Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский**

**политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Вычислительной математики, механики и биомеханики

Направление: 09.04.02 «Информационные системы и технологии»

Программа: «Информационные технологии и системная инженерия»

**ЛАБОРАТОРНЫЕ РАБОТЫ**

по дисциплине

**«Параллельное программирование»**

Выполнил:

ИТСИ-24-1м

студент гр.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Новожилов А. С.

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

(Ф.И.О.)

Принял:

Истомин Д.А.

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(должность, ФИО)*

*\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(оценка) (подпись)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(дата)*

**Пермь 2025**

**Лабораторная работа № 1**

**Условие:** реализовать функцию умножения двух массивов из 4 чисел с использованием SSE инструкций и сравнить её производительность с последовательной реализацией.

**Алгоритм:**

1. Последовательная реализация: поэлементно умножает два массива из 4 float чисел в цикле.
2. SSE реализация:

* Загружает оба массива в XMM регистры (movups).
* Выполняет параллельное умножение четырех пар чисел (mulps).
* Сохраняет результат обратно в массив (movups).

1. Обе реализации: замеряется время выполнения заданного количества итераций каждой функции для сравнения производительности.

**Реализация:**

100000 итераций 2.38 мс

100000 итераций 0.62 мс

**Выводы:** SSE реализация быстрее примерно в 4 раза, чем последовательная благодаря параллельному умножению. Использование SSE инструкций позволило оптимизировать умножение массивов.

**Лабораторная работа № 2**

**Условие:** создать программу, использующую Pthreads для запуска n потоков, выполняющих “длительную” операцию (вычисление квадратного корня). Сравнить производительность с последовательным выполнением той же операции.

**Алгоритм:**

1. Pthreads реализация:

* Создает n потоков.
* Каждый поток выполняет “длительную” операцию (вычисление квадратного корня в цикле - sqrt(i)).
* Используется pthread\_create для создания потоков.
* Используется pthread\_join для ожидания завершения всех потоков.

1. Последовательная реализация: в цикле, n раз выполняется “длительная” операция, аналогичная той, что выполняется в потоках.
2. Обе реализации: измеряется время выполнения (с помощью gettimeofday) для заданного числа потоков/итераций.

**Реализация:**

5 потоков pthreads 860.34 мс

5 потоков последовательное выполнение 3194.59 мс

**Выводы:** pthreads реализация быстрее последовательной, так как операции выполняются параллельно. Выигрыш в производительности зависит от количества потоков и от того, насколько хорошо ОС может распределять нагрузку между ядрами процессора. Использование многопоточности (Pthreads) позволяет эффективно использовать доступные вычислительные ресурсы для ускорения выполнения задач.

**Лабораторная работа № 3**

**Условие:** реализовать программу, использующую OpenMP для параллельного выполнения “длительной” операции (вычисления квадратного корня). Сравнить производительность с последовательной реализацией и реализацией на Pthreads (из предыдущей лабораторной работы).

**Алгоритм:**

1. OpenMP реализация:

* Использует директиву #pragma omp parallel for для распараллеливания цикла.
* num\_threads(thread\_num) устанавливает количество потоков OpenMP.
* Каждый поток выполняет “длительную” операцию (вычисление квадратного корня в цикле - sqrt(i)).

1. Pthreads и Последовательная реализации (используются из предыдущей лабораторной): как описано в предыдущем отчете.
2. Все реализации: измеряется время выполнения (с помощью gettimeofday) для заданного числа потоков/итераций.

**Реализация:**

5 потоков openmp 956.46 мс

**Выводы:** OpenMP, как и Pthreads, значительно быстрее последовательной реализации благодаря параллельному выполнению.

**Лабораторная работа № 4**

**Условие:** сравнение производительности и потокобезопасности Map (HashMap, Hashtable, synchronized HashMap, ConcurrentHashMap) в Java при многопоточном доступе.

**Алгоритм:** многопоточные операции чтения/записи (увеличение счетчика по ключу) для каждой реализации Map. Измерение времени выполнения и фиксация ConcurrentModificationException для HashMap.

**Реализация:**

Collections:

java.util.HashMapError accessing map:java.util.HashMap

Error accessing map:java.util.HashMap

...done.

java.util.Hashtable...done.

java.util.Collections$SynchronizedMap...done.

java.util.concurrent.ConcurrentHashMap...done.

Execution times:

HashMap: 0,338 s,

HashTable: 1,754 s,

SyncMap: 1,635 s,

ConcurrentHashMap: 0,265 s.

**Выводы:** ConcurrentHashMap обеспечивает наилучшую производительность в многопоточном окружении за счет синхронизации на уровне бакетов. HashMap не потокобезопасна. Hashtable и synchronized HashMap потокобезопасны, но медленнее из-за синхронизации на уровне методов коллекции.

**Лабораторная работа № 5**

**Условие:** реализовать считающий семафор, унаследовав его от Semaphore и используя ReentrantLock и Condition для синхронизации.

**Алгоритм:**

1. MySemaphore: собственная реализация семафора.

Использует ReentrantLock и Condition.

acquire(): блокирует, если нет разрешений, ждет через Condition. После этого блокируется стандартным Semaphore.

release(): увеличивает разрешения, сигнализирует потоку через Condition. Вызывает release() от Semaphore.

1. Использование: N потоков пытаются acquire() семафор, выполняют “работу” (sleep), затем release().
2. Контроль: ограничивает одновременную “работу” COUNT потоками. Измеряет и выводит максимальное количество одновременно работающих потоков.

**Реализация:**

Regular semaphore:

Поток pool-1-thread-3 работает. Активных потоков: 1

Поток pool-1-thread-1 работает. Активных потоков: 2

Поток pool-1-thread-2 работает. Активных потоков: 2

Поток pool-1-thread-4 работает. Активных потоков: 1

My semaphore:

Поток pool-2-thread-1 работает. Активных потоков: 1

Поток pool-2-thread-2 работает. Активных потоков: 2

Поток pool-2-thread-3 работает. Активных потоков: 1

Поток pool-2-thread-4 работает. Активных потоков: 2

**Выводы:** демонстрирует реализацию семафора с использованием ReentrantLock и Condition для синхронизации потоков и ограничения доступа к ресурсу.

**Лабораторная работа № 6**

**Условие:** разработать клиент-серверное приложение для обмена сообщениями с использованием сокетов.

**Алгоритм:**

1. Клиент:

* Подключается к серверу.
* Отправляет сообщения на сервер.
* Получает и отображает сообщения от сервера (broadcast).
* Вводит логин.

1. Сервер:

* Принимает подключения клиентов.
* Хранит список подключенных клиентов в CopyOnWriteArrayList (потокобезопасный).
* Принимает сообщения от клиентов.
* Пересылает сообщения всем подключенным клиентам (broadcast).
* Принимает ввод с консоли сервера и рассылает сообщения.

1. Обмен сообщениями: данные пересылаются через Socket (текстовые сообщения, обновления цен, и т.д.).

**Реализация:** клиент подключается к серверу.Сервер хранит информацию о клиентах.Сервер может отправлять сообщения всем клиентам одновременно.

**Выводы:** демонстрирует межпроцессное взаимодействие (IPC) с использованием сокетов в Java, обеспечивая базовый функционал чата.

**Лабораторная работа № 7**

**Условие:** запустить пример использования библиотеки MappedBus для обмена сообщениями между процессами.

**Алгоритм:**

* ObjectWriter: записывает сообщения PriceUpdate в разделяемую память (test-message). Необходимо запустить перед ObjectReader. (Должен быть отдельно)
* ObjectReader: считывает сообщения PriceUpdate из разделяемой памяти.

**Выводы:** ObjectReader выводит сообщения PriceUpdate из разделяемой памяти. Демонстрирует использование MappedBus для эффективного IPC.