

Technologies for Big Data Analytics

Εργασία 1 - Threads and Processes - Υποερώτημα 3

Antonis Prodromou

2025-11-21

Πίνακας περιεχομένων

1 Περιγραφή λύσης - Υποερώτημα 3	1
1.1 Σκοπός	1
1.2 Καθορισμός κλάσεων	2
1.3 Αρχιτεκτονική συστήματος	2
1.4 Λειτουργία Server	3
1.5 Λειτουργία client	4
1.6 Κωδικοποίηση	4
1.7 Μετατροπή λύσης για πολλαπλούς clients	5

1 Περιγραφή λύσης - Υποερώτημα 3

1.1 Σκοπός

Στην υποεργασία αυτή υλοποιείται μία εφαρμογή Java που περιλαμβάνει επικοινωνία διεργασιών με Transmission Control Protocol (TCP) sockets μεταξύ ενός πελάτη (client) και ενός διακομιστή (server). Το σύστημα επιτρέπει στον πελάτη την διαχείριση ενός πίνακα κατακερματισμού (Hashtable) με αποθηκευμένες πληροφορίες.

1.2 Καθορισμός κλάσεων

Για τον καθορισμό των κλάσεων, ακολουθούμε την αρχή της Μοναδικής Αρμοδιότητας, επομένως δημιουργούμε δύο κλάσεις, την *Server* και την *Client* οι οποίες έχουν τις εξής αρμοδιότητες:

1. Server - Ακούει για εισερχόμενες συνδέσεις σε συγκεκριμένη θύρα, την οποία περνάω ως παράμετρο. - Εκτελεί εντολές που λαμβάνει από τον *client*. - Ανάλογα με την εντολή αποθηκεύει, διαγράφει ή εμφανίζει καταχωρήσεις που αποθηκεύονται σε έναν πίνακα κατακερματισμού (*hashtable*). - Στέλνει πίσω στον *client* απάντηση, που δηλώνει την κατάσταση σε σχέση με την εντολή (π.χ. επιτυχής ή όχι).

2. Client - Συνδέεται στον *server* μέσω *sockets*. - Διαβάζει εντολές από τον χρήστη, μέσω της κονσόλας (*terminal*). - Στέλνει εντολές στον *server* και εμφανίζει τις απαντήσεις που λαμβάνει.

1.3 Αρχιτεκτονική συστήματος

Η αλληλεπίδραση μεταξύ των δύο προγραμμάτων έχει ως εξής:

1. Ο *server* ξεκινάει πρώτος, και περιμένει τον *client* να συνδεθεί.
2. Ο *client* ανοίγει *socket* σύνδεσης προς τον *server*.
3. Ο χρήστης πληκτρολογεί εντολές στην κονσόλα.
4. Ο *client* στέλνει κάθε εντολή στον *server* ως απλό κείμενο μέσω *PrintWriter*.
5. Ο *server* διαβάζει την εντολή με *BufferedReader*.
6. Ο *server* εκτελεί την εκάστοτε λειτουργία (*insert*, *delete*, *search*, *exit*).
7. Ο *server* στέλνει απάντηση.
8. Ο *client* εμφανίζει το αποτέλεσμα.
9. Η διαδικασία επαναλαμβάνεται μέχρι την εντολή *0 0*.

Η σύνδεση παραμένει ανοιχτή καθ' όλη τη διάρκεια που τρέχουν τα δύο προγράμματα.

1.4 Λειτουργία Server

Ξεκινάμε αρχικοποιώντας (initialize) τον πίνακα κατακερματισμού. Δημιουργούμε έναν `serverSocket` και εφαρμόζουμε ένα blocking call με τη μέθοδο `accept()`, που α) επιστρέφει ένα socket και β) σηματοδοτεί πως η εκτέλεση του προγράμματος μπαίνει σε παύση έως ότου ένας client προσπαθήσει να συνδεθεί στην θύρα του server.

Όταν επικοινωνήσει επιτυχώς ο client μέσω ενός socket, ολοκληρώνεται και το three-way handshake, το οποίο ονομάζεται έτσι γιατί πρέπει να ανταλλάγουν πληροφορίες με την ανταλλαγή τριών πακέτων (segments), τα οποία είναι τα εξής:

- Βήμα 1 (*Client to Server*): Ο client εκκινεί τη σύνδεση στέλνοντας ένα πακέτο TCP στον server, έχοντας στο πεδίο σημαίες το SYN (από το synchronize) ίσο με την τιμή 1.
- Βήμα 2 (*Server to Client*): Ο server λαμβάνει το μήνυμα με σημαία SYN, και για να αποδεχτεί τη σύνδεση αποκρίνεται με ένα πακέτο με τις σημαίες SYN/ACK (από το acknowledge) ίσες με 1 (Server to Client).
- Βήμα 3 (*Client to Server*): Ο client δέχεται το SYN/ACK και στέλνει το τελευταίο πακέτο, έχοντας στο πεδίο σημαίες μόνο το ACK ίσο με 1.

Στη συνέχεια δημιουργούμε 2 κανάλια:

- Ένα εισερχόμενο κανάλι `getInputStream()` το οποίο το εσωκλείουμε σε α) ένα `InputStreamReader` το οποίο θα μετατρέψει τα byte σε χαρακτήρες, το αντίστροφο του `getOutputStream` με το οποίο έφυγαν από τον client δηλαδή και β) ένα `BufferedReader`, το οποίο μου επιτρέπει να διαβάζω το κείμενο του χρήστη σε μπλοκ μέσω ενός μόνο call, αντί να διαβάζω έναν-έναν τους χαρακτήρες που πληκτρολογεί, και άρα να εναλλάσσομαι διαρκώς μεταξύ της μηχανής της Java (Java Virtual Machine) και τον πυρήνα (kernel) του κεντρικού συστήματος, ξοδεύοντας πόρους και χρόνο.
- Ένα εξερχόμενο κανάλι `getOutputStream()` που, καθώς εσωκλείεται σε `PrintWriter`, κρατάει μια επικοινωνία ανοιχτή που μεταφέρει αυτόματα δεδομένα στον client, μετατρέποντας τους χαρακτήρες σε bytes. Η χρησιμότητα του `PrintWriter` ως περιτύλιγμα έγκειται στο ότι προσφέρει μεθόδους (όπως το `println()`) για να εισάγεται το κείμενο με συμβατικούς τρόπους.

1.5 Λειτουργία client

Επιχειρεί μια σύνδεση στη διεύθυνση και την θύρα του server αρχικοποιώντας ένα αντικείμενο `Socket` (`new Socket()`). Έπειτα δημιουργούμε το κανάλι επικοινωνίας για την αποστολή των δεδομένων μέσω του `getOutputStream()`, που επιστρέφει ένα κανάλι για την συγκεκριμένη θύρα. Επειδή το κανάλι περιβάλλεται από ένα `PrintWriter`, είναι σαν να έχω μια ανοιχτή επικοινωνία μέσω της οποίας μπορώ να στείλω αυτόματα τα δεδομένα στον δέκτη.

Προκειμένου να λάβουμε τις εντολές από τον χρήστη, δημιουργούμε ένα κανάλι επικοινωνίας `InputStreamReader` που διαβάζει από την κονσόλα χάρη στο `System.in`, που του περνάμε ως παράμετρο. Αυτά εσωκλείονται σε ένα `BufferedReader` που - όμοια με όσα περιγράφηκαν για τον server - το οποίο μου επιτρέπει να διαβάσω το κείμενο του χρήστη σε μπλοκ μέσω ενός μόνο call. Το κείμενο που πληκτρολογεί ο χρήστης α) διαβάζεται γραμμή - γραμμή (`readLine()`) και β) μετατρέπεται σε array μέσω `split()` και στέλνεται στον server.

Με παρόμοιο τρόπο δημιουργούμε και ένα δεύτερο κανάλι για να λαμβάνει ο client απαντήσεις από τον server (δηλαδή χρησιμοποιώντας `getInputStream()`) και `BufferedReader`.

1.6 Κωδικοποίηση

Ο client στέλνει στον server μια τριάδα ακέραιων αριθμών της μορφής `A B C`, όπου:

- `A` είναι ο κωδικός λειτουργίας και παίρνει τις εξής τιμές:
 - `0`: τέλος επικοινωνίας και τερματισμός
 - `1`: εισαγωγή
 - `2`: διαγραφή
 - `3`: αναζήτηση
- `B` είναι το κλειδί με το οποίο αποθηκεύεται η πληροφορία στον πίνακα κατακερματισμού.
- `C` η τιμή που αποθηκεύεται.

Η επικοινωνία ολοκληρώνεται όταν ο client στείλει `0 0`.

1.7 Μετατροπή λύσης για πολλαπλούς clients

Σε περίπτωση πολλαπλών clients, θα έπρεπε καταρχήν να κάνουμε αλλαγές στον server ώστε να μπορεί να δεχθεί τα αιτήματα επικοινωνίας τους. Τώρα έχουμε ένα `accept()` το οποίο για να δεχθεί παραπάνω clients θα μπορούσε να είναι μέσα σε ένα `while statement`, οπότε θα δημιουργούνταν 3 διαφορετικά sockets (`clientSocket = serverSocket.accept()`)

Για να γίνεται παράλληλη και ανεξάρτητη επικοινωνία με τους client, μπορούμε να χρησιμοποιήσουμε `threads`, καθένα από τα οποία θα διαχειρίζονταν τα `incoming` και `outgoing` κανάλια με τις εντολές εισαγωγής στο `hashtable` κλπ. Για τη διαχείριση του πίνακα κατακερματισμού συγκεκριμένα, θα μπορούσαμε να χρησιμοποιήσουμε `synchronized (this)` για το μπλοκ του.