Практикум по курсу

Базы данных и параллелизм 2020 г.

Задание №1

Напишите программу, которая создаёт нить.
Родительская и вновь созданная нити должны распечатать десять строк текста.

Задание №2

Модифицируйте программу из “Задания №1” так, чтобы вывод родительской нити производился после завершения дочерней.

Задание №3

Напишите программу, которая создаёт четыре нити, исполняющие один и тот же метод. Этот метод должен распечатать последовательность текстовых строк, переданных как параметр. Каждая из созданных нитей должна распечатать различные последовательности строк.

Задание №4

Дочерняя нить должна распечатывать текст на экран. Через две секунды после создания дочерней нити, родительская нить должна прервать её.

Задание №5

Модифицируйте программу из “Задания 4” так, чтобы дочерняя нить перед завершением распечатывала сообщение об этом.

Задание №6

В приложении №1 находится код программы, которую вам необходимо доработать.

Легенда:
1. Класс Company характеризует компанию, разделённую на отделы.
2. Каждый отдел (класс Department) умеет вычислять сумму от 1 до n, где n — это рандомное число от 1 до 6.
3. Каждая итерация суммирования занимает 1 секунду. Поэтому: если n равно 3, то на вычисление суммы 0 + 1 + 2 уйдёт 3 секунды (метод performCalculations).

Вам необходимо дописать реализацию класса Founder, в котором:
1. У вас будет находиться список со всеми Worker’ами (Runnable).
2. За каждым Worker’ом должен быть закреплён свой Department.
3. Каждый Worker должен инициировать запуск вычислений.
4. После того как во всех нитях завершатся вычисления, нужно вывести результат (метод showCollaborativeResult)

Примечание:
Следует разобраться как работают барьеры в Java.

Задание №7

Напишите программу, которая вычисляет число Пи при помощи ряда Лейбница (рис. 1). Количество потоков программы должно определяться параметром командной строки. Количество итераций может определяться во время компиляции.

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \ldots = \frac{\pi}{4}$$

рис. 1 — Ряд Лейбница

Задание №8

Модифицируйте программу из “Задания №7” так, чтобы сама по себе она не завершалась. Вместо этого, после получения сигнала SIGINT программа должна как можно скорее завершаться, собирать частичные суммы ряда и выводить полученное приближение числа.

Рекомендации:
Ожидаемое решение состоит в том, что вы установите обработчик SIGINT. Подумайте, как минимизировать ошибку, обусловленную тем, что разные потоки к моменту завершения успели пройти разное количество итераций. Скорее всего, такая минимизация должна обеспечиваться за счет увеличения времени между получением сигнала и выходом.

Задание №9

Напишите программу, которая будет симулировать известную задачу про обедающих философов. Пять философов сидят за круглым столом и едят спагетти. Спагетти едят при помощи двух вилок. Каждые двое философов, сидящих рядом, пользуются общей вилкой. Философ некоторое время размышляет, потом пытается взять вилки и принимается за еду. Съев некоторое количество спагетти, философ освобождает вилки и снова начинает размышлять. Еще через некоторое время он снова принимается за еду, и т.д., пока спагетти не кончатся. Если одну из вилок взять не получается, философ ждет, пока она освободится. Как вариант реализации: философы симулируются при помощи нитей, периоды размышлений и еды – при помощи засыпаний, а вилки – при помощи мьютексов. Философы всегда берут сначала левую вилку, а потом правую.

Дополнительно:
Объясните, при каких обстоятельствах это может приводить к мертвой блокировке. Измените протокол взаимодействия философов с вилками таким образом, чтобы мертвых блокировок не происходило.

Задание №10

Модифицируйте программу из “Задания №1” так, чтобы вывод родительской и дочерней нитей был синхронизирован: сначала родительская нить выводила первую строку, затем дочерняя, затем родительская вторую строку и т.д. Используйте мьютексы.

Примечание:
Явные и неявные передачи управления между нитями и холостые циклы разрешается использовать только на этапе инициализации.

Задание №11

Решите “Задание №10” с использованием двух семафоров-счетчиков.

Задание №12

Родительская нить программы должна считывать вводимые пользователем строки и помещать их в начало связанного списка. Строки длиннее 80 символов можно разрезать на несколько строк. При вводе пустой строки программа должна выдавать текущее состояние списка. Дочерняя нить пробуждается каждые пять секунд и сортирует список в лексикографическом порядке (используйте пузырьковую сортировку).

Примечание:
Все операции над списком должны синхронизоваться при помощи синхронизованных блоков на объектах головы.

Задание №13

Решите задачу из “Задания №9” при помощи атомарного захвата вилок. Когда философ может взять одну вилку, но не может взять другую, он должен положить вилку на стол и ждать, пока освободятся обе вилки.

Рекомендация: создайте еще мьютекс forks и условную переменную. При попытке взять вилку философ должен захватывать forks и проверять доступность обоих вилок. Если одна из вилок недоступна, философ должен освободить вторую вилку (если он успел ее захватить) и заснуть на условной переменной. Освобождая вилки, философ должен оповещать остальных философов об этом при помощи условной переменной. Тщательно продумайте процедуру захвата и освобождения мьютексов, чтобы избежать ошибок потерянного пробуждения.

Задание №14

Разработайте имитатор производственной линии, изготавливающей винтики (widget). Винтик собирается из детали C и модуля, который, в свою очередь, состоит из деталей A и B. Для изготовления детали A требуется 1 секунда, B – две секунды, C – три секунды. Задержку изготовления деталей имитируйте при помощи засыпания. Используйте семафоры-счётчики.

Задание №15

Реализуйте сервер, который принимает TCP соединения и транслирует их.

Сервер должен получать из командной строки следующие параметры:
1. Номер порта P, на котором следует слушать.
2. Имя или IP-адрес узла N, на который следует транслировать соединения.
3. Номер порта P', на который следует транслировать соединения.

Сервер принимает все входящие запросы на установление соединения на порт P. Для каждого такого соединения он открывает соединение с портом P' на сервере N. Затем он транслирует все данные, получаемые от клиента, серверу N, а все данные, получаемые от сервера N – клиенту. Если сервер N или клиент разрывают соединение, наш сервер также должен разорвать соединение. Если сервер N отказывает в установлении соединения, следует разорвать клиентское соединение.

Сервер не должен быть многопоточным и никогда не должен блокироваться при операциях чтения и записи. Не следует использовать неблокирующиеся сокеты.

Задание №16

Реализуйте простой HTTP-клиент. Он принимает один параметр командной строки – URL. Клиент делает запрос по указанному URL и выдает тело ответа на терминал как текст (т.е. если в ответе HTML, то распечатывает его исходный текст без форматирования). Вывод производится по мере того, как данные поступают из HTTP-соединения. Когда будет выведено более экрана (более 25 строк) данных, клиент должен продолжить прием данных, но должен остановить вывод и выдать приглашение Press space to scroll down. При нажатии пользователем клиент должен вывести следующий экран данных.

Приложение №1

Код класса Company.java

```
import java.util.ArrayList;
import java.util.List;

public final class Company {

    private final List<Department> departments;

    public Company(final int departmentsCount) {
        this.departments = new ArrayList<>(departmentsCount);
        for (int i = 0; i < departmentsCount; i++) {
            departments.add(i, new Department(i));
        }
    }

    /**
     * Вывод результата по всем отделам.
     * P.S. Актуально после того, как все отделы выполнят свою работу.
     */
    public void showCollaborativeResult() {
        System.out.println("All departments have completed their work.");

        final int result = departments.stream()
                .map(Department::getCalculationResult)
                .reduce(Integer::sum)
                .orElse(-1);

        System.out.println("The sum of all calculations is: " + result);
    }

    /**
     * @return Количество доступных отделов для симуляции выполнения
работы.
     */
    public int getDepartmentsCount() {
        return departments.size();
    }

    /**
     * @param index Индекс для текущего свободного отдела.
     * @return Свободный отдел для симуляции выполнения работы.
     */
    public Department getFreeDepartment(final int index) {
        return departments.get(index);
    }
}
```

Код класса Department.java

```java
import java.util.concurrent.ThreadLocalRandom;
import java.util.concurrent.TimeUnit;

public class Department {

    private final int identifier;
    private final int workingSeconds;
    private int calculationResult = 0;

    public Department(final int identifier) {
        this.identifier = identifier;
        this.workingSeconds = ThreadLocalRandom.current().nextInt(1, 6);
    }

    /**
     * Симуляция работы длительностью в workingSeconds секунд.
     * В данном случае просто вычисляем сумму.
     */
    public void performCalculations() {
        for (int i = 0; i < workingSeconds; i++) {
            try {
                Thread.sleep(TimeUnit.SECONDS.toMillis(1));
            } catch (final InterruptedException ignored) {
                // Ignored case.
            }
            calculationResult += i;
        }
    }

    /**
     * @return Уникальный идентификатор для отдела,
     * по которому мы можем отличить его от других.
     */
    public int getIdentifier() {
        return identifier;
    }

    /**
     * ВАЖНО!
     * Далеко не самый правильный способ вычисления и получения данных,
     * но для демонстрации работы барьера пойдёт.
     *
     * @return Результат вычислений.
     */
    public int getCalculationResult() {
        return calculationResult;
    }
}
```

Код класса Founder.java (незавершённый)

```
public final class Founder {

    private final List<Runnable> workers;

    public Founder(final Company company) {
        this.workers = new ArrayList<>(company.getDepartmentsCount());
    }

    public void start() {
        for (final Runnable worker : workers) {
            new Thread(worker).start();
        }
    }
}
```