**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: "Смена состояний процессов"**

**Описание:**

* На вертикальной оси — идентификаторы процессов: reader\_xxxxx и writer\_xxxxx.
* На горизонтальной оси — время (в миллисекундах).

**Комментарии:**

* График показывает, как процессы читают и записывают данные с использованием общей памяти.
* Все процессы проводят значительное время в состоянии "Ожидание", что типично при использовании мьютекса для синхронизации доступа.
* Чтение и запись происходят короткими всплесками между длинными периодами ожидания.

**2: "Занятость страниц по времени (heatmap)"**

**Описание:**

* Вертикальная ось — номера страниц в общей памяти.
* Горизонтальная ось — момент времени (с шагом 100 мс).

**Комментарии:**

* Чтение (светлые полоски) и запись (ярко-красные полосы) происходят на разных страницах в случайные моменты.
* Нет систематического конфликта между чтением и записью: это подтверждает, что защита доступа (мьютекс) работает корректно.
* Некоторые страницы используются чаще, из-за случайного выбора страниц в генераторе случайных чисел (std::mt19937).

**Комментарий по коду**

Код представлен в двух частях:

1. **Writer**:
   * Создаёт и мапит файл общей памяти.
   * Пишет в случайную страницу после захвата мьютекса.
   * Записывает лог с отметками времени (ожидание, запись, освобождение мьютекса).
   * Использует VirtualLock, чтобы не выгружать страницы.
2. **Reader**:
   * Открывает существующий мьютекс и разделяемую память.
   * Читает случайную страницу после захвата мьютекса.
   * Логирует действия аналогично writer-процессу.

**Вывод**В ходе эксперимента была реализована система взаимодействия потоков чтения и записи через разделяемую память с применением мьютекса для синхронизации. Анализ логов и визуализация данных показали, что синхронизация работает корректно: одновременный доступ к памяти не происходит, состояния процессов последовательно чередуются, не создавая гонок и конфликтов.

Графики подтвердили, что:

* доступ к памяти строго упорядочен благодаря использованию одного мьютекса;
* чтение и запись происходят равномерно, без систематических задержек или "зависаний";
* процессы эффективно используют общее время выполнения, чередуя активные и пассивные фазы;
* страницы памяти используются равномерно и случайно, что соответствует логике генератора случайных индексов.

Таким образом, поставленная цель — продемонстрировать корректную работу синхронизированного доступа к общей памяти несколькими процессами — была успешно достигнута. Реализация может служить базой для построения более сложных моделей межпроцессного взаимодействия.

**Текст программ**

**Writer**

#include "header.h"

#include <iostream>

#include <fstream>

#include <random>

#include <thread>

HANDLE InitMutex()

{

HANDLE mutex = CreateMutexW(nullptr, FALSE, SYNC\_OBJECT\_NAME);

if (!mutex)

{

std::wcerr << L"[InitMutex] Ошибка: " << GetLastError() << std::endl;

}

return mutex;

}

HANDLE InitMapping(HANDLE& outHandle, LPVOID& outMemory)

{

outHandle = CreateFileMappingW(INVALID\_HANDLE\_VALUE, nullptr, PAGE\_READWRITE, 0, TOTAL\_PAGES \* PAGE\_SIZE, SHARED\_MEMORY\_NAME);

if (!outHandle)

{

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

return nullptr;

}

outMemory = MapViewOfFile(outHandle, FILE\_MAP\_WRITE, 0, 0, 0);

if (!outMemory)

{

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

return nullptr;

}

VirtualLock(outMemory, TOTAL\_PAGES \* PAGE\_SIZE);

return outHandle;

}

void WriteToRandomPage(LPVOID memoryBase, const std::wstring& logFile, std::mt19937& rng)

{

DWORD tickStart = timeGetTime();

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

WaitForSingleObject(InitMutex(), INFINITE);

int pageIndex = rng() % TOTAL\_PAGES;

char\* pagePtr = static\_cast<char\*>(memoryBase) + (pageIndex \* PAGE\_SIZE);

DWORD tickWrite = timeGetTime();

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

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

memcpy(pagePtr, text.c\_str(), min(text.size(), PAGE\_SIZE - 1));

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

DWORD tickRelease = timeGetTime();

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

ReleaseMutex(InitMutex());

}

int main()

{

HANDLE mutex = InitMutex();

if (!mutex) return 1;

HANDLE mapping;

LPVOID memory;

if (!InitMapping(mapping, memory)) return 1;

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

std::mt19937 rng(static\_cast<unsigned>(time(nullptr)));

while (true)

{

WriteToRandomPage(memory, logFile, rng);

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

}

return 0;

}

**Reader**

#include "header.h"

#include <iostream>

#include <fstream>

#include <random>

#include <thread>

HANDLE GetExistingMutex()

{

HANDLE mutex = OpenMutexW(MUTEX\_ALL\_ACCESS, FALSE, SYNC\_OBJECT\_NAME);

if (!mutex)

{

std::wcerr << L"[GetExistingMutex] Ошибка: " << GetLastError() << std::endl;

}

return mutex;

}

HANDLE GetSharedMemory(LPVOID& outPtr)

{

HANDLE mappingHandle = OpenFileMappingW(FILE\_MAP\_READ, FALSE, SHARED\_MEMORY\_NAME);

if (!mappingHandle)

{

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

return nullptr;

}

outPtr = MapViewOfFile(mappingHandle, FILE\_MAP\_READ, 0, 0, 0);

if (!outPtr)

{

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

return nullptr;

}

VirtualLock(outPtr, TOTAL\_PAGES \* PAGE\_SIZE);

return mappingHandle;

}

void ReadRandomPage(LPVOID memoryBase, const std::wstring& logFile, std::mt19937& rng)

{

DWORD tickBeforeWait = timeGetTime();

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

HANDLE mutex = GetExistingMutex();

WaitForSingleObject(mutex, INFINITE);

int pageIndex = rng() % TOTAL\_PAGES;

const char\* page = static\_cast<char\*>(memoryBase) + (pageIndex \* PAGE\_SIZE);

DWORD tickRead = timeGetTime();

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

std::string content(page, page + strnlen\_s(page, PAGE\_SIZE));

std::wcout << L"[" << GetCurrentProcessId() << L"] Чтение страницы " << pageIndex << L": "

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

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

DWORD tickUnlock = timeGetTime();

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

ReleaseMutex(mutex);

}

int main()

{

HANDLE mutex = GetExistingMutex();

if (!mutex) return 1;

HANDLE mapping;

LPVOID sharedMemory;

if (!GetSharedMemory(sharedMemory)) return 1;

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

std::mt19937 generator(static\_cast<unsigned>(time(nullptr)));

while (true)

{

ReadRandomPage(sharedMemory, logPath, generator);

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

}

return 0;

}

**Header**

#pragma once

#include <windows.h>

#include <string>

const wchar\_t\* SHARED\_MEMORY\_NAME = L"Global\\CustomSharedMem";

const wchar\_t\* SYNC\_OBJECT\_NAME = L"Global\\AccessMutex";

const size\_t PAGE\_SIZE = 4096;

const int TOTAL\_PAGES = 12;

inline void LogToFile(const std::wstring& filepath, const std::wstring& entry)

{

FILE\* file;

\_wfopen\_s(&file, filepath.c\_str(), L"a+");

if (file)

{

fwprintf(file, L"%s\n", entry.c\_str());

fclose(file);

}

}