

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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Зобнин И.М.

(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;

namespace Lab3

{

class Program

{

static void OutputInConsole(IEnumerable<int> enumerable)

{

foreach (var element in enumerable)

Console.Write(element + " ");

}

static int FIFO(Tuple<List<int>, List<int>> pagesAndRequests, int blocksNum)

{

ColorPrint.PrintWithColor(ConsoleColor.Green, "FIFO");

Console.WriteLine(" with " + blocksNum + " blocks");

var pages = pagesAndRequests.Item1;

var requests = pagesAndRequests.Item2;

var pagesQueue = new Queue<int>(pages);

int pageFaults = 0;

OutputInConsole(pagesQueue);

Console.WriteLine();

foreach (var request in requests)

{

if (pagesQueue.Contains(request))

{

OutputInConsole(pagesQueue);

Console.Write("<- " + request);

}

else if (!pagesQueue.Contains(request) && pagesQueue.Count == blocksNum)

{

pagesQueue.Dequeue();

pagesQueue.Enqueue(request);

OutputInConsole(pagesQueue);

ColorPrint.PrintWithColor(ConsoleColor.Red, "<- ");

Console.Write(request);

ColorPrint.PrintWithColor(ConsoleColor.Red, " p");

++pageFaults;

}

else if (!pagesQueue.Contains(request) && pagesQueue.Count < blocksNum)

{

pagesQueue.Enqueue(request);

OutputInConsole(pagesQueue);

ColorPrint.PrintWithColor(ConsoleColor.Red, "<- ");

Console.Write(request);

ColorPrint.PrintWithColor(ConsoleColor.Red, " p");

++pageFaults;

}

Console.WriteLine();

}

Console.WriteLine("Page faults: " + pageFaults);

return pageFaults;

}

static void MoveAndCompress(int indexOfValue, int indexToMoveTo, List<int> collection)

{

int value = collection[indexOfValue];

for (int i = indexOfValue; i < indexToMoveTo; ++i)

collection[i] = collection[i + 1];

collection[indexToMoveTo] = value;

}

static int LRU(Tuple<List<int>, List<int>> pagesAndRequests, int blocksNum)

{

ColorPrint.PrintWithColor(ConsoleColor.Yellow, "LRU");

Console.WriteLine(" with " + blocksNum + " blocks");

var pages = pagesAndRequests.Item1;

var requests = pagesAndRequests.Item2;

int pageFaults = 0;

OutputInConsole(pages);

Console.WriteLine();

foreach (var request in requests)

{

if (pages.Contains(request))

{

var indexOfRequest = pages.IndexOf(request);

MoveAndCompress(indexOfRequest, pages.Count - 1, pages);

OutputInConsole(pages);

Console.Write("<- " + request);

}

else if (!pages.Contains(request) && pages.Count == blocksNum)

{

pages.RemoveAt(0);

pages.Add(request);

OutputInConsole(pages);

ColorPrint.PrintWithColor(ConsoleColor.Red, "<- ");

Console.Write(request);

ColorPrint.PrintWithColor(ConsoleColor.Red, " p");

++pageFaults;

}

else if (!pages.Contains(request) && pages.Count < blocksNum)

{

pages.Add(request);

OutputInConsole(pages);

ColorPrint.PrintWithColor(ConsoleColor.Red, "<- ");

Console.Write(request);

ColorPrint.PrintWithColor(ConsoleColor.Red, " p");

++pageFaults;

}

Console.WriteLine();

}

Console.WriteLine("Page faults: " + pageFaults);

return pageFaults;

}

static void OutputBestAlgorithm(int pageFaultsFIFO, int pageFaultsLRU)

{

if (pageFaultsFIFO < pageFaultsLRU)

Console.WriteLine("Наиболее подходящий алгоритм: FIFO");

else if (pageFaultsLRU < pageFaultsFIFO)

Console.WriteLine("Наиболее подходящий алгоритм: LRU");

else if (pageFaultsFIFO == pageFaultsLRU)

Console.WriteLine("Алгоритмы справились одинаково.");

}

static void Main()

{

var pagesAndRequests = GetPagesAndRequests.ReadFromFile();

var pageFaultsFIFO = FIFO(pagesAndRequests, 4);

Console.WriteLine();

var pageFaultsLRU = LRU(pagesAndRequests, 4);

OutputBestAlgorithm(pageFaultsFIFO, pageFaultsLRU);

Console.WriteLine();

pagesAndRequests = GetPagesAndRequests.ReadFromFile();

pageFaultsFIFO = FIFO(pagesAndRequests, 5);

Console.WriteLine();

pageFaultsLRU = LRU(pagesAndRequests, 5);

OutputBestAlgorithm(pageFaultsFIFO, pageFaultsLRU);

Console.Read();

}

}

}

[конец файла Program.cs]

[файл ColorPrint.cs]

using System;

namespace Lab3

{

static class ColorPrint

{

public static void PrintWithColor(ConsoleColor color, string message)

{

Console.ForegroundColor = color;

Console.Write(message);

Console.ResetColor();

}

public static void ErrorPrint(string errorMessage, string askMessage)

{

PrintWithColor(ConsoleColor.Red, errorMessage);

PrintWithColor(ConsoleColor.Yellow, askMessage);

}

}

}

[конец файла ColorPrint.cs]

[файл GetPagesAndRequests.cs]

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace Lab3

{

public static class GetPagesAndRequests

{

public static Tuple<List<int>, List<int>> ReadFromFile()

{

var pages = new List<int>();

var requests = new List<int>();

using (var reader = new StreamReader(@".\PagesAndRequests.txt"))

{

var str = reader.ReadLine();

pages = str.Split(' ').ToList().Select(a => int.Parse(a)).ToList();

str = reader.ReadLine();

requests = str.Split(' ').ToList().Select(a => int.Parse(a)).ToList();

}

return new Tuple<List<int>, List<int>>(pages, requests);

}

}

}

[конец файла GetPagesAndRequests.cs]

**2.1 Примеры выполнения программы**

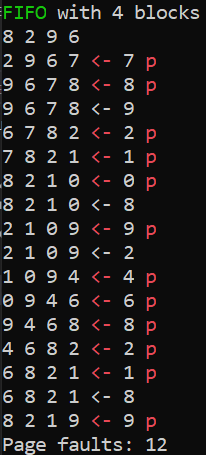


Рисунок 1 – Алгоритм FIFO с 4 страничными блоками

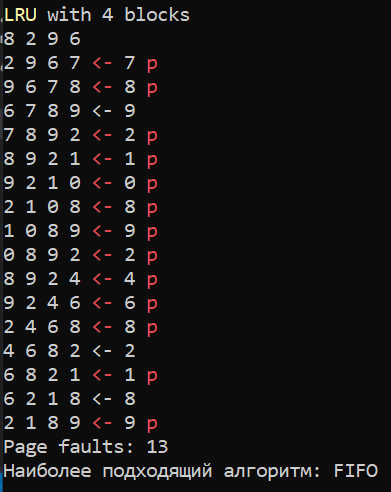


Рисунок 2 – Алгоритм LRU с 4 страничными блоками

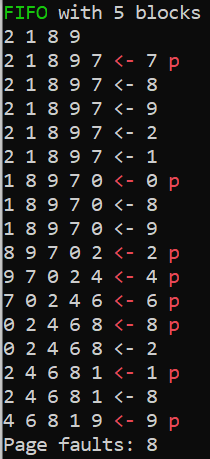


Рисунок 3 – Алгоритм FIFO с 5 страничными блоками

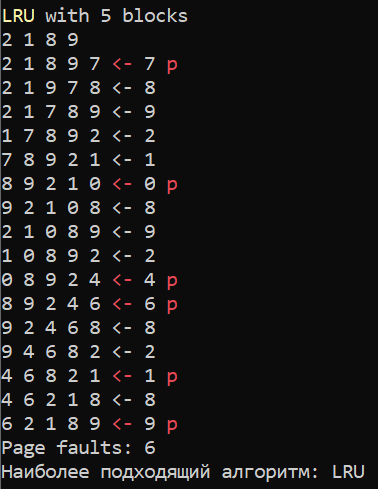


Рисунок 4 – Алгоритм LRU с 5 страничными кадрами

**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.

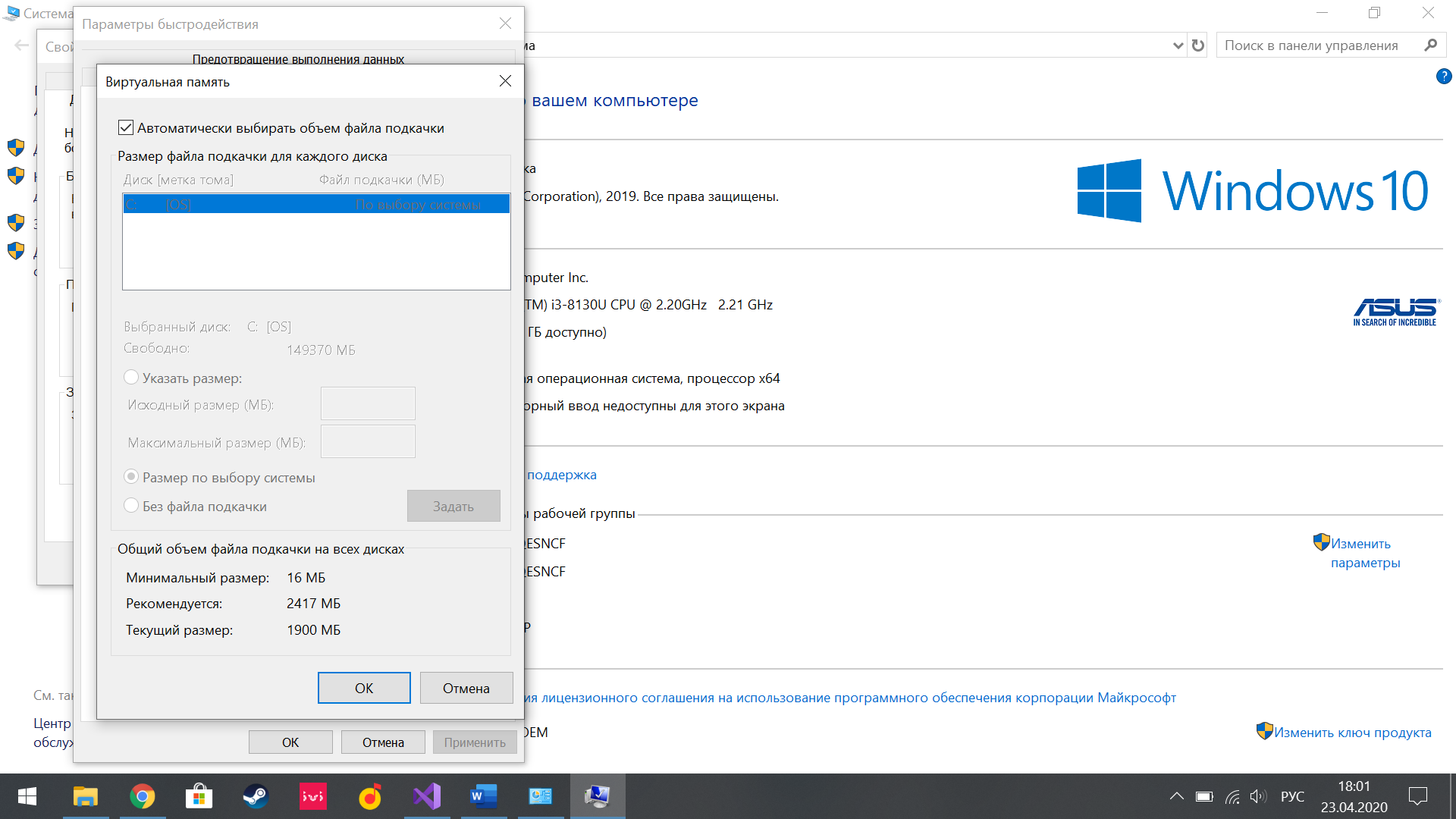
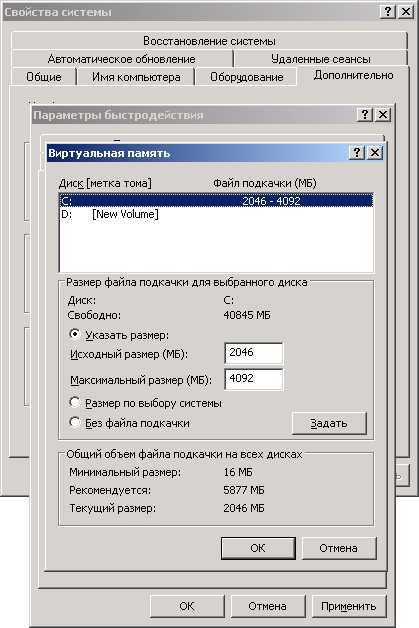


Рисунок 5 – Настройка файла подкачки на Windows 10

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.