# Министерство Образования Республики Беларусь

Учреждение образования

«Гомельский государственный технический университет

имени П.О.Сухого»

Кафедра: «Информатика»

**Лабораторная работа №8**

по дисциплине: **«Операционные системы и среды»**

**Реализация файловой системы**

Выполнило студент

группы ИП-31

*Соловьев Д.С.*

Проверил преподаватель

*Самовендюк Н.В.*

Гомель 2018

**Цель работы:** разработать модель файловой системы

**Ход работы:**

Разработать приложение, создающее виртуальный файл и позволяющее

- форматировать виртуальный файл с возможностью задания размера кластера;

- создавать каталоги в виртуальном файле;

- производить учёт свободного пространства;

- реализовывать поиск файлов и директорий;

- сохранять в виртуальный файл файлы с жёсткого диска;

- удалять файлы из виртуального файла;

- записывать на жёсткий диск файлы из виртуального файла;

- создавать в виртуальном файле текстовые файлы;

- предоставлять возможность редактировать текстовые файлы внутри виртуального файла.

Файловую систему внутри виртуального файла выбрать согласно варианта.

Таблица 1 - Варианты заданий

|  |  |  |  |
| --- | --- | --- | --- |
| **Вариант** | **Условие задачи** | **Учёт свободных блоков** | **Поиск файлов и папок** |
| 17 | Индексно-последовательная файловая система. Связанный список. | Битовый вектор | Бинарный поиск |

**Листинг программы**

FileSystemForm.cpp:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.IO;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace lab8  {  public partial class FileSystemForm : Form  {  SystemAddressSpace SystemAddressSpace { get; }  public FileSystemForm()  {  InitializeComponent();  var size = 1024;  var blockSize = 8;  SystemAddressSpace = new SystemAddressSpace(size, blockSize);  FillSystemInfo();  FillTree(filesTree.TopNode, 0);  }  void FillTree(TreeNode treeNode, int index)  {  var systemNode = SystemAddressSpace.GetAddressSpaceBlock(index) as SystemNode;  TreeNode newNode = null;  if (treeNode == null)  newNode = filesTree.Nodes.Add(systemNode.Name);  else  newNode = treeNode.Nodes.Add(systemNode.Name);  newNode.Tag = systemNode;  if (systemNode.Type == "Dictionary")  {  var dict = systemNode as SystemNodeDirectory;  foreach (var elem in dict.innerFilesAddresses)  FillTree(newNode, elem);  }  }  void FillSystemInfo()  {  systemClasterSize.Text = SystemAddressSpace.BlockSize.ToString();  systemAllSpace.Text = SystemAddressSpace.Size.ToString();  systemFreeSpace.Text = SystemAddressSpace.FreeSpace().ToString();  }  private void editButton\_Click(object sender, EventArgs e)  {  if ((filesTree.SelectedNode.Tag as SystemNode).Type != "Directory")  {  var prevNewNodeIndex = (filesTree.SelectedNode.Parent.Tag as SystemNode).Index;  var newNodeIndex = (filesTree.SelectedNode.Tag as SystemNode).Index;  (SystemAddressSpace.GetAddressSpaceBlock(prevNewNodeIndex) as SystemNodeDirectory).RemoveNode(newNodeIndex);  if (radioButtonDirectory.Checked)  (SystemAddressSpace.GetAddressSpaceBlock(prevNewNodeIndex) as SystemNodeDirectory)  .AddNode(new SystemNodeDirectory  (  SystemAddressSpace,  newNodeIndex,  newNodeName.Text  ), newNodeIndex);  else  (SystemAddressSpace.GetAddressSpaceBlock(prevNewNodeIndex) as SystemNodeDirectory)  .AddNode(new SystemNodeFile  (  SystemAddressSpace,  newNodeIndex,  newNodeName.Text,  fileText.Text  ), newNodeIndex);  SystemNode newNode = SystemAddressSpace.GetAddressSpaceBlock(newNodeIndex) as SystemNode;  filesTree.SelectedNode.Tag = newNode;  filesTree.SelectedNode.Text = newNode.Name;  FillSystemInfo();  Refresh();  }  else  MessageBox.Show("This is directory");  }  private void addNewButton\_Click(object sender, EventArgs e)  {  if (newNodeName.Text.Length != 0)  {  var prevNewNodeIndex = 0;  var newNodeIndex = SystemAddressSpace.GetWasteBlockAdress();  if (filesTree.SelectedNode == null)  filesTree.SelectedNode = filesTree.Nodes[0];  if (filesTree.SelectedNode.Parent != null)  prevNewNodeIndex = (filesTree.SelectedNode.Parent.Tag as SystemNode).Index;  if ((filesTree.SelectedNode.Tag as SystemNode).Type == "Directory")  prevNewNodeIndex = (filesTree.SelectedNode.Tag as SystemNode).Index;  else  prevNewNodeIndex = (filesTree.SelectedNode.Parent.Tag as SystemNode).Index;    if (radioButtonDirectory.Checked)  (SystemAddressSpace.GetAddressSpaceBlock(prevNewNodeIndex) as SystemNodeDirectory)  .AddNode(new SystemNodeDirectory  (  SystemAddressSpace,  newNodeIndex,  newNodeName.Text  ), newNodeIndex);  else  (SystemAddressSpace.GetAddressSpaceBlock(prevNewNodeIndex) as SystemNodeDirectory)  .AddNode(new SystemNodeFile  (  SystemAddressSpace,  newNodeIndex,  newNodeName.Text,  fileText.Text  ), newNodeIndex);  SystemNode newNode = SystemAddressSpace.GetAddressSpaceBlock(newNodeIndex) as SystemNode;  if ((filesTree.SelectedNode.Tag as SystemNode).Type == "Directory")  {  var node = filesTree.SelectedNode.Nodes.Add(newNode.Name);  node.Tag = newNode;  }  else  {  var node = filesTree.SelectedNode.Parent.Nodes.Add(newNode.Name);  node.Tag = newNode;  }  FillSystemInfo();  Refresh();  }  else  MessageBox.Show("Enter name of the file or the directory");  }  private void deleteButton\_Click(object sender, EventArgs e)  {  var deletingNodeIndex = (filesTree.SelectedNode.Tag as SystemNode).Index;  SystemAddressSpace.Remove(deletingNodeIndex);  filesTree.SelectedNode.Remove();  FillSystemInfo();  Refresh();  }  private void filesTree\_AfterSelect(object sender, TreeViewEventArgs e)  {  var treeNode = filesTree.SelectedNode?.Tag as SystemNode;  if (treeNode != null)  {  if (treeNode.Type == "Directory")  {  var systemNodeDirectory = treeNode as SystemNodeDirectory;  nodeName.Text = systemNodeDirectory.Name;  nodeType.Text = systemNodeDirectory.Type;  nodeAddress.Text = systemNodeDirectory.Index.ToString();  nodeSize.Text = systemNodeDirectory.GetSize().ToString();  }  else  {  newNodeName.Text = treeNode.Name;  var systemNodeFile = treeNode as SystemNodeFile;  nodeName.Text = systemNodeFile.Name;  nodeType.Text = systemNodeFile.Type;  nodeAddress.Text = systemNodeFile.Index.ToString();  nodeSize.Text = systemNodeFile.GetSize().ToString();  fileText.Text = systemNodeFile.ToString();  }  }  }  private void searchButton\_Click(object sender, EventArgs e)  {  var address = SystemAddressSpace.FindByBinarySearch(searchNode.Text);  if (address < 0)  return;  var node = FindNode(filesTree.Nodes[0], address);  filesTree.SelectedNode = node;  filesTree\_AfterSelect(sender, null);  }  TreeNode FindNode(TreeNode treeNode, int address)  {  if ((treeNode.Tag as SystemNode).Index == address)  return treeNode;  foreach (TreeNode node in treeNode.Nodes)  {  var tmp = FindNode(node, address);  if (tmp != null)  return tmp;  }  return null;  }  private void saveToHardDisk\_Click(object sender, EventArgs e)  {  if ((filesTree.SelectedNode.Tag as SystemNode).Type == "Directory")  {  return;  }  saveFileDialog1.Filter = "txt files (\*.txt)|\*.txt";  saveFileDialog1.FileName = (filesTree.SelectedNode.Tag as SystemNode).Name;  saveFileDialog1.RestoreDirectory = true;  if (saveFileDialog1.ShowDialog() == DialogResult.OK)  {  var path = saveFileDialog1.FileName;  var text = (filesTree.SelectedNode.Tag as SystemNodeFile).ToString();  File.WriteAllText(path, text);  }  }  private void loadFromDisk\_Click(object sender, EventArgs e)  {  openFileDialog1.Filter = "Text files(\*.txt)| \*.txt";  openFileDialog1.RestoreDirectory = true;  if (openFileDialog1.ShowDialog() == DialogResult.OK)  {  var path = openFileDialog1.FileName;  var name = openFileDialog1.SafeFileName;  var text = File.ReadAllText(path);  if (filesTree.SelectedNode == null)  filesTree.SelectedNode = filesTree.Nodes[0];  var newNodeIndex = SystemAddressSpace.GetWasteBlockAdress();  int prevNewNodeIndex = 0;  if ((filesTree.SelectedNode.Tag as SystemNode).Type == "Directory")  prevNewNodeIndex = (filesTree.SelectedNode.Tag as SystemNode).Index;  else  prevNewNodeIndex = (filesTree.SelectedNode.Parent.Tag as SystemNode).Index;  (SystemAddressSpace.GetAddressSpaceBlock(prevNewNodeIndex) as SystemNodeDirectory)  .AddNode(new SystemNodeFile  (  SystemAddressSpace,  newNodeIndex,  name,  text  ), newNodeIndex);  SystemNode newNode = SystemAddressSpace.GetAddressSpaceBlock(newNodeIndex) as SystemNode;  if ((filesTree.SelectedNode.Tag as SystemNode).Type == "Directory")  {  var node = filesTree.SelectedNode.Nodes.Add(newNode.Name);  node.Tag = newNode;  }  else  {  var node = filesTree.SelectedNode.Parent.Nodes.Add(newNode.Name);  node.Tag = newNode;  }  }  FillSystemInfo();  Refresh();  }  }  } |

SystemAdressSpace:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace lab8  {  class SystemAddressSpace  {  public SystemAddressSpaceBlock[][] systemNodes;  int[][] bitVectorOfFreeBlocks;  //List<List<int>> hashTable;  List<KeyValuePair<string, SystemAddressSpaceBlock>> binarySearchTable;  public int Size { get; set; }  public int BlockSize { get; set; }  public SystemAddressSpace(int size, int blockSize)  {  Size = size;  BlockSize = blockSize;  var blockMatrixRank = size / (blockSize \* blockSize);  systemNodes = new SystemAddressSpaceBlock[blockMatrixRank][]  .Select(e => new SystemAddressSpaceBlock[blockMatrixRank])  .ToArray();  bitVectorOfFreeBlocks = new int[blockMatrixRank][]  .Select(e => new int[blockMatrixRank].  Select(t => 0)  .ToArray())  .ToArray();  FormBinarySearchTable();  new SystemNodeDirectory(this, 0, "Root");  bitVectorOfFreeBlocks[0][0] = 1;  }  void FormBinarySearchTable()  {  binarySearchTable = new List<KeyValuePair<string, SystemAddressSpaceBlock>>  (systemNodes.Length \* systemNodes[0].Length);  for (var i = 0; i < systemNodes.Length; i++)  {  for (var j = 0; j < systemNodes[i].Length; j++)  {  if (bitVectorOfFreeBlocks[i][j] == 1)  {  if (systemNodes[i][j] is SystemNode node)  binarySearchTable  .Add(new KeyValuePair<string, SystemAddressSpaceBlock>(node.Name, node));  }  }  }  }  public int Insert(SystemAddressSpaceBlock systemAddressSpaceBlock, int index = -1)  {  var address = GetWasteBlockAdress();  var i = address / systemNodes.Length;  var j = address % systemNodes.Length;  if(index >= 0)  {  i = index / systemNodes.Length;  j = index % systemNodes.Length;  }  bitVectorOfFreeBlocks[i][j] = 1;  systemNodes[i][j] = systemAddressSpaceBlock;    FormBinarySearchTable();  systemAddressSpaceBlock.Adress = address;  return address;  }  public int GetWasteBlockAdress()  {  for(var i = 0; i < bitVectorOfFreeBlocks.Length; i++)  for(var j = 0; j < bitVectorOfFreeBlocks[i].Length; j++)  if(bitVectorOfFreeBlocks[i][j] == 0)  return i \* systemNodes.Length + j;  return 1;  }  public void Remove(int address)  {  systemNodes[address / systemNodes.Length][address % systemNodes.Length].Remove();  systemNodes[address / systemNodes.Length][address % systemNodes.Length] = null;  bitVectorOfFreeBlocks[address / BlockSize][address % BlockSize] = 0;  FormBinarySearchTable();  }  public void Remove(SystemAddressSpaceBlock prev, SystemAddressSpaceBlock deletable)  {  if (prev.NextBlock == deletable)  {  prev.NextBlock = deletable.NextBlock;  Remove(deletable.Adress);  }  }  public SystemAddressSpaceBlock GetAddressSpaceBlock(int address)  {  return systemNodes[address / systemNodes.Length][address % systemNodes.Length];  }  public int FreeSpace()  {  var result = bitVectorOfFreeBlocks.Select(e => e.Sum()).ToArray().Sum();  return Size - result \* systemNodes.Length;  }  public int FindByBinarySearch(string name)  {  binarySearchTable.Sort(new KvpSpaceBlockComparer());  return binarySearchTable[binarySearchTable.BinarySearch  (new KeyValuePair<string, SystemAddressSpaceBlock>(name, null),  new KvpSpaceBlockComparer())].Value.Adress;  }  }  class KvpSpaceBlockComparer : IComparer<KeyValuePair<string, SystemAddressSpaceBlock>>  {  public int Compare(KeyValuePair<string, SystemAddressSpaceBlock> x, KeyValuePair<string, SystemAddressSpaceBlock> y)  {  return x.Key.CompareTo(y.Key);  }  }  } |

SystemAdressSpaceBlock:

|  |
| --- |
| namespace lab8  {  abstract class SystemAddressSpaceBlock  {  public SystemAddressSpace SystemAddressSpace { get; set; }  public SystemAddressSpaceBlock NextBlock { get; set; } = null;  public int Adress { get; set; }  public SystemAddressSpaceBlock(SystemAddressSpace systemAddressSpace)  {  SystemAddressSpace = systemAddressSpace;  }  public abstract int GetSize();  public abstract void Remove();  }  } |

SystemNode:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace lab8  {  abstract class SystemNode: SystemAddressSpaceBlock  {  public int Index { get; }  public int IndexNext { get; set; }  public string Name { get; }  public string Type { get; }    public SystemNode(SystemAddressSpace systemAddressSpace, int index, string name, string type): base(systemAddressSpace)  {  Index = index;  Name = name;  Type = type;  IndexNext = -1;  }  public override abstract int GetSize();  public override abstract void Remove();  }  } |

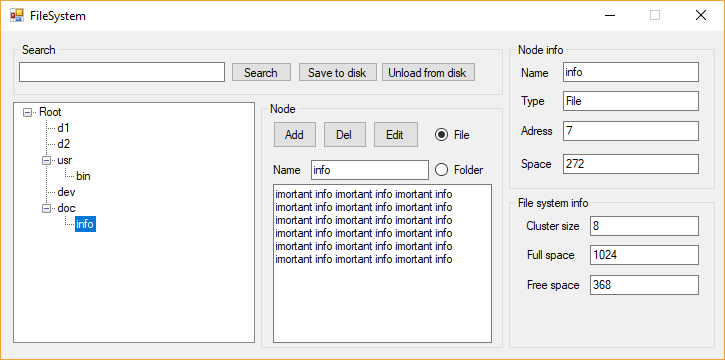
SystemNodeDirectory:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace lab8  {  class SystemNodeDirectory : SystemNode  {  public List<int> innerFilesAddresses = new List<int>();  public SystemNodeDirectory(SystemAddressSpace systemAddressSpace, int index, string name)  : base(systemAddressSpace, index, name, "Directory")  {  systemAddressSpace.Insert(this, index);  }  public void AddNode(SystemAddressSpaceBlock systemAddressSpaceBlock, int index = -1)  {  if(innerFilesAddresses.Count > 1)  (SystemAddressSpace.GetAddressSpaceBlock(innerFilesAddresses.Last()) as SystemNode)  .IndexNext = innerFilesAddresses.Last();  innerFilesAddresses.Add(index);  }  public override void Remove()  {  while (innerFilesAddresses.Count > 0)  RemoveNode(innerFilesAddresses.First());  }  public void RemoveNode(int address)  {  if (innerFilesAddresses.Contains(address))  {  innerFilesAddresses.Remove(address);  SystemAddressSpace.Remove(address);  }  }  public override int GetSize()  {  var result = 0;  foreach (var item in innerFilesAddresses)  {  var block = SystemAddressSpace.GetAddressSpaceBlock(item);  if (block != null)  result += block.GetSize();  }  return result + SystemAddressSpace.BlockSize;  }  }  } |

SystemNodeFile:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace lab8  {  class SystemNodeFile : SystemNode  {  int rootBlockAddress = -1;  public SystemNodeFile(SystemAddressSpace systemAddressSpace, int index, string name, string text)  :base(systemAddressSpace, index, name, "File")  {  systemAddressSpace.Insert(this);  rootBlockAddress = systemAddressSpace.Insert(new SystemFileBlock(systemAddressSpace, (text ?? "")));  }    public override int GetSize()  {  return (SystemAddressSpace.GetAddressSpaceBlock(rootBlockAddress) as SystemFileBlock).GetSize();  }  public override void Remove()  {  if(rootBlockAddress >= 0)  SystemAddressSpace.Remove(rootBlockAddress);  }  public override string ToString()  {  return (SystemAddressSpace.GetAddressSpaceBlock(rootBlockAddress) as SystemFileBlock).ToString();  }  }  } |

**Результат:**



**Вывод:**

В ходе выполнения данной лабораторной работе была разработана модель файловой системы