Обзор урока

— зачем нужны потоки
— создание потоков, таймер
— критическая секция, атомарность операций, монитор
— безопасный счетчик (synchronized, atomic)
— deadlock
— wait()/notify()
— producer/consumer (blocking queue)

Материалы

базовые:

habrahabr.ru/post/164487/

www.skipy.ru/technics/synchronization.html

Хорстманн К., Корнелл Г. Java 2. Том 2. Тонкости программирования — Глава 1

javaigrun.ru/2010/04/09/potoki-v-java/

Продвинутые темы (не обязательно, но в будущем очень желательно)

обзор java.util.concurrent

habrahabr.ru/post/133981/

модель памяти java (видео доклада)

Задание

Сетевой чат

— свежим взглядом и с полным пониманием деталей посмотреть на код server/client с прошлого урока
— реализовать метод остановки потоков-хэндлеров на сервере (interrupt) и на клиенте - обрабатывать InterruptedException, который вылетает при прерывании блокирующей операции чтения/записи в сокет

— разберитесь с Executors/ExecutorService и переделайте серверную часть на FixedThreadPool. Попробуйте написать нагрузочный тест и увидеть, как отваливается клиент при заполненном пуле

Producer/Consumer

Это упражнение достаточно большое, предстоит разобраться с новыми понятиями. Заложите на него 2-5 часа.
Задача — быстрый поиск подстроки во всех файлах директории. Например, такой есть в FAR manager(win) и есть утилита grep (unix). Наш поисковик должен быть многопоточным.

Простая часть:
— из программы открыть большой текстовый файл file, на вход подать строку для поиска
— **[Producer]** создать поток, который обращается к file, начинает его считывать блоками (размер блока на ваше усмотрение, например 4096 байт). Как только блок прочитан, поток складывает его в BlockingQueue (java.util.concurrent.ArrayBlockingQueue)

- Размер BlockingQueue должен быть небольшим. Размер - это количество элементов, которое вмещает очередь. В нашей задаче размер не больше 10.

— **[Consumer]** создать поток, который читает BlockingQueue, если там есть данные, начинает их обрабатывать (искать подстроку). Реализация ArrayBlockingQueue автоматически блокируется, если очередь пустая.

- Реализация алгоритма поиска подстроки в этой задаче не так важна, может быть тривиальной. Важен момент передачи данных между потоками через очередь

— Замерьте время обработки всех файлов с помощью System.nanoTime(). Нужно дождаться окончания работы всех потоков.

```
long start = System.nanoTime();
// стартуем все потоки start()
// ждем окончания всех join()
long elapsedTime = System.nanoTime() - start; // сколько времени заняло
```

Сложная часть:
— сгенерировать текстовые файлы (можно бессмысленные) в директории. Размер файлов пусть будет около 10 мб (Можете создать руками, но вы же программисты ;)). Итого должны получить около 10-20 файлов размером 10 мб каждый, в сумме 100-200 мб текста
— вашей программе на вход подается путь к директории и строка для поиска
— считать имена файлов из директории и создать объекты типа File, положить эти объекты в некий список Collection files. Используйте синхронизированную коллекцию ConcurrentLinkedQueue — иначе несколько потоков могут взять один и тот же файл. Здесь перед нами встает задача синхронизации, есть общий ресурс - список файлов, нужно предотвратить ситуацию, когда несколько потоков берут один и тот же файл. Можно ограничить доступ к списку, сделав все его методы синхронизированными, либо воспользоваться готовой реализацией.
— Вместо одного потока создайте пул потоков producer и пул потоков consumer — по 4 потока. producer-потоки берут файл из коллекции, пока она не пуста. Пул
— Замерьте время заново