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

**ОТЧЕТ**

к лабораторной работе № 6

на тему «Средства синхронизации и взаимного исключения (Windows). Изучение и использование средств синхронизации и взаимного исключения»

Выполнил:

студент гр. 153504

Михалевич M.П.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Цели работы 3](#_Toc147863804)

[2 Kраткие теоретические сведения 4](#_Toc147863805)

[3 Результаты выполнения лабораторной работы 5](#_Toc147863806)

[Вывод 6](#_Toc147863807)

[Список использованных источников 7](#_Toc147863808)

[Приложение А 8](#_Toc147863809)

## 1 ЦЕЛИ РАБОТЫ

Целью данной лабораторной работы является разработка системы взаимного исключения для обеспечения безопасного доступа к общим ресурсам из нескольких потоков.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Инструменты синхронизации в операционной системе Windows выполняют ключевую функцию в поддержании целостности и безопасности параллельной обработки задач. Эти инструменты позволяют координировать взаимодействие множества потоков и процессов при доступе к ресурсам, которые они разделяют, включая данные, файлы, устройства или определенные участки кода. Синхронизация задач не только предотвращает конфликты, но и обеспечивает стабильную и последовательную работу программного обеспечения.

Механизмы синхронизации, такие как мьютексы, семафоры, события и критические секции, широко используются в системе Windows для контроля доступа к ресурсам. Мьютексы и семафоры управляют доступом, разрешая только одному потоку или процессу использовать ресурс в данный момент времени. События служат сигналами для потоков, которые ожидают наступления определенных условий, в то время как критические секции предлагают простой способ блокировки для защиты общих данных от одновременного доступа.

В процессе выполнения данной лабораторной работы, были применены следующие теоретические сведения:

1. Многопоточность: изучение и использование потоков для выполнения параллельных вычислений.
2. Синхронизация потоков: использование мьютексов для обеспечения синхронного доступа к общим ресурсам и предотвращения состояния гонки.
3. Критические секции: понимание необходимости защиты участков кода, где потоки взаимодействуют с общим ресурсом.
4. Работа с WinAPI для управления потоками: использование функций CreateThread, WaitForSingleObject, ReleaseMutex и других для управления потоками в Windows.
5. Обработка ошибок: использование GetLastError и проверки возвращаемых значений функций для обработки ошибок при работе с потоками и мьютексами.
6. Рекурсия: применение рекурсивной функции для вычисления факториала.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе лабораторной работы было разработано приложение взаимного исключения для обеспечения безопасного доступа к общим ресурсам из нескольких потоков. Результат работы показан на рис. 3.1.

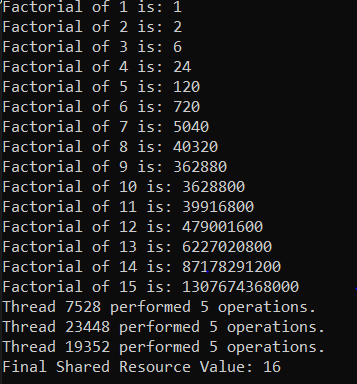


Рисунок 3.1 – Результат работы программы

## ВЫВОД

В ходе выполнения лабораторной работы была разработана системы взаимного исключения для обеспечения безопасного доступа к общим ресурсам из нескольких потоков.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.
2. [Электронный ресурс]. – Режим доступа: [https://learn.microsoft.com/ru-ru/windows/win32/api – Дата доступа 23.10.2023](https://learn.microsoft.com/ru-ru/windows/win32/api%20–%20Дата%20доступа%2023.10.2023)
3. [Электронный ресурс]. – Режим доступа:  [https://learn.microsoft.com/ru-ru/cpp/parallel/multithreading-with-c-and-win32?view=msvc-170 – Дата доступа 24.10.2023](%20https://learn.microsoft.com/ru-ru/cpp/parallel/multithreading-with-c-and-win32?view=msvc-170%20–%20Дата%20доступа%2024.10.2023)

## ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг кода**

#include <windows.h>

#include <iostream>

// Глобальный мьютекс

HANDLE hMutex;

// Вспомогательная функция для вычисления факториала

unsigned long long factorial(int n) {

if (n <= 1) return 1;

else return n \* factorial(n - 1);

}

// Функция потока, выполняющая сложные вычисления и изменяющая общий ресурс

DWORD WINAPI ThreadFunction(LPVOID lpParam) {

int\* sharedResource = (int\*)lpParam;

int operationsPerformed = 0;

for (int i = 0; i < 5; ++i) {

WaitForSingleObject(hMutex, INFINITE); // Захват мьютекса

// Начало критической секции

std::cout << "Thread " << GetCurrentThreadId() << " is entering critical section." << std::endl;

// Выполнение сложной операции: расчет факториала

unsigned long long factorialResult = factorial(\*sharedResource);

std::cout << "Factorial of " << \*sharedResource << " is: " << factorialResult << std::endl;

(\*sharedResource)++;

operationsPerformed++;

// Конец критической секции

std::cout << "Thread " << GetCurrentThreadId() << " is leaving critical section." << std::endl;

ReleaseMutex(hMutex); // Освобождение мьютекса

Sleep(1000); // Имитация работы вне критической секции

}

std::cout << "Thread " << GetCurrentThreadId() << " performed " << operationsPerformed << " operations." << std::endl;

return 0;

}

int main() {

int sharedResource = 1; // Начальное значение ресурса установим в 1 для корректного вычисления факториала

// Создание мьютекса

hMutex = CreateMutex(NULL, FALSE, NULL);

if (hMutex == NULL) {

std::cerr << "CreateMutex error: " << GetLastError() << std::endl;

return 1;

}

// Создание трех потоков

const int threadCount = 3;

HANDLE hThreads[threadCount];

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

hThreads[i] = CreateThread(NULL, 0, ThreadFunction, &sharedResource, 0, NULL);

if (hThreads[i] == NULL) {

std::cerr << "CreateThread error: " << GetLastError() << std::endl;

return 1;

}

}

// Ожидание завершения потоков

WaitForMultipleObjects(threadCount, hThreads, TRUE, INFINITE);

// Закрытие дескрипторов потоков

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

CloseHandle(hThreads[i]);

}

// Вывод результата

std::cout << "Final Shared Resource Value: " << sharedResource << std::endl;

// Закрытие мьютекса

CloseHandle(hMutex);

return 0;

}