Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ-7)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Исследование применения потоков для решения задачи «Читатели-писатели»

Студент ИУ7-78 (Группа)	(Подпись, дата)	А.В. Иванников (И.О.Фамилия)
Руководитель курсовой работы	(Подпись, дата)	Н.Ю. Рязанова (И.О.Фамилия)
Консультант	(Подпись, дата)	Н.Ю. Рязанова (И.О.Фамилия)

2020 г.

Содержание

Введение

На сегодняшний день все современные операционные системы (далее в тексте – ОС) используют алгоритмы и методы синхронизации процессов и потоков.

Процесс обычно определяют как экземпляр выполняемой программы. В каждом процессе есть минимум один поток.

Поток является программным компонентом, который выполняется независимо от других потоков, но решает свои задачи в общей области памяти. Потоки возникли в ОС как средство распараллеливания вычислений.

Данная работа посвящена изучению основ работы с потоками в ОС Windows и методам их синхронизации.

В аналитическом разделе курсовой работы будет рассмотрена постановка задачи и методы ее решения.

В конструкторском разделе курсовой работы будет приведено описание структуры приложения.

В технологическом разделе курсовой работы будет рассмотрен выбор языка программирования и интегрированной среды разработки приложения.

В исследовательском разделе будет исследовано изменение времени выполнения приложения при увеличении количества «читателей» с фиксированным количеством «писателей» и при увеличении количества «писателей» с фиксированным количеством «читателей».

В заключении приводятся общие выводы о проделанной работе.

В приложении приведен листинг программного кода приложения.

1. Аналитический раздел

1.1. Постановка задачи

В соответствии с заданием на курсовую работу необходимо провести исследование применения потоков для решения задачи «Читатели - писатели». Схема задачи приведена на рисунке 1.

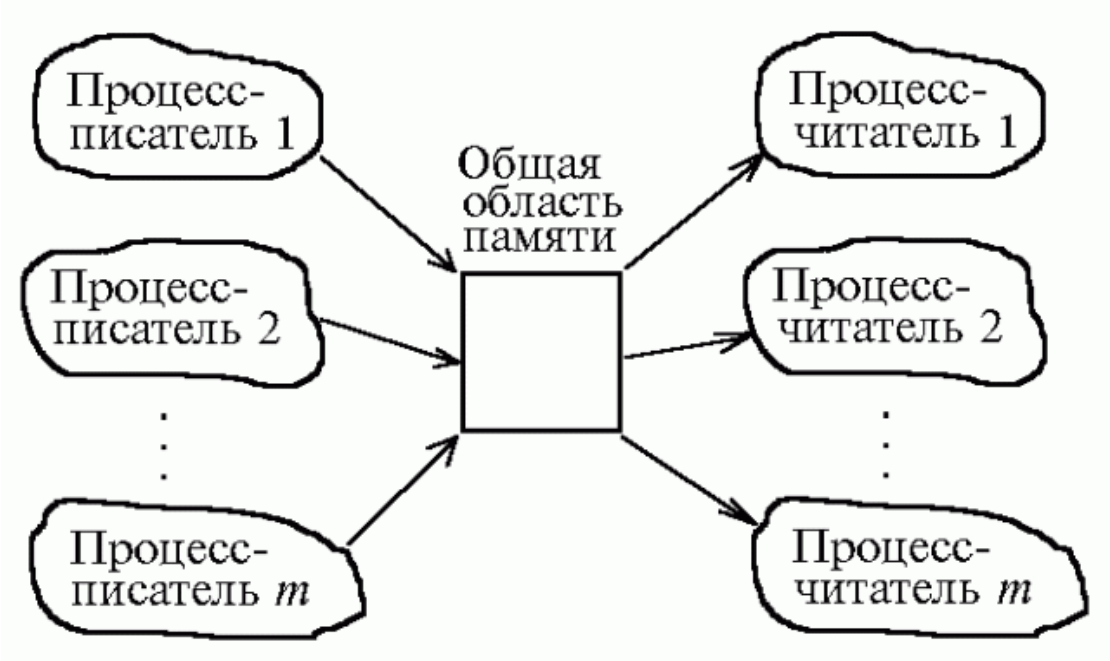

Рисунок 1 – Схема задачи «Читатели – писатели»

Для решения поставленной задачи необходимо проанализировать:

- особенности потоков в ОС Windows;

- реализацию задачи «Читатели-писатели»;

- средства синхронизации потоков ОС Windows.

1.2 Анализ особенностей потоков ОС Windows

Любой поток состоит из двух компонентов:

- объекта ядра, через который ОС управляет потоком. Там же хранится статистическая информация о потоке;

- стека потока, который содержит параметры всех функций и локальные переменные, необходимые потоку для выполнения кода.

Потоки всегда создаются в контексте какого-либо процесса. На практике это означает, что потоки исполняют код и манипулируют данными в адресном пространстве процесса. Поэтому, если два и более потоков выполняется в контексте одного процесса, все они делят одно адресное пространство. Потоки могут исполнять один и тот же код и манипулировать одними и теми же данными.

У каждого потока собственный набор регистров процессора, называемый контекстом потока. Контекст отражает состояние регистров процессора на момент последнего исполнения потока.

При завершении потока объект ядра «поток» переводится в свободное состояние. Если данный поток является последним активным потоком в процессе, завершается и сам процесс. При завершении потока сопоставленный с ним объект ядра «поток» не освобождается до тех пор, пока не будут закрыты все внешние ссылки на этот объект.

1.3 Анализ реализации задачи «Читатели – писатели»

Задача «Читатели–писатели» заключается в обеспечении согласованного доступа нескольких потоков к разделяемым данным.

Является одной из важнейших задач параллельного программирования, в которой моделируется доступ к общей базе данных.

Особенность задачи заключается в наличии двух типов потоков:

- «читатель» — только читает данные, не меняет содержимого базы данных, поэтому несколько таких потоков могут обращаться к такой базе данных одновременно;

- «писатель» — модифицирует данные, может изменять данные в базе, и поэтому он должен обладать к ней исключительным, монопольным доступом.

Пока с базой данных работает «писатель» — никакие другие потоки (и «писатели» и «читатели») работать с базой данных не должны. Такой режим монопольного доступа организуется для отдельных записей баз данных, а не для базы данных целиком.

Корректное решение задачи должно удовлетворять следующим условиям:

- потоки выполняются параллельно;

- во время выполнения потоком операции записи, данные не могут использоваться другими потоками;

- во время выполнения потоком операции чтения, другие потоки также могут выполнять операцию чтения;

- потоки должны завершить работу в течение конечного времени.

1.4 Анализ средств взаимоисключения ОС Windows

В ОС Windows существуют следующие средства взаимоисключения:

– событие (event):

уведомляет об окончании какой-либо операции. Содержит счетчик числа пользователей и две булевы переменные: одна сообщает тип данного объекта-события, другая — его состояние (свободен или занят). Бывает двух типов: со сбросом вручную (manual–reset events) и с автосбросом (auto–reset events). Первые позволяют возобновлять выполнение сразу нескольких ждущих потоков, вторые — только одного;

–семафор (semaphore):

введен Эдсгером Вибе Дейкстрой в 1965 году. Содержит счетчик числа пользователей и поддерживает два 32–битных значения со знаком: одно определяет максимальное число ресурсов (контролируемое семафором и равное 5), другое используется как счетчик текущего числа ресурсов.

Алгоритм работы семафора приведен на рисунке 2.

Рисунок 2 – Алгоритм работы семафора

Операция P блокирует семафор, операция V деблокирует семафор;

–мьютекс (взаимоисключение, mutual exclusion, mutex):

гарантирует потокам взаимоисключающий доступ к единственному ресурсу. Содержит счетчик числа пользователей, счетчик рекурсии и переменную, в которой запоминается идентификатор потока и гарантирует, что любой поток получает монопольный доступ к блоку памяти, обеспечивая тем самым целостность данных. Является аналогом одноместного семафора, но отличается от семафора тем, что допускает только один поток к критическому ресурсу, заставляя другие потоки, которые пытаются получить доступ к этому ресурсу ждать, пока первый поток не закончит свою работу с общим ресурсом. Принимает два значения: открыт (поток получает доступ к критическому ресурсу), закрыт –(поток не может получить доступ к критическому ресурсу);

Событие, семафор, мьютекс и событие являются объектами ядра операционной системы.

– критическая секция (critical section):

является частью программы, в которой используется доступ к общей памяти. Выполняет те же задачи, что и мьютекс, но объектом ядра не является.

– монитор Хоара (Hoare’smonitor):

является механизмом организации параллелизма, который содержит как данные, так и процедуры, необходимые для реализации динамического распределения конкретного общего ресурса или группы общих ресурсов. Включает набор разделяемых переменных и процедур доступа к ним, которым процессы пользуются в режиме разделения, причём в каждый момент им может пользоваться только один процесс.

1.5 Заключение

Для решения задачи «Читатели – писатели» был выбран монитор Хоара по ряду причин:

- монитор Хоара в своей конструкции может содержать несколько средств взаимоисключения, что обеспечивает гибкость при построении сложных схем синхронизации;

- монитор Хоара имеет важное свойство, позволяющее успешно справляться со взаимными исключениями: в любой момент времени в мониторе может быть активен только один процесс.

2. Конструкторский раздел

2.1 Основные функции приложения

В приложении реализован монитор Хоара. Основными функциями приложения являются:

- startRead( ) — функция блокировки разделяемого ресурса для выполнения операции чтения;

- endRead( )- функция снятия блокировки с разделяемого ресурса после завершения операции чтения;

- startWrite( ) - функция блокировки разделяемого ресурса для его инкремента;

- endWrite( )- функция снятия блокировки с разделяемого ресурса после записи.

В качестве разделяемого ресурса используется счетчик, который в процессе работы «писателя» инкрементируется на единицу.

Диаграмма IDEF0 приложения приведена на рисунке 3.

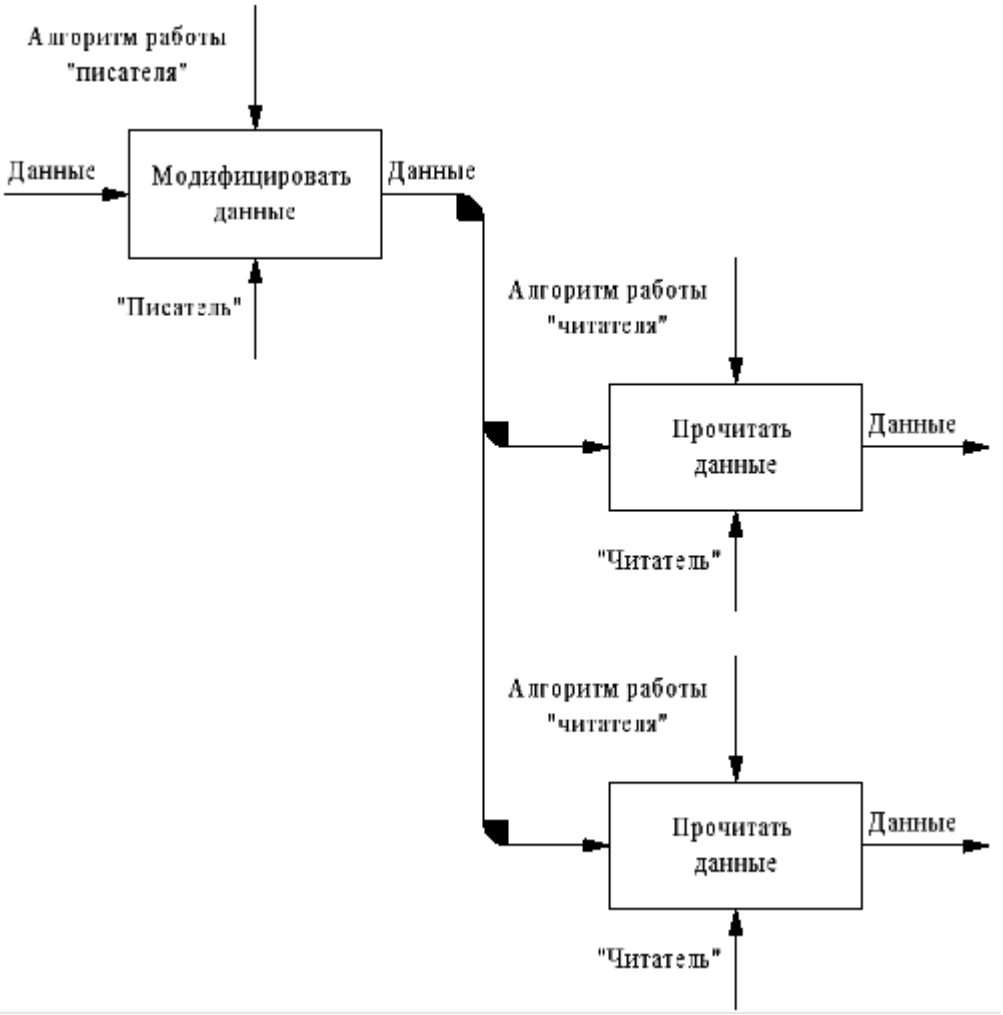

Рисунок 3 – Диаграмма IDEF0 приложения

2.2 Алгоритм работы «читателя»

Когда «читателю» необходимо выполнить чтение данных, он вызывает процедуру входа в монитор startRead( ), затем инкрементирует переменную queueReaders, показывая свое присутствие в очереди на чтение данных новому «писателю». «Читатель» может находиться внутри функции startRead(), пока никакой другой «писатель» не пишет данные и не претендует на выполнение записи данных. Выполнение этих условий определяются двумя переменными writeLock и queueWriters. Если выполняется какое-либо из условий, «читатель» ожидает наступления события canRead внутри монитора. После наступления события, «читатель» убирает себя из очереди читателей, декрементируя переменную

queueReaders и инкрементируя переменную readers. Функция startRead( ) завершается оповещением переменной canRead, чтобы другие ждущие потоки смогли продолжить чтение данных.

По окончании процесса чтения «читатель» вызывает функцию endRead( ), в которой «читатель» уменьшает переменную readers на 1, сокращая тем самым количество активных читателей. Когда в системе не остается ни одного читателя, поток выдает команду оповещения canWrite, чтобы ждущий своей очереди «писатель» смог продолжить работу, исключив тем самым бесконечное откладывание ждущих «писателей».

Графическое представление алгоритма работы «читателя» приведено на рисунке 4.

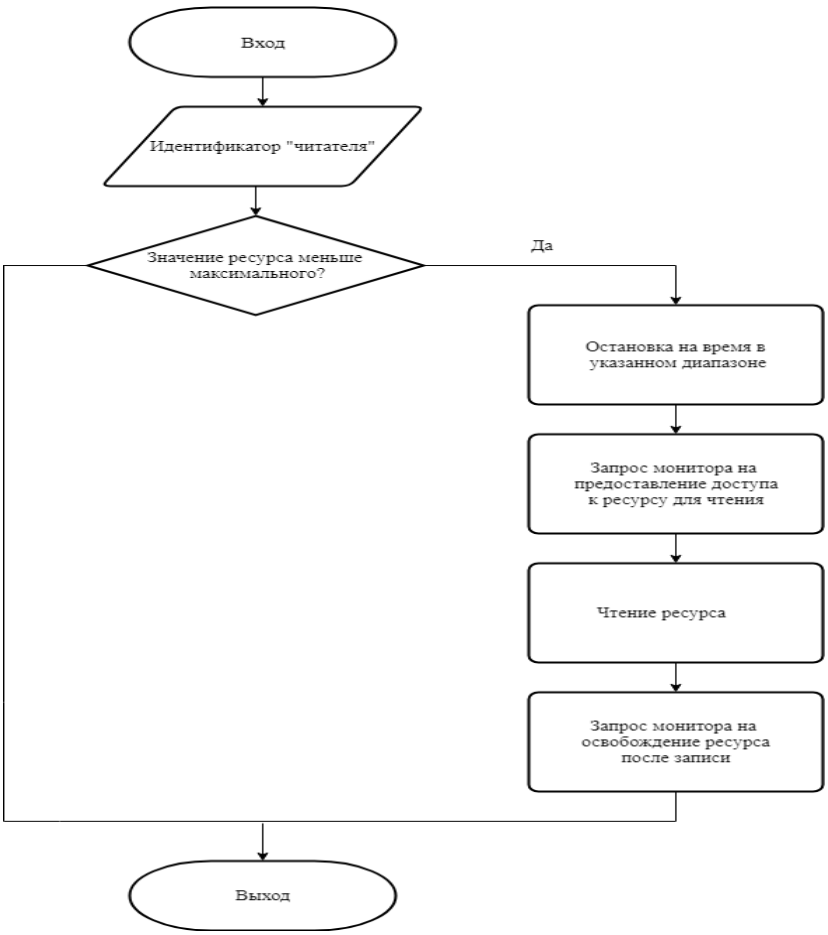

Рисунок 4 – Графическое представление алгоритма работы «читателя»

2.3 Алгоритм работы «писателя»

Когда «писателю» необходимо произвести запись, он вызывает процедуру входа в монитор startWrite( ), инкрементирует переменную queueWriters, показывая всем остальным потокам свое намерение произвести запись данных. Так как «писатель» обладает монопольным доступом к разделяемому ресурсу, необходимо проверить, чтобы в текущий момент в системе не находились работающие «читатели» или какой-либо активный «писатель», на что указывают переменные readers и writeLock. В противном случае «писателю» придется подождать наступления события canWrite. После того, как событие canWrite наступит, «писатель» уменьшает на 1 переменную queueWriters и присваивает значение true переменной writeLock, блокируя тем самым доступ к разделяемому ресурсу остальным потокам.

Когда «писатель» заканчивает свою работу, он вызывает функцию endWrite( ) и устанавливает для переменной writeLock ложное значение. После этого он проверяет наличие/отсутствие ожидающего «читателя», на что указывает переменная queueReaders. Если имеется ожидающий «читатель», инициируется событие canRead, в противном случае событие canWrite.

Графическое представление алгоритма работы работы «писателя» приведено на рисунке 5.

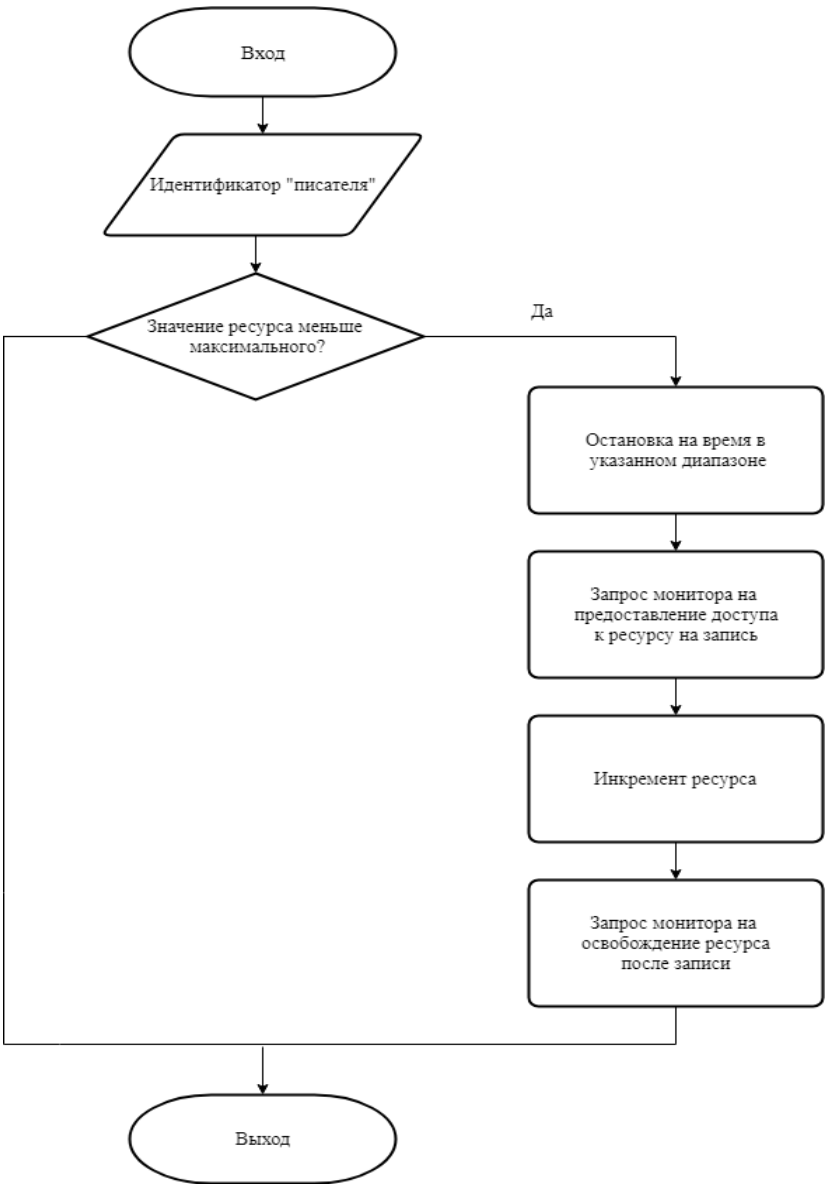

Рисунок 5 – Графическое представление алгоритма работы «писателя»

3. Технологический раздел

3.1 Выбор языка программирования и среды разработки

В качестве языка программирования для написания приложения был выбран язык C++.

Язык C++ является компилируемым статически типизированным языком программирования общего назначения. Поддерживает разные парадигмы программирования, но, в сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования. Язык C++ имеет синтаксис, основанный на синтаксисе C.

Язык C++ обладает достоинств:

— высокая совместимость с языком C, позволяющая использовать весь существующий C-код;

—поддержка различных стилей и технологий программирования, включая традиционное директивное программирование, ООП, обобщенное программирование, метапрограммирование (шаблоны, макросы);

— наличие возможности работы на низком уровне с памятью, адресами, портами;

— возможность создания обобщённых контейнеров и алгоритмов для разных типов данных, их специализация и вычисления на этапе компиляции, при использовании шаблонов;

— кроссплатформенность;

— эффективность.

В то же самое время язык C++ не лишен недостатков, ряд из которых унаследованы от языка C:

— синтаксис, провоцирующий ошибки;

— примитивный препроцессор;

— плохая поддержка модульности.

В качестве среды разработки приложения была выбрана Visual Studio 2017.

Данная среда обладает рядом достоинств:

—поддержка множества языков при разработке;

—интуитивный стиль кодирования;

— более высокая скорость разработки;

— возможности отладки.

В качестве недостатка можно отметить невозможность отладчика (Microsoft Visual Studio Debugger) отслеживать в коде режима ядра. Отладка в Windows в режиме ядра в общем случае выполняется при использовании Win Dbg, KD или Soft ICE.

Приложение спроектировано с использованием интерфейса Windows API.

3.2 Использованные API-функции

Список использованных в приложении API—функций приведен в таблице 1.

Таблица 1 — API-функции

Функция	Параметры	Что означает	Примечание
CreateEvent	NULL	Объекту «событие» присвоены атрибуты защиты по умолчанию	
	FALSE	Событие со сбросом вручную	
	TRUE	Начальное состояние события - свободное	
	NULL	Объект неименованный	
CreateMutex	NULL	Объекту «мьютекс» присвоены атрибуты защиты по умолчанию	
	FALSE	Мьютекс не принадлежит ни одному из потоков, то есть находится в свободном состоянии	
	NULL	Объект неименованный	
InterlockedDecrement	&readers	Адрес декрементируемой переменной	Входит в семейство Interlocked—функций
InterlockedIncrement	&queueReaders	Адрес инкрементируемой переменной	Входит в семейство Interlocked—функций

Продолжение таблицы 1

Функция	Параметры	Что означает	Примечание
ResetEvent	CanRead	Событие, состоянием которого осуществляется управление	
ReleaseMutex	Mutex	Мьютекс, состоянием которого осуществляется управление	
SetEvent	CanRead	Событие, состоянием которого осуществляется управление	
WaitForSingleObject	CanRead	Объект, поддерживающий состояния «свободен—занят»	
	INFINITE	Время, которое ждет вызывающий поток объект не определено	

Продолжение таблицы 1

Функция	Параметры	Что означает	Примечание
_beginthreadex	NULL	Объекту «поток» присвоены атрибуты защиты по умолчанию	Windows — функцией не является Является функцией из библиотеки VisualC++
	0	Стек для потока создается с использованием информации, встроенной компоновщиком в EXE-файл	
	Read	Функция, с которой начинается выполнение потока	
	(PVOID)i	Идентификатор потока	
	0	Исполнение потока начинается немедленно	
	NULL	32—разрядная переменная, получающая идентификатор потока, не используется	

Процесс работы приложения продемонстрирован на рисунке 6.

Рисунок 6 – Демонстрация работы приложения

4. Исследовательский раздел

В исследовательском разделе исследовалось изменение времени выполнения приложения в двух случаях:

- при увеличении количества «читателей» с одним «писателем»;

- при увеличениколичества «писателей» содним «читателем».

Результаты исследования приведены в таблице 2, на рисунке 7 и рисунке 8.

Таблица 2 – Результаты исследования

Количество «читателей»	Время выполнения, с	Количество «писателей»	Время выполнения, с
1	22,49	1	18,19
2	22,95	2	18,52
3	23,22	3	19,01
4	23,34	4	19,15
5	23,77	5	19,35

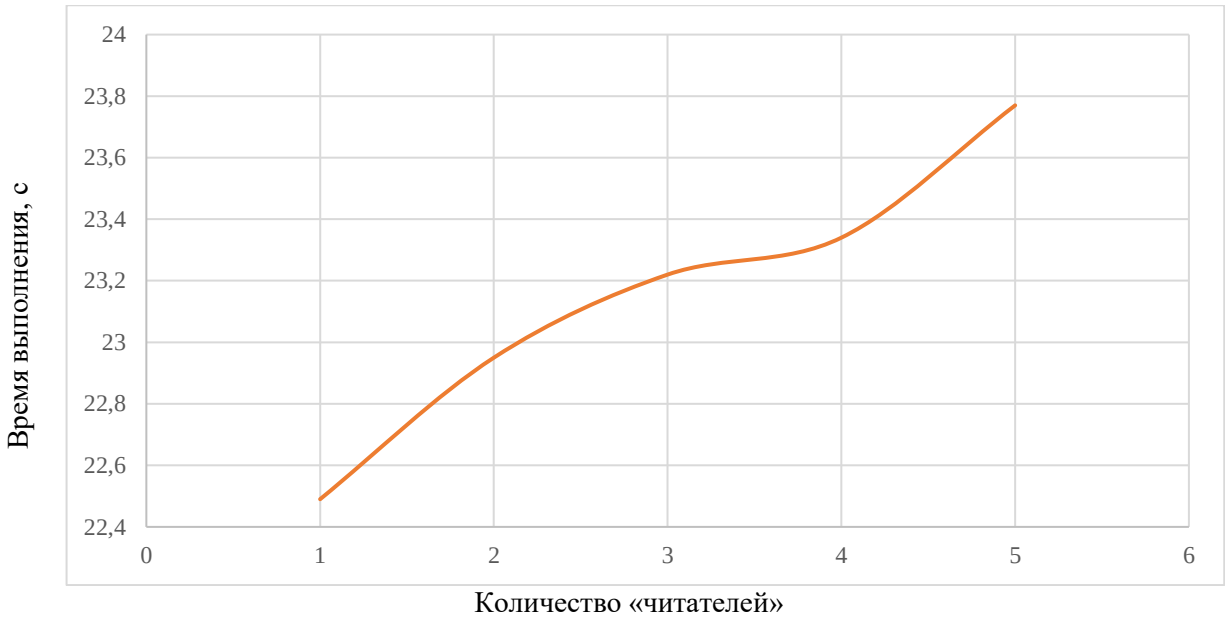

Рисунок 7 – Изменение времени выполнения приложения при увеличении количества «читателей»

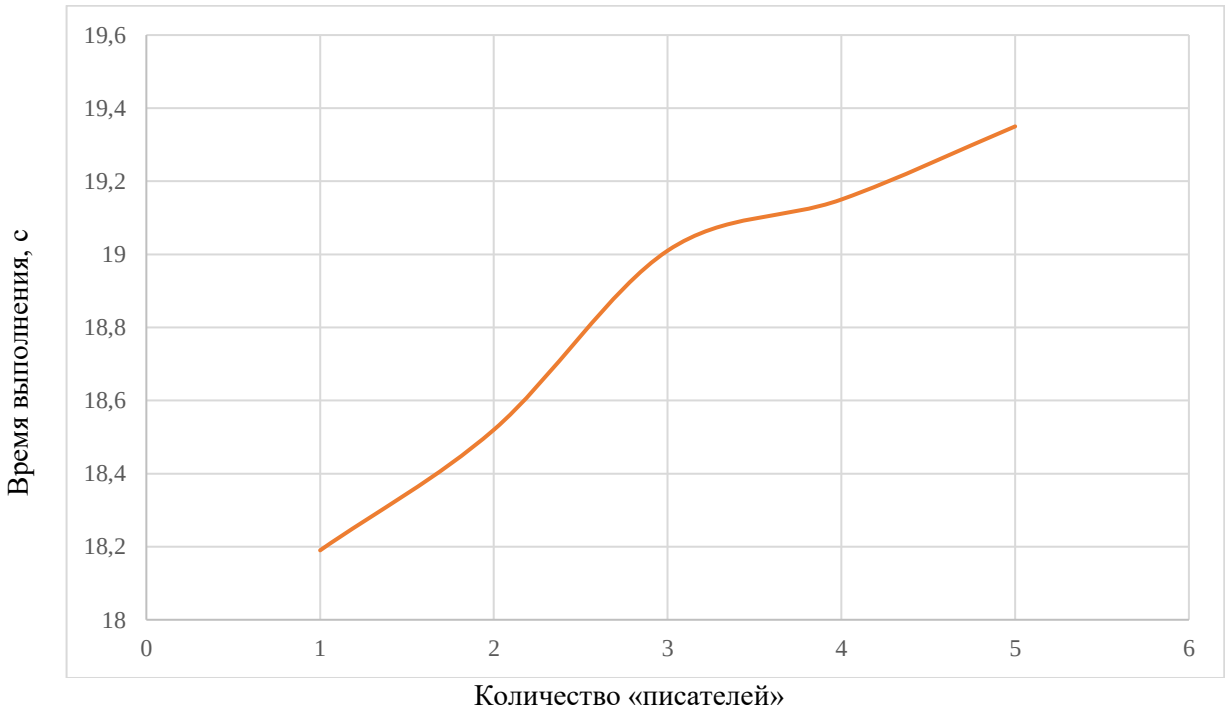

Рисунок 8 – Изменение времени выполнения приложения при увеличении количества «писателей»

Исследование показало, что увеличение количества «читателей» с одним «писателем» и увеличение количества «писателей» с одним «читателем» увеличивает время работы приложения. Увеличение времени выполнения работы приложения составило ≈6%

Заключение

В ходе выполнения данной курсовой работы были выполнены следующие задачи:

— изучены способы создания потоков и возможности для их синхронизации;

— изучен теоретический материал по синхронизации потоков;

— изучены исходные тексты примеров использования объектов синхронизации и спецификации функций Win API по работе с объектами синхронизации;

— проведена синхронизация работы потоков «читателей» и «писателей» с помощью монитора Хоара;

— исследовано влияние на время выполнения приложения увеличение количества «читателей» с фиксированным количеством «писателей» и увеличение количества «писателей» с фиксированным количеством «читателей».

Список использованных источников

1. Г. Шилдт. C++. Базовый курс. Издательство «Вильямс», 2015
2. Дж. Рихтер. WINDOWS ДЛЯ ПРОФЕССИОНАЛОВ**.** Создание эффективных Win32-приложенийс учетом специфики 64-разрядной версии Windows, 4-е изд. Издательство «Питер», 2008
3. Э. Таненбаум, Х. Бос. Современные операционные системы, 4-е изд. Издательство «Питер», 2019
4. [Электронный ресурс] ru.wikipedia.org/
5. [Электронный ресурс] https://msdn.microsoft.com/

Приложение А

Листинг программного кода приложения

```
#include "pch.h"
#include <iostream>
#include <cmath>
#include <windows.h>
#include <process.h>
#include <mutex>

#define READERS 3
#define WRITERS 3

using namespace std;

HANDLE canRead;
HANDLE canWrite;
HANDLE MUTEX;

LONG val = 0;
LONG maxVal = 10;
BOOL stop = false;

LONG readers = 0;

LONG queueReaders = 0;
LONG queueWriters = 0;

BOOL writeLock = false;

void startRead(int);
void stopRead();
unsigned __stdcall read(PVOID);

void startWrite(int);
void stopWrite();
unsigned __stdcall write(PVOID);

LONGShow(LONG&);
```

```
int main() {
	canRead = CreateEvent(NULL, TRUE, FALSE, NULL);
	canWrite = CreateEvent(NULL, FALSE, TRUE, NULL);
	MUTEX = CreateMutex(NULL, FALSE, NULL);

	for (int j = 0; j < READERS; j++) {
		_beginthreadex(NULL, 0, read, (PVOID)j, 0, NULL);
	}

	for (int i = 0; i < WRITERS; i++) {
		_beginthreadex(NULL, 0, write, (PVOID)i, 0, NULL);
	}

	getchar();

	return 0;
}


unsigned __stdcall read(PVOID Id) {
	int id = (int)Id;

	while (stop!=true) {
		startRead(id);
		Sleep(2000);
		printf("%d\n\n", Show(val));
		stopRead();
	}

	return 0;
}

voidstartRead(intid) {
	InterlockedIncrement(&queueReaders);

	if (writeLock == true || queueWriters> 0) {
		ResetEvent(canRead);
	}

	WaitForSingleObject(canRead, INFINITE);
	InterlockedIncrement(&readers);
	InterlockedDecrement(&queueReaders);
	SetEvent(canRead);
}
```

```
voidstopRead() {
     InterlockedDecrement(&readers);

     if (readers == 0) {
          SetEvent(canWrite);
     }
}

unsigned__stdcall write(PVOIDId) {
     int id = (int)Id;

     while (stop!=true) {
          startWrite(id);

          if (val<maxVal) {
               ++val;
               cout<<"Writer "<< id <<"\n\nWritten value: "<< Show(val)
<<"\n"<<endl;
               cout<<"Readers are reading...\n"<<endl;
          }
          else {
               WaitForSingleObject(MUTEX, INFINITE);
               stop = true;
               ReleaseMutex(MUTEX);
          }

          stopWrite();
     }

     return 0;
}

voidstartWrite(intid) {
     InterlockedIncrement(&queueWriters);

     if (readers > 0 || writeLock == true) {
          ResetEvent(canWrite);
     }

     WaitForSingleObject(canWrite, INFINITE);
     writeLock = true;
     InterlockedDecrement(&queueWriters);
}
```

```
void stopWrite() {
     writeLock = false;

     if (queueReaders> 0)
          SetEvent(canRead);
     else
          SetEvent(canWrite);
}


LONGShow(LONG&val) {

     returnval;
}
```