|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
| Институт информационных технологий (ИТ) |
| Кафедра инструментального и прикладного программного обеспечения |

|  |  |
| --- | --- |
| **ОТЧЕТ**  **ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 1** | |
| **по дисциплине** |  |
| **«Архитектура клиент-серверных приложений»**  **Тема: Многопоточные приложения, как основа**  **распределённых клиент-серверных вычислений** | |
| Выполнил студент группы ИКБО-20-22 | Шумахер М.Е. |
| Принял преподаватель кафедры ИиППО | Волков М.Ю. |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | «04» сентября 2024 г. |  |
| «Зачтено» | «04» сентября 2024 г. |  |

Москва 2024

**Цель работы:** ознакомится с моделью многопоточного программирования в Java.

**Теоретическое введение:**

В отличие от многих других языков программирования, java предлагает встроенную поддержку многопоточного программирования. Многопоточная программа содержит две или более частей, которые могут выполняться одновременно. Каждая часть такой программы называется потоком (thгead), и каждый поток задает отдельный путь выполнения. То есть, многопоточность ­ это специализированная форма многозадачности.

Существуют два отдельных типа многозадачности: многозадачность, основанная на процессах, и многозадачность, основанная на потоках. Важно понимать разницу между ними. Большинству читателей многозадачность, основанная на процессах, является более знакомой формой.

**Класс Thread.** В Java функциональность отдельного потока заключается в классе Thread. И чтобы создать новый поток, нам надо создать объект этого класса. Но все потоки не создаются сами по себе. Когда запускается программа, начинает работать главный поток этой программы. От этого главного потока порождаются все остальные дочерние потоки.

**Реализация интерфейса Runnable.** Другой способ определения потока представляет реализация интерфейса Runnable. Этот интерфейс имеет один метод run. В методе run() собственно определяется весь тот код, который выполняется при запуске потока.

**Синхронизация потоков. Оператор synchronized.** При работе потоки нередко обращаются к каким-то общим ресурсам, которые определены вне потока, например, обращение к какому-то файлу. Если одновременно несколько потоков обратятся к общему ресурсу, то результаты выполнения программы могут быть неожиданными и даже непредсказуемыми.

**Методы wait и notify.** Иногда при взаимодействии потоков встает вопрос о извещении одних потоков о действиях других. Например, действия одного потока зависят от результата действий другого потока, и надо как-то известить один поток, что второй поток произвел некую работу.

**Семафоры.** Семафоры представляют еще одно средство синхронизации для доступа к ресурсу. В Java семафоры представлены классом Semaphore, который располагается в пакете java.util.concurrent.

Для управления доступом к ресурсу семафор использует счетчик, представляющий количество разрешений. Если значение счетчика больше нуля, то поток получает доступ к ресурсу, при этом счетчик уменьшается на единицу. После окончания работы с ресурсом поток освобождает семафор, и счетчик увеличивается на единицу. Если же счетчик равен нулю, то поток блокируется и ждет, пока не получит разрешение от семафора.

**Блокировки. ReentrantLock.** Для управления доступом к общему ресурсу в качестве альтернативы оператору synchronized мы можем использовать блокировки. Функциональность блокировок заключена в пакете java.util.concurrent.locks.

Вначале поток пытается получить доступ к общему ресурсу. Если он свободен, то на него накладывает блокировку. После завершения работы блокировка с общего ресурса снимается. Если же ресурс не свободен и на него уже наложена блокировка, то поток ожидает, пока эта блокировка не будет снята.

**Постановка задачи:** используя материалы данной практической работы необходимо написать многопоточную программу, в которой два потока записывают строку в стандартный вывод, по образцу PING PONG PING PONG PING PONG. Программа должна работать следующим образом:

1-й поток печатает «Ping» и переходит в состояние ожидания.

2-й поток выходит из состояния ожидания, печатает «Pоng», уведомляет 1-й поток, возвращается в состояние ожидания.

1-й поток выходит из состояния ожидания, печатает «Pшng», уведомляет 2-й поток, возвращается в состояние ожидания.

Шаги 2 и 3 повторяются и печатают «Ping Pong».

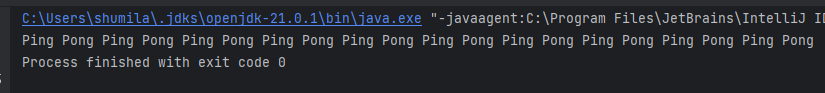
Программа должна быть реализована только с использованием Wait Notify, либо ReentrantLock.

**Программный код:**

Main.java:

package prac1;  
  
import java.util.concurrent.locks.Condition;  
import java.util.concurrent.locks.ReentrantLock;  
public class Main {  
 static final ReentrantLock *lock* = new ReentrantLock();  
 static final Condition *pingCondition* = *lock*.newCondition();  
 static final Condition *pongCondition* = *lock*.newCondition();  
 static boolean *pingTurn* = true; // флаг  
 public static void main(String[] args) {  
  
 Thread pingThread = new Thread(() -> {  
 for (int i = 0; i < 10; i++) {  
 *lock*.lock();  
 try {  
 while (!*pingTurn*) {  
 *pingCondition*.await();  
 }  
 System.*out*.print("Ping ");  
 *pingTurn* = false;  
 *pongCondition*.signal();  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 } finally {  
 *lock*.unlock();  
 }  
 }  
 });  
  
 Thread pongThread = new Thread(() -> {  
 for (int i = 0; i < 10; i++) {  
 *lock*.lock();  
 try {  
 while (*pingTurn*) {  
 *pongCondition*.await();  
 }  
 System.*out*.print("Pong ");  
 *pingTurn* = true;  
 *pingCondition*.signal();  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 } finally {  
 *lock*.unlock();  
 }  
 }  
 });  
  
 pingThread.start();  
 pongThread.start();  
 }  
}

Вывод программы:



**Вывод**: в результате выполнения практической работы была освоена работа с моделью многопоточного программирования в Java.

**БИБЛИОГРАФИЧЕСКИЙ СПИСОК**

1. Многопоточность в Java / [Электронный ресурс] // Habr.com : [сайт]. — URL: https://habr.com/ru/articles/164487/ (дата обращения: 04.09.2024).
2. Wait and notify Methods in Java / [Электронный ресурс] // Baeldung.com : [сайт]. — URL: https://www.baeldung.com/java-wait-notify (дата обращения: 04.09.2024).
3. В чем разница между мьютексом, монитором и семафором / [Электронный ресурс] // Javarush.com : [сайт]. — URL: https://javarush.com/groups/posts/2174-v-chem-raznica-mezhdu-mjhjuteksom-monitorom-i-semaforom (дата обращения: 04.09.2024).