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

Хрищанович А.К.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

[3 Результаты выполнения лабораторной работы 6](#_Toc146631500)

[Выводы 7](#_Toc146631501)

[Список использованных источников 8](#_Toc146631502)

[Приложение А (обязательное) Листинг кода 9](#_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.[1]

Средства синхронизации и взаимного исключения в операционной системе Windows играют критическую роль в обеспечении безопасности и корректности многозадачного выполнения программ. Они позволяют управлять доступом нескольких потоков или процессов к общим ресурсам, таким как общие данные, файлы, устройства или критические секции кода. Средства синхронизации обеспечивают правильную координацию выполнения задач, что предотвращает состояния гонки и гарантирует надежное выполнение программ.

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.

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. Потоки (Threads): при написании кода были созданы и использованы потоки с помощью функций Windows API, таких как CreateThread и WaitForMultipleObjects. Это позволяет выполнить задачи параллельно, улучшая производительность и эффективность программы.

2. Синхронизация доступа к общим данным: для обеспечения безопасности при работе с общими данными используется мьютекс (mutex). Функции WaitForSingleObject и ReleaseMutex используются для захвата и освобождения мьютекса. Это предотвращает одновременный доступ нескольких потоков к общим данным и исключает конфликты при изменении.

3. Структуры данных и векторы: структура Task используется для передачи данных между потоками. Вектор std::vector используется для хранения общих данных, которые будут обработаны потоками.

4. Функции работы с временем: функция srand(time(NULL)) используется для инициализации генератора случайных чисел на основе текущего времени.

5. Работа с дескрипторами: программа управляет дескрипторами потоков и мьютекса с помощью функций CloseHandle.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы была создана система взаимного исключения для обеспечения безопасного доступа к общим ресурсам из нескольких потоков. Результат работы программы предоставлен на рисунке 3.1.

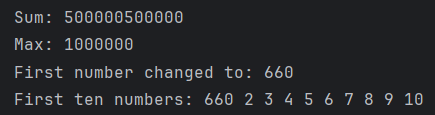


Рисунок 3.1 – Результат работы программы

## 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 24.10.2023](https://learn.microsoft.com/ru-ru/windows/win32/procthread/process-and-thread-functions%20–%20Дата%20доступа%2024.10.2023)**ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг кода

**main.cpp**

#include <windows.h>

#include <iostream>

#include <algorithm>

#include <vector>

struct Task {

int taskType; // Номер задачи (1 - сумма, 2 - максимум, 3 - минимум)

std::vector<int> data; // Данные для выполнения задачи

};

std::vector<int> sharedData(1000000); // Общий ресурс

HANDLE hMutex; // Мьютекс для синхронизации доступа к общим данным

DWORD WINAPI ThreadFunction(LPVOID lpParam) {

Task\* task = static\_cast<Task\*>(lpParam);

WaitForSingleObject(hMutex, INFINITE); // Захватываем мьютекс для синхронизации доступа

if (task->taskType == 1) {

// Задача для вычисления суммы

long long sum = 0;

for (int num : task->data) {

sum += num;

}

std::cout << "Sum: " << sum << std::endl;

} else if (task->taskType == 2) {

// Задача для нахождения максимума

int max = \*std::max\_element(task->data.begin(), task->data.end());

std::cout << "Max: " << max << std::endl;

} else if (task->taskType == 3) {

// Задача для изменения первого числа на случайное

if (!task->data.empty()) {

srand(time(NULL));

task->data[0] = rand() % 1000; // Заменяем первое число на случайное от 0 до 999

std::cout << "First number changed to: " << task->data[0] << std::endl;

std::cout << "First ten numbers: ";

for (int i = 0; i < 10 && i < task->data.size(); ++i) {

std::cout << task->data[i] << " ";

}

std::cout << std::endl;

}

}

ReleaseMutex(hMutex); // Освобождаем мьютекс после завершения задачи

return 0;

}

int main() {

// Инициализация общих данных

for (int i = 0; i < 1000000; ++i) {

sharedData[i] = i + 1;

}

// Создаем мьютекс

hMutex = CreateMutex(NULL, FALSE, NULL);

// Создаем задачи для суммы, минимума и максимума

Task tasks[3];

for (int i = 0; i < 3; ++i) {

tasks[i].taskType = i + 1;

tasks[i].data = sharedData;

}

// Запускаем потоки для выполнения задач

HANDLE hThreads[3];

for (int i = 0; i < 3; ++i) {

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

}

// Дожидаемся завершения потоков

WaitForMultipleObjects(3, hThreads, TRUE, INFINITE);

// Закрываем дескрипторы потоков

for (int i = 0; i < 3; ++i) {

CloseHandle(hThreads[i]);

}

// Закрываем дескриптор мьютекса

CloseHandle(hMutex);

return 0;

}