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:

студент группы АВТ-214

Крипаков Леонид Дмитриевич

г. Новосибирск 2024

**Задание**

Реализовать класс для управления виртуальной памятью. Объект этого класса моделирует массив целого типа очень большой размерности (>10000 элементов). При создании объекта инициализируется (заполняется 0) файл подкачки (двоичный файл прямого доступа). Файл содержит сигнатуру - два байта, представляющие символы ‘VМ’, 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 в файле определяется путём выравнивания суммарного объема памяти в байтах моделируемого массива на границу страницы.

Тестирующая программа (консольное приложение):

- вводит несколько (2 или 3) значений, попадающих на разные страницы, чтобы обеспечить

перемещение страниц;

- выводит на экран значения элементов массива до записи в массив и после чтения из

массива.

- при завершении не уничтожает файл виртуального массива.

**Описание всех структур, используемых в системе управления виртуальной памятью**

**1. Таблица страниц:**

* Содержит записи для каждой страницы виртуального адресного пространства.
* Запись страницы содержит:
  + **Физический адрес:** адрес страницы в физической памяти.
  + **Флаги:** информация о состоянии страницы (доступна ли она, модифицирована ли она и т. д.).

**2. Таблица дескрипторов сегментов:**

* Содержит дескрипторы для каждого сегмента виртуального адресного пространства.
* Дескриптор сегмента содержит:
  + **Базовый адрес:** виртуальный адрес начала сегмента.
  + **Предельный адрес:** виртуальный адрес конца сегмента.
  + **Права доступа:** информация о том, кто может читать, писать и выполнять код в сегменте.

**3. Блок управления памятью (MMU):**

* Аппаратное устройство, которое выполняет трансляцию виртуальных адресов в физические адреса.
* Использует таблицу страниц для преобразования виртуальных адресов в физические адреса.

**4. Файл подкачки:**

* Область на диске, которая используется для хранения данных, которые не помещаются в физическую память.
* Когда странице требуется доступ, но она находится в файле подкачки, она загружается в физическую память.

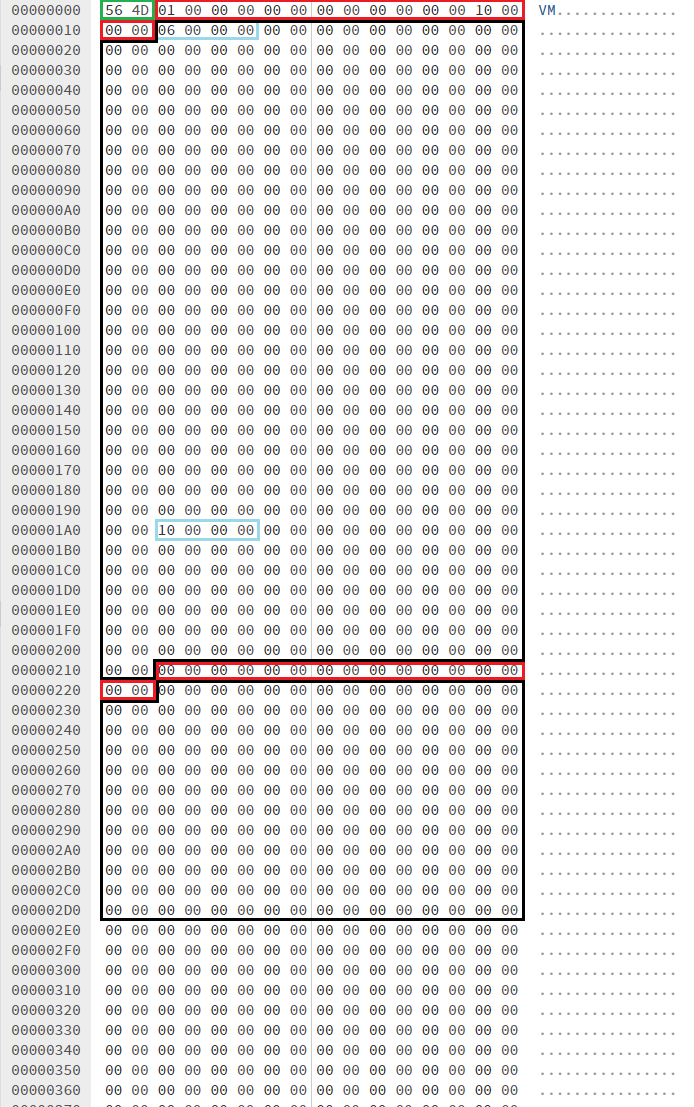
**5. Алгоритмы замены страниц:**

* Определяют, какие страницы следует выгрузить из физической памяти, когда требуется место для новой страницы.

**Исходный код класса и тестирующей программы**

|  |
| --- |
| VirtualArray.cs |
| using System;  using System.IO;  using System.Collections.Generic;  using System.Linq;  using System.Text;  public class VirtualArray  {  // Размер виртуального массива  private readonly uint size;  // Список страниц в памяти  private readonly List<Page> pages = new();  // Путь к файлу, используемому для хранения данных  private readonly string path;  // Конструктор класса VirtualArray.  // size: размер виртуального массива.  // path: путь к файлу, по умолчанию "file.bin".  public VirtualArray(uint size, string path = "file.bin")  {  this.size = size;  this.path = path;  // Открытие или создание бинарного файла.  FileStream stream = File.Open(path, FileMode.OpenOrCreate);  // Запись метки 'VM' в начало файла  stream.Write(Encoding.ASCII.GetBytes("VM"));  // Инициализация пустых страниц в файле.  while (size != 0)  {  size -= size >= 128 ? 128 : size;  byte[] bitmap = new byte[16];  stream.Write(bitmap);  int[] content = new int[128];  foreach (int value in content) stream.Write(BitConverter.GetBytes(value));  }  // Установка указателя на начало данных страниц.  stream.Position = 2;  // Чтение страниц из файла в память.  while (pages.Count != 3)  {  byte[] bitmap = new byte[16];  stream.Read(bitmap);  List<int> content = new();  while (content.Count != 128)  {  byte[] value = new byte[4];  try  {  stream.Read(value);  }  catch  {  break;  }  content.Add(BitConverter.ToInt32(value));  }  // Создание объекта страницы и добавление в список страниц.  Page page = new() { Number = (ushort)pages.Count, State = 0, DateOfApplication = DateTime.Now.ToString(), Bitmap = bitmap, Content = content.ToArray() };  pages.Add(page);  }  // Закрытие бинарного файла.  stream.Close();  }  // Метод для загрузки страницы, содержащей указанный элемент.  // elementIndex: индекс элемента.  // Возвращает номер загруженной страницы.  public int LoadPageOfElement(ushort elementIndex)  {  // Проверка на корректность индекса элемента.  if (elementIndex >= size) throw new Exception("Индекс элемента вне допустимых границ массива!");  // Определение номера страницы, содержащей элемент.  ushort pageNumber = (ushort)(elementIndex / 128);  // Поиск страницы в памяти.  List<Page> pageList = (from p in pages  where p.Number == pageNumber  select p).ToList();  // Проверка наличия страницы в памяти и ее состояния.  if (pageList.Any())  {  if (pageList.First().State == 1)  {  // Если страница изменена, записываем ее обратно в файл.  FileStream stream = File.Open(path, FileMode.Open);  stream.Position = 2 + pageList.First().Number \* 528;  stream.Write(pageList.First().Bitmap);  foreach (int value in pageList.First().Content) stream.Write(BitConverter.GetBytes(value));  stream.Close();  // Сбрасываем флаг изменения страницы.  Page page = pages.Find(x => x.Number == pageList.First().Number);  page.State = 0;  pages[pages.IndexOf(pages.Find(x => x.Number == pageList.First().Number))] = page;  }  // Возврат индекса страницы  return pages.IndexOf(pages.Find(x => x.Number == pageList.First().Number));  }  // Если страница не найдена в памяти, загружаем ее из файла.  if (!pageList.Any())  {  // Находим самую старую страницу в памяти.  List<Page> oldestPageList = (from p in pages  orderby p.DateOfApplication  select p).ToList();  FileStream stream = File.Open(path, FileMode.Open);  // Выгружаем старую страницу, если она была изменена.  if (oldestPageList.First().State == 1)  {  stream.Position = 2 + oldestPageList.First().Number \* 528;  stream.Write(oldestPageList.First().Bitmap);  foreach (int value in oldestPageList.First().Content) stream.Write(BitConverter.GetBytes(value));  }  // Удаляем самую старую страницу из памяти.  pages.Remove(oldestPageList.First());  // Загружаем требуемую страницу из файла.  stream.Position = 2 + pageNumber \* 528;  byte[] bitmap = new byte[16];  stream.Read(bitmap);  List<int> content = new();  while (content.Count != 128)  {  byte[] value = new byte[4];  try  {  stream.Read(value);  }  catch  {  break;  }  content.Add(BitConverter.ToInt32(value));  }  stream.Close();  // Создаем объект страницы и добавляем его в память.  Page page = new() { Number = pageNumber, State = 0, DateOfApplication = DateTime.Now.ToString(), Bitmap = bitmap, Content = content.ToArray() };  pages.Add(page);  // Возврат индекса страницы  return pages.IndexOf(page);  }  // Возвращаем номер загруженной страницы.  return -1;  }  // Метод для чтения значения элемента по индексу.  // elementIndex: индекс элемента.  // Возвращает значение элемента.  public int ReadValueOfElement(ushort elementIndex)  {  // Проверка на корректность индекса  if (elementIndex >= size) throw new Exception("Element index was outside the bounds of array!");  // Подгрузка требуемой страницы и получение индекса в памяти  int index = LoadPageOfElement(elementIndex);  // Возврат значения элемента  return pages[index].Content[elementIndex % 128];  }  // Метод для записи значения элемента по индексу.  // elementIndex: индекс элемента.  // elementValue: значение элемента.  public void WriteValueOfElement(ushort elementIndex, int elementValue)  {  // Проверка на корректность индекса элемента.  if (elementIndex >= size) throw new Exception("Индекс элемента вне допустимых границ массива!");  // Подгрузка требуемой страницы и получение индекса в памяти  int index = LoadPageOfElement(elementIndex);  // Обновление значений страницы.  Page page = pages[index];  page.Bitmap[elementIndex % 128 / 8] = (byte)(page.Bitmap[elementIndex % 128 / 8] | (byte)Math.Pow(2, elementIndex % 128 % 8));  page.Content[elementIndex % 128] = elementValue;  page.State = 1;  page.DateOfApplication = DateTime.Now.ToString();  pages[index] = page;  }  } |
| Program.cs |
| using System;  class Program  {  static void Main(string[] args)  {  // Создание виртуального массива.  VirtualArray virtualArray = new(10000);  // Запись элементов.  virtualArray.WriteValueOfElement(0, 6);  virtualArray.WriteValueOfElement(100, 16);  virtualArray.WriteValueOfElement(5000, 36);  // Вывод результатов.  Console.WriteLine("Значение нулевого элемента: " + virtualArray.ReadValueOfElement(0));  Console.WriteLine("Значение сотого элемента: " + virtualArray.ReadValueOfElement(100));  Console.WriteLine("Значение пятитысячного элемента: " + virtualArray.ReadValueOfElement(5000));  }  } |
| Page.cs |
| public struct Page  {  // Номер страницы.  public ushort Number;  // Флаг модификации(0 - не модифицирована, 1 - модифицирована).  public byte State;  // Дата последнего изменения.  public string DateOfApplication;  // Битовая карта, где каждый бит представляет элемент.  public byte[] Bitmap;  // Массив значений элементов.  public int[] Content;  } |

**Фрагменты дампа файла виртуального массива**



*Рис. 1. Начало дампа*

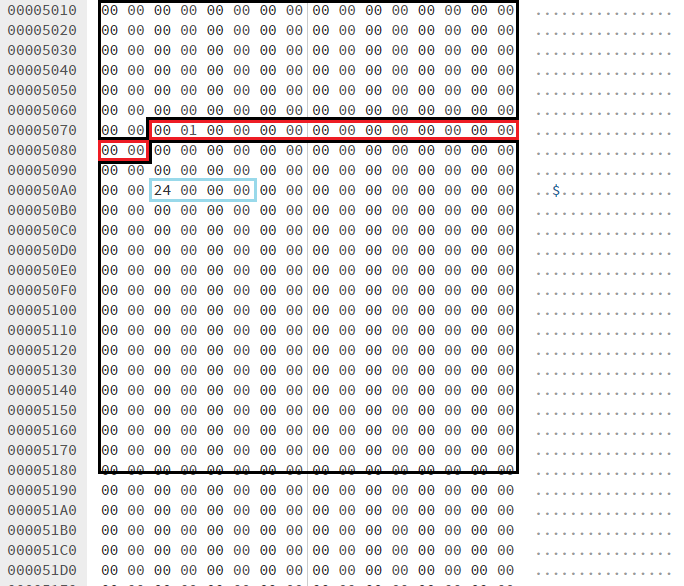
2 байта – сигнатура

16 байт – битовая карта

4 байта - 610 = 616 – нулевой элемент

4 байта – 1610 = 1016 – сотый элемент

512 байт – размер страницы

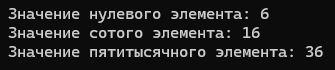


*Рис. 2. Фрагмент дампа. 40 страница файла*

16 байт – битовая карта

3610 = 2416 – пятитысячный элемент

Расчёт адреса пятитысячного элемента:



*Рис. 3. Вывод в консоли*