

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Санкт-Петербургский государственный технологический институт  
(технический университет)»  
СПбГТИ(ТУ)

|  |  |  |
| --- | --- | --- |
| УГНС | 09.00.00 | Информатика и вычислительная техника |
| Направление подготовки | 09.03.03 | Прикладная информатика |
| Направленность (профиль) |  | Прикладная информатика в химии |
| Факультет |  | Информационных технологий и управления |
| Кафедра |  | Систем автоматизированного  проектирования и управления |
| Учебная дисциплина |  | **ОПЕРАЦИОННЫЕ СИСТЕМЫ** |

Курс 2 Группа 485

**Отчет по лабораторной работе № 3**

|  |  |
| --- | --- |
| **Тема:** | **Особенности управления виртуальной памятью.**  **Реализация алгоритмов выталкивания: FIFO и LRU.** |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата) (инициалы, фамилия)

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Макарук Р.В.

(подпись, дата) (инициалы, фамилия)

Отметка о зачете \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись преподавателя)

Санкт-Петербург

2020

**1. Цель работы**

Ознакомиться с организацией основной памяти, исследовать возможности виртуальной памяти и файла подкачки.Разработать алгоритмы выталкивания страниц FIFO и LRU.

**Исходные данные для тестирования алгоритмов замещения страниц**

Операционная система использует алгоритм замещения страниц **FIFO (LRU)** в системе с четырьмя страничными блоками и восьмью страницами. Последовательность обращений к страницам приведена в таблице:

**Таблица 1 - Последовательность обращений**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| обращение | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| страница | **7** | **8** | **9** | **2** | **1** | **0** | **8** | **9** | **2** | **4** | **6** | **8** | **2** | **1** | **8** | **9** |

Программа должна показать:

* сколько страничных прерываний произойдет в системе при условии, что четыре страничных блока изначально заняты страницами: 8 2 9 6;
* сколько страничных прерываний произойдет в системе при увеличении количества страничных блоков на 1 (в данном примере пятым элементом была добавлена цифра '7');

**2 Ход работы**

Код программы:

[файл Program.cs]

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

class Program

{

static readonly string NL = Environment.NewLine;

static readonly int additionalBlock = -1;

static string MakeBlocksString(IEnumerable<int> blocks)

{

string blocksString = "";

foreach (var block in blocks)

if (block != additionalBlock)

blocksString += block + " ";

return blocksString;

}

static IEnumerable<int> AddBlock(IEnumerable<int> blocks, int blockToAdd)

{

if (!blocks.Contains(additionalBlock))

{

var queuedBlocks = new Queue<int>(blocks);

queuedBlocks.Dequeue();

queuedBlocks.Enqueue(blockToAdd);

blocks = queuedBlocks;

}

else

{

var listedBlocks = blocks.ToList();

int indexToInsert = listedBlocks.IndexOf(additionalBlock);

listedBlocks.Remove(additionalBlock);

listedBlocks.Insert(indexToInsert, blockToAdd);

blocks = listedBlocks;

}

return blocks;

}

static int FIFO(IEnumerable<int> blocks, IEnumerable<int> blocksToAdd)

{

Console.WriteLine("Алгоритм FIFO для " + blocks.Count() + " страничных блоков:");

Console.WriteLine(MakeBlocksString(blocks) + '\t' + MakeBlocksString(blocksToAdd));

int pageFaults = 0;

var blocksToAddToOutput = blocksToAdd.ToList();

foreach (var blockToAdd in blocksToAdd)

{

string blocksToOutputInConsole = "";

if (!blocks.Contains(blockToAdd))

{

blocks = AddBlock(blocks, blockToAdd);

blocksToOutputInConsole += " <-p";

++pageFaults;

}

blocksToAddToOutput.Remove(blockToAdd);

blocksToOutputInConsole = blocksToOutputInConsole.Insert(0, MakeBlocksString(blocks));

Console.WriteLine(blocksToOutputInConsole + '\t' + MakeBlocksString(blocksToAddToOutput));

}

return pageFaults;

}

static int LRU(IEnumerable<int> blocks, IEnumerable<int> blocksToAdd)

{

Console.WriteLine("Алгоритм LRU для " + blocks.Count() + " страничных блоков:");

Console.WriteLine(MakeBlocksString(blocks) + '\t' + MakeBlocksString(blocksToAdd));

int pageFaults = 0;

var blocksToAddToOutput = blocksToAdd.ToList();

foreach (var blockToAdd in blocksToAdd)

{

string blocksToOutputInConsole = "";

if (!blocks.Contains(blockToAdd))

{

blocks = AddBlock(blocks, blockToAdd);

blocksToOutputInConsole += " <-p";

++pageFaults;

}

else

{

var existedBlock = blocks.ToList().Find(b => b == blockToAdd);

var defaultValsNum = 0;

if (blocks.Contains(additionalBlock))

defaultValsNum += blocks.Where(b => b == additionalBlock).Count();

var blocksAsList = blocks.ToList();

blocksAsList.Remove(existedBlock);

blocksAsList.Insert(blocks.Count() - (defaultValsNum + 1), existedBlock);

blocks = blocksAsList;

}

blocksToAddToOutput.Remove(blockToAdd);

blocksToOutputInConsole = blocksToOutputInConsole.Insert(0, MakeBlocksString(blocks));

Console.WriteLine(blocksToOutputInConsole + '\t' + MakeBlocksString(blocksToAddToOutput));

}

return pageFaults;

}

static string CompareAlgorithmsEfficiency(int FIFOPageFaults, int LRUPageFaults)

{

if (FIFOPageFaults < LRUPageFaults)

return "FIFO оказался более эффективным алгоритмом.";

else if (LRUPageFaults < FIFOPageFaults)

return "LRU оказался более эффективным алгоритмом.";

return "Алгоритмы одинакого эффективны.";

}

static void Main()

{

var fourBlocks = new List<int> { 8, 2, 9, 6 };

var fiveBlocks = new List<int> { 8, 2, 9, 6, additionalBlock };

var blocksToAdd = new List<int> { 7, 8, 9, 2, 1, 0, 8, 9, 2, 4, 6, 8, 2, 1, 8, 9 };

var FIFOPageFaults = FIFO(fourBlocks, blocksToAdd);

Console.WriteLine("Страничных прерываний: " + FIFOPageFaults + NL);

var LRUPageFaults = LRU(fourBlocks, blocksToAdd);

Console.WriteLine("Страничных прерываний: " + LRUPageFaults + NL);

Console.WriteLine(CompareAlgorithmsEfficiency(FIFOPageFaults, LRUPageFaults) + NL);

FIFOPageFaults = FIFO(fiveBlocks, blocksToAdd);

Console.WriteLine("Страничных прерываний: " + FIFOPageFaults + NL);

LRUPageFaults = LRU(fiveBlocks, blocksToAdd);

Console.WriteLine("Страничных прерываний: " + LRUPageFaults + NL);

Console.WriteLine(CompareAlgorithmsEfficiency(FIFOPageFaults, LRUPageFaults) + NL);

}

}

}

[конец файла Program.cs]

**2.1 Примеры выполнения программы**

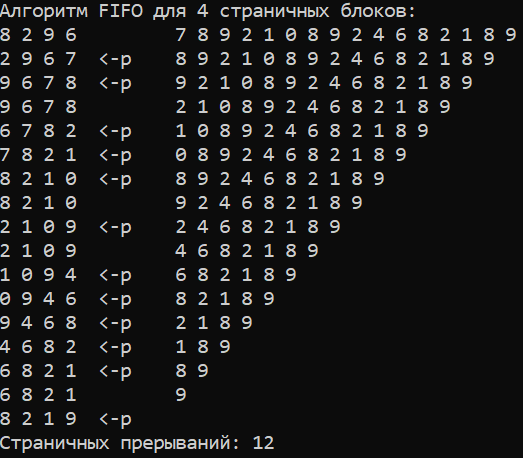


Рисунок 1 – Алгоритм FIFO с 4 страничными блоками

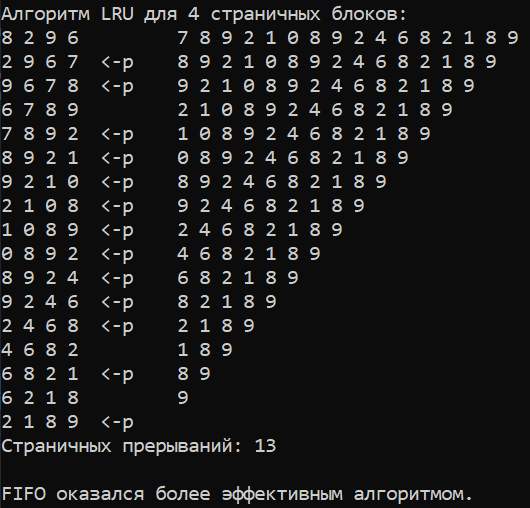


Рисунок 2 – Алгоритм LRU с 4 страничными блоками

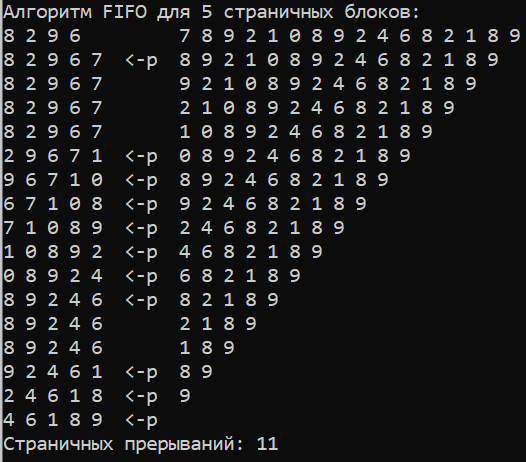


Рисунок 3 – Алгоритм FIFO с 5 страничными блоками

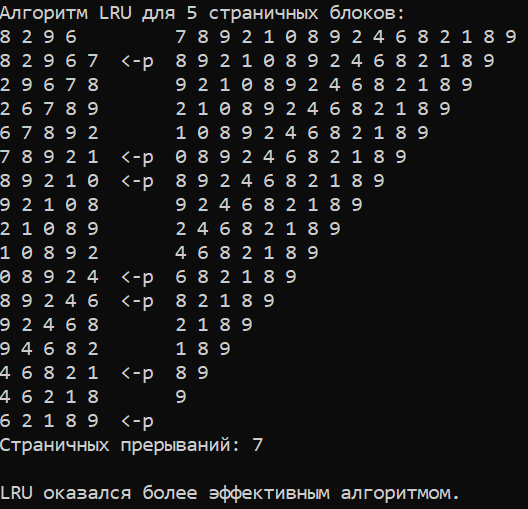


Рисунок 4 – Алгоритм LRU с 5 страничными кадрами

**2.2 Файл подкачки**

При работе системы физической оперативной памяти для всех процессов может не хватить, поэтому OC создаёт файл подкачки. Туда она помещает ненужные в соответствии со стратегией выталкивания страницы памяти, чтобы освободить место для нужных. Позже, если выгруженная в файл подкачки страница вдруг понадобится, она будет оттуда загружена обратно в ОП.

У файла подкачки есть еще одно назначение – в наиболее современных OC при полном выключение компьютера содержимое ОЗУ выгружается на диск в файл подкачки, так как ОЗУ является энергозависимой памятью. При следующей загрузке содержимое файла подкачки переносится обратно в ОЗУ

Исходя из этого, файл подкачки обычно создают с объемом равным объему ОЗУ, то есть если у вас 4 Гб ОЗУ, то и файл подкачки правильно сделать минимум на 4 Гб, а в идеале на 20-30% больше.

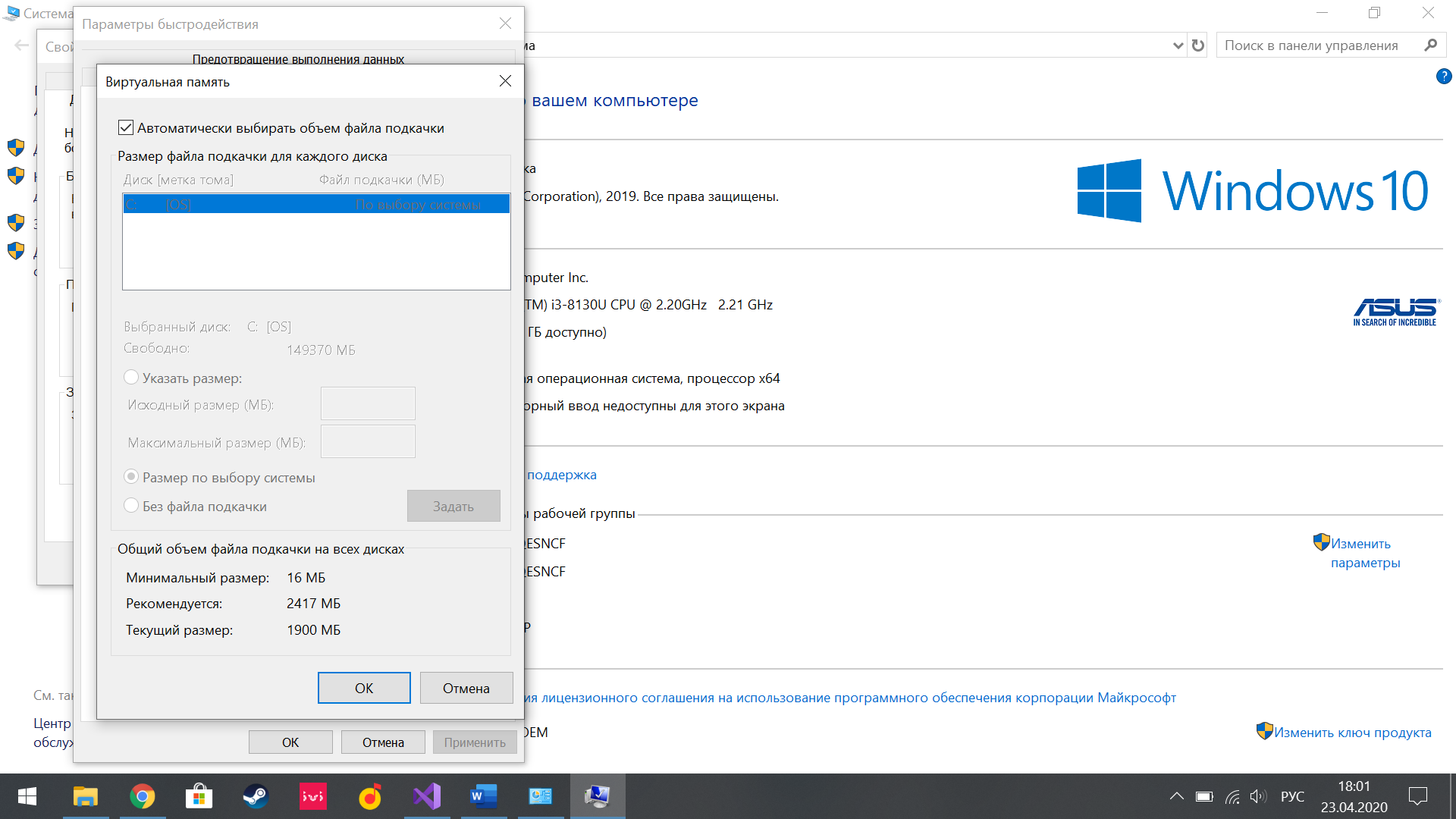
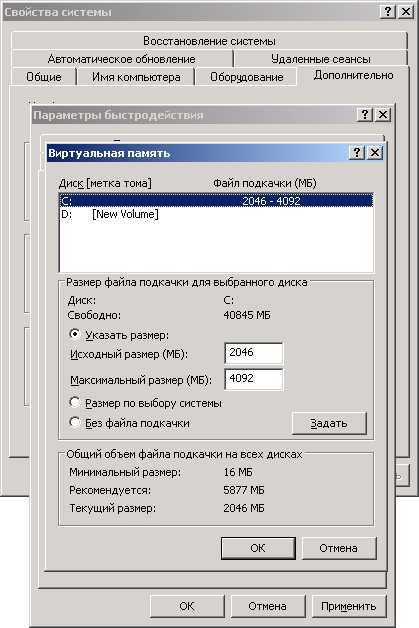


Рисунок 5 – Настройка файла подкачки на Windows 10

Рисунок 6 – Настройка файла подкачки на Windows XP

Размеры файла подкачки можно изменять, убрав галочку с автоматического определения.

**3. Ответы на контрольные вопросы**

**Поясните разные значения термина «свопинг».**

В начале под «свопингом» понимали выгрузку на внешний носитель неактивных процессов, позже с появлением виртуальной памяти это стало выгрузкой лишь неактивных страниц.

**На что влияет размер страницы? Каковы преимущества и недостатки большого размера страницы?**

Чем больше размер страницы, тем меньше будет размер структур данных, обслуживающих преобразование адресов, но тем больше будут потери, связанные с тем, что память можно выделять только постранично.Как следует выбирать размер страницы? Во-первых, нужно учитыватьразмер таблицы страниц, здесь желателен большой размер страницы (страницменьше, соответственно и таблица страниц меньше). С другой стороны, памятьлучше утилизируется с маленьким размером страницы. В среднем половинапоследней страницы процесса пропадает.

**Вывод**

В ходе выполнения лабораторной работы были изучены алгоритмы замещения страниц FIFO и LRU, исследованы возможности работы системы с файлом подкачки, на примере Windows XP и Windows 10. Его функция - выгружать неиспользуемую часть оперативной памяти на жёсткий диск и загружать обратно при необходимости.