

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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Афанасьев А.С.

(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.IO;

using System.Linq;

using System.Collections.Generic;

namespace Lab3

{

class Program

{

static void Main()

{

Console.Write("Введите путь к файлу: ");

string path = Console.ReadLine();

var blocks = GetBlocksFromFile(path);

var requests = GetRequestsFromFile(path);

Console.WriteLine("Алгоритм FIFO для 4 страничных блоков:");

FIFO(blocks, requests);

blocks = GetBlocksFromFile(path);

Console.WriteLine("Алгоритм LRU для 4 страничных блоков:");

LRU(blocks, requests);

blocks = GetBlocksFromFile(path);

blocks.Add(null);

Console.WriteLine("Алгоритм FIFO для 5 страничных блоков:");

FIFO(blocks, requests);

blocks = GetBlocksFromFile(path);

blocks.Add(null);

Console.WriteLine("Алгоритм LRU для 5 страничных блоков:");

LRU(blocks, requests);

}

static List<int?> GetBlocksFromFile(string path)

{

string array = File.ReadLines(path).First();

string[] block = array.Split(' ');

List<int?> blocks = new List<int?>();

Console.Write("Блоки: ");

for (int i = 0; i < block.Length; i++)

{

blocks.Add(Convert.ToInt32(block[i]));

}

for (int i = 0; i < block.Length; i++)

{

Console.Write(blocks[i] + " ");

}

Console.WriteLine();

return blocks;

}

static List<int> GetRequestsFromFile(string path)

{

string array = File.ReadLines(path).Skip(1).First();

string[] tabl = array.Split(' ');

List<int> requests = new List<int>();

Console.Write("Запросы: ");

for (int i = 0; i < tabl.Length; i++)

{

requests.Add(Convert.ToInt32(tabl[i]));

}

for (int i = 0; i < tabl.Length; i++)

{

Console.Write(requests[i] + " ");

}

Console.WriteLine();

return requests;

}

static void FIFO(List<int?> blocks, List<int> requests)

{

int pageFaults = 0;

for (int i = 0; i < requests.Count; i++)

{

bool flag = true;

for (int j = 0; j < blocks.Count; j++)

{

if (blocks[j] == requests[i])

{

flag = true;

break;

}

else

{

flag = false;

}

}

if (flag == false)

{

pageFaults++;

if (blocks.Contains(null))

{

var indexOfEmptyElem = blocks.IndexOf(null);

blocks[indexOfEmptyElem] = requests[i];

}

else

{

int k = 0;

while (k < blocks.Count - 1)

{

blocks[k] = blocks[k + 1];

k++;

}

blocks[blocks.Count - 1] = requests[i];

}

}

for (int k = 0; k < blocks.Count; k++)

{

Console.Write(blocks[k] + " ");

}

if (flag == false)

Console.Write(" <-p");

Console.WriteLine();

}

Console.WriteLine($"{pageFaults} страничных прерываний");

}

static void LRU(List<int?> blocks, List<int> requests)

{

int pageFaults = 0;

for (int i = 0; i < requests.Count; i++)

{

bool flag = true;

for (int j = 0; j < blocks.Count; j++)

{

if (blocks[j] == requests[i])

{

blocks.RemoveAt(j);

blocks.Add(requests[i]);

flag = true;

break;

}

else

{

flag = false;

}

}

if (flag == false)

{

pageFaults++;

if (blocks.Contains(null))

{

var indexOfEmptyElem = blocks.IndexOf(null);

blocks[indexOfEmptyElem] = requests[i];

}

else

{

int k = 0;

while (k < blocks.Count - 1)

{

blocks[k] = blocks[k + 1];

k++;

}

blocks[blocks.Count - 1] = requests[i];

}

}

for (int k = 0; k < blocks.Count; k++)

{

Console.Write(blocks[k] + " ");

}

if (flag == false)

Console.Write(" <-p");

Console.WriteLine();

}

Console.WriteLine($"{pageFaults} страничных прерываний");

}

}

}

[конец файла Program.cs]

**2.1 Примеры выполнения программы**

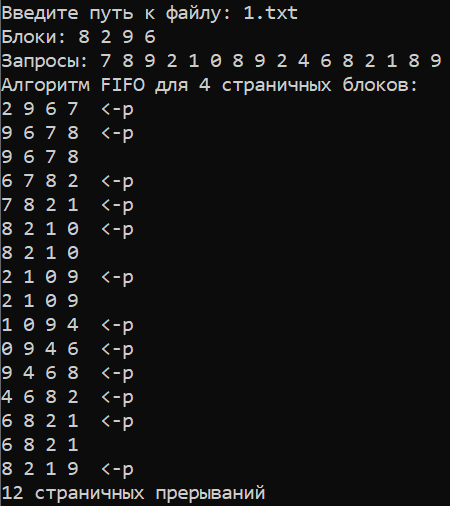


Рисунок 1 – Алгоритм FIFO с 4 страничными блоками

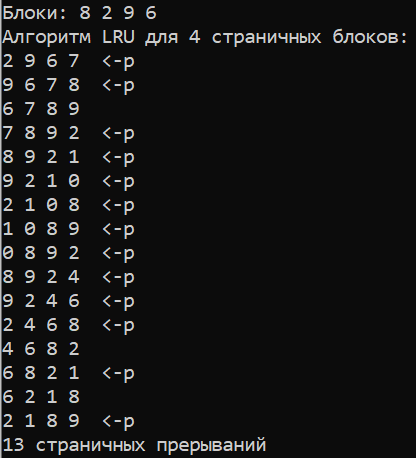


Рисунок 2 – Алгоритм LRU с 4 страничными блоками

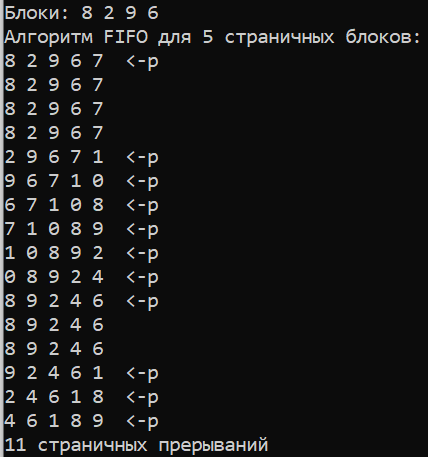


Рисунок 3 – Алгоритм FIFO с 5 страничными блоками

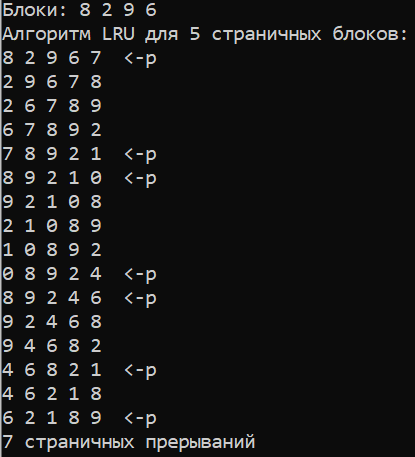


Рисунок 4 – Алгоритм LRU с 5 страничными кадрами

**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.

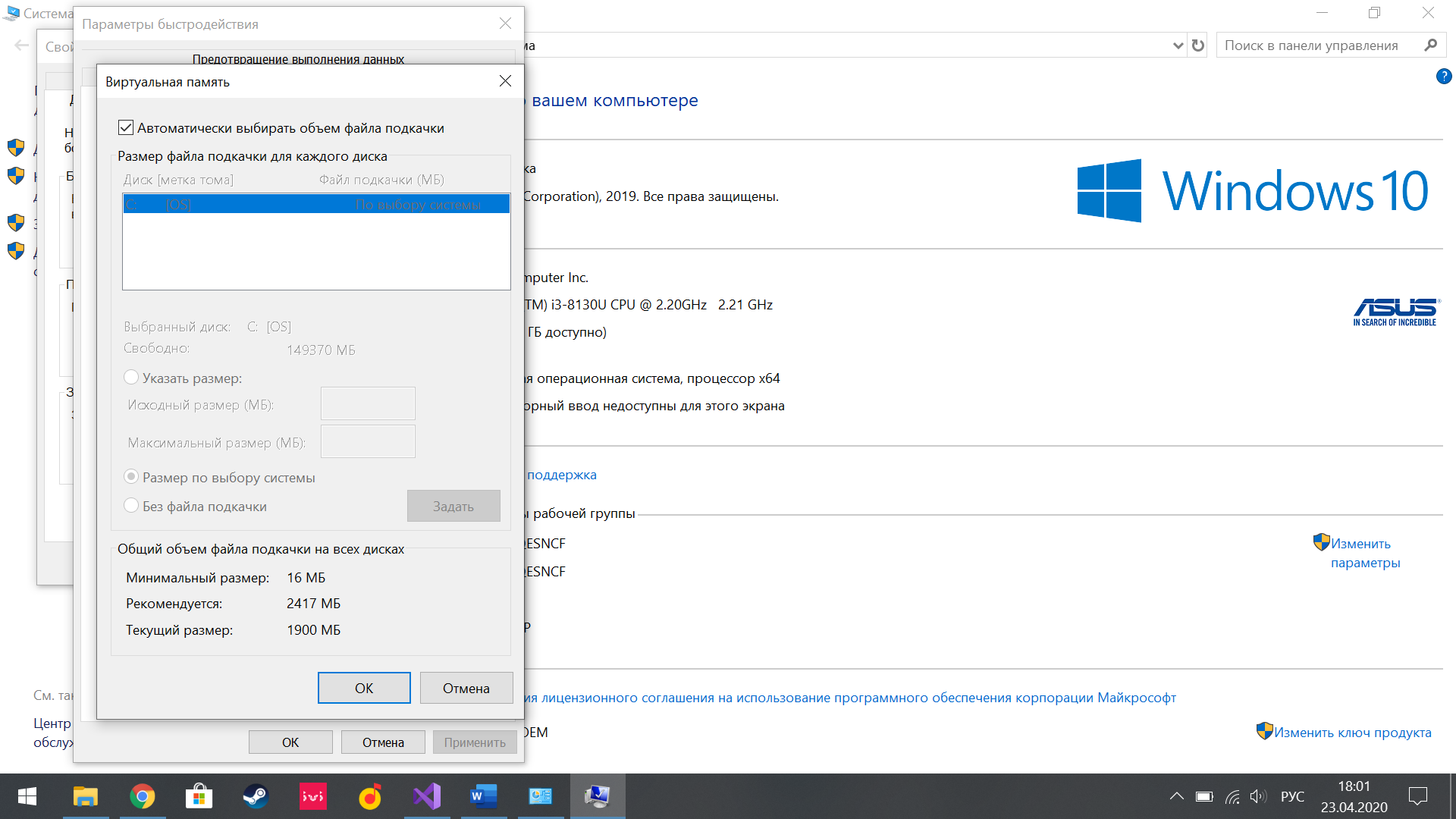
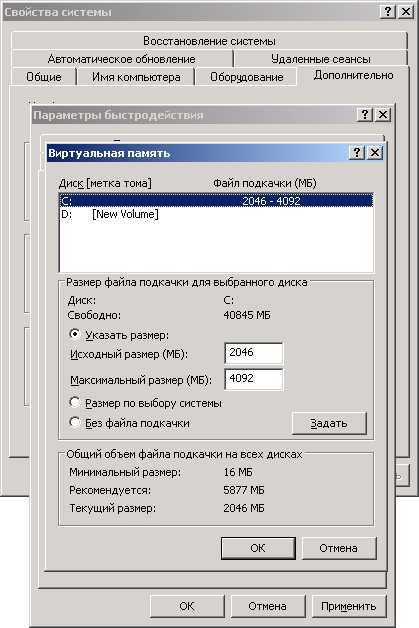


Рисунок 5 – Настройка файла подкачки на Windows 10

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.