**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9

по дисциплине: **«Операционные системы»**

на тему: «Управление виртуальной памятью»

Выполнил: студент группы ИТП-11 Гаврильченко А.Е.

Принял: преподаватель Карась О.В.

Гомель 2023

**Цель работы:** изучить понятие виртуальной памяти, страничная виртуальная память, исключительные ситуации при работе с памятью, стратегии управления страничной памятью, алгоритмы замещения страниц, алгоритм FIFO, оптимальный алгоритм (ОРТ), алгоритм LRU, выталкивание редко используемой страницы, алгоритм NFU, алгоритм Second-Chance.

**Задание**

Разработать программу, реализующую заданный алгоритм замещения страниц в памяти.

Менеджер памяти должен:

1. Разбивать память заданного размера на указанное количество страниц. На экран должна выводиться следующая информация о состоянии памяти: объем памяти, число страниц, число свободных страниц (%), размер страницы.

2. 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). Например, Pro 3. После нажатия на кнопку «ДОБАВИТЬ» страница размещается в свободной странице памяти. Если задано глобальное размещение (см. вариант задания), то выбирается любая не занятая страница. При локальном размещении страница размещается только 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.

3. Удалять из памяти заданную страницу или все страницы заданного процесса (по нажатию кнопки «УДАЛИТЬ»). Указывается номер удаляемой страницы в памяти.

4. Организовывать циклическое обращение к страницам, размещенным в памяти по нажатию на кнопку. При этом случайным образом задается количество обращений к страницам (диапазон 1...10). Для каждого обращения генерируется, случайным образом, номер страницы из диапазона [0; количество страниц памяти]. При обращении к странице в зависимости, от варианта, увеличивается ее внутренний счетчик обращений или устанавливается флаг обращения.

В процессе работы менеджера на экран должна выводиться следующая статистическая информация:

1. Общее число обращений к страницам;

2. Количество выталкиваний из памяти (дополнительно в %).

Провести исследования для разного числа страниц при постоянном объеме памяти.

Таблица 1 ‒ Варианты зданий

|  |  |
| --- | --- |
| Вариант | Задание |
| 4 | Глобальное размещение. Алгоритм замещения – LRU CLOCK. Существует глобальный счетчик обращений к страницам. При каждом обращении к станице в ее внутренний регистр заносится значение глобального счетчика. Выгружается страница с наименьшим значением счетчика. |

Результат выполнения задания представлен на рисунке 1.

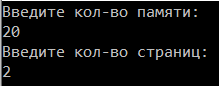


Рисунок 1 – Количество памяти и страниц

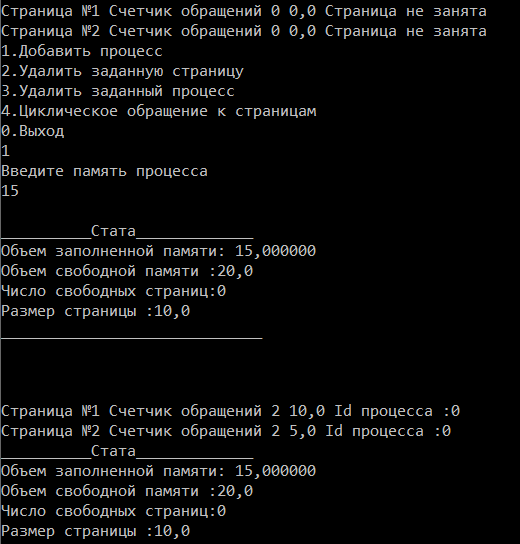


Рисунок 2- Добавление процесса

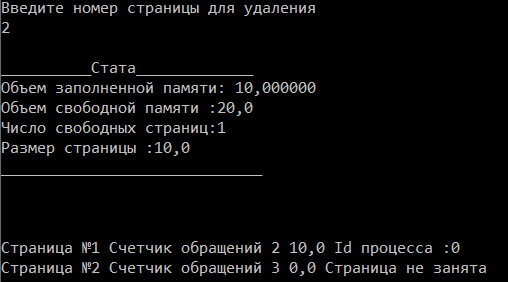


Рисунок 3- Удаление страницы

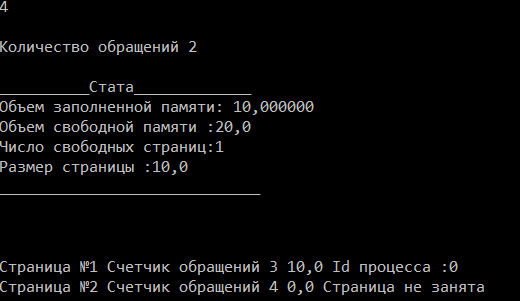


Рисунок 4- Сгенерировать обращение к странице

**Вывод:** в результате выполнения лабораторной работы изучил понятие виртуальной памяти, страничную виртуальную память, исключительные ситуации при работе с памятью, стратегии управления страничной памятью, алгоритмы замещения страниц, алгоритм FIFO, выталкивание первой пришедшей страницы, оптимальный алгоритм (ОРТ), выталкивание долго всего не использовавшейся страницы, алгоритм LRU, выталкивание редко используемой страницы, алгоритм NFU, алгоритм Second-Chance.

Приложение:

Листинг программы:

<!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">

    <title>Менеджер памяти: страничный, локальный, FIFO</title>

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

    <script src="lab10.js"></script>

</head>

<body class="body">

    <div>

        Всего памяти:<input id="mem\_size" style="width:40px" value="10000"></input>

        Число страниц:<input id="page\_count" style="width:40px" value="20"></input>

        Начальное число страниц на процесс:<input id="process\_pages" style="width:40px" value="4"></input>

        Порог увеличения страниц:<input id="access\_threshold" style="width:40px" value="10"></input>

        <br/>

        <button type="button" onclick="add\_page();">Добавить страницу</button>

        Процесс:<input id="add\_process" style="width:40px"></input>

        Номер логической страницы: <input id="add\_page" style="width:40px"></input>

        <br/>

        <button type="button" onclick="remove\_page();">Удалить страницу</button>

        Номер физической страницы: <input id="remove\_page" style="width:40px"></input>

        <button type="button" onclick="remove\_process();">Удалить процесс</button>

        Имя:<input id="remove\_process" style="width:40px"></input>

        <br/>

        <button type="button" onclick="random\_access();">Сгенерировать обращения к страницам</button>

        <br/>

        Состояние памяти:

        <br/>

        <div id="memory" class="memory"></div>

        <div id="mem\_stats"></div>

    </div>

</body>

</html>

let memman = {

    max\_memory: 10000,

    page\_count: 20,

    process\_pages: 4,

    access\_threshold: 10,

    // физические страницы памяти

    pages: [],

    // карта FIFO очередей страниц, ключ = имя процесса, значение = массив очередности страниц

    page\_order: {},

    // карта числа доступов к страницам процессов, ключ = имя процесса, значение = число доступов

    process\_accesses: {}

};

window.onload = function()

{

    init\_mem();

    print\_memory();

}

function init\_mem()

{

    // заполнить массив пустыми страницами

    memman.pages = [];

    for (let i = 0; i < memman.page\_count; i++)

    {

        memman.pages.push({

            process: null,

            logical\_page: null,

            access\_count: 0

        });

    }

}

// добавить логическую страницу к процессу.

// резервирует страницы если процесса ещё не существует

function add\_page()

{

    let name = document.getElementById("add\_process").value;

    let page = +document.getElementById("add\_page").value;

    if (page < 0)

    {

        return;

    }

    // сколько зарезервировано физ страниц для процесса?

    if (count\_logical\_pages(name) < 1)

    {

        reserve\_pages(name, memman.process\_pages);

    }

    load\_page(name, page);

    print\_memory();

}

// очистить одну физическую страницу

function remove\_page()

{

    let page = +document.getElementById("remove\_page").value;

    if (page < 0 || page >= memman.pages.length)

    {

        return;

    }

    memman.pages[page].logical\_page = null;

    memman.pages[page].access\_count = 0;

    print\_memory();

}

// удалить все страницы процесса

function remove\_process()

{

    let name = document.getElementById("remove\_process").value;

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

    {

        if (memman.pages[i].process === name)

        {

            memman.pages[i].process = null;

            memman.pages[i].logical\_page = null;

            memman.pages[i].access\_count = 0;

        }

    }

    print\_memory();

}

// загрузить страницу в память

function load\_page(process, log\_page)

{

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

    {

        if (memman.pages[i].process === process)

        {

            // страница уже существует

            if (memman.pages[i].logical\_page === log\_page)

            {

                add\_access(memman.pages[i]);

                return;

            }

            // найдена свободная страница

            if (memman.pages[i].logical\_page === null)

            {

                memman.pages[i].logical\_page = log\_page;

                add\_access(memman.pages[i]);

                update\_order(process, log\_page);

                return;

            }

        }

    }

    // все страницы заняты, выгрузить самую старую

    if (memman.page\_order[process] === undefined || memman.page\_order[process].length === 0)

    {

        // нет страниц в очереди - нечего выгружать

        return;

    }

    // логический номер вытолкнутой страницы

    let unload\_page = memman.page\_order[process].shift();

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

    {

        if (memman.pages[i].process === process && memman.pages[i].logical\_page === unload\_page)

        {

            memman.pages[i].logical\_page = log\_page;

            memman.pages[i].access\_count = 0;

            add\_access(memman.pages[i]);

            update\_order(process, log\_page);

        }

    }

}

// увеличить счётчик обращений к странице процесса

// page - объект страницы

function add\_access(page)

{

    page.access\_count++;

    if (memman.process\_accesses[page.process] === undefined)

    {

        memman.process\_accesses[page.process] = 0;

    }

    memman.process\_accesses[page.process]++;

    if (memman.process\_accesses[page.process] >= memman.access\_threshold)

    {

        reserve\_pages(page.process, memman.process\_pages);

        memman.process\_accesses[page.process] = 0;

    }

}

// добавить логическую страницу в очередь FIFO

function update\_order(process, log\_page)

{

    let o = memman.page\_order[process];

    if (o === undefined)

    {

        o = [];

    }

    o.push(log\_page);

    memman.page\_order[process] = o;

}

// сгенерировать случайное число доступов к страницам

function random\_access()

{

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

    for (let i = 0; i < num; i++)

    {

        let k = Object.keys(memman.page\_order);

        let p = Math.floor(Math.random()\*k.length);

        let process = k[p];

        let n = Math.floor(Math.random()\*memman.page\_count);

        load\_page(process, n);

    }

    print\_memory();

}

// вычислить число страниц используемых процессом

function count\_logical\_pages(process)

{

    let count = 0;

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

    {

        if (memman.pages[i].process === process)

        {

            count++;

        }

    }

    return count;

}

// зарезервировать страницы за процессом

function reserve\_pages(process, count)

{

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

    {

        if (memman.pages[i].process === null)

        {

            memman.pages[i].process = process;

            count--;

            if (count < 1)

            {

                return;

            }

        }

    }

}

function count\_free\_pages()

{

    let count = 0;

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

    {

        if (memman.pages[i].logical\_page === null)

        {

            count++;

        }

    }

    return count;

}

// вывести состояние памяти

function print\_memory()

{

    let mem\_size = +document.getElementById("mem\_size").value;

    let page\_count = +document.getElementById("page\_count").value;

    memman.process\_pages = +document.getElementById("process\_pages").value;

    memman.access\_threshold = +document.getElementById("access\_threshold").value;

    if (mem\_size > 0 && page\_count > 0)

    {

        if (mem\_size !== memman.max\_memory ||

            page\_count !== memman.page\_count)

        {

            memman.max\_memory = mem\_size;

            memman.page\_count = page\_count;

            init\_mem();

        }

    }

    let out = "";

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

    {

        let cls = "free";

        if (memman.pages[i].process !== null)

        {

            if (memman.pages[i].logical\_page === null)

            {

                cls = "reserved";

            }

            else

            {

                cls = "alloc";

            }

        }

        out += '<div class="'+ cls +'">';

        let log = memman.pages[i].logical\_page === null? "-" : memman.pages[i].logical\_page;

        out += "физ:" + i + " лог:" + log + "<br/>";

        out += memman.pages[i].process === null? "свободно" : memman.pages[i].process;

        out += '<br/>';

        out += "обращений: " + memman.pages[i].access\_count;

        out += '</div>';

    }

    document.getElementById("memory").innerHTML = out;

    out = "Всего: " + memman.max\_memory;

    out += ", страниц: " + memman.page\_count;

    out += ", свободно: " + Math.floor(count\_free\_pages() \* 100 / memman.page\_count) + "%";

    out += ", размер страницы: " + Math.floor(memman.max\_memory / memman.page\_count);

    document.getElementById("mem\_stats").innerHTML = out;

}