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 № 4

на тему «Управление процессами и потоками (Windows). Порождение, завершение, изменение приоритетов процессов и потоков, исследование эффективности»

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 многопоточного приложения для вычисления тригонометрических функций и сохранения результата в файл с использованием маппинга файлов в память.

## 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.[1]

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

Создание нового процесса или потока выполняется с использованием функций Windows API: CreateProcess для процессов и CreateThread для потоков. При порождении нового процесса происходит выделение отдельного адресного пространства памяти, файловых ресурсов и других системных ресурсов.

Процессы и потоки могут завершаться по разным причинам. Вы можете использовать функции, такие как ExitProcess для завершения процесса и ExitThread для завершения потока. Кроме того, процесс или поток могут быть завершены системой из-за ошибок или других обстоятельств.[1]

Управление процессами и потоками в Windows – это важный аспект разработки приложений, позволяющий создавать эффективные и отзывчивые многозадачные приложения, а также обеспечивать их стабильную работу.

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 создаются мьютексы, использующиеся для синхронизации доступа к общим 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 и удаления мьютексов используем функцию 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 разделяемым ресурсам между разными потоками: четыре потока могут одновременно попытаться записать свои результаты в общие переменные sinResult, cosResult, tanResult и ctgResult. 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.

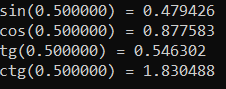


Рисунок 3.1 – Результат работы программы

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы было создано простое многопоточное консольное приложение, которое выводит результат подсчетов четырех потоков. Первый поток ведет подсчет синуса заданного угла, второй считает косинус этого угла, третий подсчитывает тангенс угла, четвертый – котангенс.

## 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)
3. [Электронный ресурс]. – Режим доступа:  [https://www.rsdn.org/article/baseserv/mt.xml – Дата доступа 12.10.2023](https://stackoverflow.com/questions/17187265/how-to-group-radio-box-buttons-using-win32-api%20–%20Дата%20доступа%2023.09.2023)

## 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(sinMutex, INFINITE); // Захватываем мьютекс для синуса

sinResult = sin(\*input);

ReleaseMutex(sinMutex); // Освобождаем мьютекс

return 0;

}

DWORD WINAPI CalculateCos(LPVOID param) {

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

WaitForSingleObject(cosMutex, INFINITE);

cosResult = cos(\*input);

ReleaseMutex(cosMutex);

return 0;

}

DWORD WINAPI CalculateTan(LPVOID param) {

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

WaitForSingleObject(tanMutex, INFINITE);

tanResult = tan(\*input);

ReleaseMutex(tanMutex);

return 0;

}

DWORD WINAPI CalculateCtg(LPVOID param) {

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

WaitForSingleObject(ctgMutex, INFINITE);

ctgResult = 1.0 / tan(\*input);

ReleaseMutex(ctgMutex);

return 0;

}

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 failed, fname = "

<< fileName << std::endl;

return;

}

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

if (hMapping == NULL) {

std::cerr << "IniteMappingFile - CreateFileMapping failed, fname = "

<< fileName << std::endl;

CloseHandle(hFile);

return;

}

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

if (pMappedData == NULL) {

std::cerr << "fileMappingCreate - MapViewOfFile failed, 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;

// Создаем мьютексы для синхронизации доступа к результатам

sinMutex = CreateMutex(NULL, FALSE, NULL);

cosMutex = CreateMutex(NULL, FALSE, NULL);

tanMutex = CreateMutex(NULL, FALSE, NULL);

ctgMutex = CreateMutex(NULL, FALSE, NULL);

if (sinMutex == NULL || cosMutex == NULL || tanMutex == NULL || ctgMutex == NULL) {

std::cerr << "Ошибка при создании мьютексов!" << std::endl;

return 1;

}

// Создаем 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;

}

// Ожидаем завершения всех потоков

WaitForSingleObject(sinThread, INFINITE);

WaitForSingleObject(cosThread, INFINITE);

WaitForSingleObject(tanThread, INFINITE);

WaitForSingleObject(ctgThread, INFINITE);

// Закрываем и удаляем мьютексы

CloseHandle(sinMutex);

CloseHandle(cosMutex);

CloseHandle(tanMutex);

CloseHandle(ctgMutex);

// Закрываем дескрипторы потоков

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;

InitMappingFile();

SaveData(result, result.size());

UninitializeMappingFile();

return 0;

}

**main.h**

#pragma once

double sinResult = 0.0;

double cosResult = 0.0;

double tanResult = 0.0;

double ctgResult = 0.0;

HANDLE sinMutex, cosMutex, tanMutex, ctgMutex; // Мьютексы для синхронизации доступа к результатам

const WCHAR\* fileName = L"result.txt";

const int fileSize = 1000;

HANDLE hFile = NULL;

HANDLE hMapping = NULL;

LPVOID pMappedData = NULL;

SIZE\_T mappedDataSize = 0;