Документация

1.Увод

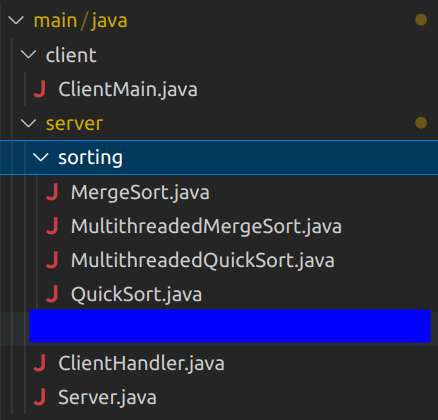
Проектът представялява многонишков сървър, който може да обсулжва множество клиента едновременно. Клиентите могат да пращат заявки на сървъра, които биват обработвани и резултатът се праща обратно на клиента. Заявките се състоят от тип на алгоритъм за сортировка, брой нишки(ако алгоритъмът е многонишков) и масив който да бъде сортитран.

2.Основни концепции и алгоритми

В проекта за сега са имплементирани четири сортиращи алгоритъма: еднонишков merge sort, многонишков merge sort (който работи с динамичен брой нишки), еднонишков quick sort и многонишков оптимизиран quick sort(който работи с броят подадени нишки от клиента). За сървъра са заделени 5 нишки, които да обслужват клиенти и техните заявки. Тези 5 нишки могат да създават допълнителни нишки при необходимост за дадена заявка. Броят на заделените нишки (5), ограничава броят на клиентите, които могат да бъдат обслужени едновременно. Повече за това в частта реализация.

3. ООП архитектура

Проекта е разделен в два главни пакета. Единият пакет е за клиентския интерфейс и логика, а другият е за сървърния интерфейс и логика, заедно със сортиращите алгоритми, които са подпакет. В класът ClientMain може да се реализира валидация на входа и по-добре изглеждащ UI. В server пакет имаме клас ClientHandler, който служи за обработването на заявки от даден клиент. Инстанции на този клас създава класът Server.



4.Реализация

Java ни дава обширен инструментариум относно многонишков код, който е разновиден и лесен за ползване.

Server класът е реализиран ползвайки вградения клас Socket за мрежовата част на проекта и чрез класа ExecutorService, който е инициализиран с фиксиран нишков басейн (5 нишки). Когато клиент се свърже чрез сокета, той бива приет от сървъра и се създава обект от клас ClientHandler. Този обект се добавя към опашка (неявно). Тази опашка бива обработена от фиксираните 5 нишки, където всяка нишка взима задача, която да изпълнява.

Многонишковия merge sort е реализиран чрез ForkJoinPool фреймуърк. Този фреймуърк е полезен, защото позволява “work-stealing”, което балансира работата измежду нишките.

Многонишковия quick sort е реализиран чрез същия фреймуърк, но с фиксиран брой нишки подадени от клиента. При масиви с елементи по малко от 1000, не се заделят допълнителни нишки за отрязъците, а една нишка сортира целия масив. Това се прави, за да се спести време и ресурси, защото създаването на нишки и тяхното приспиване, терминиране отнема не малко време (Това са Kernel извиквания които са скъпи).

5.Резултати от тествания

Подаден е масив от 10 000 000 елемента. Времето в дясно е за колко време е сортиран дадения масив с конкретния алгоритъм.

Еднонишков mergesort -> 205ms

Многонишков mergesort -> 136ms

Еднонишков quicksort -> 110ms

Многонишков quicksort с 4 нишки -> 70ms

Многонишков quicksort с 8 нишки -> 32ms

Аномалията относно резултатите на еднонишковия qs и многонишковия ms, най-вероятно се дължи на факта, че самия meregesort не може да бъде in-place за разлика от quicksort, което означава, че трябва динамично да се заделя и копира памет, което бави алгоритъма.

6. Заключение

Този проект дава възможността да бъде изследвана ефективността на различни алгоритми от многобройни клиенти паралелно. Проектът има проста йерархия, което позволява лесното надграждане на самия проект. В бъдещ план ще е добре да има повече алгоритми за сравнение и самия сървър да бъде реализиран чрез селектор.