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

Мурашко М.Д.

Принял: преподаватель

Карась О. В.

Гомель 2022

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

**Ход работы**

**Задание**: 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.

**Условие для варианта №20:** Свопинг. Удаляется процесс с наименьшим приоритетом.

В соответствии с вариантом были разработаны *html*, *js* и *css*-файлы, листинг к которым представлен в приложениях А, Б и С.

Результат выполнения задания представлен на рисунках 1-9 ниже:

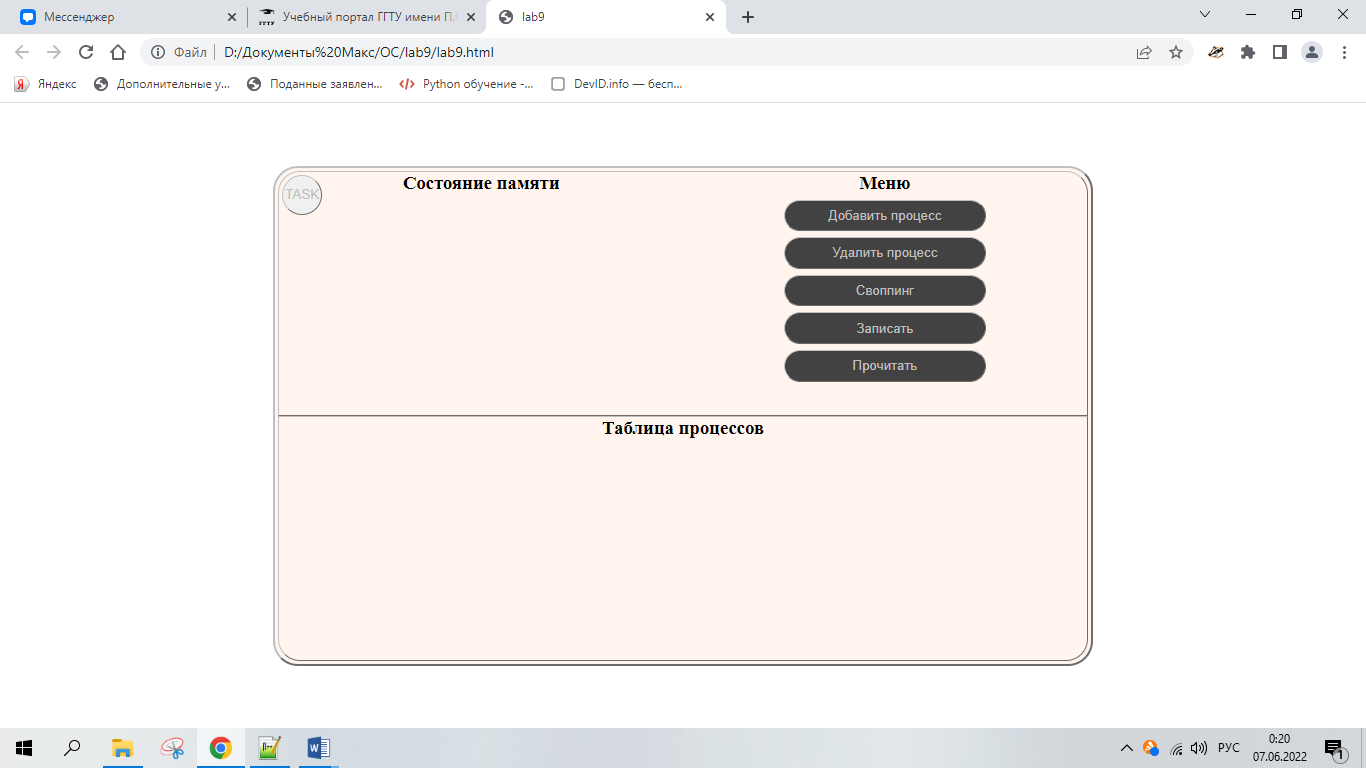


Рисунок 1 – Общий вид программы

На рисунках 2-5 представлен процесс добавления процесса.

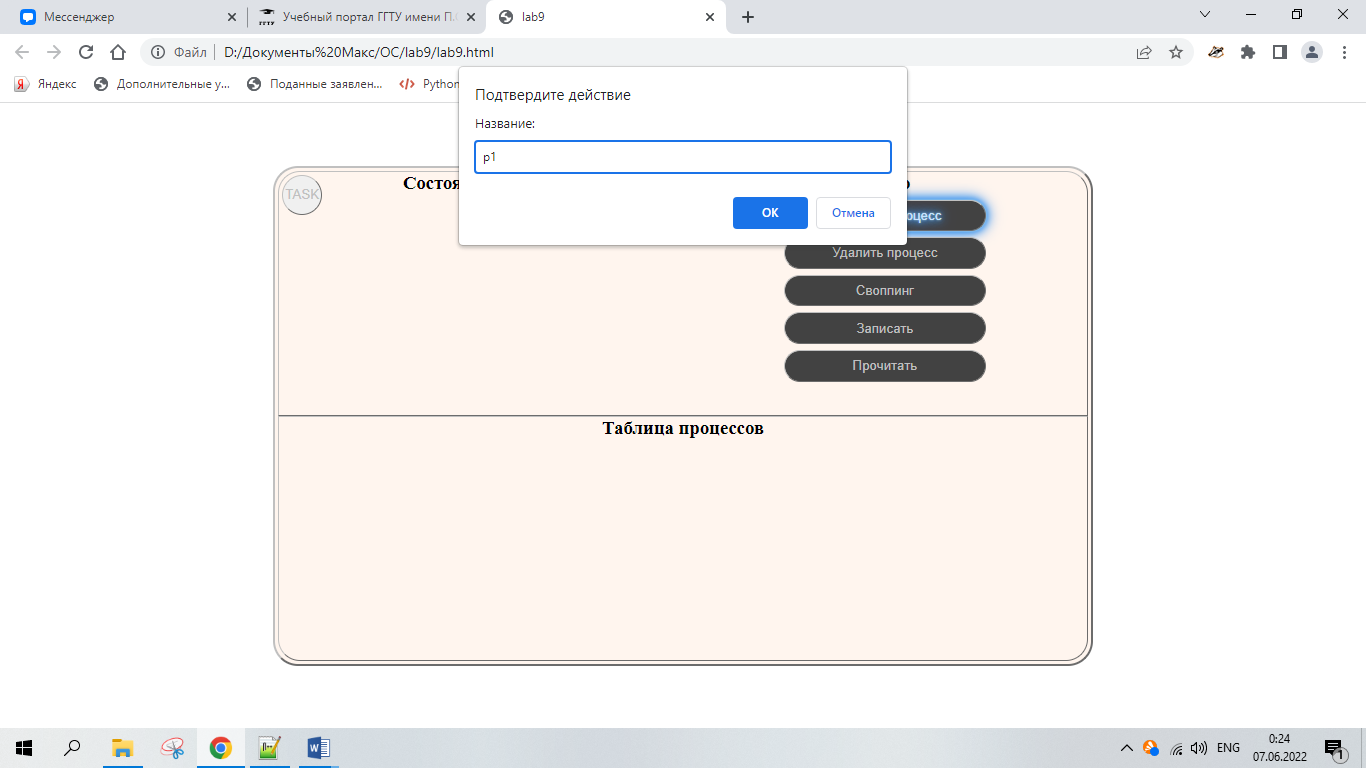


Рисунок 2 – Ввод имени процесса

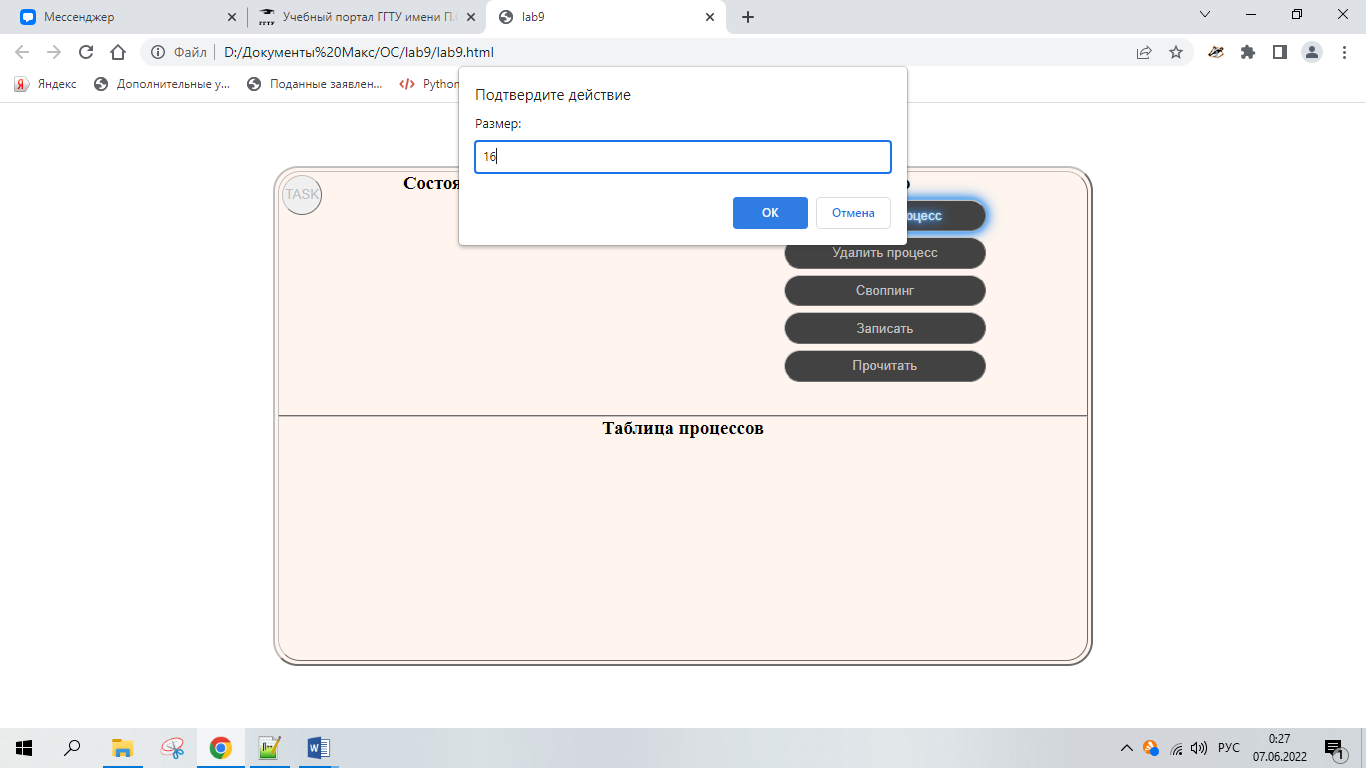


Рисунок 3 – Ввод объёма памяти занимаемого процессом

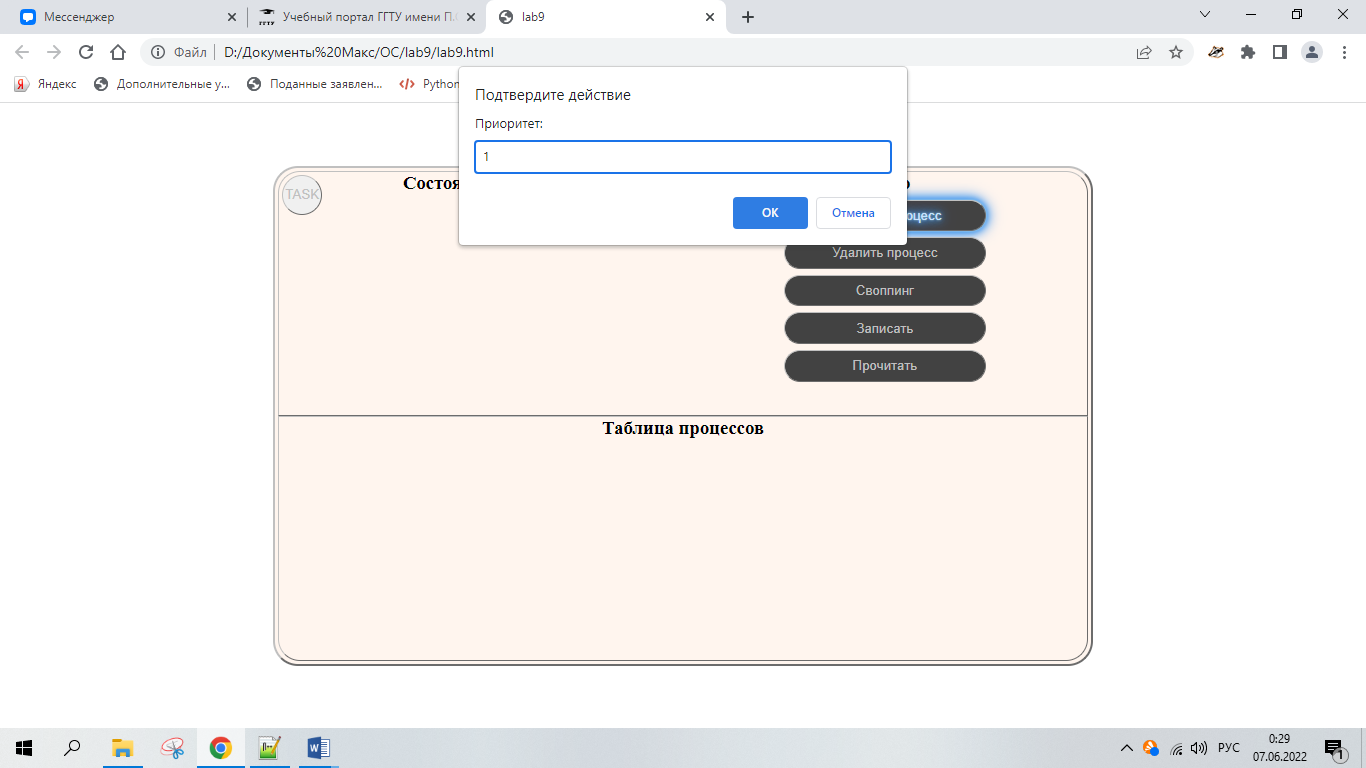


Рисунок 4 – Ввод значения приоритета

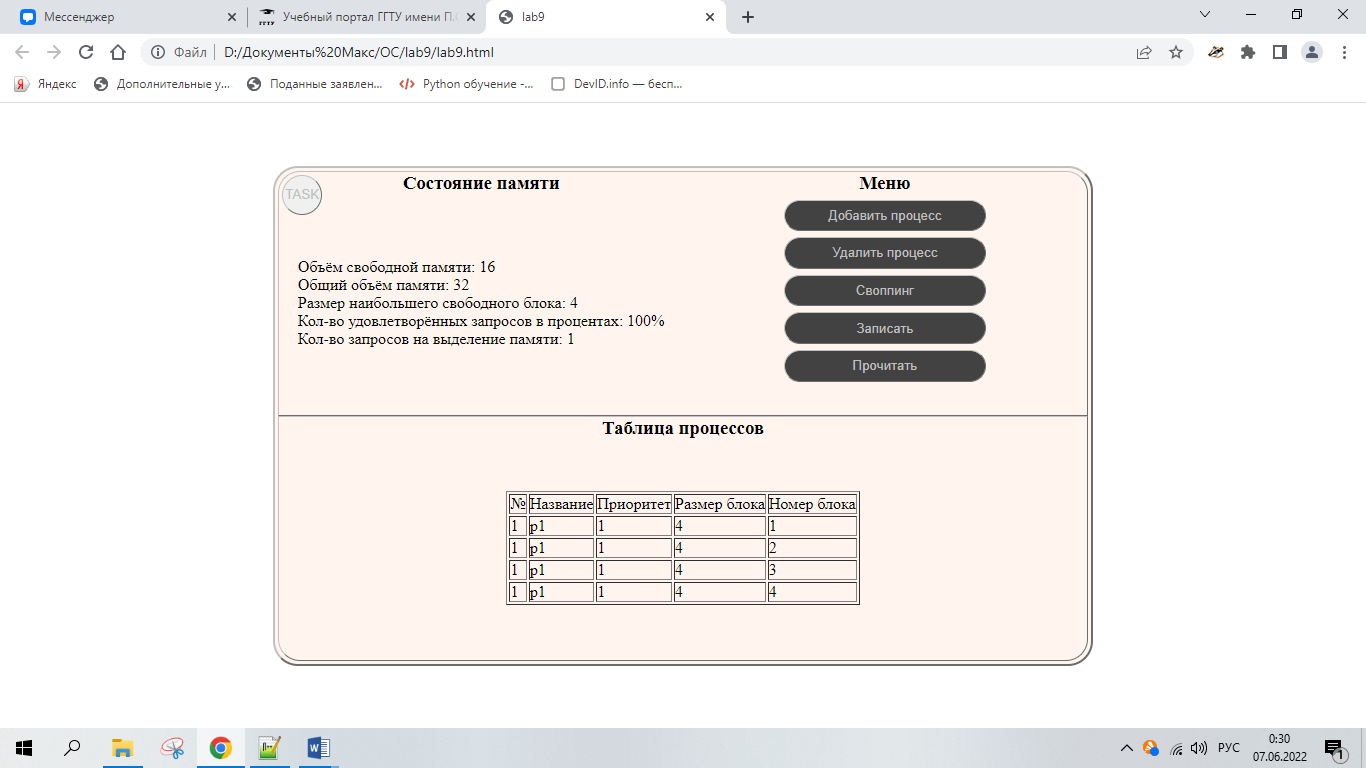


Рисунок 5 – Результат добавления процесса

Удаление процессов, а также запись и чтение осуществляются посредством ввода имени процесса:

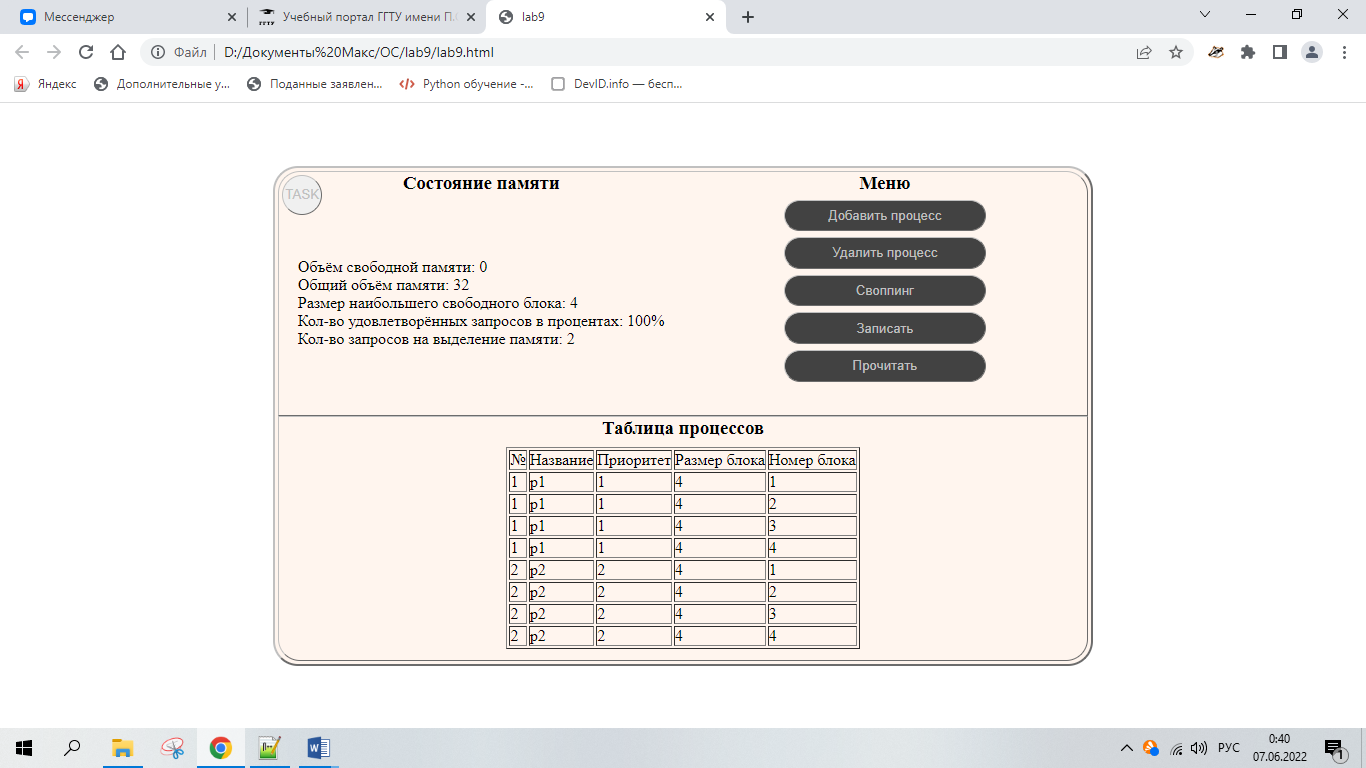


Рисунок 6 – В памяти находятся два процесса

Удалим процесс *p1* из памяти:

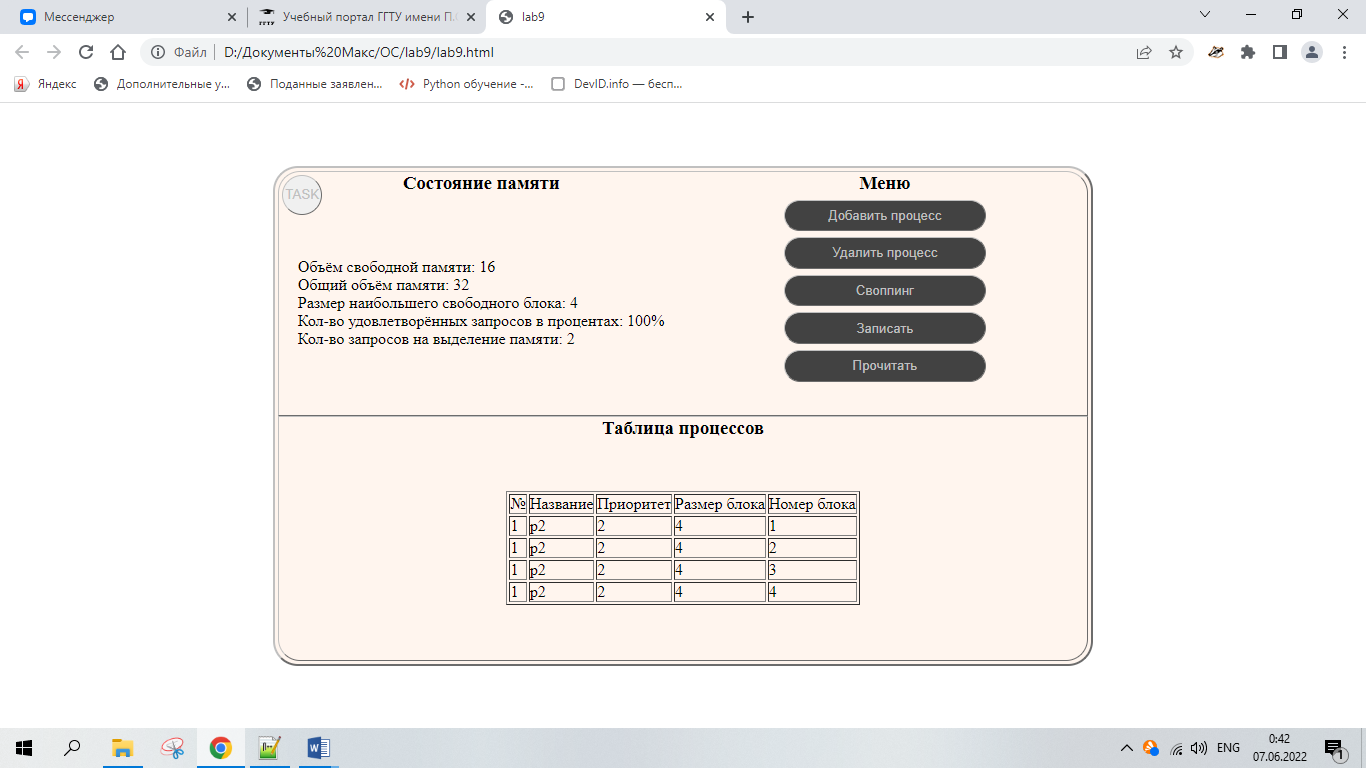


Рисунок 7 – Результат удаления процесса *p1*

Ниже представлена работа алгоритма свопинг:

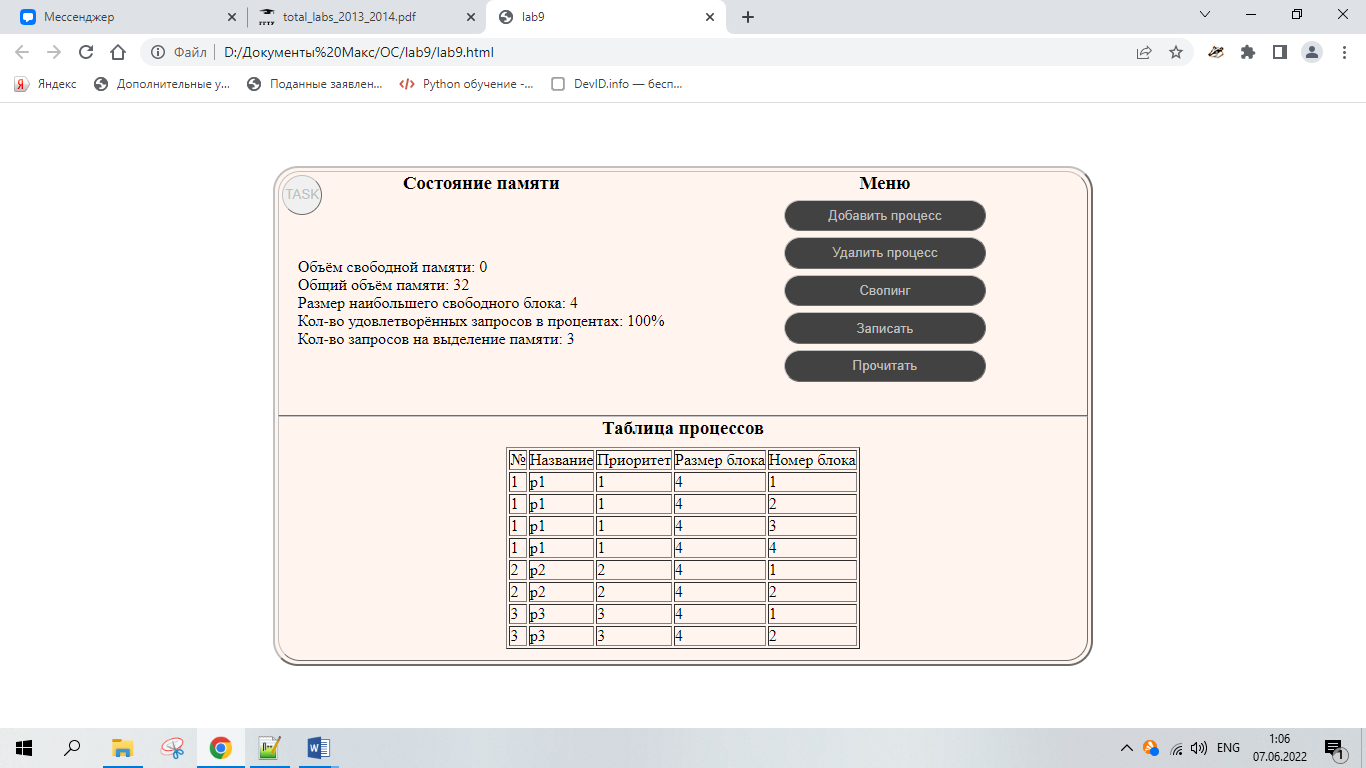


Рисунок 8 – Изначально задано 3 процесса

Добавим процесс *p4* с приоритетом 4 и размером 4, появится окно сообщением о том, что память заполнена, при нажатии на кнопку “Свопинг” будет удалён процесс *p1*, так как имеет наименьший приоритет:

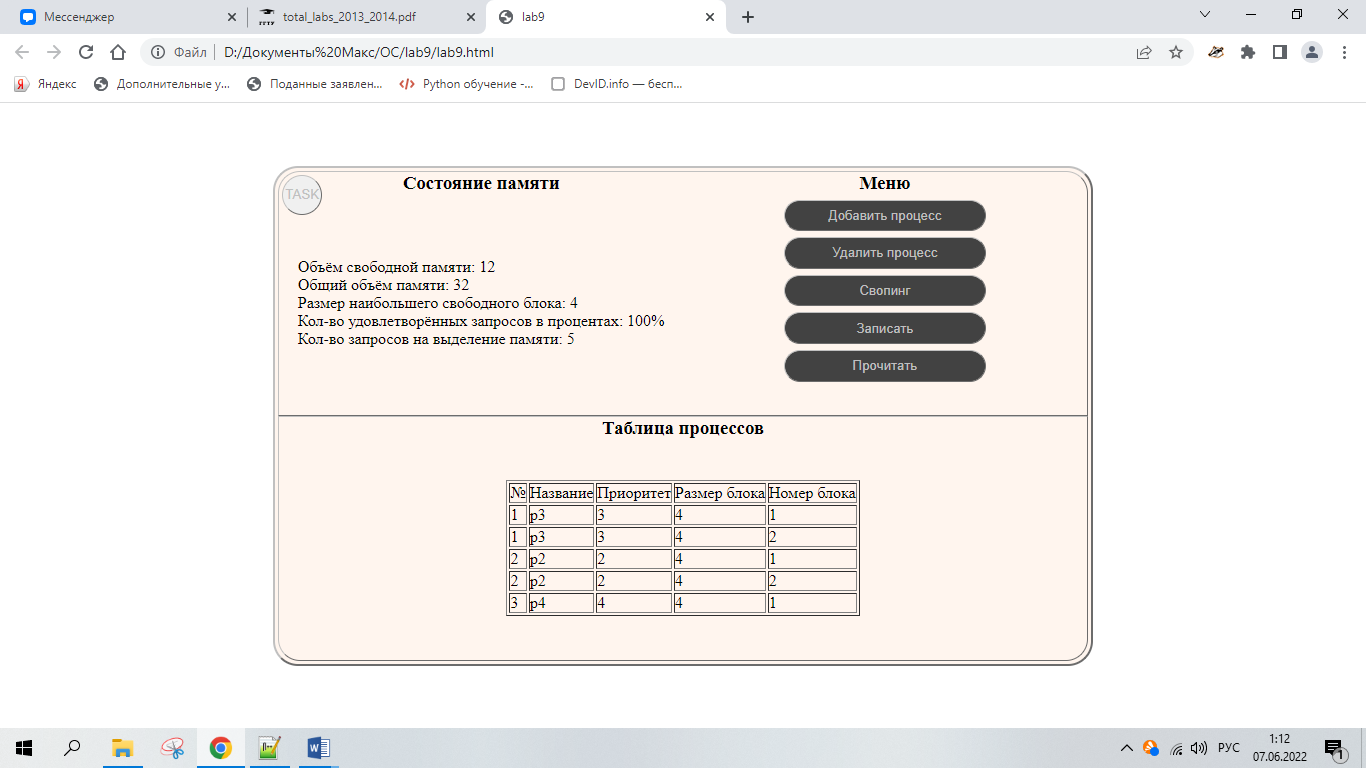


Рисунок 9 – Результат свопинга

Для симуляции работы схемы управления памятью с динамическими разделами– Свопинг– была создана отдельная кнопка “Свопинг”, по нажатию на которую из памяти удаляется процесс с наименьшим приоритетом, при условии, что последний добавленный процесс не помещается в памяти.

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

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

**Код js-документа**

let memory =

{

total: 32,

maxBlockSize: 4,

processes: [],

waitings: [],

queries: 0,

getAmountOfFreeMemory() {

let freeMemory = 0;

this.processes.forEach(process => freeMemory += process.size);

return this.total - freeMemory;

}

}

//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:\nСвопинг. Выгружается процесс, с наименьшим приоритетом.");

});

//Описание основных переменных

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

let del\_process=document.getElementById("del\_process");

let clean\_memory=document.getElementById("clean\_memory");

let processesTable=document.getElementById("table\_processes");

let memoryInfo=document.getElementById("memory\_info");

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

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

//Описание функций для кнопок

add\_process.addEventListener("click", (e) =>

{

addProcess();

});

del\_process.addEventListener("click", (e) =>

{

deleteProcess();

});

clean\_memory.addEventListener("click", (e) =>

{

doSwopping();

});

write.addEventListener("click", (e) =>

{

writeToProcess();

});

read.addEventListener("click", (e) =>

{

readFromProcess();

});

//функция удаления процесса

function deleteProcess() {

let name = prompt("Название процесса:");

if (!memory.processes.some(process => process.name == name)) {

alert("Такого процесса нет!");

return;

}

memory.processes = memory.processes.filter(process => process.name != name);

updateInfo();

}

//Функция записи в процесс

function writeToProcess() {

let name = prompt("Название процесса:");

if (!memory.processes.some(process => process.name == name)) {

alert("Такого процесса нет!");

return;

}

let max = 0;

let ind = 0;

memory.processes.forEach((process, i) =>{

if (process.name == name) {

max = process.size;

ind = i;

}

})

let val = prompt("Введите какие-нибудь символы(максимум "+max+"): ");

if (val.length > max) {

alert("Кол-во символов больше максимального!");

return;

}

memory.processes[ind].value = val;

}

//Функция чтения

function readFromProcess() {

let name = prompt("Название процесса:");

if (!memory.processes.some(process => process.name == name)) {

alert("Такого процесса нет!");

return;

}

let ind = 0;

memory.processes.forEach((process, i) =>{

if (process.name == name) {

ind = i;

}

})

alert(`Процессе с названием ${name} содержит следующую запись: ${memory.processes[ind].value}`);

}

//Функция выводящая сведения о процессах

function updateInfo() {

showProcessesTable();

showMemoryInfo();

}

//Функция добавления процесса

function addProcess(n, s, p) {

let name = n || prompt("Название: ");

let size = s || +prompt("Размер: ");

let priority = p || +prompt("Приоритет: ");

if (memory.getAmountOfFreeMemory() < size) {

alert("Память заполнена!");

memory.waitings.push({name, size, priority});

memory.queries++;

updateInfo();

return;

}

if (memory.processes.some(process => process.name == name)) {

alert("Процесс с таким названием уже есть!");

return;

}

let process = {name, size, priority, blocks: [], value: ""};

if (size > memory.maxBlockSize) {

let wholeIterations = Math.floor(size / memory.maxBlockSize);

let remains = size % memory.maxBlockSize;

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

process.blocks.push(memory.maxBlockSize);

if (remains)

process.blocks.push(remains);

} else

process.blocks.push(size);

memory.processes.push(process);

memory.queries++;

updateInfo();

}

//Функция выводящая таблицу процессов

function showProcessesTable() {

let str = "<table border='1'><thead><tr><td>№</td><td>Название</td><td>Приоритет</td><td>Размер блока</td><td>Номер блока</td></tr></thead><tbody>";

memory.processes.forEach((process, ind) => {

process.blocks.forEach((block, blockInd) => {

str += `<tr><td>${ind + 1}</td><td>${process.name}</td>

<td>${process.priority}</td><td>${block}</td><td>${blockInd + 1}</td></tr>`;

});

});

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

processesTable.innerHTML = str;

}

//Функция для выода информации о состоянии памяти

function showMemoryInfo() {

let maxFreeSize = -1;

let amountOfFulfilled = (100 \* memory.waitings.length) / (memory.waitings.length + memory.processes.length);

console.log(amountOfFulfilled);

memory.processes.forEach(process => {

process.blocks.forEach(block => {

if (maxFreeSize != -1 && memory.maxBlockSize - block > maxFreeSize)

maxFreeSize = memory.maxBlockSize - block;

else

maxFreeSize = block;

})

})

let str = `<p>Объём свободной памяти: ${memory.getAmountOfFreeMemory()}`;

str += `<br>Общий объём памяти: ${memory.total}<br>`;

str += `Размер наибольшего свободного блока: ${maxFreeSize}`;

str += `<br>Кол-во удовлетворённых запросов в процентах: ${memory.waitings.length != 0 ? amountOfFulfilled.toFixed(2) : 100}%`;

str += `<br>Кол-во запросов на выделение памяти: ${memory.queries}</p>`;

memoryInfo.innerHTML = str;

}

//Функция удаляющая процесс с наименьшим приоритетом

function doSwopping() {

if (!memory.waitings.length) {

alert("Нет ожидающих процессов!");

return;

}

let addedProcess = memory.waitings.pop();

let sortProcesses = memory.processes.sort((pr1, pr2) => pr2.priority - pr1.priority);

let del = "";

while (addedProcess.size > memory.getAmountOfFreeMemory()) {

del += sortProcesses.pop().name + ";";

}

memory.processes = sortProcesses;

alert("Были удалены следующие процессы: "+del);

console.log(sortProcesses);

addProcess(addedProcess.name, addedProcess.size, addedProcess.priority);

updateInfo();

}

**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>lab9</title>

</head>

<body>

<div id="wrapper">

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

<div id="main\_block">

<div id="memory\_condition">

<h3>Состояние памяти</h3>

<div id="memory\_info">

</div>

</div>

<div id="button\_box">

<h3>Меню</h3>

<button class="buttons" id="add\_process">Добавить процесс</button>

<button class="buttons" id="del\_process">Удалить процесс</button>

<button class="buttons" id="clean\_memory">Свопинг</button>

<button class="buttons" id="write">Записать</button>

<button class="buttons" id="read">Прочитать</button>

</div>

</div>

<div id="table\_block">

<h3>Таблица процессов</h3>

<div id="table\_processes">

</div>

</div>

</div>

<script src="lab9.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);

}

#main\_block{

height:50%;

width:100%;

border:1px outset #C0C0C0;

border-radius:22px 22px 0px 0px;

}

#memory\_condition{

height:100%;

width:50%;

float:left;

border-radius:22px 0px 0px 0px;

}

#memory\_info{

display:flex;

align-items:center;

justify-content:center;

height:90%;

width:100%;

}

#button\_box{

height:100%;

width:50%;

float:right;

border-radius:0px 22px 0px 0px;

}

.buttons{

display:block;

height:13%;

width:50%;

margin:1.5% auto;

border:1.5px outset #C0C0C0;

border-radius:20px;

color:#C0C0C0;

background-color:#424242;

transition:0.3s;

}

.buttons:hover{

color:#c6e2ff;

text-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);

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);

}

#table\_block{

height:50%;

width:100%;

border:1px outset #C0C0C0;

border-radius:0px 0px 22px 22px;

overflow:auto;

}

#table\_processes{

display:flex;

align-items:center;

justify-content:center;

height:90%;

width:100%;

}

h3{

text-align:center;

}