Technologies for Big Data Analytics

Εργασία 1 - Threads and Processes - Υποερώτημα 4

Antonis Prodromou

2025-11-21

Πίνακας περιεχομένων

1 Περιγραφή Λύσης - Υποερώτημα 4

1.1 Σκοπός

Ο σκοπός αυτής της εργασίας είναι η δημιουργία ενός συστήματος διεργασιών που αποτελείται από διεργασίες τύπου consumer, producer και server οι οποίες:

- Επικοινωνούν με TCP sockets.
- Οι server (πάνω από ένας) μπορούν να αποθηκεύσουν προϊόντα που κυμαίνονται μεταξύ 1 και 1000 σε αριθμό.
- Ο κάθε παραγωγός (πάνω από ένας) μπορεί να προσθέσει μεταξύ 10 και 100 μονάδων στην αποθήκευση στο server.

- Ο κάθε consumer (πάνω από ένας) συνδέεται τυχαία σε έναν από τους servers και αφαιρεί μεταξύ 10 και 100 μονάδων από την αποθήκευση.
- Κάθε server μπορεί να υποστηρίζει ταυτόχρονα πολλούς consumers και πολλούς producers, άρα θα πρέπει να υποστηρίζει multi-threading. Αυτό στην πράξη σημαίνει πως όποτε ένας client προσπαθεί να συνδεθεί, ένα διαφορετικό νήμα αναλαμβάνει να χειριστεί το αίτημά του.

1.2 Κλάσεις

Δημιουργούμε τις κλάσεις `Server`, `Producer` και `Consumer`, με επιπρόσθετη την κλάση `ClientHandler`, για να χειρίζεται τα αιτήματα των πελατών.

1.2.1 Server

Ο Server δέχεται ως παράμετρο μια θύρα, εν προκειμένω τις 8881, 8882, 8883.

Η μέθοδος main() του server αναλαμβάνει αφενός να δημιουργήσει ένα στιγμιότυπο Server και έπειτα να καλέσει επ' αυτού τη μέθοδο `start()`. Η `start()` ανοίγει ένα server socket, το οποίο δέχεται τα αιτήματα σύνδεσης των client (`.accept()`) για όσο ο διακόπτης λειτουργίας running είναι `True` (ο διακόπτης επεξηγείται λεπτομερώς παρακάτω).

Επιπλέον, δημιουργεί ένα νέο νήμα πάνω στο οποίο τρέχει η εφαρμογή ClientHandler. Το νήμα χρειάζεται προκειμένου ο server να είναι multi-threaded και να μπορεί να χειρίζεται αιτήματα από διαφορετικούς clients:

```
new Thread(new ClientHandler(clientSocket, this, MAX_STORAGE)).start();
```

Το `this` αναφέρεται στο στιγμιότυπο του Server, καθώς επί αυτού εφαρμόζεται η `start()`, που με τη σειρά της δημιουργεί έναν νέο `ClientHandler`.

1.2.2 ClientHandler

Η `ClientHandler` τρέχει σε ένα νήμα, και λειτουργεί ως σύνδεσμος επικοινωνίας μεταξύ client και server, τροποποιώντας την αποθήκη του server, βάση των εντολών που δέχεται από τον client. Παίρνει τρεις παραμέτρους: τα socket του server και του πελάτη, καθώς και τη μέγιστη επιτρεπόμενη χωρητικότητα του server. Ως κλάση, υλοποιεί την διεπαφή Runnable, που σημαίνει πως μπορούν να την χρησιμοποιήσουν πολλά νήματα. Όταν τρέχει, δημιουργεί:

- Ένα κανάλι εισερχομένων `InputStreamReader` που εσωκλείεται σε ένα `BufferReader` για να διαβάζει τις εντολές client και consumer ανά μπλοκ και όχι χαρακτήρα-χαρακτήρα.
- Ένα κανάλι εξερχομένων μέσω του `getOutputStream`, που εσωκλείεται σε `PrintWriter` για να απαντάει στον πελάτη.

Ακολούθως οι εντολές από producer και consumer διαβάζονται γραμμή γραμμή (`in.readLine()`) και καλείται η `handleCommand`. Αυτή η μέθοδος ανάλογα με το εισερχόμενο μήνημα κάνει τα εξής:

- `"ADD"`: προσθέτει την τιμή στην αποθήκευση του storage με `server.storage += addValue`
- `"SUB"`: αφαιρεί την τιμή από την αποθήκευση του storage με `server.storage -= subValue;`

Προτού εκτελέσει οποιαδήποτε από τις παραπάνω εντολές, η `ClientHandler` αποκτά τον έλεγχο του monitor του αντικειμένου `storageLock` όπως εξηγείται αναλυτικά στην παράγραφο «Συγχρονισμός» παρακάτω.

1.2.3 Producer

Ο Producer ως κλάση υλοποιεί την διεπαφή Runnable, που σημαίνει πως μπορούν να την χρησιμοποιήσουν πολλά νήματα. Στη συγκεκριμένη εφαρμογή δημιουργούμε 3 νήματα μέσω μιας for loop στη `main()`.

Στην κομμάτι της κυρίως εκτέλεσής της (`run()`), διαλέγουμε ως θύρα μια τυχαία θέση index της λίστας με τις θύρες του server [8881, 8882, 8883], καθώς και έναν τυχαίο αριθμό μεταξύ 10 και 100 (με `10 + random.nextInt(91)`). Στη συνέχεια, βάζουμε το prefix `ADD` - εφόσον ο producer προσθέτει μόνο στον server - ακολουθούμενη από τον αριθμό που θέλουμε να προστεθεί. Αυτά τα περνάμε μαζί στο κανάλι εξερχομένων που καταλήγει στον ClientHandler (`out.println(command);`). Εάν δεν έχουμε απάντηση από τον ClientHandler,

σπάμε τη λούπα, ολοκληρώνεται η run() για το συγκεκριμένο thread, και αυτό σταματιέται από την JVM.

1.2.4 Consumer

Στον consumer ακολουθείται ακριβώς η ίδια διαδικασία με τον Producer, με τη διαφορά της αφαίρεσης από το στοκ (SUB) αντί για αποθήκευση.

1.3 Συγχρονισμός

Για τον σωστό συγχρονισμό των νημάτων θα χρησιμοποιήσουμε έναν μηχανισμό συγχρονισμού block, για τον έλεγχο της κλειδαριάς (lock) του monitor ενός αντικειμένου. Το σκεπτικό βασίζεται στο ότι το στιγμιότυπο κάθε κλάσης στην Java έχει εξ' ορισμού έναν εσωτερικό μηχανισμό που ονομάζεται `monitor`, που διασφαλίζει την αποκλειστική πρόσβαση στην κατάσταση του αντικειμένου ανά πάσα στιγμή. Ζητάμε λοιπόν από τα νήματα που θέλουμε να εκτελέσουν ένα κομμάτι κώδικα, να αποκτήσουν πρώτα το `monitor` για ένα τυχαίο αντικείμενο, χρησιμοποιώντας ένα `synchronized` statement, που έχει την παρακάτω μορφή:

```
synchronized (server.storageLock) {
  ///
}
```

Στην προκειμένη περίπτωση, το νήμα ζητά το `monitor` του αντικειμένου `storageLock`. Όσο διαρκεί η εκτέλεση του κώδικα ανάμεσα στις αγκύλες, το `monitor` κατέχεται από το νήμα. Εάν σε αυτό το διάστημα ένα άλλο νήμα προσπαθήσει να πάρει το monitor, δεν μπορεί, καθώς δύο διεργασίες δεν επιτρεπεται να έχουν ταυτόχρονη πρόσβαση στο ίδιο monitor. Μόλις ελευθερωθεί το `monitor`, είναι στη διάθεση του κάθε νήματος για να το αποκτήσει, χωρίς να τηρείται κάποια προτεραιότητα.

Στη δική μας εφαρμογή, χρησιμοποιούμε synchronized statements σε κάθε περίπτωση που θέλουμε πρόσβαση στη μεταβλητή `storage`, που είναι ο ακέραιος που συμβολίζει την αποθήκη του server.

1.4 Έλεγχος λειτουργίας του προγράμματος

Ο έλεγχος της λειτουργίας του προγράμματος γίνεται μέσω ενός διακόπτη: πρόκειται για μια μεταβλητή τύπου `AtomicBoolean` με την ονομασία `running`. Η επιλογή αυτού του τύπου γίνεται γιατί η AtomicBoolean διασφαλίζει την ατομική πρόσβαση σε αυτήν, δηλαδή ένα μόνο νήμα ανά δεδομένη στιγμή μπορεί να δει την τιμή της (μέσω της μεθόδου `get()`) και να την ενημερώσει (`set()`), που δεν θα ήταν η περίπτωση με μια κοινή boolean μεταβλητή. Για να δεχθεί ο server αιτήματα σύνδεσης (δημιουργώντας client sockets) και να δημιουργήσει ένα thread για κάθε σύνδεση, θέτουμε ως προϋπόθεση η τιμή του running να είναι ίση με `True`, μέσω ενός while statement:

```
while (running.get()) {
  // δημιουργία συνδέσεων σε νήματα
}
```

1.5 Τερματισμός προγράμματος

Προκειμένου το πρόγραμμα να μην τρέχει αένεα, αλλά να διακόπτει με ομαλό τρόπο, δημιουργούμε καταρχήν μια μέθοδο του Server με όνομα `stop()`. Η λειτουργία της έγκειται στο να θέτει την τιμή της προαναφερθείσας AtomicBoolean running ως False (`running.set(false);`). Η `stop()` καλείται με δύο τρόπους:

1. Θέτουμε εξαρχής έναν πεπερασμένο χρόνο λειτουργίας στα 60 δευτερόλεπτα, μέσω του πεδίου SERVER_LIFETIME στην κλάση του server. Σε μια πλήρη εφαρμογή, όπου θα θέλαμε την συνεχή λειτουργία του προγράμματος, αυτός ο περιορισμός θα μπορούσε να αρθεί, εδώ όμως επιτρέπει τον τερματισμό του χωρίς να χρειαστεί χειροκίνητη διακοπή του από την κονσόλα. Με την έναρξη του προγράμματος, και μέσα στη μέθοδο `start()`, δημιουργούμε ένα νήμα το οποίο βάζουμε σε κατάσταση παύσης (χρησιμοποιώντας το `sleep()`) για χρόνο ίσο με SERVER_LIFETIME, και το οποίο μόλις ξυπνήσει, ενεργοποιεί τη μέθοδο `stop()`.

```
new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            Thread.sleep(SERVER_LIFETIME);
            stop();
        } catch (InterruptedException ignored) {
        }
```

```
        }
    }, "Shutdown-Timer").start();
```

Είναι σημαντικό ότι θεωρούμε τη διακοπή του ύπνου του νήματος (όταν κάποιος καλέσει τη μέθοδό του `interrupt()` χρησιμοποιώντας ένα `InterruptedException`) ως θεμιτή, δεν δίνουμε κάποιο error δηλαδή. Τέλος, δίνουμε στο νήμα και όνομα (`Shutdown-Timer`) ώστε να μπορούμε να κάνουμε σωστό trace σε περίπτωση σφάλματος.

2. Με την πληκτρολόγηση στην κονσόλα του `Ctrl+C`. Για να το πετύχω αυτό, καλώ καταρχήν την κλάση `Runtime` (χρησιμοποιώντας τη μέθοδό της `getRuntime`). Η `Runtime` είναι μια εγγενής κλάση της Java, ένα στιγμιότυπο της οποίας δημιουργείται αυτόματα σε κάθε εφαρμογή Java, και η οποία επιτρέπει την αλληλεπίδραση με το περιβάλλον της.

Επί του στιγμιότυπου καλούμε την μέθοδο `addShutdownHook` της Runtime, η οποία ενεργοποιείται με τον συνδυασμό `Ctrl+C`, εξ ορισμού της. Αυτή δημιουργεί ένα νήμα, το οποίο στην συγκεκριμένη περίπτωση καλεί τη μέθοδο `stop()`.

```
Runtime.getRuntime().addShutdownHook(new Thread(new Runnable() {
    @Override
    public void run() {
        server.stop();
    }
}));
```

1.6 Πώς εκτελούμε το πρόγραμμα

Τρέχω τις παρακάτω εντολές σε ένα terminal την καθεμία, για την εκκίνηση των server:

```
java Subtask_4_ProConServer.Server 8881
java Subtask_4_ProConServer.Server 8882
java Subtask_4_ProConServer.Server 8883
```

Σε ένα τέταρτο terminal τρέχω:

```
java Subtask_4_ProConServer.Producer
```

Και σε πέμπτο:

```
java Subtask_4_ProConServer.Consumer
```