УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

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

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 9

по дисциплине «Операционные системы»

На тему **«**Простейшие схему управления памятью»

Выполнил: студент гр. ИТП-11

Воробьев В.А.

Принял: преподаватель

Карась О. В.

Гомель 2022

**Цель**: изучение алгоритмов управления памятью, разработка программы менеджера памятью.

**Ход работы**

**Задание 1**. Разработать программу, реализующий заданный алгоритм выделения памяти.

Менеджер памяти должен:

* По запросу выделять память согласно заданного алгоритма. На экран выводится информация о состоянии памяти: объем памяти, объем свободной памяти, размер наибольшего свободного блока, количество запросов на выделение памяти, количество удовлетворенных запросов (%).
* Для выделения памяти указывается имя процесса и размер блока. После нажатия на кнопку «Добавить» память выделяется или выдаётся сообщение о невозможности выделения.
* Удалить из памяти заданный блок или все блоки заданного процесса по нажатию кнопки удалить
* Реализовать возможность последовательной записи/чтения информации в/из выделенную память по логическому адресу. Вывести физического адреса ячейки памяти, в которую была осуществлена запись.
* Организовать циклическое выделение и освобождение памяти. При этом случайным образом задаётся количество выделяемых блоков и их размер.

Алгоритм выделения памяти своппинг, удаляется тот процесс, у которого меньший приоритет. Листинг программы представлен в приложении A.

Окно программы будет выглядеть следующим образом:

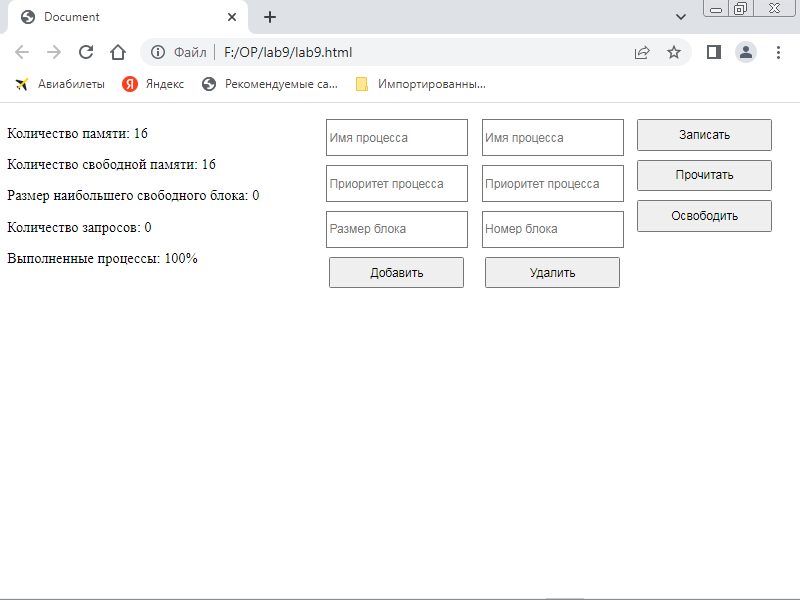


Рисунок 1 – Окно программы

Например, добавим несколько процессов, дабы полностью занять память. Если для процесса не будет хватать памяти, то он будет помещен в очередь и ждать, пока не будет освобождена память, либо удален вручную процесс из памяти. Нажав на кнопку записать, мы записываем некоторое значение в блок памяти, и оно должно быть меньше размера блока для процесса. Очевидно, что если мы сейчас добавим процесс, то ему не хватит памяти, поэтому он будет помещен в очередь. Окно после добавления нескольких процессов будет выглядеть следующим образом:

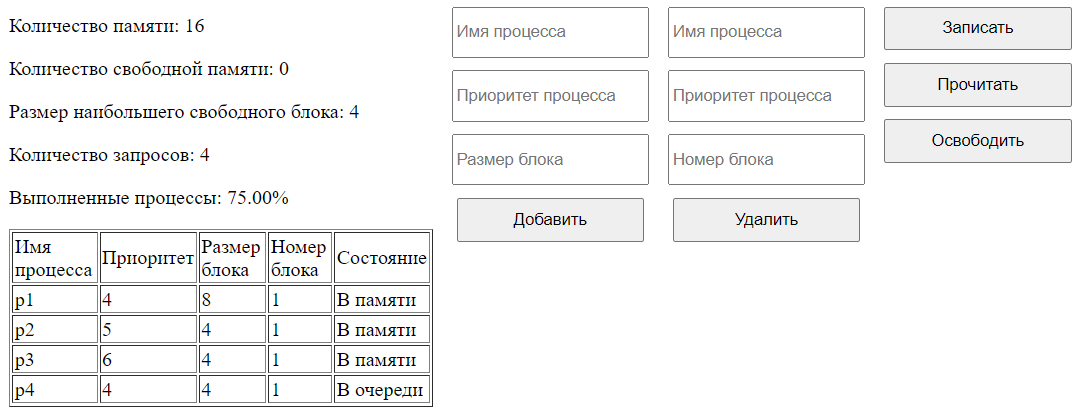


Рисунок 2 – Окно программы после

добавления процессов

Чтобы процессу хватило памяти, мы можем либо удалить блок 1 процесса p3 вручную, либо просто очистить память. При очищении памяти удалится процесс p3, так как у него самый меленький приоритет.



Рисунок 3 – Окно программы после

освобождения памяти

Также мы можем записать некоторое значения в блок, можно, например, запишем в процесс p4 в блок 1 значение: 1001, так как размер блока 4. Затем прочитаем значение:

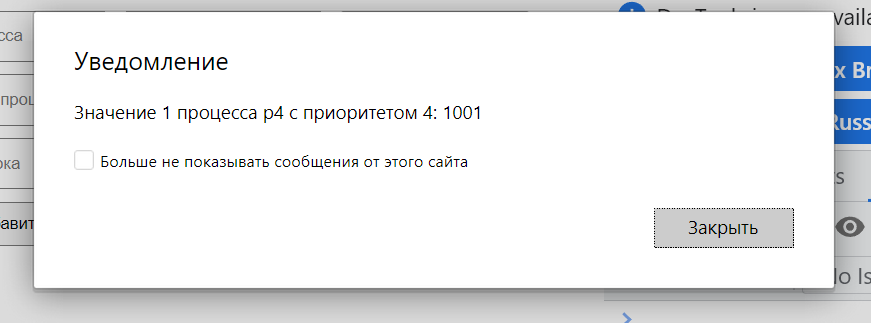


Рисунок 4 – Результат чтения из блока

**Вывод**: в ходе работы были изучены алгоритмы управления памятью, разработана программа менеджера памятью.

**ПРИЛОЖЕНИЕ А**

**КОД МЕНЕДЖЕРА ПАМЯТИ**

**"use strict";**

**let memory =**

**{**

**total: 16,**

**maxBlockSize: 8,**

**blocks: [],**

**waitings: [],**

**queries: 0,**

**getAmountOfFreeMemory() {**

**let freeMemory = 0;**

**this.blocks.forEach(block => freeMemory += block.size);**

**return this.total - freeMemory;**

**}**

**}**

**let processInfoField = document.getElementById("processInfoField");**

**let processNameAdd = document.getElementById("processNameAdd");**

**let blockSizeAdd = document.getElementById("blockSizeAdd");**

**let addProcessBtn = document.getElementById("addProcessBtn");**

**let processPriorityAdd = document.getElementById("processPriorityAdd");**

**let blockNumberDel = document.getElementById("blockNumberDel");**

**let deleteProcessBtn = document.getElementById("deleteProcessBtn");**

**let processPriorityDel = document.getElementById("processPriorityDel");**

**let processNameDel = document.getElementById("processNameDel");**

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

**let writeDownBtn = document.getElementById("writeDownBtn");**

**let readBtn = document.getElementById("readBtn");**

**writeDownBtn.addEventListener("click", (e) => {**

**let name = prompt("Имя процесса:");**

**let priority = +prompt("Приоритет процесса:");**

**let blockNumber = +prompt("Номер блока:");**

**let isFound = false;**

**let isError = true**

**let num = 0;**

**memory.blocks.forEach((block) => {**

**if (block.name == name && block.priority == priority)**

**{**

**num++;**

**if (num == blockNumber && !isFound)**

**{**

**let value = prompt(`Значение(до ${block.size} символов):`);**

**if (value.length > block.size)**

**{**

**isError = false;**

**alert("Вы ввели больше символов, чем нужно!");**

**}**

**else**

**block.value = value;**

**isFound = true;**

**}**

**}**

**});**

**});**

**readBtn.addEventListener("click", (e) => {**

**let name = prompt("Имя процесса:");**

**let priority = +prompt("Приоритет процесса:");**

**let blockNumber = +prompt("Номер блока:");**

**let isFound = false;**

**let isError = true;**

**let num = 0;**

**memory.blocks.forEach((block) => {**

**if (block.name == name && block.priority == priority)**

**{**

**num++;**

**if (num == blockNumber && !isFound)**

**{**

**alert(`Значение ${blockNumber} процесса ${name} с приоритетом ${priority}: ${block.value}`);**

**isError = false;**

**isFound = true;**

**}**

**}**

**});**

**});**

**showMemoryState(memory);**

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

**if (!memory.blocks.length)**

**{**

**alert("Нет исполняющихся процессов! Память свободная");**

**return;**

**}**

**memory.blocks = memory.blocks.sort((bl1, bl2) => bl2.priority - bl1.priority).slice(1);**

**if (memory.waitings.length && memory.waitings[0].size <= memory.getAmountOfFreeMemory())**

**{**

**memory.blocks.push(memory.waitings[0]);**

**memory.waitings = del(memory.waitings, 0);**

**}**

**showMemoryState(memory);**

**flushFields([blockNumberDel]);**

**});**

**addProcessBtn.addEventListener("click", (e) =>**

**{**

**if (!blockSizeAdd.value || !processNameAdd.value || !processPriorityAdd.value)**

**{**

**alert("Вы не ввели либо имя блока, либо имя процесса, либо его приоритет!");**

**return;**

**}**

**let freeMemory = 0;**

**memory.blocks.forEach(block => freeMemory += block.size);**

**let blockSize = +blockSizeAdd.value;**

**let processName = processNameAdd.value;**

**let processPriority = +processPriorityAdd.value;**

**if (blockSize > memory.maxBlockSize)**

**{**

**alert("Максим памяти на блок - 8");**

**return;**

**}**

**if (blockSize <= memory.total - freeMemory)**

**memory.blocks.push({name: processName, priority: processPriority, size: blockSize});**

**else**

**{**

**memory.waitings.push({name: processName, priority: processPriority, size: blockSize, value: ""});**

**alert("Не хватает памяти, удалите какой-либо из процессов! Ваш процесс будет ожидать своей очереди!");**

**}**

**flushFields([blockSizeAdd, processNameAdd, processPriorityAdd]);**

**memory.queries++;**

**showMemoryState(memory);**

**});**

**deleteProcessBtn.addEventListener("click", (e) =>**

**{**

**if (!blockNumberDel.value || !processNameDel.value || !processPriorityDel.value)**

**{**

**alert("Вы не ввели либо имя блока, либо имя процесса")**

**return;**

**}**

**let blockNumber = blockNumberDel.value;**

**let processName = processNameDel.value;**

**let processPriority = processPriorityDel.value;**

**if (!memory.blocks.some(block => block.name == processName))**

**{**

**alert("Процесса с таким именем нет!");**

**return;**

**}**

**let blocks = 0;**

**memory.blocks.forEach(block => block.name == processName && blocks++);**

**if (blocks > blockNumber)**

**{**

**alert(`У процесса ${processName} максимум ${blocks[processName]} блоков!`);**

**return;**

**}**

**else**

**{**

**let num = 0;**

**let isDel = false;**

**memory.blocks.forEach((block, ind) => {**

**if (block.name == processName && block.priority == processPriority)**

**{**

**num++;**

**if (num == blockNumber && !isDel)**

**{**

**memory.blocks = del(memory.blocks, ind);**

**if (memory.waitings.length)**

**if (memory.getAmountOfFreeMemory >= memory.waitings[0].size)**

**{**

**memory.blocks.push(memory.waitings[0]);**

**memory.waitings = del(memory.waitings, 0);**

**isDel = true;**

**}**

**showMemoryState(memory);**

**flushFields([blockNumberDel, processNameDel, processPriorityDel]);**

**}**

**}**

**});**

**}**

**});**

**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)];**

**}**

**function flushFields(fields)**

**{**

**fields.forEach(field => field.value = "");**

**}**

**function showProcesses(memory)**

**{**

**let pattern = `<table border='1'><thead><tr><td>Имя процесса</td><td>Приоритет</td><td>Размер блока</td><td>Номер блока</td><td>Состояние</td></tr></thead>`;**

**pattern += "<tbody>";**

**let blocks = {};**

**let blocksW = {};**

**memory.blocks.forEach(block => {**

**if (!blocks[block.name])**

**{**

**blocks[block.name] = [];**

**blocks[block.name].push(block);**

**}**

**else**

**blocks[block.name].push(block);**

**});**

**memory.waitings.forEach(block => {**

**if (!blocksW[block.name])**

**{**

**blocksW[block.name] = [];**

**blocksW[block.name].push(block);**

**}**

**else**

**blocksW[block.name].push(block);**

**});**

**for (let prop in blocks)**

**blocks[prop].forEach((block, ind) => {**

**pattern += `<tr><td>${block.name}</td><td>${block.priority}</td><td>${block.size}</td><td>${ind + 1}</td><td>В памяти</td></tr></tr>`;**

**});**

**for (let prop in blocksW)**

**blocksW[prop].forEach((block, ind) => {**

**pattern += `<tr><td>${block.name}</td><td>${block.priority}</td><td>${block.size}</td><td>${ind + 1}</td><td>В очереди</td></tr></tr>`;**

**});**

**pattern += `</tbody></table>`;**

**processInfoField.innerHTML += pattern;**

**}**

**function showMemoryState(memory)**

**{**

**let freeMemory = 0;**

**let maxBlock = memory.blocks.length ? memory.blocks.map(block => 8 - block.size).sort((el1, el2) => el2 - el1)[0] : 0;**

**memory.blocks.forEach(block => freeMemory += block.size);**

**let pattern = `<p>Количество памяти: ${memory.total}</p>`;**

**pattern += `<p>Количество свободной памяти: ${memory.getAmountOfFreeMemory()}</p>`;**

**pattern += `<p>Размер наибольшего свободного блока: ${maxBlock}</p>`;**

**pattern += `<p>Количество запросов: ${memory.queries}</p>`;**

**let c = "";**

**if (memory.blocks.length && memory.waitings.length)**

**c = 100 - (memory.waitings.length \* 100 / memory.blocks.length);**

**else if (memory.blocks.length)**

**c = 100;**

**else**

**c = 0;**

**let percent = 100 - (100 \* memory.waitings.length) / (memory.waitings.length + memory.blocks.length);**

**pattern += `<p>Выполненные процессы: ${memory.waitings.length != 0 ? percent.toFixed(2) : 100}%</p>`;**

**processInfoField.innerHTML = pattern;**

**if (memory.blocks.length)**

**showProcesses(memory);**

**}**