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. 153503

Филипеня А.Д.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

[3 Результаты выполнения лабораторной работы 5](#_Toc146631500)

[Выводы 6](#_Toc146631501)

[Список использованных источников 7](#_Toc146631502)

[Приложение А (обязательное) Листинг кода 8](#_Toc146631503)

## 1 ПОСТАНОВКА ЗАДАЧИ

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

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений.

Процесс – это совокупность ресурсов, необходимых для выполнения программы. Поток – это основная выполняемая единица, для которой операционная система выделяет процессорное время. [1]

Одним из наиболее распространенных средств синхронизации в Windows являются мьютексы (mutexes), семафоры (semaphores), события (events) и критические секции (critical sections). Мьютексы и семафоры обеспечивают взаимное исключение, позволяя только одному потоку (или процессу) одновременно доступ к общему ресурсу. События позволяют потокам ожидать определенных условий или уведомлений от других потоков. Критические секции предоставляют легкий механизм блокировки для обеспечения безопасности доступа к общим данным. [2]

При разработке многозадачных приложений под Windows, правильное использование средств синхронизации становится критически важным. Это позволяет избегать состояний гонки, повышать производительность, улучшать отзывчивость и обеспечивать надежность программ. Ошибки в синхронизации могут привести к непредсказуемым результатам и сбоям, поэтому понимание и использование этих средств является ключевой компетенцией для разработчиков под Windows.

Для выполнения данной лабораторной работы, были использованы следующие теоретические сведения и концепции:

1. В программе создаются четыре потока с помощью функции CreateThread(), входными функциями которых являются функции подсчета синуса, косинуса, тангенса и котангенса.

2. В программе создается мьютекс, использующийся для синхронизации доступа к общему ресурсам, с помощью функции CreateMutex().

3. Для каждого потока ожидается его завершение с использованием функции WaitForSingleObject(). Это гарантирует, что главная программа будет ждать завершения всех четырех потоков, прежде чем продолжить выполнение.

4. Функция ReleaseMutex() используется для освобождения мьютекса. Для закрытия дескриптора мьютекса используется функция CloseHandle().

6. После завершения всех четырех потоков закрываются их дескрипторы с помощью функции CloseHandle().

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было реализовано простое консольное многопоточное приложение, которое использует многопоточность для вычисления синуса, косинуса, тангенса и котангенса заданного угла. Программа порождает четыре потока, каждый из которых имеет свою входную функцию. Первый поток ведет подсчет синуса заданного угла, используя функцию CalculateSin(), второй считает косинус этого угла с использованием функции CalculateCos(), третий подсчитывает тангенс угла при помощи функции CalculateTan(), четвертый – котангенс угла, используя функцию CalculateCtg().

В данном проекте используются мьютексы для обеспечения синхронизации доступа к разделяемому ресурсам между разными потоками. Без синхронизации это может привести к состоянию гонки данных, когда несколько потоков пытаются одновременно изменить общие переменные. Результат работы программы показан на рисунке 3.1.

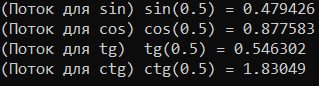


Рисунок 3.1 – Результат работы программы

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы была создана простая система 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/procthread/process-and-thread-functions – Дата доступа 11.10.2023](https://learn.microsoft.com/ru-ru/windows/win32/procthread/process-and-thread-functions%20–%20Дата%20доступа%2008.10.2023)

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**main.cpp**

#include <iostream>

#include <Windows.h>

#include <cmath>

#include <string>

#include "main.h"

DWORD WINAPI CalculateSin(LPVOID param) {

double\* input = (double\*)param;

WaitForSingleObject(mutex, INFINITE); // Захватываем мьютекс для синуса

sinResult = sin(\*input);

std::cout << "(Поток для sin) sin(" << \*input << ") = " << sinResult << std::endl;

ReleaseMutex(mutex); // Освобождаем мьютекс

return 0;

}

DWORD WINAPI CalculateCos(LPVOID param) {

double\* input = (double\*)param;

WaitForSingleObject(mutex, INFINITE);

cosResult = cos(\*input);

std::cout << "(Поток для cos) cos(" << \*input << ") = " << cosResult << std::endl;

ReleaseMutex(mutex);

return 0;

}

DWORD WINAPI CalculateTan(LPVOID param) {

double\* input = (double\*)param;

WaitForSingleObject(mutex, INFINITE);

tanResult = tan(\*input);

std::cout << "(Поток для tg) tg(" << \*input << ") = " << tanResult << std::endl;

ReleaseMutex(mutex);

return 0;

}

DWORD WINAPI CalculateCtg(LPVOID param) {

double\* input = (double\*)param;

WaitForSingleObject(mutex, INFINITE);

ctgResult = 1.0 / tan(\*input);

std::cout << "(Поток для ctg) ctg(" << \*input << ") = " << ctgResult << std::endl;

ReleaseMutex(mutex);

return 0;

}

// Lab3

void InitMappingFile() {

hFile = CreateFile(fileName, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile == INVALID\_HANDLE\_VALUE) {

std::cerr << "IniteMappingFile - CreateFile не работает, fname = "

<< fileName << std::endl;

return;

}

hMapping = CreateFileMapping(hFile, NULL, PAGE\_READWRITE, 0, fileSize, NULL);

if (hMapping == NULL) {

std::cerr << "IniteMappingFile - CreateFileMapping не работает, fname = "

<< fileName << std::endl;

CloseHandle(hFile);

return;

}

pMappedData = MapViewOfFile(hMapping, FILE\_MAP\_ALL\_ACCESS, 0, 0, fileSize);

if (pMappedData == NULL) {

std::cerr << "fileMappingCreate - MapViewOfFile не работает, fname = "

<< fileName << std::endl;

CloseHandle(hMapping);

CloseHandle(hFile);

return;

}

}

void UninitializeMappingFile() {

if (pMappedData != NULL) {

UnmapViewOfFile(pMappedData);

pMappedData = NULL;

}

if (hMapping != NULL) {

CloseHandle(hMapping);

hMapping = NULL;

}

if (hFile != INVALID\_HANDLE\_VALUE) {

CloseHandle(hFile);

hFile = INVALID\_HANDLE\_VALUE;

}

}

void SaveData(std::string str, int size) {

if (pMappedData != NULL) {

size\_t dataSize = size \* sizeof(char);

if (mappedDataSize + dataSize >= fileSize)

{

UninitializeMappingFile();

InitMappingFile();

mappedDataSize = 0;

}

memcpy((CHAR\*)pMappedData + mappedDataSize, str.c\_str(), dataSize);

mappedDataSize += dataSize;

}

}

int main() {

double input\_value = 0.5;

mutex = CreateMutex(NULL, FALSE, NULL);

if (mutex == NULL) {

std::cerr << "Ошибка при создании мьютекса" << std::endl;

return 1;

}

// отдельные потоки для каждой функции

HANDLE sinThread = CreateThread(NULL, 0, CalculateSin, &input\_value, 0, NULL);

HANDLE cosThread = CreateThread(NULL, 0, CalculateCos, &input\_value, 0, NULL);

HANDLE tanThread = CreateThread(NULL, 0, CalculateTan, &input\_value, 0, NULL);

HANDLE ctgThread = CreateThread(NULL, 0, CalculateCtg, &input\_value, 0, NULL);

if (sinThread == NULL || cosThread == NULL || tanThread == NULL || ctgThread == NULL) {

std::cerr << "Ошибка при создании потоков" << std::endl;

return 1;

}

// ожидается завершения всех потоков

WaitForSingleObject(sinThread, INFINITE);

WaitForSingleObject(cosThread, INFINITE);

WaitForSingleObject(tanThread, INFINITE);

WaitForSingleObject(ctgThread, INFINITE);

// Закрываем дескрипторы мьтекса и потоков

CloseHandle(mutex);

CloseHandle(sinThread);

CloseHandle(cosThread);

CloseHandle(tanThread);

CloseHandle(ctgThread);

std::string result = "sin(" + std::to\_string(input\_value) + ") = " + std::to\_string(sinResult) + "\n" +

"cos(" + std::to\_string(input\_value) + ") = " + std::to\_string(cosResult) + "\n" +

"tg(" + std::to\_string(input\_value) + ") = " + std::to\_string(tanResult) + "\n" +

"ctg(" + std::to\_string(input\_value) + ") = " + std::to\_string(ctgResult) + "\n";

//std::cout << result;

//Lab3

InitMappingFile();

SaveData(result, result.size());

UninitializeMappingFile();

return 0;

}