МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

Отчет по лабораторной работе № 10

по дисциплине:” Системное программирование”

на тему:” Многозадачность потоков,

автосборка сложных проектов”

Вариант 6

Выполнила**:** студент группы 10701222 Дяденькина Т.М.

Принял**:** ст. пр. Давыденко Н.В.

Минск 2024

Цель работы

Изучить понятие потоков и их особенности. Создание и завершение по-

тока. Синхронизация потоков. Получение информации о потоках. Обмен

данными между потоками. Изучить встроенный инструментарий для разработки приложений под семейство ОС Linux и фундаментальные основы системного программирования с использованием компиляторов gcc/g++, отладчика gdb и других для проектирования, компиляции, отладки и запуска приложений на языке программирования С/C++. Научиться эффективно использовать специальные средства для автоматизации процесса компиляции, сборки и запуска многофайловых проектов.

Задание 2

Проведите рефакторинг программы из 1-го задания так чтобы синхронизировать соответствующие потоки.

Соберите и запустите приложение, сравните результаты с результатами предыдущей программы. Дайте объяснения.

Выполнение

# **Код мэйна**

#include <iostream>

#include "AddressBook.h"

#include "ThreadedOperations.h"

#include <pthread.h>

pthread\_mutex\_t mutex;

int main() {

pthread\_mutex\_init(&mutex, NULL);

AddressBook addressBook;

addressBook.loadFromFile("example1.txt"); // Load from file

ThreadedOperations operations;

auto future = operations.performComplexCalculation(addressBook);

pthread\_mutex\_lock(&mutex); // блокировка доступа к ресурсу

std::cout << "Adding people...\n";

addressBook.addEntry(AddressBookEntry("Mary Mahn", "888-456-8520", "mary.mahn@example.com"));

addressBook.addEntry(AddressBookEntry("Tanya Dzad", "789-123-7410", "tanya.dzad@example.com"));

addressBook.addEntry(AddressBookEntry("Alya Shkrob", "159-852-7563", "alya.shkrob@example.com"));

addressBook.addEntry(AddressBookEntry("John Doe", "555-123-4567", "john.doe@example.com"));

std::cout << "Finding John Doe...\n";

addressBook.findEntry("John Doe");

std::cout << "Deleting John Doe...\n";

addressBook.deleteEntry("John Doe");

std::cout << "Finding John Doe...\n";

addressBook.findEntry("John Doe");

pthread\_mutex\_unlock(&mutex); // разблокировка доступа к ресурсу

future.wait(); // Wait for the calculation to finish

addressBook.saveToFile("example1.txt"); // Save to file

pthread\_mutex\_destroy(&mutex); // освобождение ресурсов мьютекса

return 0;

}

**Второй поток**

#include "ThreadedOperations.h"

#include "AddressBook.h"

#include <future>

#include <thread>

#include <chrono>

#include <iostream>

std::future<void> ThreadedOperations::performComplexCalculation(const AddressBook& addressBook) {

return std::async(std::launch::async, [addressBook]() {

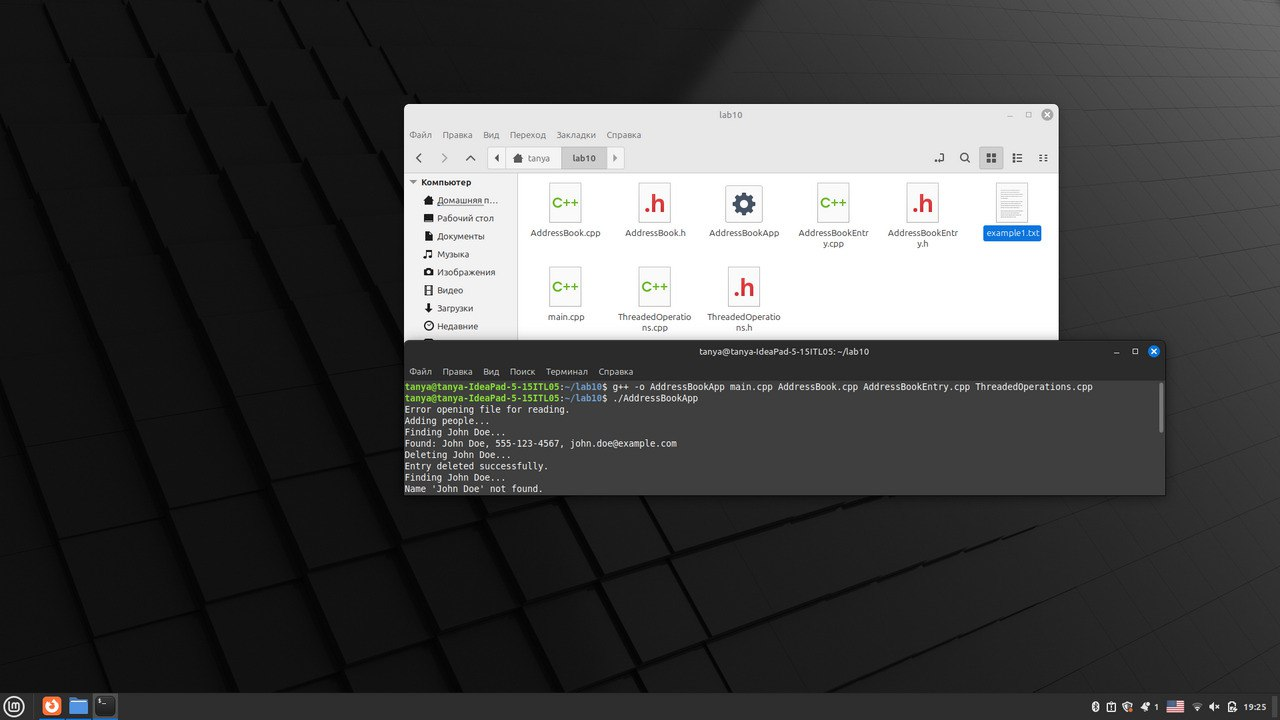
// Simulate a complex calculation

std::this\_thread::sleep\_for(std::chrono::seconds(2)); // Example: 2 seconds

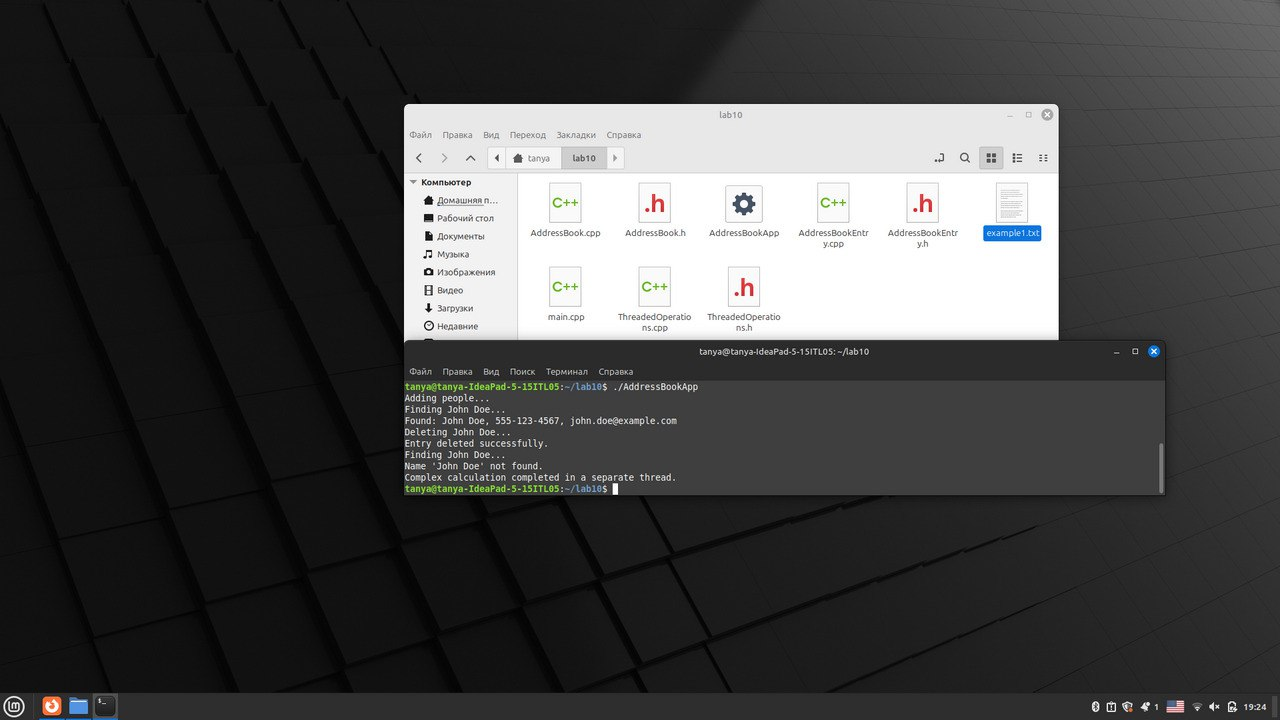
std::cout << "Complex calculation completed in a separate thread." << std::endl;

});

}

**Компиляция**

**Выполнение**



Контрольные вопросы

В чем различие компиляции и интерпритации программы?

Ответ:

1. Процесс выполнения

* **Компиляция**:

Программа полностью переводится в машинный код до выполнения.

Результат компиляции — исполняемый файл, который можно запускать на целевой платформе.

* **Интерпретация**:

Программа переводится в машинный код построчно.

Нет промежуточного исполняемого файла; интерпретатор читает и выполняет код напрямую.

2. Время выполнения

Компиляция программы медленнее на этапе запуска, но быстрее на выполнении программы, по сравнению с интерпретацией.

3. Отладка

* **Компиляция**:

Ошибки выявляются на этапе компиляции.

Сложная отладка из-за необходимости повторного компилирования после внесения изменений.

* **Интерпретация**:

Ошибки могут быть обнаружены только во время выполнения.

Легче вносить изменения.

4. Примеры языков

Просто применятся к разным языкам программирования

**Компилируемые языки**: C, C++

**Интерпретируемые языки:** Python, JavaScript

5. Портируемость

Исполняемый файл, созданный на одной платформе после компиляции, обычно не будет работать на другой платформе без перекомпиляции, в отличии от кода, написанного на интерпретируемом языке, он более портируемый.