**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной техники**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Операционные системы»**

**Тема: «Межпроцессное взаимодействие»**

Студента гр. 3312 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Поляков А.И.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимофеев А.В.

Санкт-Петербург

2025

**Цель работы.**

исследовать инструменты и механизмы взаимодействия процессов в Windows.

**Задание 4.1.**

Реализация решения задачи о читателях-писателях.

**Указания к выполнению.**

1. Выполнить решение задачи о читателях-писателях, для чего необходимо разработать консольные приложения «Читатель» и «Писатель»: одновременно запущенные экземпляры процессов-читателей и процессов писателей должны совместно работать с буферной памятью в виде проецируемого файла:

o размер страницы буферной памяти равен размеру физической страницы оперативной памяти;

o число страниц буферной памяти равно сумме цифр в номере студенческого билета без учета первой цифры.

страницы буферной памяти должны быть заблокированы в оперативной памяти (функция VirtualLock);

длительность выполнения процессами операций «чтения» и «записи» задается случайным образом в диапазоне от 0,5 до 1,5 сек.;

для синхронизации работы процессов необходимо использовать объекты синхронизации типа «семафор» или «мьютекс»;

процессы-читатели и процессы-писатели ведут свои журнальные файлы, в которые регистрируют переходы из одного «состояния» в другое (начало ожидания, запись или чтение, переход к освобождению) с указанием кода времени (функция TimeGetTime). Для состояний «запись» и «чтение» необходимо также запротоколировать номер рабочей страницы.

2. Запустите приложения читателей и писателей, суммарное количество одновременно работающих читателей и писателей должно быть не менее числа страниц буферной памяти. Проверьте функционирование приложений, проанализируйте журнальные файлы процессов, постройте сводные графики смены «состояний» для не менее 5 процессов-читателей и 5 процессов-писателей, дайте свои комментарии относительно переходов процессов из одного состояния в другое.

3. Постройте графики в Excel/Python и дайте свои комментарии:

 Смена состояний процессов (пример: читатель 1 → ожидание → чтение

→ освобождение).

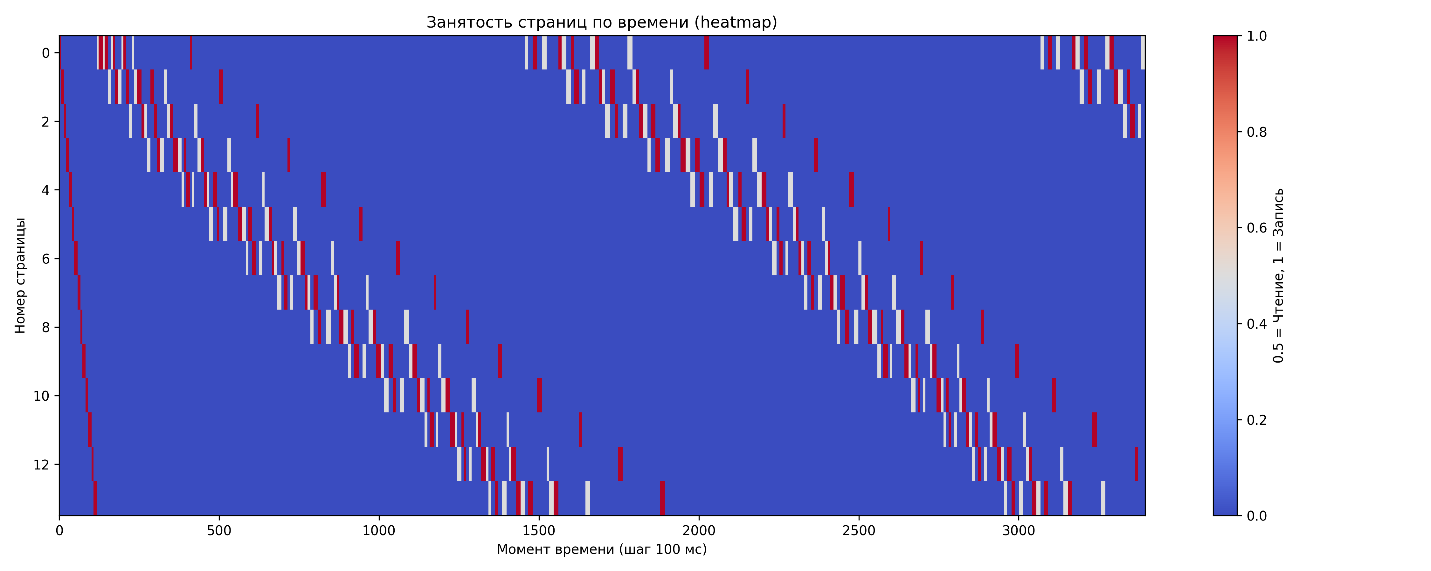
 Занятость страниц во времени (визуализация через heatmap).

4. Подготовьте итоговый отчет с развернутыми выводами по

заданию.

Комментарии к графикам

1. **График "Занятость страниц по времени (heatmap)"**

****

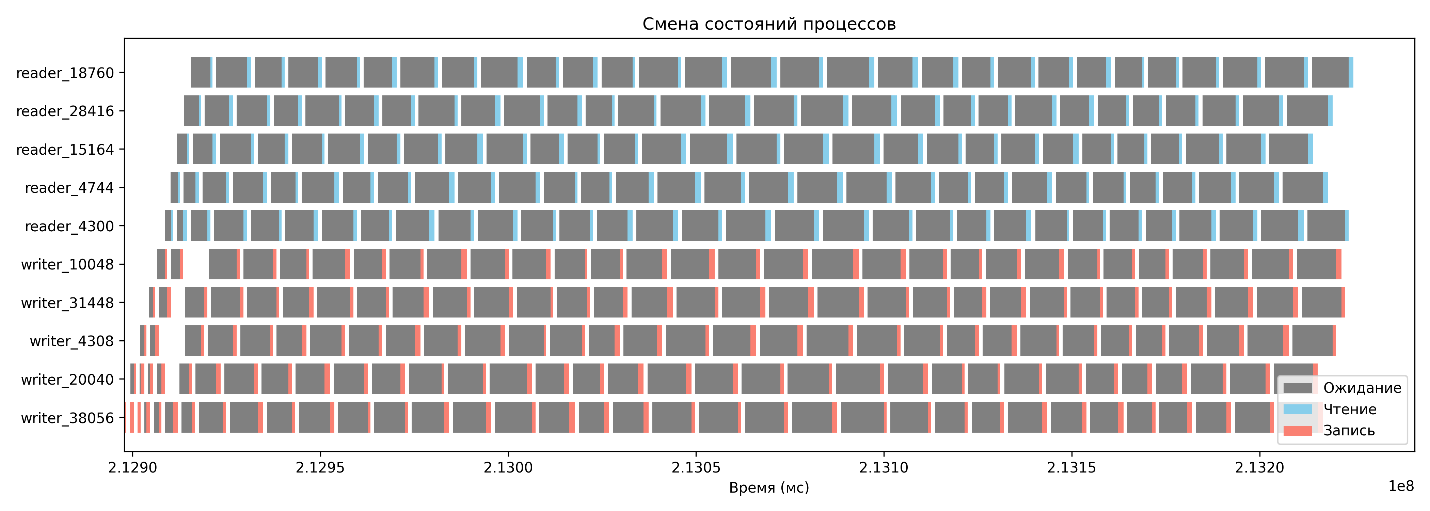
**Описание:**

* Ось X: моменты времени (шаг — 100 мс).
* Ось Y: номера страниц.
* Цвет:
  + Синий (0.0): нет активности.
  + Белый (0.5): чтение.
  + Красный (1.0): запись.

**Комментарии:**

* Активность равномерно распределена по страницам — writer и reader обходят страницы по кругу, избегая постоянной работы с одной и той же страницей.
* Записи происходят с заметно меньшей частотой, чем чтения. Это связано с тем, что writer дожидается освобождения страниц, и его паузы между итерациями длиннее, особенно при отсутствии свободных страниц.
* Чтения зачастую сгруппированы — можно заметить, что на некоторых страницах чтение происходит более плотно. Это указывает на периодическое накопление записей и последующее «снятие» их блоками.
* Отсутствие наложений чтения и записи подтверждает корректную синхронизацию. Благодаря использованию Mutex, writer и reader поочерёдно получают доступ к разделяемой памяти. Это особенно видно по чёткому чередованию белых и красных блоков без наложений.
* Цикличный характер использования страниц — диагональный узор на графике показывает, что страницы используются последовательно и по кругу, что соответствует логике lastPage в коде.

1. **График "Смена состояний процессов"**

****

**Описание:**

* Ось X: время (в мс).
* Ось Y: процессы (читатели и писатели).
* Цвета:
  + Серый — **ожидание**.
  + Голубой — **чтение**.
  + Красный — **запись**.

**Что видно:**

* У читателей (reader\_\*) чередуются периоды ожидания и чтения.
* У писателей (writer\_\*) — ожидание и запись.
* Видно, что чтения и записи не перекрываются — это признак правильной синхронизации (один процесс в критической секции в каждый момент времени).
* Иногда у писателей длительные периоды ожидания (серый), что говорит о том, что **чтения "захватывают" мьютекс** на продолжительное время.
* У читателей состояние чтения (голубой) бывает плотным и частым, особенно у некоторых потоков — возможно, они активнее других.
* Баланс между чтением и записью немного смещён в сторону чтения, что логично, т.к. в типичных системах читателей больше, чем писателей.

**Вывод по переходам состояний:**

1. **Все процессы корректно используют мьютекс** — видно, что чтение и запись не перекрываются.
2. **Писатели чаще находятся в ожидании**, потому что читающие процессы занимают мьютекс.
3. **Чтения распределены по времени плотнее и активнее**, что указывает на их высокую активность.
4. **Состояния "Ожидание → Чтение/Запись → Ожидание"** происходят циклично у всех процессов, что соответствует логике кода (захват мьютекса — работа — отпуск — пауза).

**Выводы**

По результатам визуализации можно сделать вывод, что система синхронизации между процессами (читателями и писателями) работает корректно. Все переходы между состояниями происходят в строгом соответствии с логикой "читатель–писатель":

* В каждый момент времени в критической секции находится только один процесс — либо читатель, либо писатель.
* Параллельного чтения нескольких читателей не происходит, что подтверждает использование обычного мьютекса вместо более сложной reader-writer блокировки.
* Процессы последовательно захватывают ресурс, ожидая своей очереди. Особенно это заметно у писателей, которым приходится дольше ждать из-за частых чтений.
* Распределение активности по страницам достаточно равномерное, но наблюдаются "горячие" зоны — отдельные страницы используются чаще.
* Программа демонстрирует реалистичную модель конкуренции, где чтения преобладают, а записи происходят реже, но требуют исключительного доступа.

Таким образом, поведение системы подтверждает правильность реализации модели с точки зрения взаимного исключения и чередования доступа к разделяемому ресурсу.

Текст программы

Writer

#include "shared.h"

#include <iostream>

#include <fstream>

#include <random>

#include <thread>

int main()

{

HANDLE hMutex = CreateMutexW(nullptr, FALSE, MUTEX\_NAME);

if (!hMutex)

{

std::wcerr << L"Ошибка CreateMutex: " << GetLastError() << std::endl;

return 1;

}

HANDLE hMapFile = CreateFileMappingW(INVALID\_HANDLE\_VALUE, nullptr, PAGE\_READWRITE, 0, TOTAL\_SIZE, FILE\_MAPPING\_NAME);

if (!hMapFile)

{

std::wcerr << L"Ошибка CreateFileMapping: " << GetLastError() << std::endl;

return 1;

}

LPVOID pBase = MapViewOfFile(hMapFile, FILE\_MAP\_WRITE, 0, 0, 0);

if (!pBase)

{

std::wcerr << L"Ошибка MapViewOfFile: " << GetLastError() << std::endl;

return 1;

}

VirtualLock(pBase, TOTAL\_SIZE);

BYTE\* meta = (BYTE\*)pBase;

char\* dataBase = (char\*)pBase + META\_SIZE;

std::wstring logFile = L"writer\_" + std::to\_wstring(GetCurrentProcessId()) + L".log";

int lastPage = 0;

while (true)

{

DWORD tickStart = timeGetTime();

LogToFile(logFile, L"[WAITING] " + std::to\_wstring(tickStart));

WaitForSingleObject(hMutex, INFINITE);

int page = -1;

for (int i = 0; i < PAGE\_COUNT; ++i)

{

int candidate = (lastPage + i) % PAGE\_COUNT;

if (meta[candidate] == 0)

{

page = candidate;

meta[page] = 1;

lastPage = (page + 1) % PAGE\_COUNT;

break;

}

}

if (page == -1)

{

ReleaseMutex(hMutex);

std::this\_thread::sleep\_for(std::chrono::milliseconds(500));

continue;

}

char\* pageAddr = dataBase + (page \* PAGE\_SIZE);

DWORD tickWrite = timeGetTime();

LogToFile(logFile, L"[WRITING] Page " + std::to\_wstring(page) + L" at " + std::to\_wstring(tickWrite));

std::string data = "PID " + std::to\_string(GetCurrentProcessId());

memcpy(pageAddr, data.c\_str(), min(data.size(), PAGE\_SIZE - 1));

std::this\_thread::sleep\_for(std::chrono::milliseconds(500 + rand() % 1000));

DWORD tickRelease = timeGetTime();

LogToFile(logFile, L"[RELEASE] " + std::to\_wstring(tickRelease));

ReleaseMutex(hMutex);

std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

}

return 0;

}

Reader

#include "shared.h"

#include <iostream>

#include <fstream>

#include <random>

#include <thread>

int main()

{

HANDLE hMutex = OpenMutexW(MUTEX\_ALL\_ACCESS, FALSE, MUTEX\_NAME);

if (!hMutex)

{

std::wcerr << L"Ошибка OpenMutex: " << GetLastError() << std::endl;

return 1;

}

HANDLE hMapFile = OpenFileMappingW(FILE\_MAP\_READ | FILE\_MAP\_WRITE, FALSE, FILE\_MAPPING\_NAME);

if (!hMapFile)

{

std::wcerr << L"Ошибка OpenFileMapping: " << GetLastError() << std::endl;

return 1;

}

LPVOID pBase = MapViewOfFile(hMapFile, FILE\_MAP\_READ | FILE\_MAP\_WRITE, 0, 0, 0);

if (!pBase)

{

std::wcerr << L"Ошибка MapViewOfFile: " << GetLastError() << std::endl;

return 1;

}

VirtualLock(pBase, TOTAL\_SIZE);

BYTE\* meta = (BYTE\*)pBase;

char\* dataBase = (char\*)pBase + META\_SIZE;

std::wstring logFile = L"reader\_" + std::to\_wstring(GetCurrentProcessId()) + L".log";

int lastPage = 0;

while (true)

{

DWORD tickStart = timeGetTime();

LogToFile(logFile, L"[WAITING] " + std::to\_wstring(tickStart));

WaitForSingleObject(hMutex, INFINITE);

int page = -1;

for (int i = 0; i < PAGE\_COUNT; ++i)

{

int candidate = (lastPage + i) % PAGE\_COUNT;

if (meta[candidate] == 1)

{

page = candidate;

meta[page] = 0;

lastPage = (page + 1) % PAGE\_COUNT;

break;

}

}

if (page == -1)

{

ReleaseMutex(hMutex);

std::this\_thread::sleep\_for(std::chrono::milliseconds(500));

continue;

}

char\* pageAddr = dataBase + (page \* PAGE\_SIZE);

DWORD tickRead = timeGetTime();

LogToFile(logFile, L"[READING] Page " + std::to\_wstring(page) + L" at " + std::to\_wstring(tickRead));

std::string readData(pageAddr, pageAddr + strnlen\_s(pageAddr, PAGE\_SIZE));

std::wcout << L"[" << GetCurrentProcessId() << L"] Прочитал с страницы " << page << L": "

<< std::wstring(readData.begin(), readData.end()) << std::endl;

std::this\_thread::sleep\_for(std::chrono::milliseconds(500 + rand() % 1000));

DWORD tickRelease = timeGetTime();

LogToFile(logFile, L"[RELEASE] " + std::to\_wstring(tickRelease));

ReleaseMutex(hMutex);

std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

}

return 0;

}

Shared.cpp

#pragma once

#include <windows.h>

#include <string>

const wchar\_t\* FILE\_MAPPING\_NAME = L"Global\MySharedMemory";

const wchar\_t\* MUTEX\_NAME = L"Global\PageMutex";

const size\_t PAGE\_SIZE = 4096;

const int PAGE\_COUNT = 14;

const size\_t META\_SIZE = PAGE\_COUNT;

const size\_t TOTAL\_SIZE = META\_SIZE + PAGE\_COUNT \* PAGE\_SIZE;

inline void LogToFile(const std::wstring& filename, const std::wstring& message)

{

FILE\* f;

\_wfopen\_s(&f, filename.c\_str(), L"a+");

if (f)

{

fwprintf(f, L"%s\n", message.c\_str());

fclose(f);

}

}