МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра защиты информации



**ЛАБОРАТОРНАЯ РАБОТА №4**

«Многопоточность»

**по дисциплине: « Программирование »**

|  |  |
| --- | --- |
| Выполнил:  *Студент гр. «АБ-321», «АВТФ»*  *Морозов Даниил Иванович*  «21» февраля 2025г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Ассистент кафедры ЗИ*  *Исаев Глеб Андреевич*  «\_\_\_» \_\_\_\_\_\_ 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

****Цели и задачи работы:****

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

****Задание к работе:****

**Самостоятельно решить задачи в соответствии с индивидуальным вариантом.**

**Текст программ доступен по ссылке** <https://github.com/Mirmordan/S3L3-6>

****Задание 1.****

**Реализовать параллельный запуск заданного числа потоков, содержащих случайные символы из ASCII таблицы в формате гонки. Проанализировать особенности (сравнительный анализ):**

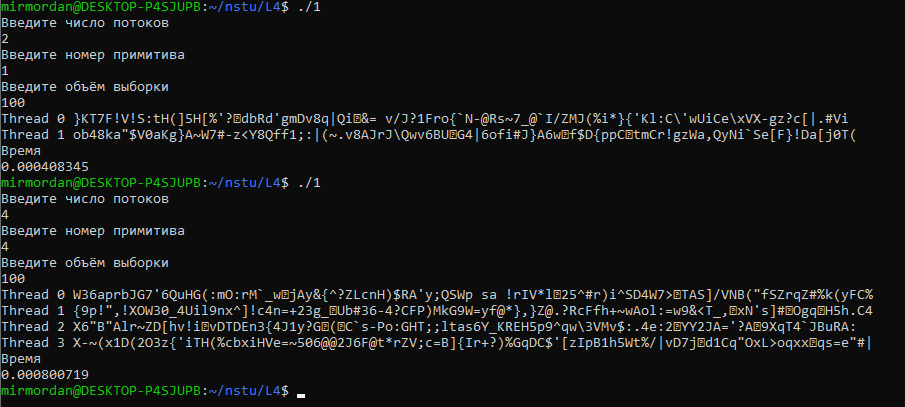
* **Mutexes;**
* **Semaphore;**
* **SemaphoreSlim;**
* **Barrier;**
* **SpinLock;**
* **SpinWait;**
* **Monitor.**

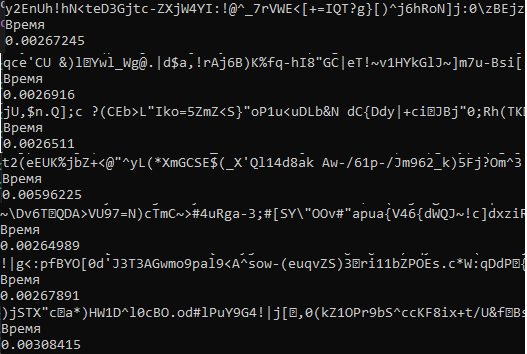
**Проанализировать скорости работы примитивов с использованием StopWatch и BenchMarkDotNet.**

**Большинство упомянутых примитивов занимаются блокировкой ресурсов (потока) пока ресурс не освободится. Есть единичные блокировки (Mutex,Monitor,Spinlock,Spinwait), которые блокируют поток пока ресурс не освободится, есть те , которые пропускают лишь ограниченное число потоков (semaphore,semaphore slim), например одновременно лишь 2.**

**Из всех примитивов выделяется только барьер, который останавливает потоки на определённом участке кода, пока все остальные потоки не достигнут его**

**Работа примитивов реализвуется классом Locker, в котором при инициализации задаётся тип примитива, методы lock и unlock делают разные действия в зависимости от вида примитива (во всех случаях кроме барьера они осуществляют блокировку и разблокировку по примитиву, барьер в методе lock простаивает).**

**Рис.1. Работа программы.**

**Рис.2. Разное время работы примитивов по порядку в задании.**

**Как мы видим, практически все примитивы обладают одним и тем же временем обработки примерно 0.0026с, потому что примитивы настроенны на вывод только одного потока в одно время и не нагружены частой синхронизацией.**

**Выделяется барьер, тут он не обеспечивает потокобезопасность и выполняет свою функцию — задерживает потоки пока все не достигнут точки синхронизации, поэтому время обработки больше.**

**Также монитор имеет повышенные затраты на синхронизацию, поэтому потребляет немного больше ресурсов.**

****Задание 2.****

**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.**

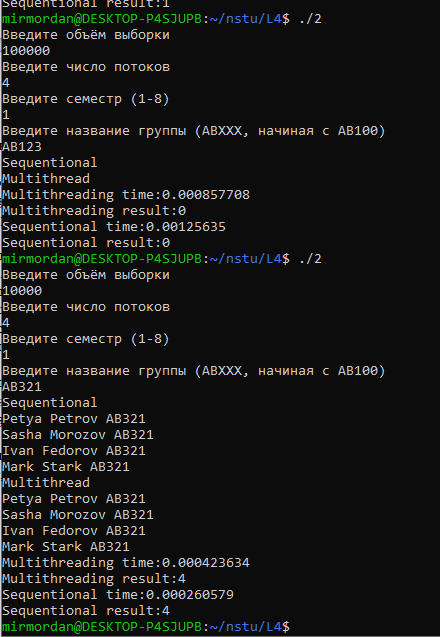
**Вариант 14.**

**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 список студентов группы Г, получивших стипендию за семестр С, и количество таких студентов.**

**Конструктор структуры (студент) задаёт ему все параметры студента и результатов сессий псевдослучайно (группы раздаются последовательно на 30 человек в формате AB<число>, начиная с AB100) Таким образом, при создании массива с данными мы имеем кучу групп по 30 студентов со случайными результатами сессии.**

**Многопоточная обработка состоит в том, что у определённого потока свои границы обработки массива данных, которые задаются при инициализации индивидуально, таким образом все данные делятся на непересекающиеся сектора, в которых отдельно работает один поток.**

**При большем количестве примитивов ( у меня только обязательный мьютекс для вывода) время многопоточной обработки страдает, а сейчас ощутимый прирос скорости наблюдается только при огромной выборке. До этого каждый поток атомарно брал себе уникальную строку и работал с ней, но этот подход создаёт необходимость большей синхронизации и добавляет времени обработки, текущим подходом я сократил расходы на синхронизацию получения данных и уменьшил затраты синхронизации на количество совершаемой потоком работы.**

**Рис.3. Многопоточная и простая обработка.**

****Задание 3.****

**Задача обедающих философов**

**Пять безмолвных философов сидят вокруг круглого стола, перед каждым философом стоит тарелка спагетти. На столе между каждой парой ближайших философов лежит по одной вилке.**

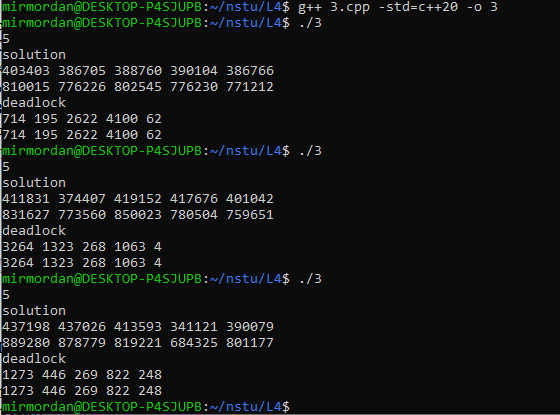
Каждый философ может либо есть, либо размышлять. Приём пищи не ограничен количеством оставшихся спагетти — подразумевается бесконечный запас. Тем не менее, философ может есть только тогда, когда держит две вилки — взятую справа и слева (альтернативная формулировка проблемы подразумевает миски с рисом и [палочки для еды](https://ru.wikipedia.org/wiki/Палочки_для_еды) вместо тарелок со спагетти и вилок).

Каждый философ может взять ближайшую вилку (если она доступна) или положить — если он уже держит её. Взятие каждой вилки и возвращение её на стол являются раздельными действиями, которые должны выполняться одно за другим.

Вопрос задачи заключается в том, чтобы разработать модель поведения ([параллельный](https://ru.wikipedia.org/wiki/Параллелизм_(информатика)) [алгоритм](https://ru.wikipedia.org/wiki/Алгоритм)), при котором ни один из философов не будет голодать, то есть будет вечно чередовать приём пищи и размышления.

Я решил задачу очень просто, каждый философ пытается взять вилку слева, затем справа, кушает и кладёт обе вилки обратно. Чтобы исключить взаимную блокировку был добавлен семафор, таким образом философы никогда не попадут в блокировку.

В качестве ресурсов выступает счётчик для каждого философа, который считает сколько он поел. Роль вилок выполняют мьютексы, которые потоки-философы поочерёдно блокируют. В качестве примера представлено две модели — философы, которые попадают в блокировку и философы, которые избегают блокировки семафором. Результаты представлены спустя одну и две секунды работы программы соответственно.

Рис.4. Философы без и с блокировкой.

Как мы видим, на каком-то этапе все философы берут левые вилки и ожидают вилки справа, вариант с семафором (Solution) не позволяет им этого сделать и избегает блокировки.

Вывод.

В результате выполнения лабораторной работы я изучил работу примитивов синхронизации, многопоточности в C++ и некоторые проблемы синхронизации не только в виде теоретического материала, но и на практике. Данный материал оказался очень полезен, поскольку правильная многопоточная обработка может сократить время выполнения в разы, однако это должно быть уместно и потокобезопасно.