива, находящейся на страницеПри создании объекта инициализируется файл подкачки (двоичный файл прямого доступа). Файл содержит сигнатуру - два байта, представляющие символы ‘ВМ’, и блоки (страницы) размерности 512 байт, Каждая страница состоит из элементов, соответствующих ячейкам моделируемого массива, и содержащих байты значений целого типа. Каждой странице предшествует битовая карта (массив байтов), в которой каждый бит соответствует ячейке моделируемого массива, находящейся на странице. Значение бита 0 означает, что в эту ячейку ничего не записано.

Количество страниц определяется путём выравнивания суммарного объема памяти моделируемого массива на границу страницы.

**2. ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ СТРУКТУР**



**3. ИСХОДНЫЙ КОД КЛАССА И ТЕСТИРУЮЩЕЙ ПРОГРАММЫ**

|  |
| --- |
| Класс Page |
| using System;  namespace ProgrammingLab1Sharp  {  class Page  {  public bool pageStatus; //false страница не модифицировалась, true если была запись  public int pageNum;  public DateTime lastAccessTime;  public int[] valueArray;  public Page()  {  pageStatus = false;  pageNum = -1;  lastAccessTime = DateTime.Now;  valueArray = new int[128];  }  }  } |
| Класс LargeArray |
| using System;  using System.IO;  using System.Text;  namespace ProgrammingLab1Sharp  {  public enum FileOpenMode  {  Open = 0,  Create = 1  }    class LargeArray : IDisposable  {  const int pageSize = 512; //Размер страницы в байтах  const int valuesInPage = 128;  const int pagesInMemory = 8; //Количество страниц в оперативной памяти  private byte[][] flagMap; //Битовая карта  private Page[] pages;  private readonly string filename;  private readonly long size;  private FileStream fileStream;  public LargeArray(long \_size, string \_filename, FileOpenMode fileOpenMode = FileOpenMode.Create)  {  if (\_size < 0)  {  throw new Exception("Отрицательный размер массива!");  }  if (\_size > 250000000)  {  throw new Exception("Для хранения массива такого размера требуется слишком много памяти!");  }  filename = \_filename;  size = \_size;  pages = new Page[pagesInMemory];  flagMap = new byte[pagesInMemory][];  for (int i = 0; i < pagesInMemory; i++)  {  flagMap[i] = new byte[pageSize / 4];  }  switch (fileOpenMode)  {  case FileOpenMode.Open:  {  FileChecking();  int activePages = (int)Math.Ceiling(size / (double)valuesInPage);  if (activePages > pagesInMemory)  {  activePages = pagesInMemory;  }  for (int i = 0; i < activePages; i++)  {  LoadPage(i, i);  }  for (int i = activePages; i < pagesInMemory; i++)  {  pages[i] = new Page();  pages[i].pageNum = i;  }  }  break;  case FileOpenMode.Create:  {  FileInitialization();  for (int i = 0; i < pagesInMemory; i++)  {  pages[i] = new Page();  pages[i].pageNum = i;  }  for (int i = 0; i < pagesInMemory; i++)  {  for (int j = 0; j < pageSize / 4; j++)  {  flagMap[i][j] = 0;  }  }  }  break;  default:  throw new Exception("Error fileOpenMode enum");  }  }  public int this[long index]  {  get  {  if (index < 0)  {  throw new Exception("Отрицательный индекс");  }  if (index >= size)  {  throw new Exception("Индекс выходит за пределы массива!");  }  int arrValue = 0;  int result = GetValue(index, ref arrValue);  if (result == 1)  {  return arrValue;  }  else  {  throw new Exception($"Нет доступа к элементу индекса {index}");  }  }  set  {  int result = SetValue(index, value);  if (result != 1)  {  throw new Exception($"Не возможно записать значение {value} в ячейку индекса {index}.");  }  }  }  public void PrintAllArray()  {  for (long i = 0; i < size; i++)  {  int value = -1;  int result = GetValue(i, ref value);  if (result == 1)  {  Console.WriteLine($"Индекс: {i}, значение: {value}");  }  }  }  private ref int GetAddress(long index)  {  int pageNum = GetPageNum(index);  int pageIndex = GetPageIndex(pageNum);  if (pageIndex == -1)  {  int oldestPage = 0;  for (int i = 0; i < pagesInMemory; i++)  {  if (pages[oldestPage].lastAccessTime < pages[i].lastAccessTime)  {  oldestPage = i;  }  }  pageIndex = oldestPage;  if (pages[pageIndex].pageStatus)  {  SavePage(pageIndex);  }  LoadPage(pageNum, pageIndex);  pages[pageIndex].lastAccessTime = DateTime.Now;  }  int relativeAddress = (int)index % valuesInPage;  ref int absoluteAddress = ref pages[pageIndex].valueArray[relativeAddress];  return ref absoluteAddress;  }  public int GetValue(long index, ref int result)  {  try  {  ref int absoluteAddress = ref GetAddress(index);  int pageNum = GetPageNum(index);  int pageIndex = GetPageIndex(pageNum);  if (flagMap[pageIndex][index % valuesInPage] == 1)  {  result = absoluteAddress;  return 1;  }  else if (flagMap[pageIndex][index % valuesInPage] == 0)  {  return 0;  }  else  {  throw new Exception("Неожиданное значение в битовой карте (не 00 и не 01)");  }  }  catch (Exception)  {  return 0;  }  }  public int SetValue(long index, int value)  {  try  {  ref int absoluteAddress = ref GetAddress(index);  absoluteAddress = value;  int pageNum = GetPageNum(index);  int pageIndex = GetPageIndex(pageNum);  flagMap[pageIndex][index % valuesInPage] = 1;  pages[pageIndex].pageStatus = true;  return 1;  }  catch (Exception)  {  return 0;  }  }  public long GetArraySize()  {  return size;  }  private int GetPageIndex(int pageNum)  {  for (int i = 0; i < pagesInMemory; i++)  {  if (pages[i].pageNum == pageNum)  {  return i;  }  }  return -1;  }  private int GetPageNum(long index)  {  return (int)index / (pageSize / 4);  }  private void SavePage(int pageIndex)  {  int shift = 2 + pages[pageIndex].pageNum \* (valuesInPage \* sizeof(byte) + valuesInPage \* sizeof(int));  byte[] flags = new byte[valuesInPage];  int[] values = new int[valuesInPage];  for (int i = 0; i < valuesInPage; i++)  {  flags[i] = flagMap[pageIndex][i];  }  for (int i = 0; i < valuesInPage; i++)  {  values[i] = pages[pageIndex].valueArray[i];  }  using (fileStream = new FileStream(filename, FileMode.Open, FileAccess.ReadWrite))  {  using (BinaryWriter bw = new BinaryWriter(fileStream, Encoding.Default))  {  bw.Seek(shift, SeekOrigin.Begin);  bw.Write(flags);  for (int i = 0; i < valuesInPage; i++)  {  bw.Write(values[i]);  }  }  }  }  private void LoadPage(int pageNum, int pageIndex)  {  byte[] flags = new byte[valuesInPage];  int[] values = new int[valuesInPage];  using (fileStream = new FileStream(filename, FileMode.Open, FileAccess.ReadWrite))  {  using (BinaryReader br = new BinaryReader(fileStream, Encoding.Default))  {  long shift = 2 + pageNum \* (valuesInPage \* sizeof(byte) + valuesInPage \* sizeof(int));  if (shift > int.MaxValue) //Если сдвиг по байтам превышает размер int, функция ReadBytes требует нескольких вызовов для полной обработки сдвига  {  do  {  int partOfShift;  if (shift > int.MaxValue)  {  partOfShift = int.MaxValue;  shift -= int.MaxValue;  br.ReadBytes(partOfShift);  }  else if (shift >= 0)  {  partOfShift = (int)shift;  shift = -1;  br.ReadBytes(partOfShift);  }  } while (shift > 0);  }  else  {  br.ReadBytes((int)shift);  }  flags = br.ReadBytes(valuesInPage);  for (int i = 0; i < valuesInPage; i++)  {  values[i] = br.ReadInt32();  }  }  }  Page page = new Page();  page.lastAccessTime = DateTime.Now;  page.pageNum = pageNum;  page.pageStatus = false;  for (int i = 0; i < valuesInPage; i++)  {  page.valueArray[i] = values[i];  }  pages[pageIndex] = page;  for (int i = 0; i < valuesInPage; i++)  {  flagMap[pageIndex][i] = flags[i];  }  }  private void FileChecking()  {  using (fileStream = new FileStream(filename, FileMode.Open, FileAccess.Read))  {  using (BinaryReader br = new BinaryReader(fileStream, Encoding.Default))  {  byte[] signature;  signature = br.ReadBytes(2);  if (signature[0] != (byte)'B' || signature[1] != (byte)'M')  {  throw new Exception("Invalid signature");  }  long count = size;  while (count > 0)  {  byte[] flags = br.ReadBytes(valuesInPage);  for (int i = 0; i < valuesInPage; i++)  {  if (!(flags[i] == 1 || flags[i] == 0))  {  throw new Exception("Неожиданное значение в битовой карте (не 00 и не 01)");  }  }  br.ReadBytes(valuesInPage \* sizeof(int));  count -= valuesInPage;  }  }  }  }  private void FileInitialization()  {  using (fileStream = new FileStream(filename, FileMode.Create, FileAccess.Write))  {  using (BinaryWriter bw = new BinaryWriter(fileStream, Encoding.Default))  {  bw.Write((byte)'B');  bw.Write((byte)'M');  int pagesInFile = (int)Math.Round((double)(size / valuesInPage), 0, MidpointRounding.AwayFromZero) + 1;  long fileSize = pagesInFile \* valuesInPage \* (sizeof(int) + sizeof(byte));  for (long i = 0; i < fileSize; i++)  {  bw.Write((byte)0);  }  }  }  }  private bool disposed = false;  public void Dispose()  {  Dispose(true);  GC.SuppressFinalize(this);  }  protected virtual void Dispose(bool disposing)  {  if (disposed)  {  return;  }  if (disposing)  {  int activePages = (int)Math.Ceiling(size / (double)valuesInPage);  if (activePages > pagesInMemory)  {  activePages = pagesInMemory;  }  for (int i = 0; i < activePages; i++)  {  if (pages[i].pageNum >= -1)  {  SavePage(i);  }  }  }  disposed = true;  }  public static int DeleteFile(string \_filename)  {  string path = \_filename;  FileInfo fileInf = new FileInfo(path);  if (fileInf.Exists)  {  fileInf.Delete();  return 1;  }  else  {  return 0;  }  }  ~LargeArray()  {  Dispose(false);  }  }  } |
| Тестирующая программа |
| using System;  namespace ProgrammingLab1Sharp  {  class Program  {  static void Main(string[] args)  {  Random rand = new Random();  int ArraySize = 10001;  int ValueSize = 50;  Console.Write("Введите название файла, в который будет записываться массив: ");  try  {  string filename = Console.ReadLine();  LargeArray largeArray = new LargeArray(ArraySize, filename);  try  {  for (int i = 0; i < ValueSize; i++)  {  int value = rand.Next(1000);  int index = rand.Next(0, ArraySize - 1);  Console.WriteLine($"Значение до записи: {value}.");  largeArray[index] = value;  Console.WriteLine($"\tЗначение после записи: {largeArray[index]}, на {index} индексе.");  Console.WriteLine();  }  }  catch (Exception e)  {  Console.WriteLine($"Произошло исключение: {e.Message}");  }  finally  {  largeArray.Dispose();  }  }  catch (Exception e)  {  Console.WriteLine($"Произошло исключение!: {e.Message}");  }  }  }  } |

**4. ФРАГМЕНТ ДАМПА ФАЙЛА**

