Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №4

на тему

**ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ (ПОТОКОВ): ВЗАИМНОЕ ИСКЛЮЧЕНИЕ И СИНХРОНИЗАЦИЯ**

Выполнил: студент гр.253505 Гимпель К.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc178780699)

[2 Краткие теоритические сведения 4](#_Toc178780700)

[3 Результат запуска программы 5](#_Toc178780701)

[3.1 Шаги запуска программы 5](#_Toc178780702)

[3.2 Иллюстрация запуска и работы программы 5](#_Toc178780703)

Заключение 6

Список использованных источников 7

Приложение А (обязательное) Исходный код программы 8

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является изучение и практическое применение механизмов взаимного исключения и синхронизации в многопоточных приложениях с использованием WinAPI. В процессе выполнения работы необходимо разработать приложение, которое демонстрирует корректное и согласованное взаимодействие потоков в модели «Писатели-читатели». Основные задачи включают:

– Реализация модели «Писатели-читатели» с возможностью параметризации;

– Анализ корректности работы модели (отсутствие коллизий);

– Наглядное представление результатов.

Специальных требований к приложению не предъявляется; в частности, оно может быть не обязательно оконными, но также и консольным.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Взаимодействие процессов (потоков) в многозадачных системах требует решения задач взаимного исключения и синхронизации для предотвращения некорректного совместного доступа к критическим ресурсам. Критический ресурс – это общий ресурс, к которому могут обращаться несколько потоков, а критическая секция – это участок кода, где происходит доступ к такому ресурсу. Проблема атомарности доступа для проверки и модификации глобальных объектов, используемых для синхронизации, решается с помощью механизмов синхронизации, таких как CRITICAL\_SECTION, Mutex и Semaphore в WinAPI.

В WinAPI для обеспечения синхронизации используются различные механизмы. CRITICAL\_SECTION обеспечивает взаимное исключение для потоков внутри одного процесса, что позволяет избежать конфликтов при доступе к общим данным. Mutex расширяет эту возможность на взаимодействие между разными процессами. Semaphore ограничивает количество потоков, которые могут одновременно обращаться к ресурсу, а Event позволяет потокам ожидать наступления определенного события. Функции ожидания, такие как WaitForSingleObject, используются для ожидания освобождения объектов синхронизации.

Теоретические модели для описания ситуаций синхронизации, такие как модель «Писатели-читатели», помогают понять и решить проблемы, связанные с параллельным доступом к общим ресурсам. В этой модели несколько потоков (читатели) могут одновременно читать данные, но только один поток (писатель) может писать данные в любой момент времени. Это обеспечивает корректность работы системы и предотвращает грязное чтение данных.

Данные механизмы взаимодействия процессов и синхронизации подробно рассмотрены в книгах А. Таненбаума «Современные операционные системы» [1] и А. Сильбершаца «Операционные системы: принципы и практика» [2], где описываются теоретические аспекты и практическая реализация межпроцессного взаимодействия в современных ОС.

3 РЕЗУЛЬТАТ ЗАПУСКА ПРОГРАММЫ

## 3.1 Шаги запуска программы

Программа для модели "Писатели-читатели" была успешно разработана, скомпилирована и запущена. Основные шаги запуска включают:

– Открытие командной строки;

– Компиляцию файла main.cpp: g++ main.cpp -o main.exe;

– Запуск программы main.exe.

В процессе работы программы писатели и читатели параллельно обращаются к общему массиву данных. Писатели записывают данные в массив, а читатели считывают их. Программа обеспечивает корректное взаимодействие между потоками, предотвращая грязное чтение и запись данных.

## 3.2 Иллюстрация запуска и работы программы

На рисунке 3.1 показан консольный вывод программы. Писатели и читатели работают параллельно, записывая и считывая данные из общего массива. Программа корректно синхронизирует доступ к ресурсам, предотвращая коллизии.

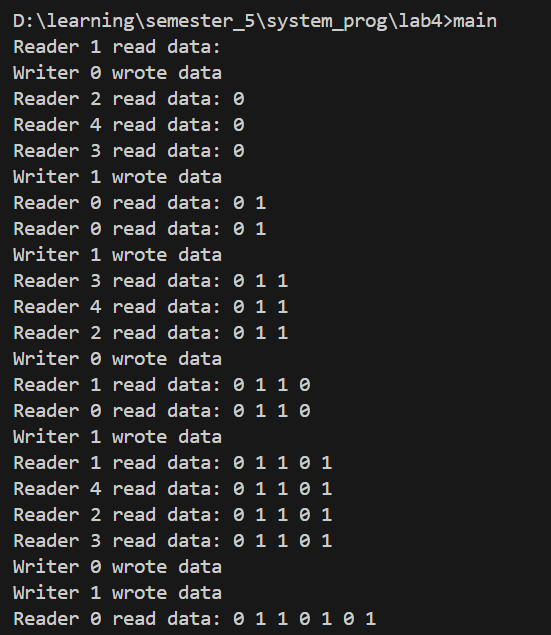


Рисунок 3.1 – Результат работы программ

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы по дисциплине «Системное программирование» была разработана и реализована модель «Писатели-читатели» с использованием WinAPI. Основная задача заключалась в демонстрации параллельной согласованной работы процессов (потоков) и их взаимодействия с использованием механизмов синхронизации.

В процессе выполнения работы были изучены и применены механизмы взаимного исключения и синхронизации, такие как CRITICAL\_SECTION, Mutex, Semaphore, и Event. Программа была успешно протестирована, и результаты показали корректность и эффективность используемых механизмов синхронизации.

Лабораторная работа не только углубила теоретические знания о способах взаимодействия процессов, но и предоставила практические навыки, которые являются основой для разработки надёжных и эффективных систем многопоточности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Таненбаум Э. С. Современные операционные системы. — 4-е изд. — СПб.: Питер, 2018. — 1120 с.

[2] Сильбершац А., Галвин П., Гейдж Г. Операционные системы: принципы и практика. — 9-е изд. — М.: Вильямс, 2015. — 1056 с.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <windows.h>

#include <iostream>

#include <vector>

#include <thread>

#include <mutex>

#include <condition\_variable>

CRITICAL\_SECTION cs;

HANDLE hMutex;

HANDLE hSemaphoreReaders;

HANDLE hSemaphoreWriters;

std::vector<int> sharedData;

int numReaders = 5;

int numWriters = 2;

int maxReaders = 3;

int maxWriters = 1;

void WriterThread(int id) {

while (true) {

WaitForSingleObject(hSemaphoreWriters, INFINITE);

EnterCriticalSection(&cs);

sharedData.push\_back(id);

std::cout << "Writer " << id << " wrote data\n";

LeaveCriticalSection(&cs);

ReleaseSemaphore(hSemaphoreWriters, 1, NULL);

Sleep(1000);

}

}

void ReaderThread(int id) {

while (true) {

WaitForSingleObject(hSemaphoreReaders, INFINITE);

EnterCriticalSection(&cs);

std::cout << "Reader " << id << " read data: ";

for (int data : sharedData) {

std::cout << data << " ";

}

std::cout << "\n";

LeaveCriticalSection(&cs);

ReleaseSemaphore(hSemaphoreReaders, 1, NULL);

Sleep(1000);

}

}

int main() {

InitializeCriticalSection(&cs);

hMutex = CreateMutex(NULL, FALSE, NULL);

hSemaphoreReaders = CreateSemaphore(NULL, maxReaders, maxReaders, NULL);

hSemaphoreWriters = CreateSemaphore(NULL, maxWriters, maxWriters, NULL);

std::vector<std::thread> writers;

for (int i = 0; i < numWriters; ++i) {

writers.push\_back(std::thread(WriterThread, i));

}

std::vector<std::thread> readers;

for (int i = 0; i < numReaders; ++i) {

readers.push\_back(std::thread(ReaderThread, i));

}

for (auto& writer : writers) {

writer.join();

}

for (auto& reader : readers) {

reader.join();

}

DeleteCriticalSection(&cs);

CloseHandle(hMutex);

CloseHandle(hSemaphoreReaders);

CloseHandle(hSemaphoreWriters);

return 0;

}