УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

**“ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО”**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 10

по дисциплине «Операционные системы»

На тему **«**Управление виртуальной памятью.

Алгоритмы замещения страниц»

Выполнил: студент гр. ИТП-11

Мурашко М.Д.

Принял: преподаватель

Карась О. В.

Гомель 2022

**Цель**: изучить приемы управления виртуальной памятью, а также алгоритмы замещения страниц и реализовать программу, управляющую виртуальной памятью.

**Ход работы**

**Задание:**. Разработать программу, реализующий заданный алгоритм замещения страниц в памяти.

Менеджер памяти должен:

* Разбивать память заданного размера на указанное количество страниц. На экран должна выводиться информация о состоянии памяти: объем памяти, число страниц, число свободных страниц (%), размер страницы;
* Размещать в памяти страницу заданного процесса с замещением занятой по заданному алгоритму (по нажатию кнопки «Добавить»). Для размещения страницы в памяти указывается имя процесса и ее номер (вводятся отдельно). После нажатия на кнопку «Добавить» страница размещается в свободной странице памяти. Если задано глобальное размещение, то выбирается любая незанятая страница. При локальном размещении страница помещается среди виртуальных страниц, выделенных этому процессу. Выделение страниц в памяти выполняется при первом ее занесении процесса в память. Алгоритм замещения выполняется только при отсутствии свободных страниц под процесс;
* Удалить из памяти заданную страницу или все страницы заданного процесса (по нажатию кнопки «Удалить»). Указывается номер удаляемой страницы в памяти.
* Организовать циклическое обращение к страницам размещенным в памяти по нажатию на кнопку, при этом случайным образом задается количество обращений к страницам. Для каждого обращения генерируется номер страницы из диапазона от 0 до количества страниц памяти. При обращении к странице в зависимости от варианта увеличивается ее внутренний счетчик обращений или устанавливается флаг обращения.

**Условие для варианта №20:** Глобальное размещение, алгоритм замещения – *NFU*. По таймеру 2 (с большей задержкой, чем Т1) происходит опрос всех страниц. Для каждой страницы с установленным флагом обращений увеличивается соответствующий счетчик, флаг обращений сбрасывается. Выгружается страница с наименьшим значением счетчика.

Для выполнения задания были разработаны *html*, *js* и *css*-файлы, листинг к которым приведен в приложениях А, Б и С. Ниже представлен результат выполнения задания на рисунках 1-7.

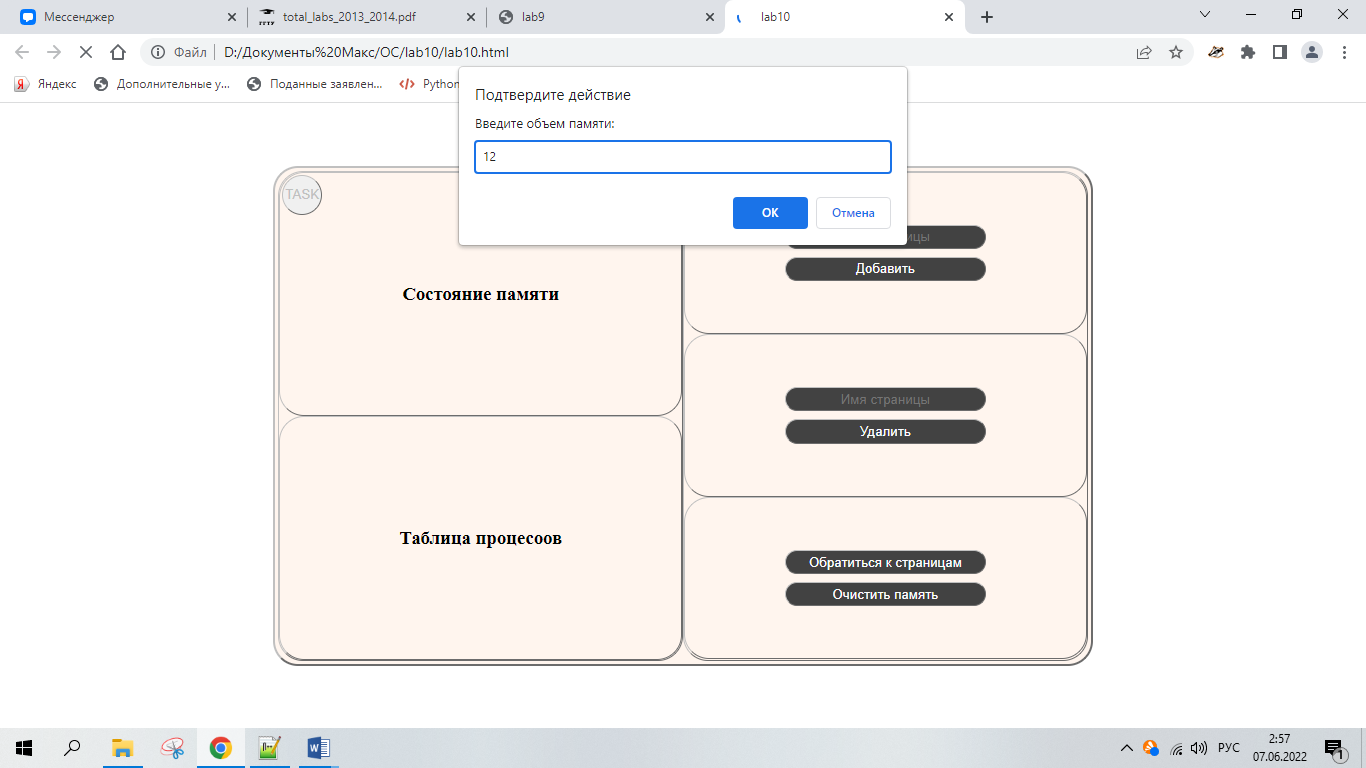


Рисунок 1 – Ввод значения объёма памяти

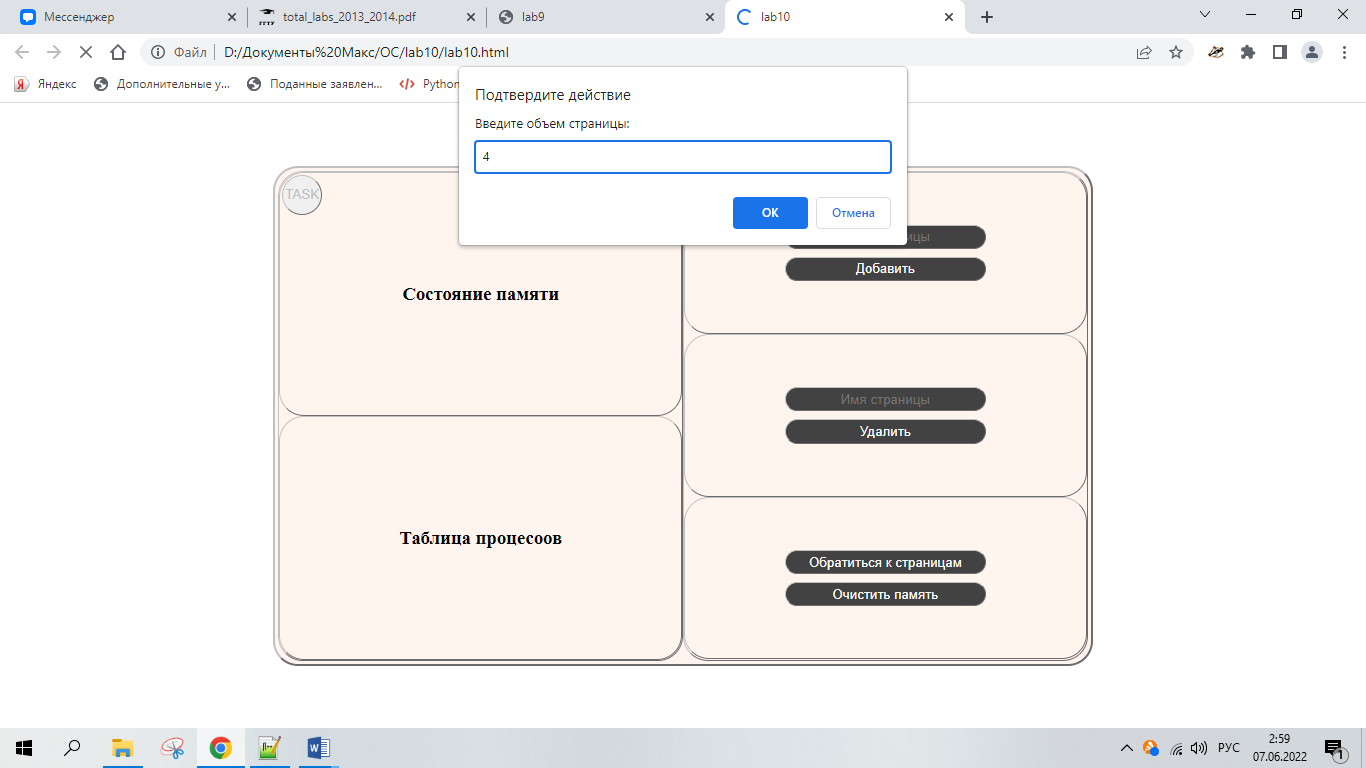


Рисунок 2 – Ввод размера одной страницы

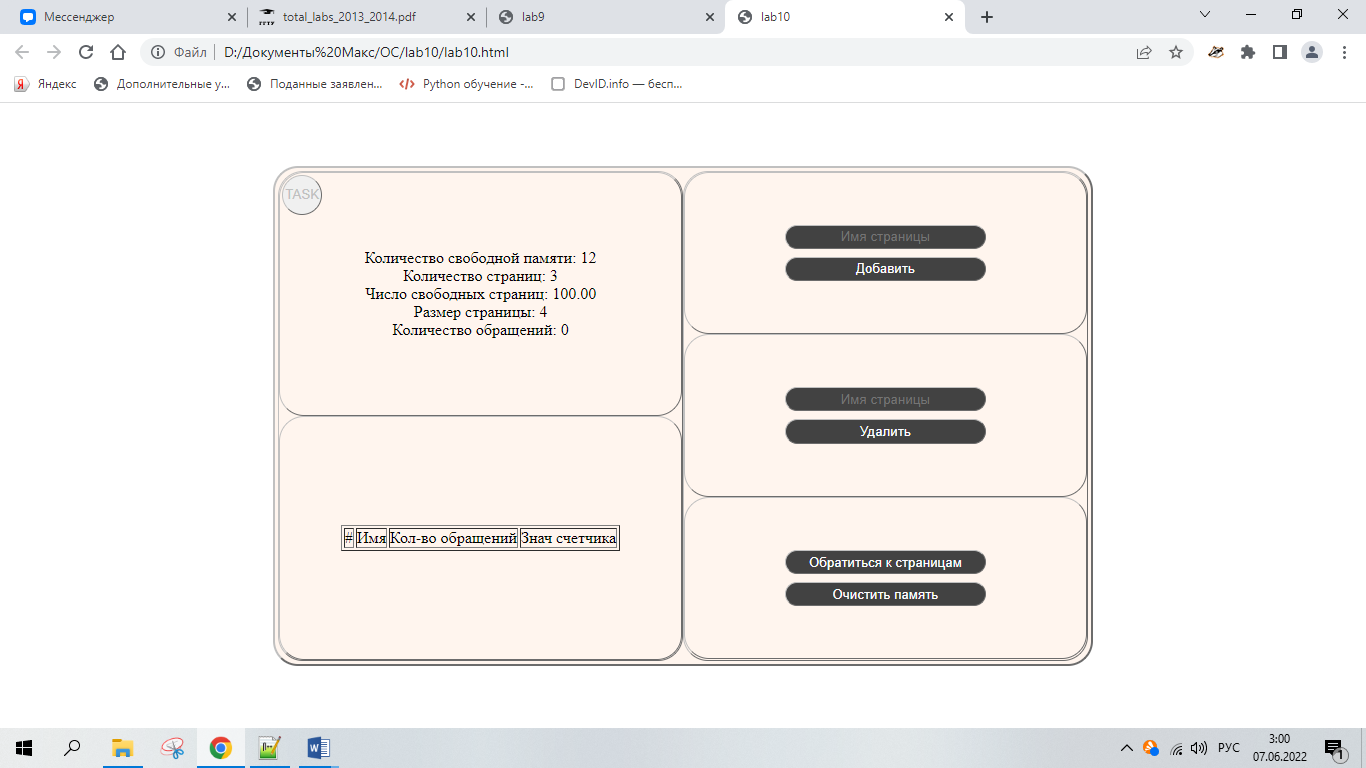


Рисунок 3 – Вид страницы после ввода начальных данных

Теперь добавим несколько страниц: *p1*, *p2*, *p3,* для проверки работы алгоритма замещения страниц, суть которого в следующем: через некоторые промежутки времени мы обращаемся к страницам, увеличивается их специальный счетчик, выгрузится та страница, у которой значение этого счетчика самое большое, то есть страница является самой «Старой». Страницу можно выгрузить по нажатию на кнопку «Очистить память» или же это произойдет, если при добавлении страницы памяти будет недостаточно.

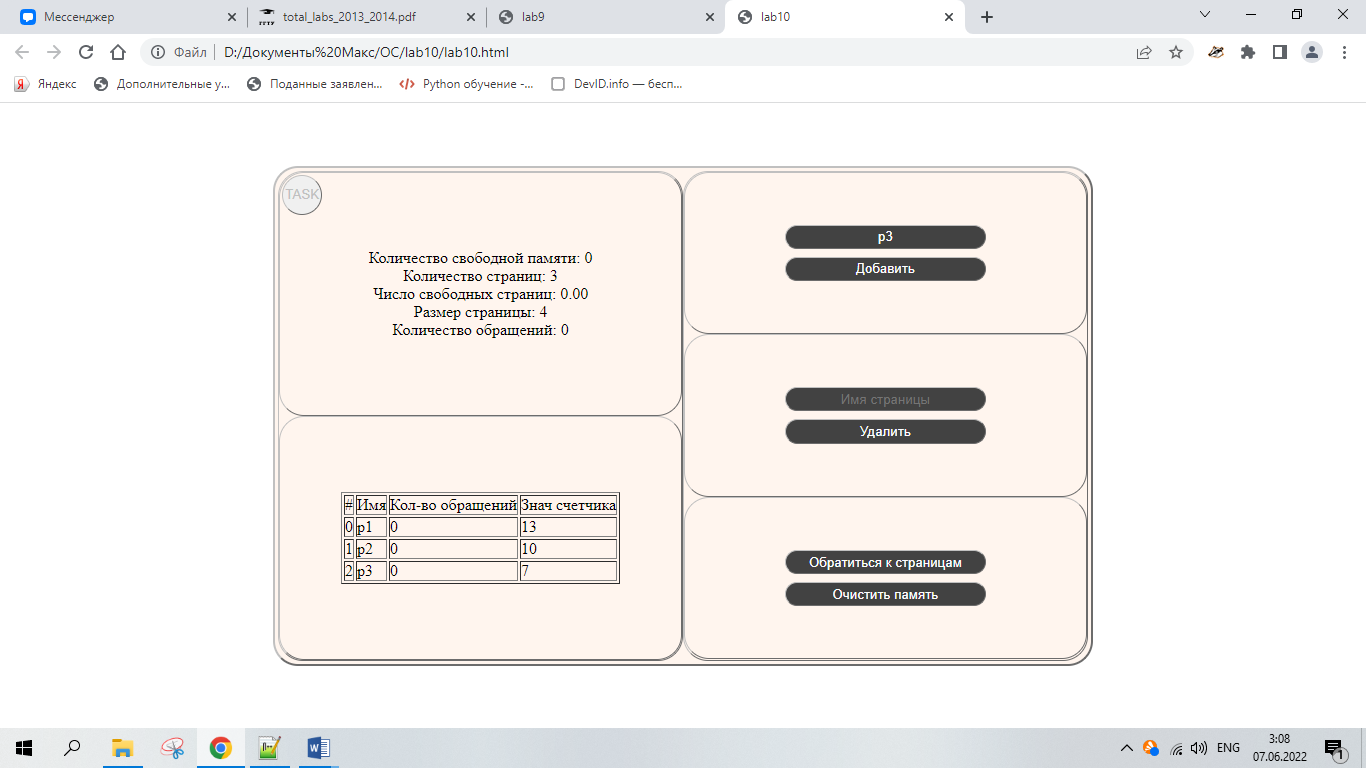


Рисунок 4 – Таблица после добавления процессов

Самое большое значение счетчика будет у первой страницы, так как она самая «старая». Для имитации обращения к страницам, нажмём кнопку «Обратиться страницам», которая сгенерирует случайно число от 1 до 10 – количество обращений, и для каждого обращения сгенерирует случайное число от 1 до количества страниц – страница, к которой обратятся.

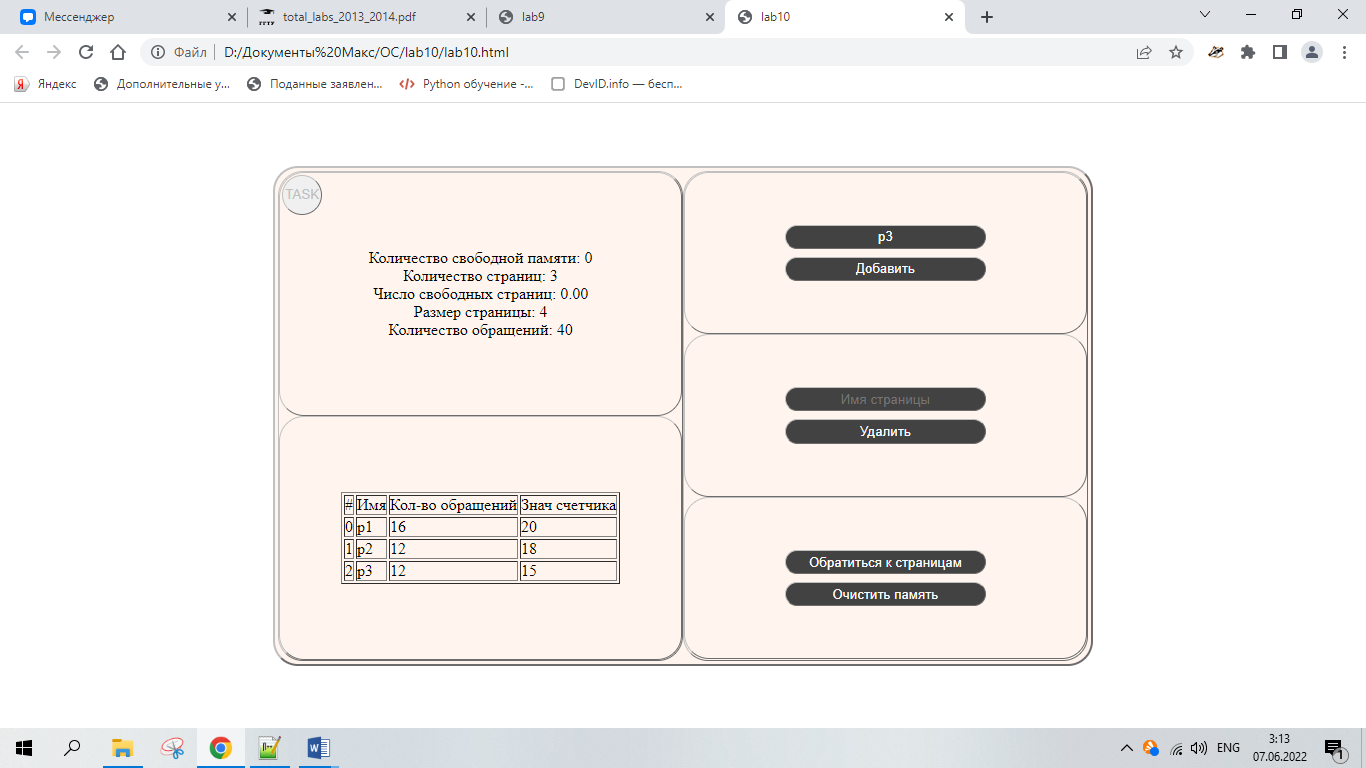


Рисунок 5 – Таблица после обращения к страницам

Логично, что при очистке памяти очисти удалится первая страница, так как она самая «старая»:

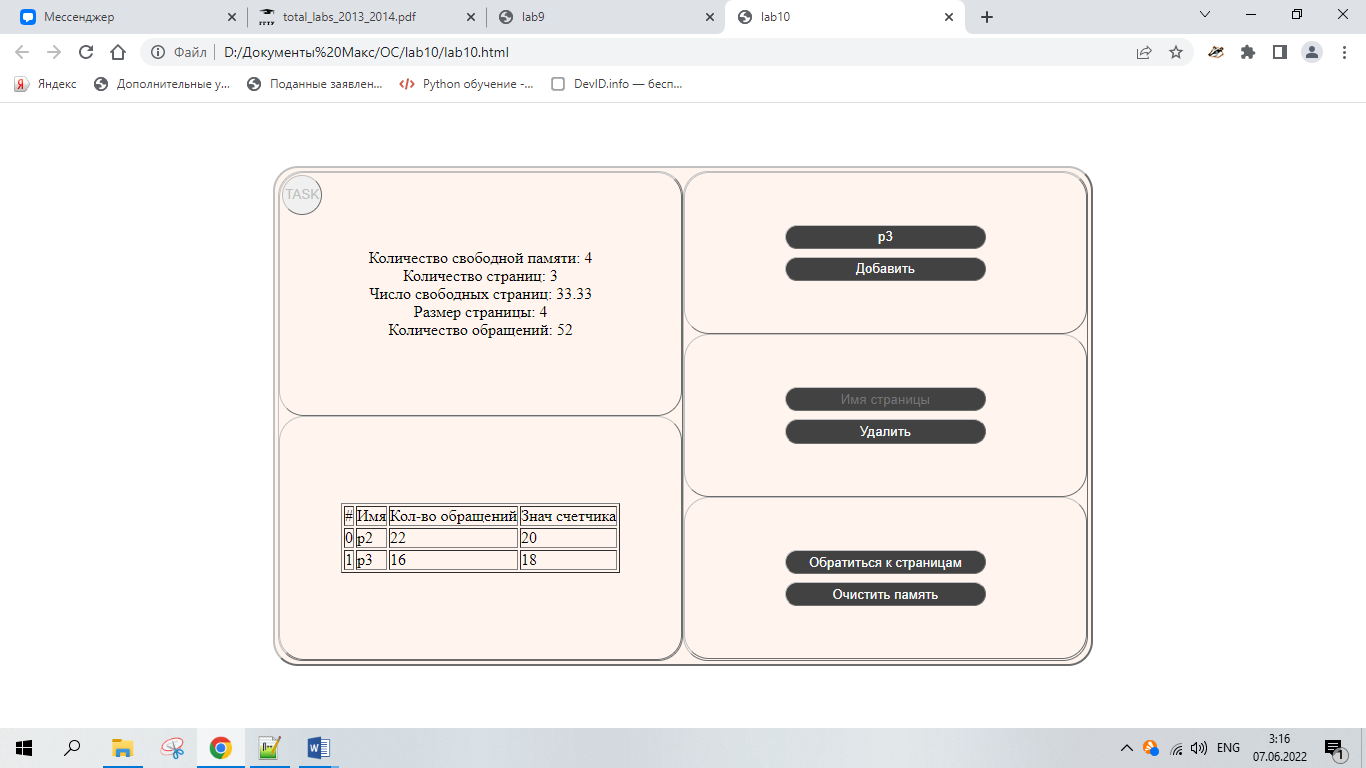


Рисунок 6 – Результат очистки памяти

При добавлении процесса *p5* страница *p2* удалится автоматически, так как максимальное количество страниц равно 3:

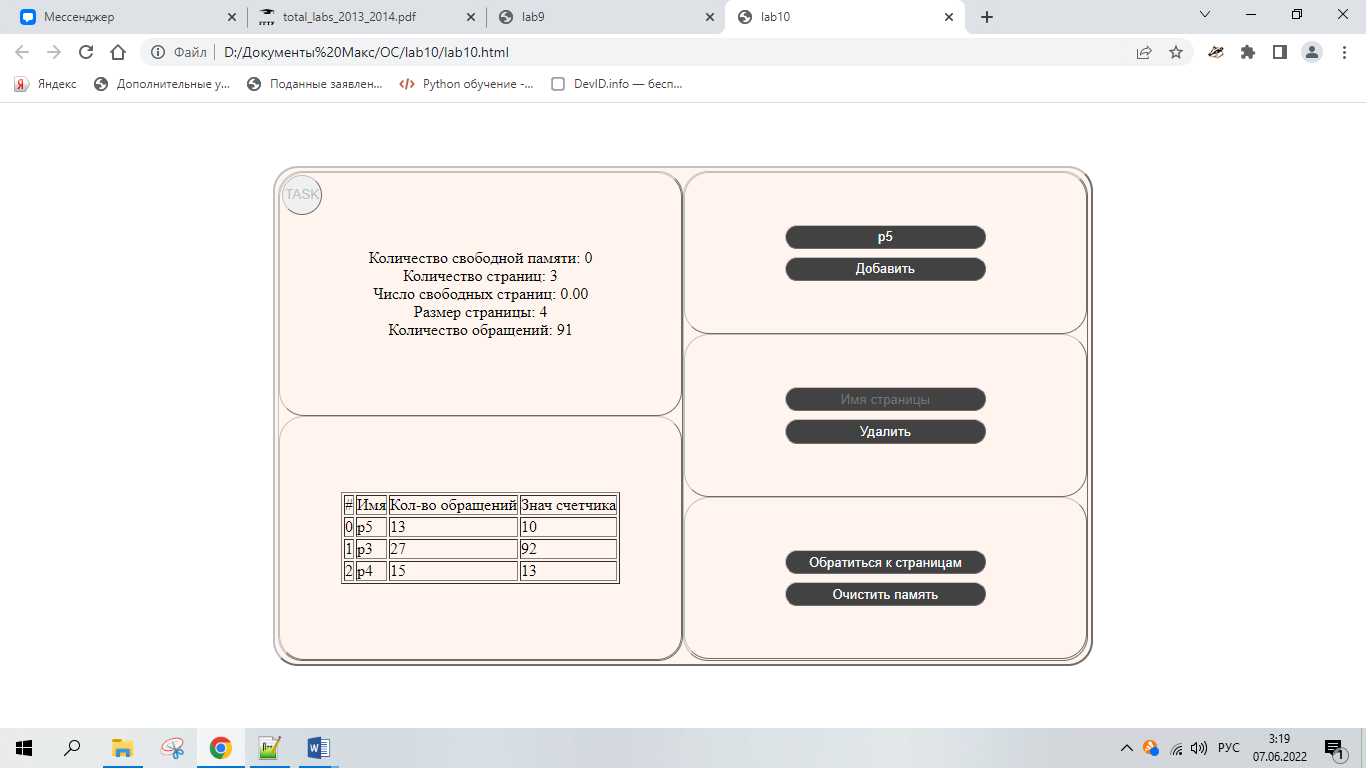


Рисунок 7 – Автоматическое удаление страницы *p2*

Опять же, страница *p2* была удалена алгоритмом как самая “старая”.

**Вывод:** в ходе работы были изучены приемы управления виртуальной памятью, а также алгоритмы замещения страниц и была реализована программа, управляющую виртуальной памятью.

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

**Код менеджера виртуальной памяти**

//Описание переменной и функции для вывода задания по нажатию на кнопку "TASK"

let show\_task=document.getElementById("task");

show\_task.addEventListener("click", (e) => {

alert("Задание для варианта №20:\nГлобальное размещение. Алгоритм замещения-NFU.\nВыгружается страница с наименьшим значением счётчика.");

});

let pagesInfo = document.getElementById("pagesInfo");

let pagesInfoTable = document.getElementById("pagesInfoTable");

let pageNameAdd = document.getElementById("pageNameAdd");

let addPageBtn = document.getElementById("addPageBtn");

let pageNameDel = document.getElementById("pageNameDel");

let deletePageBtn = document.getElementById("deletePageBtn");

let callPageBtn = document.getElementById("callPageBtn");

let freeMemoryBtn = document.getElementById("freeMemoryBtn");

let timerID = 0;

let memory = {

total: 0,

pagesAmount: 0,

pageSize: 0,

pages: [],

totalQuieries: 0

}

let total = +prompt("Введите объем памяти: ");

let pageSize = +prompt("Введите объем страницы: ");

memory.total = total;

memory.pageSize = pageSize;

memory.pagesAmount = Math.floor(total / pageSize);

showPagesInfo(memory);

callCountPage();

freeMemoryBtn.addEventListener("click", (e) => {

if (!memory.pages.length) {

alert("Страниц нет!");

return;

} else

freeMemory(memory, false)

});

callPageBtn.addEventListener("click", (e) => {

if (!memory.pages.length) {

alert("Страниц нет!");

return;

} else

callPage(memory)

});

addPageBtn.addEventListener("click", (e) => {

let name = pageNameAdd.value;

if (!name)

{

alert("Вы не ввели имя страницы!");

return;

} else

addPage(memory, name)

});

deletePageBtn.addEventListener("click", (e) => {

let name = pageNameDel.value;

if (!name)

{

alert("Вы не ввели имя страницы!");

return;

} else

deletePage(memory, name)

});

function callCountPage() {

timerID = setTimeout(() => {

memory.pages.forEach(page => page.count++);

showPagesTable(memory);

showPagesInfo(memory);

callCountPage();

}, 1000);

}

function freeMemory(memory, newPage) {

let forDel = memory.pages.sort((p1, p2) => p2.count - p1.count)[0];

for (let i = 0; i < memory.pages.length; i++)

if (memory.pages[i].name == forDel.name && newPage)

memory.pages[i] = newPage;

else if (!newPage)

{

memory.pages = del(memory.pages, i);

break;

}

showPagesTable(memory);

showPagesInfo(memory);

}

function addPage(memory, pageName) {

if (memory.pages.some(page => page.name == pageName)) {

alert("Страница с таким именем уже есть!")

return;

}

if (memory.pages.length >= memory.pagesAmount)

freeMemory(memory, {

name: pageName,

queries: 0,

count: 0

});

else

memory.pages.push({

name: pageName,

queries: 0,

count: 0

});

showPagesTable(memory);

showPagesInfo(memory);

}

function callPage(memory)

{

let amountOfQueries = Math.floor(Math.random()\*10 + 1);

for (let i = 0; i < amountOfQueries; i++)

{

memory.pages[Math.floor(Math.random() \* memory.pages.length)].queries++;

memory.totalQuieries++;

}

showPagesTable(memory);

showPagesInfo(memory);

}

function deletePage(memory, pageName) {

if (memory.pages.some(page => page.name == pageName)) {

for (let i = 0; i < memory.pages.length; i++)

if (memory.pages[i].name == pageName)

{

memory.pages = del(memory.pages, i);

showPagesTable(memory);

showPagesInfo(memory);

}

} else {

alert("Такой страницы нет!");

return;

}

}

function showPagesInfo(memory) {

let str = `<p>Количество свободной памяти: ${memory.total - memory.pages.length \* memory.pageSize}</p><p>Количество страниц: ${memory.pagesAmount}</p><p>Число свободных страниц: ${(memory.pages.length ? 100 - (100 \* memory.pages.length) / memory.pagesAmount : 100).toFixed(2)}</p><p>Размер страницы: ${memory.pageSize}</p><p>Количество обращений: ${memory.totalQuieries}</p>`;

pagesInfo.innerHTML = str;

}

function showPagesTable(memory) {

let str = "<table border='1'><thead><tr><td>#</td><td>Имя</td><td>Кол-во обращений</td><td>Знач счетчика</td></tr></thead><tbody>";

memory.pages.forEach((page, ind) => {

str += `<tr>`;

str += `<td>${ind}</td><td>${page.name}</td><td>${page.queries}</td><td>${page.count}</td`

str += `</tr>`;

})

str += "</tbody></table>";

pagesInfoTable.innerHTML = str;

}

function del(arr, pos) {

if (pos + 1 == arr.length)

return [...arr.slice(0, pos)]

else if (pos == 0)

return [...arr.slice(1)]

else

return [...arr.slice(0, pos), ...arr.slice(pos + 1, arr.length)];

}

**ПРИЛОЖЕНИ В**

**Листинг программы**

**Код HTML-документа**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link href="css/minimal-table.css" rel="stylesheet" type="text/css">

<link rel="stylesheet" href="style.css">

<title>lab10</title>

</head>

<body>

<div id="wrapper">

<button id="task">TASK</button>

<div id="info\_block">

<div class="pagesInfo" id="pagesInfo">

<h3>Состояние памяти</h3>

</div>

<div class="pagesInfoTable" id="pagesInfoTable">

<h3>Таблица процесоов</h3>

</div>

</div>

<div id="menu\_block">

<div class="block">

<input type="text" placeholder="Имя страницы" id="pageNameAdd" class="input">

<input type="button" value="Добавить" id="addPageBtn" class="input">

</div>

<div class="block">

<input type="text" placeholder="Имя страницы" id="pageNameDel" class="input">

<input type="button" value="Удалить" id="deletePageBtn" class="input">

</div>

<div class="block">

<input type="button" value="Обратиться к страницам" id="callPageBtn" class="input">

<input type="button" value="Очистить память" id="freeMemoryBtn" class="input">

</div>

</div>

</div>

<script src="lab10.js"></script>

</body>

</html>

**ПРИЛОЖЕНИЕ С**

**Листинг программы**

**Код CSS-документа**

\*{

margin:0;

padding:0;

text-decoration:none;

box-sizing:border-box;

}

html,body{

display:flex;

justify-content:center;

align-items:center;

height:100%;

width:100%;

overflow:hidden;

}

#wrapper{

height:80%;

width:60%;

padding:3px;

position:relative;

background-color:#FFF5EE;

border:2px outset #C0C0C0;

border-radius:25px;

}

#task{

height:40px;

width:40px;

position:absolute;

margin-top:0.5%;

margin-left:0.5%;

border-radius:50%;

color:#C0C0C0;

border:1.5px outset #C0C0C0;

transition:0.2s;

}

#task:hover{

box-shadow:0 0 1px rgba(202,228,225,0.92),0 0 9px rgba(202,228,225,0.34),0 0 3px rgba(30,132,242,0.52),0 0 6px rgba(30,132,242,0.92),0 0 10px rgba(30,132,242,0.78),0 0 15px rgba(30,132,242,0.92);

}

h3{

text-align:center;

}

#info\_block{

height:100%;

width:50%;

float:left;

border:1px outset #C0C0C0;

border-radius:25px;

}

#pagesInfo{

display:flex;

justify-content:center;

align-items:center;

flex-direction:column;

height:50%;

width:100%;

border:1px outset #C0C0C0;

border-radius:25px;

}

#pagesInfoTable{

display:flex;

justify-content:center;

align-items:center;

height:50%;

width:100%;

border:1px outset #C0C0C0;

border-radius:25px;

}

#menu\_block{

height:100%;

width:50%;

float:right;

border:1px outset #C0C0C0;

border-radius:25px;

}

.block{

display:flex;

flex-direction:column;

justify-content:center;

align-items:center;

height:33.3333333%;

width:100%;

border:1px outset #C0C0C0;

border-radius:25px;

}

.input{

text-align:center;

height:15%;

width:50%;

border:1px outset #C0C0C0;

border-radius:25px;

color:white;

background-color:#424242;

transition:0.3s;

margin:1%;

}

.input:hover{

color:#c6e2ff;

box-shadow:0 0 2px rgba(202,228,225,0.92),0 0 10px rgba(202,228,225,0.34),0 0 4px rgba(30,132,242,0.52),0 0 7px rgba(30,132,242,0.92),0 0 11px rgba(30,132,242,0.78),0 0 16px rgba(30,132,242,0.92);

}