Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное 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 технологии и системная инженерия
Профиль: «Информационные технологии и системная инженерия»

Лабораторные работы по дисциплине: «Параллельное программирование»

Выполнил
студент гр. ИТСИ-24-1м
Слаутин Максим Егорович

Принял Преподаватель кафедры ВММБ
Истомин Денис Андреевич

Пермь 2025

Лабораторная работа №1

Задание:

Необходимо:

1. При помощи SSE инструкций написать программу (или функцию), которая перемножает массив из 4х чисел размером 32 бита.
2. Написать аналогичную программу (или функцию), которая решает ту же задачу последовательно.
3. Сравнить производительность
4. Проанализировать сгенерированный ассемблер: gcc -S sse.c

Результаты:

```
10000000 последовательных итераций за 41.401 мс.
```

```
10000000 параллельных итераций за 8.541 мс.
```

1) Была написана программа, выполняющая перемножение массива из 4х чисел размером 32 бита при помощи SSE инструкций.

(10000000 параллельных итераций за 8.541 мс.)

2) Была написана программа, выполняющая эти действия последовательно.

(10000000 последовательных итераций за 41.401 мс.)

3) В результате сравнения производительности было выявлено, что программа, использующая SSE инструкции выполняется в 4.5 раза быстрее.

4) В сгенерированном коде ассемблера можно видеть, что операции, которые прописаны с помощью SSE были вставлены в код ассемблера, а прописанная последовательно операция умножения, была улучшена.

Лабораторная работа №2

Задание:

Необходимо:

1. При помощи Pthreads написать программу (или функцию), которая создает n потоков и каждый из потоков выполняет длительную операцию.
2. Написать аналогичную программу (или функцию), которая решает ту же задачу последовательно.
3. Сравнить производительность

Результаты:

```
MAIN: starting thread 0
MAIN: starting thread 1
MAIN: starting thread 2
MAIN: starting thread 3
MAIN: starting thread 4
MAIN: starting thread 5
```

```
MAIN: starting thread 6
MAIN: starting thread 7
MAIN: starting thread 8
MAIN: starting thread 9
        Thread #6 finished
        Thread #2 finished
        Thread #7 finished
        Thread #9 finished
        Thread #5 finished
        Thread #1 finished
        Thread #0 finished
        Thread #4 finished
        Thread #3 finished
        Thread #8 finished
10 потоков за 395.812 мс.
```

```
MAIN: starting thread 0
        Thread #0 finished
MAIN: starting thread 1
        Thread #1 finished
MAIN: starting thread 2
        Thread #2 finished
MAIN: starting thread 3
        Thread #3 finished
MAIN: starting thread 4
        Thread #4 finished
MAIN: starting thread 5
        Thread #5 finished
MAIN: starting thread 6
        Thread #6 finished
MAIN: starting thread 7
        Thread #7 finished
MAIN: starting thread 8
        Thread #8 finished
MAIN: starting thread 9
        Thread #9 finished
10 последовательных потоков за 3618.464 мс.
```

1) Была написана программа, которая используется pthreads, создавая многопоточное вычисление задачи.

(10 потоков за 395.812 мс.)

2) Была написана программа, выполняющая вычисление задачи последовательно.

(10 последовательных потоков за 3618.464 мс.)

3) При сравнении производительности выяснилось, что программа, использующая Pthreads, выполняется быстрее в 9 раз, чем та, что делает это последовательно.

Лабораторная работа №3

Задание:

Необходимо:

1. При помощи OpenMP написать программу (или функцию), которая создает n потоков и каждый из потоков выполняет длительную операцию.

2. Сравнить с последовательной программой и программой с Pthreads из предыдущей лабораторной работы.

Результаты:

```
OpenMP thread: 1
OpenMP thread: 3
OpenMP thread: 0
OpenMP thread: 9
OpenMP thread: 8
OpenMP thread: 5
OpenMP thread: 2
OpenMP thread: 7
OpenMP thread: 6
OpenMP thread: 4
        Thread #6 finished
        Thread #0 finished
        Thread #9 finished
        Thread #2 finished
        Thread #7 finished
        Thread #3 finished
        Thread #4 finished
        Thread #8 finished
        Thread #1 finished
        Thread #5 finished
10 потоков за 387.244 мс.
```

1) Была написана программа, при помощи OpenMP, которая создаёт n потоков, создавая многопоточное вычисление задачи.

(10 потоков за 387.244 мс.)

2) При сравнении производительности выяснилось, что программа, использующая OpenMP, выполняется чуть быстрее в 9.3 раз, чем та, что делает это последовательно, а также чуть быстрее чем та, что использует Pthreads.

Лабораторная работа №4

Задание:

Необходимо:

1. Написать программу, которая запускает несколько потоков
2. В каждом потоке считывает и записывает данные в HashMap, Hashtable, synchronized HashMap, ConcurrentHashMap
3. Модифицировать функцию чтения и записи элементов по индексу так, чтобы в многопоточном режиме использование непотокобесопасной коллекции приводило к ошибке
4. Сравнить производительность

Результаты:

```
Collections:
    java.util.HashMap...done. Errors: 89
    java.util.Hashtable...done. Errors: 0
    java.util.Collections$SynchronizedMap...done. Errors: 0
    java.util.concurrent.ConcurrentHashMap...done. Errors: 0
Execution times:
    HashMap: 0,067 s,
```

```
    HashTable: 0,270 s,
    SyncMap: 0,153 s,
    ConcurrentHashMap: 0,014 s.
```

Были рассмотрены 4 коллекции. Использовалось 50 потоков.

При работе с HashMap случались ошибки, связанные с конкурентным доступом к определенным ресурсам

Hashtable уже является потокобезопасной коллекцией и может обеспечить корректный результат при конкурентном доступе к бакетам.

Collections.synchronizedMap также является потокобезопасной коллекцией и может обеспечить корректный результат при конкурентном доступе.

ConcurrentHashMap также является потокобезопасной коллекцией и может обеспечить корректный результат при конкурентном доступе.

Также было произведено сравнение времени выполнения кода при использовании различных коллекций и кол-во ошибках при их работе:

Как видно из результатов, HashMap не стоит использовать, когда необходима коллекция, к которой будет конкурентный доступ. Наиболее быстрой и при этом безопасной коллекцией является ConcurrentHashMap. Связано это с тем, что в отличие от HashTable и synchronizedMap, которые блокируются полностью при доступе к ним, в ConcurrentHashMap блокируются лишь отдельные бакеты, что позволяет другим потокам в это время вести работу с другими бакетами.

Лабораторная работа №5

Задание:

Необходимо:

1. Написать программу, которая демонстрирует работу считающего семафора
2. Написать собственную реализацию семаформа (наследование от стандартного с переопределением функций) и использовать его

Результаты:

```
Regular semaphore:
Поток pool-1-thread-1 работает. Активных потоков: 1
Поток pool-1-thread-2 работает. Активных потоков: 2
Поток pool-1-thread-3 работает. Активных потоков: 3
Поток pool-1-thread-5 работает. Активных потоков: 1
Поток pool-1-thread-6 работает. Активных потоков: 3
Поток pool-1-thread-4 работает. Активных потоков: 2
Поток pool-1-thread-7 работает. Активных потоков: 1
Максимальное количество активных потоков: 3

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

```
Поток pool-2-thread-4 работает. Активных потоков: 1
Поток pool-2-thread-6 работает. Активных потоков: 3
Поток pool-2-thread-5 работает. Активных потоков: 2
Поток pool-2-thread-7 работает. Активных потоков: 3
Максимальное количество активных потоков: 3
```

Создана реализации семафора коллекции - с применением ReentrantLock и с применением мониторов (synchronized методов). Использовалось 7 потоков, а вместимость семафора была задана равной 3.

Как видно из результатов, оба семафора работают корректно и обеспечивают ограничение на максимальное количество активных потоков.

Лабораторная работа №6

Задание:

Необходимо создать клиент-серверное приложение:

1. Несколько клиентов, каждый клиент - отдельный процесс
2. Серверное приложение - отдельный процесс
3. Клиенты и сервер общаются с использоваени Socket

Необходимо релизовать функционал:

1. Клиент подключается к серверу
2. Сервер запоминает каждого клиента в java.util.concurrent.CopyOnWriteArrayList
3. Сервер читает ввод из консоли и отправляет сообщение всем подключенным клиентам

Результаты:

```
Server is running and waiting for connections...
New client connected: Socket[addr=/127.0.0.1,port=53527,localport=12345]
New client connected: Socket[addr=/127.0.0.1,port=53533,localport=12345]
User Максим connected.
User Точно не Максим connected.
[Максим]: Здорова
[Точно не Максим]: Привет
```

```
Connected to the chat server!
Enter your username:
Максим
Welcome to the chat, Максим!
Type Your Message
Здорова
[Точно не Максим]: Привет
```

```
Connected to the chat server!
Enter your username:
Точно не Максим
Welcome to the chat, Точно не Максим!
Type Your Message
```

```
[Максим]: Здорова
Привет
```

В рамках задачи выполнялось ознакомление и работа с Java IPC. Созданы клиент и сервер, которые обмениваются между собой сообщениями через socket.

Как видно из результатов, процессы-клиенты и сервер взаимодействуют корректно, выводя сообщения.

Лабораторная работа №7

Задание:

Необходимо:

1. Изучить библиотеку mappedbus
2. Запустить готовые тестовые примеры

Результаты:

```
Read: PriceUpdate [source=0, price=146, quantity=292], hasRecovered=true
Read: PriceUpdate [source=1, price=136, quantity=272], hasRecovered=true
Read: PriceUpdate [source=0, price=148, quantity=296], hasRecovered=true
Read: PriceUpdate [source=1, price=138, quantity=276], hasRecovered=true
Read: PriceUpdate [source=0, price=150, quantity=300], hasRecovered=true
Read: PriceUpdate [source=1, price=140, quantity=280], hasRecovered=true
Read: PriceUpdate [source=0, price=152, quantity=304], hasRecovered=true
Read: PriceUpdate [source=1, price=142, quantity=284], hasRecovered=true
Read: PriceUpdate [source=0, price=154, quantity=308], hasRecovered=true
Read: PriceUpdate [source=1, price=144, quantity=288], hasRecovered=true
Read: PriceUpdate [source=0, price=156, quantity=312], hasRecovered=true
Read: PriceUpdate [source=1, price=146, quantity=292], hasRecovered=true
```

Запускается 2 ObjectWriter, которые пишут данные в memory mapped файл. После этого запускается ObjectReader, считывающий данные из этого файла, при этом выводя данные в консоль.