МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ образовательное учреждениевысшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации



**ОТЧЁТ по Лабораторной работе №4**

**по дисциплине «ПРОГРАММИРОВАНИЕ»**

**«Многопоточность»**

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АБс-222  Студент: Гатауллин Д.Р. | Преподаватель: Архипова А.Б. |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

**Цели и задачи работы**: Изучение принципов реализации многопоточности и практические применение возможностей языков высокого уровня при моделировании бизнес-процессов.

**Задание к работе**:

1. Реализовать параллельный запуск заданного числа потоков, содержащих случайные символы из ASCII таблицы в формате гонки.

Проанализировать особенности работы примитивов синхронизации (сравнительный анализ):

• Mutexes

• Semaphore

• SemaphoreSlim

• Barrier

• SpinLock

• SpinWait

• Monitor

Проанализировать скорости работы примитивов с использованием StopWatch и BenchMarkDotNet.

1. Многопоточная обработка данных.

Задаются данные: размер массива данных, количество параллельных потоков, значения атрибутов структуры, дополнительные данные (в зависимости от варианта).

Результатом работы программы являются вывод времени обработки без использования многопоточности, времени обработки с использованием многопоточности и результатов обработки.

Структура содержит данные о студентах (ФИО, номер группы, список результатов сессий). Результат сессии содержит номер семестра, сведения о названии дисциплины и отметку за экзамен (от 2 до 5). Необходимо вывести среднюю успеваемость студентов группы Г за семестр С. Сопоставление с паттерном.

1. Задание обедающих философов.

**Текст программы**

==========================================================

LB4.1.cpp

==========================================================

#include <iostream>

#include <vector>

#include <thread>

#include <mutex>

#include <semaphore>

#include <barrier>

#include <atomic>

#include <random>

#include <chrono>

// Генератор случайных чисел для символов ASCII

char random\_ascii() {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(33, 126);

return static\_cast<char>(dis(gen));

}

// Функция для потока, использующая Mutex

void thread\_func\_mutex(std::mutex& mtx, int thread\_num) {

std::lock\_guard<std::mutex> lock(mtx);

// Критическая секция

std::cout << "Thread " << thread\_num << ": " << random\_ascii() << std::endl;

}

// Функция для потока, использующая Semaphore

void thread\_func\_semaphore(std::counting\_semaphore<>& sem, int thread\_num) {

sem.acquire();

// Критическая секция

std::cout << "Thread " << thread\_num << ": " << random\_ascii() << std::endl;

sem.release();

}

// Функция для потока, использующая Barrier

void thread\_func\_barrier(std::barrier<>& bar, int thread\_num) {

bar.arrive\_and\_wait();

// Критическая секция

std::cout << "Thread " << thread\_num << ": " << random\_ascii() << std::endl;

}

// Функция для потока, использующая SpinLock

void thread\_func\_spinlock(std::atomic\_flag& flag, int thread\_num) {

while (flag.test\_and\_set(std::memory\_order\_acquire)); // Spin

// Критическая секция

std::cout << "Thread " << thread\_num << ": " << random\_ascii() << std::endl;

flag.clear(std::memory\_order\_release);

}

// Функция для потока, использующая Monitor (через std::unique\_lock)

void thread\_func\_monitor(std::mutex& mtx, int thread\_num) {

std::unique\_lock<std::mutex> lock(mtx);

// Критическая секция

std::cout << "Thread " << thread\_num << ": " << random\_ascii() << std::endl;

}

int main() {

setlocale(LC\_ALL, "Rus");

int num\_threads;

std::string sync\_primitive;

std::cout << "Введите количество потоков: ";

std::cin >> num\_threads;

std::cout << "Введите примитив синхронизации (Mutexes, Semaphore, Barrier, SpinLock, Monitor): ";

std::cin >> sync\_primitive;

std::vector<std::thread> threads;

threads.reserve(num\_threads);

// Выбор примитива синхронизации

if (sync\_primitive == "Mutexes") {

std::mutex mtx;

for (int i = 0; i < num\_threads; ++i) {

threads.emplace\_back(thread\_func\_mutex, std::ref(mtx), i);

}

}

else if (sync\_primitive == "Semaphore") {

std::counting\_semaphore<> sem(1); // Разрешить одновременный доступ одного потока

for (int i = 0; i < num\_threads; ++i) {

threads.emplace\_back(thread\_func\_semaphore, std::ref(sem), i);

}

}

else if (sync\_primitive == "Barrier") {

std::barrier<> bar(num\_threads);

for (int i = 0; i < num\_threads; ++i) {

threads.emplace\_back(thread\_func\_barrier, std::ref(bar), i);

}

}

else if (sync\_primitive == "SpinLock") {

std::atomic\_flag flag = ATOMIC\_FLAG\_INIT;

for (int i = 0; i < num\_threads; ++i) {

threads.emplace\_back(thread\_func\_spinlock, std::ref(flag), i);

}

}

else if (sync\_primitive == "Monitor") {

std::mutex mtx;

for (int i = 0; i < num\_threads; ++i) {

threads.emplace\_back(thread\_func\_monitor, std::ref(mtx), i);

}

}

else {

std::cout << "Неверный ввод примитива синхронизации." << std::endl;

return 1;

}

auto start = std::chrono::high\_resolution\_clock::now(); // Начало измерения времени

// Запуск и ожидание завершения всех потоков

for (auto& th : threads) {

th.join();

}

auto end = std::chrono::high\_resolution\_clock::now(); // Конец измерения времени

std::chrono::duration<double, std::milli> elapsed = end - start; // Вычисление затраченного времени

std::cout << "Время выполнения: " << elapsed.count() << " мс\n";

return 0;

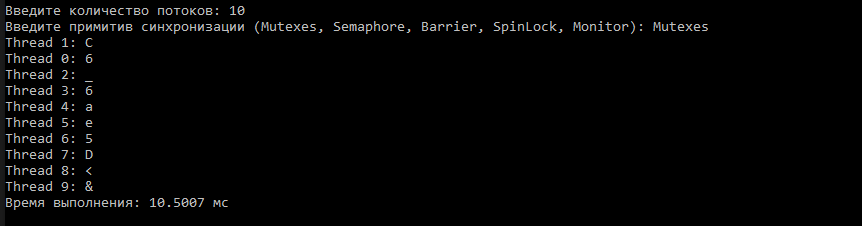
}

==========================================================

Остальной код в моём репозитории <https://github.com/SpongebobGatik/GataullinLB4>

==========================================================

**Результат работы программы**



Изображение выглядит как снимок экрана, текст

Автоматически созданное описание

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

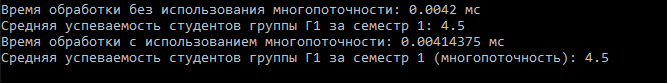
Автоматически созданное описание

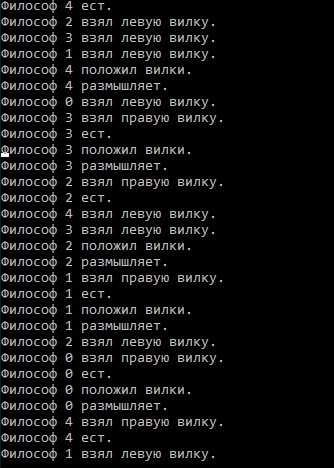
Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание





**Вывод**: В первой задании мы сравнили несколько видов синхронизации, получили то, что примерно они выполняются за одинаковое количество времени, однако SpinLock и Monitor на 1 секунду дольше выполняются, чем остальные. Во втором задании мы сравнили использование многопоточности и её неиспользование, если объём данных большой, то многопоточность окупает свои затраты на создание потока, а если нет, то она является невыгодным вложением в наше хорошее будущее. В третьем задании у нас бесконечный цикл с использованием потоков, грубо говоря, у нас едят философы и размышляют о жизни, если им разрешает поток. Все программы прекрасно работают: в контрольном примере предоставлены их результаты работы.