ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

(2020-2021)

3η Εργαστηριακή Άσκηση

"Κρυπτογραφική συσκευή VirtIO για QEMU-KVM"

Ημερομηνία Επίδειξης:

- Ζητούμενα Z1, Z2 : 7/1/2021 | 13:45 – 14:45
- Ζητούμενα Z3 : 22/1/2021 | 14:00 – 15:00

Ομάδα:

- 40

Ονοματεπώνυμο, Α.Μ., στοιχεία επικοινωνίας:

- Τόφαλος Φίλιππος
  - 03117087
  - el17087@mail.ntua.gr
- Τσούφης Χρήστος
  - 03117176
  - el17176@mail.ntua.gr

Ζήτημα 1 – Εργαλείο chat πάνω από TCP/IP sockets

Θεωρία & Περιγραφή

Σκοπός του ερωτήματος αυτού, είναι η επικοινωνία δύο διεργασιών (client και server) με χρήση TCP/IP sockets. Η μόνη διαφοροποίηση στην συμπεριφορά του client και του server είναι στο πώς διαχειρίζονται τα sockets αφού τα δημιουργήσουν. Συγκεκριμένα, από τη μεριά του server, βλέπουμε ότι πραγματοποιείται το bind και μετά βρίσκεται σε κατάσταση listen, περιμένοντας συνεχώς νέες συνδέσεις από τον client τις οποίες κάνει accept. Για κάθε accept που κάνει προκύπτει ένα νέο socket descriptor, μέσω του οποίου επικοινωνεί με τον client. Ο client το μόνο που κάνει είναι αίτημα για connect στο socket του server.

Το chat υλοποιείται μέσω της συνάρτησης chat, που βρίσκεται στο αρχείο socket-common.h και καλείται και από την μεριά του client και από την μεριά του server. Χρησιμοποιούμε την συνάρτηση select για να αποφασίσουμε κάθε φορά από ποιον file descriptor θα διαβάσουμε, δηλαδή αν θα διαβάσουμε από το stdin ή από τον socket descriptor. Σε περίπτωση που διαβάζουμε από stdin, ό,τι διαβάζουμε το προωθούμε στο socket descriptor, ώστε να περάσουμε το μήνυμα στον συνομιλητή μας. Σε περίπτωση που διαβάζουμε από το socket descriptor, τυπώνουμε ό,τι διαβάζουμε από τον συνομιλητή μας στο stdout, ώστε να μπορεί να το δει ο χρήστης

Επικοινωνία με BSD Sockets: Είναι ένα interface, με το οποίο γίνεται διεργασιακή επικοινωνία μέσω δικτύου διεργασιών που εκτελούνται σε απομακρυσμένα ή και στο ίδιο μηχάνημα.

TCP Protocol:

- Αξιόπιστη (ACK), αμφίδρομη μεταφορά δεδομένων.
- Δεν διατηρεί πληροφορία για όρια του μηνύματος.
- Connection-oriented επικοινωνία.
- PF_INET (domain) + SOCK_STREAM (type) + 0 (Protocol) = TCP
- Διαχωρισμός συνδέσεων με: {Address, Source Port}, {Address, Destination Port}.
- Στα Linux υλοποιείται από τον πυρήνα.

Ανάλυση κώδικα

Client:

- `connect_to_server()`:
  Δημιουργεί το socket , που θα χρησιμοποιηθεί ως κανάλι επικοινωνίας και επιστρέφει τον file descriptor. Κλήση συστήματος που συνδέει τον socket descriptor που επέστρεψε η socket() με την (απομακρυσμένη) διεύθυνση που προσδιορίζεται από το όρισμα addr -σε εμάς είναι στο ίδιο μηχάνημα. Καλούμε την connect, διότι χρησιμοποιούμε TCP πρωτόκολλο.
- `read()/write()`:
  Από/Στον file descriptor.
- `close()`:
  Κλείνει το ανοιχτό αρχείο στο οποίο δείχνει ο file descriptor, ο οποίος τώρα μπορεί να επαναχρησιμοποιηθεί.

Server:

- `accept_new_client()`:
  Αυτή η κλήση συστήματος χρησιμοποιείται σε connection-oriented sockets, όπου πρέπει να προηγηθεί σύνδεση για τη μεταφορά δεδομένων. Ουσιαστικά αφαιρεί το πρώτο αίτημα εισερχόμενης σύνδεσης από την ουρά, δημιουργεί ένα νέο file descriptor (χωρίς να επηρεάζει τον παλιό), ο οποίος και χρησιμοποιείται αργότερα για την διεργασιακή επικοινωνία με τον client. Σε περίπτωση που δεν υπάρχουν αιτήματα σύνδεσης στην ουρά αναμονής του socket, η accept μπλοκάρει μέχρι να δεχθεί αίτημα για εισερχόμενη σύνδεση.
- `handle_client()`:
  Δημιουργεί το socket , που θα χρησιμοποιηθεί ως κανάλι επικοινωνίας και επιστρέφει τον file descriptor. Ο file descriptor που επιστρέφει η socket() υπάρχει σε ένα χώρο διευθύνσεων, χωρίς όμως να του έχει ανατεθεί συγκεκριμένη διεύθυνση. Η bind() δένει τον file descriptor με μία τη διεύθυνση που δίνεται στο όρισμα *addr.
- `int listen()`:
  Χαρακτηρίζει το socket ως παθητικό, δηλαδή το καθιστά κατάλληλο να δεχτεί συνδέσεις. Σε περίπτωση που η ουρά αυτή είναι γεμάτη, ο client είτε θα δεχτεί error, είτε το αίτημα θα αγνοηθεί ώστε να πραγματοποιηθεί νέα προσπάθεια μελλοντικά (αναλόγως το πρωτόκολλο).
- `read()/write()`:
  Από/Στον νέο file descriptor.
- `close()`:
  Κλείνει το νέο socket descriptor που δημιούργησε η accept().

`select()`:

Η κλήση συστήματος select() χρησιμοποιείται στις περιπτώσεις που έχουμε > 1 file descriptors από/στους οποίους διαβάζουμε/γράφουμε. Στην περίπτωση των server socket, client socket έχουμε 2 file descriptors, ένας είναι ο socket descriptor, ενώ ο δεύτερος είναι ο standard input. Από το standard input διαβάζουμε αυτό που γράφει ο χρήστης από το πληκτρολόγιο και ύστερα το γράφουμε στον socket descriptor, ενώ από τον socket descriptor διαβάζουμε δεδομένα που ήρθαν μέσω δικτύου και τα γράφουμε στο standard output για να τα δει ο χρήστης. Προφανώς δεν είναι πάντα και οι δύο file descriptors έτοιμοι (δηλαδή δεν έχουν πάντα δεδομένα). Σε ένα chat όμως είναι σημαντικό να υπάρχει αμφίδρομη επικοινωνία χωρίς περιορισμούς. Για αυτό κάναμε χρήση της select() , η οποία διαχειρίζεται κάποια set από file descriptors. Έτσι πρακτικά καταφέραμε να μην πρέπει ο ένας χρήστης να περιμένει την απάντηση του άλλου για να στείλει κι άλλο μήνυμα.

Ζήτημα 2 – Κρυπτογραφημένο chat πάνω από TCP/IP

Θεωρία & Περιγραφή

Το ζητούμενο αυτής της άσκησης είναι πρακτικά η κρυπτογράφηση/ αποκρυπτογράφηση του μηνύματος που λαμβάνει/στέλνει ο χρήστης. Η διαδικασία σύνδεσης και διεργασιακής επικοινωνίας γίνεται ακριβώς με τον ίδιο τρόπο που περιγράφηκε στο 1ο ζητούμενο. Η μόνο διαφορά είναι ότι πριν γραφτεί στον socket descriptor το μήνυμα και σταλεί μέσω TCP, γίνεται κρυπτογράφηση του μηνύματος. Αντίστοιχα μόλις ένα εισερχόμενο μήνυμα διαβαστεί από τον socket descriptor, γίνεται πρώτα αποκρυπτογράφηση και κατόπιν τυπώνεται στο standard output στην κανονική του (αναγνώσιμη) μορφή. Η (απο)κρυπτογράφηση επιτυγχάνεται με κλήσεις συστήματος ioctl() στο ειδικό αρχείο /dev/crypto της κρυπτογραφικής συσκευής χαρακτήρων. Συγκεκριμένα για τις ανάγκες της άσκησης θα ασχοληθούμε με τις εξής 3 κλήσεις ioctl(): `CIOCGSESSION` (αρχίζει ένα session με τη συσκευή και επιστρέφει ένα session id, το οποίο αποθηκεύεται και χρησιμοποιείται στις κλήσεις ioctl() (απο)κρυπτογράφησης), `CIOCCRYPT` (η κλήση αυτή ζητά από τη συσκευή κρυπτογράφησης να (απο)κρυπτογραφήσει δεδομένα) & `CIOCFSESSION` (η κλήση αυτή τερματίζει ένα session με τη συσκευή όταν κάποιος από τους server-client αποσυνδέεται).

Ανάλυση Κώδικα

Client:

- `connect_to_server()`:
  Με την υπόθεση ότι όταν συνδέεται ο client θα παίρνει το δικό του PID και αφότου έχει κάνει τη σύνδεση, θα στέλνει το PID του σαν string στον server και εκεί αυτός θα το αποθηκεύει ώστε να μην χρειάζεται κάθε φορά εκ νέου αυτή η διαδικασία. Επίσης γίνεται και μια αξιολόγηση με κατάλληλα μηνύματα λάθους. Ακόμη, έχουμε ορίσει ένα time out σε περίπτωση που κάτι δυσλειτουργεί να μην κολλάει η εφαρμογή ή ο server αντιστοίχως από την άλλη πλευρά. Τέλος, επιχειρούμε το connect και μετά να στείλουμε το PID και να λάβουμε μια απάντηση με acknowledgement.
- `initialize_window` & `place_window` & `reset_input`:
  Έχει να κάνει με τα end curses.
- `main`:
  Το cfd δίνει τον fd του /dev/crypto οπότε αρχικά ανοίγουμε αυτό το αρχείο και ελέγχουμε ότι είναι όντως εγκατεστημένο το crypto dev. Μετά, επιχειρούμε να συνδεθούμε στον server σύμφωνα με τη συνάρτηση παραπάνω και με την select ελέγχουμε αν υπάρχει input από τον ίδιο τον χρήστη στον standard input και παίρνουμε περιπτώσεις. Αν δώσει Enter, τότε αν το έχουμε ορίσει μέσω της make κάνει κρυπτογράφηση του μηνύματος και το στέλνει αναλόγως αλλιώς, αν φτάσουμε σε ένα όριο χαρακτήρων που έχουμε ορίσει για ευκολία στην end curses τότε σταματά να δέχεται input γενικά. Αν λάβουμε ειδοποίηση ότι είναι έτοιμος ο fd του server τότε κάνουμε decrypt το μήνυμα που λάβαμε από την read, και το τοποθετούμε στο window και είμαστε εντάξει. Βάζουμε και μια καθυστέρηση γιατί δεν βάλαμε time out στην select.

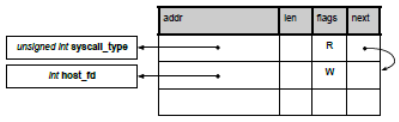
Server:

- `reformat_message()`:
  Αυτή η συνάρτηση υπάρχει για όταν λαμβάνει ένα μήνυμα ο server και θέλει, για να μπορεί ο client να αναγνωρίσει ποιας διεργασίας είναι, να βάλει τα σωστά χρώματα και να ξεχωρίσει την αρχή ενός μηνύματος. Επίσης δείχνει την αρχή μηνύματος και αναλόγως κάνει και την κρυπτογράφηση του μέρους που κάνει append. Απαραίτητη προϋπόθεση είναι να συμφωνούν ο client και ο server.
- `void accept_new_client()`:
  Κάνει accept την νέα σύνδεση και λαμβάνουμε τη διεύθυνση και το μήκος διεύθυνσης για να την εκτυπώσουμε παρακάτω. Λαμβάνουμε το PID και το αποθηκεύσουμε στον client pid. Στέλνουμε το ACK και στη συνέχεια δείχνουμε ότι έχουμε λάβει ένα νέο connection, με τα στοιχεία του connection και ενημερώνουμε τα αντίστοιχα στοιχεία. Βρίσκουμε το πρώτο διαθέσιμο index για τον client, και αναθέτουμε τα ανάλογα στοιχεία.
- `void handle_client()`:
  Έχουμε λάβει μήνυμα, το διαβάζουμε και αν είναι μηδενικού μήκους, σημαίνει ότι έγινε αποσύνδεση οπότε λαμβάνουμε από την get prename τα στοιχεία για τον fd που πρόκειται να αποδεσμεύσουμε και το εκτυπώνουμε αναλόγως στο τερματικό και κάνουμε close τον fd και μηδενίζουμε τις ανάλογες τιμές στους πίνακες. Διαφορετικά, διαβάζουμε το μήνυμα, το κάνουμε reformat όπως παραπάνω και ενημερώνουμε όλους τους υπόλοιπους ενεργούς fd ώστε να λάβουν οι αντίστοιχοι clients το μήνυμα μαζί με τον ίδιο.
- `main`:
  Μέσα στο while, κάνουμε bind & listen στη συγκεκριμένη διεύθυνση ώστε να ακούει εισερχόμενες συνδέσεις. Βάζουμε πάντα το master socket να ακούει για νέες συνδέσεις και αρχικοποιούμε, επειδή για την select θέλουμε να βρούμε το max fd, σαν το master socket και κρατάμε ποιο είναι το μέγιστο ως εκείνη τη στιγμή ώστε, καλώντας τη select να περιμένουμε μόνο για τις εγγραφές και να ελέγχουμε αν έχει τεθεί το master socket οπότε θα έχουμε νέα σύνδεση ενώ αν έχουμε από κάποιον client, κάνουμε την ανάλογη διαχείριση.
- `int listen()`:
  Χαρακτηρίζει το socket ως παθητικό, δηλαδή το καθιστά κατάλληλο να δεχτεί συνδέσεις. Σε περίπτωση που η ουρά αυτή είναι γεμάτη, ο client είτε θα δεχτεί error, είτε το αίτημα θα αγνοηθεί ώστε να πραγματοποιηθεί νέα προσπάθεια μελλοντικά (αναλόγως το πρωτόκολλο).

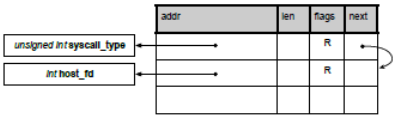
Ζήτημα 3 – Υλοποίηση συσκευής cryptodev με VirtIO

Θεωρία & Περιγραφή

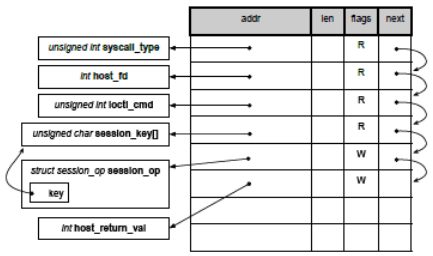
Εφόσον η τεχνική που χρησιμοποιούμε μέχρι τώρα είναι πλήρης εικονικοποίηση η συσκευή κρυπτογράφησης δεν χρησιμοποιείται πραγματικά αλλά προσομοιώνεται μέσω λογισμικού και ενσωματώνεται σε μία εικονική συσκευή του QEMU. Ο τρόπος αυτός έχει ως αποτέλεσμα πολύ αργή απόκριση της συσκευής καθώς απαιτεί άριστη γνώση των λειτουργιών κρυπτογράφησης. Έτσι στο 3ο ζητούμενο θα δημιουργήσουμε μια νέα εικονική συσκευή κρυπτογράφησης για εικονικές μηχανές που εκτελούνται από το QEMU, έτσι ώστε να μπορούμε να εκμεταλλευτούμε την πραγματική συσκευή που έχουμε διαθέσιμη στο host μηχάνημα. Απαιτείται δηλαδή επικοινωνία της πραγματικής συσκευής (host) με την εικονική μηχανή (guest). Για τον σκοπό αυτό θα χρησιμοποιήσουμε το πρότυπο VirtIO, το οποίο επιτρέπει την αποδοτική επικοινωνία του ΛΣ του guest με κώδικα που εκτελείται στο host, μέσω κοινών πόρων (VirtQueues) για ανταλλαγή δεδομένων. Έτσι μια κλήση συστήματος ioctl() που θα έκανε μια διεργασία στη συσκευή χαρακτήρων cryptodev, την κάνει τώρα στην εικονική συσκευή virtio-cryptodev, της οποίας οι drivers έχουν ρυθμιστεί έτσι ώστε η κλήση συστήματος ioctl() να μεταφέρεται μέσω VirtQueues από το guest στον hypervisor (QEMU). Στη συνέχεια ο hypervisor, που εκτελείται σε χώρο χρήστη host, καλεί την αντίστοιχη ioctl() για την πραγματική συσκευή, και επιστρέφει το αποτέλεσμα στον guest πάλι μέσω VirtQueues. Για τη μεταφορά των δεδομένων θα χρησιμοποιήσουμε scatter-gather λίστες,τις οποίες θα προσθέτουμε στην VirtQueue της συσκευής, για DMA μεταφορές δεδομένων, μέσω των συναρτήσεων που προσφέρει το πρωτόκολλο virtio_ring για το πρότυπο VirtIO.



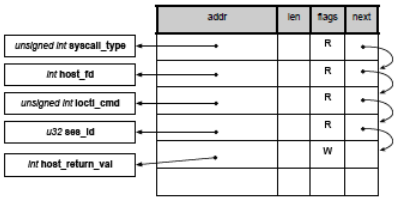
(α′) open()



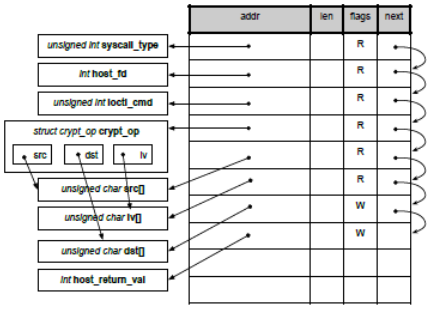
(β′) close()



(γ′) ioctl(... CIOCGSESSION ...)



(δ′) ioctl(... CIOCFSESSION ...)



(ε′) ioctl(... CIOCCRYPT ...)

Ανάλυση Κώδικα

Frontend (Χώρος πυρήνα εικονικής μηχανής):

- `crypto_dev_nodes.sh`:
  Αυτό το script δημιουργεί τα ειδικά αρχεία της εικονικής συσκευής, που θα χρησιμοποιηθούν από τον guest ως συσκευές κρυπτογράφησης/ αποκρυπτογράφησης.
- `static int crypto_chrdev_open(struct inode *inode, struct file *filp)`:
  Το μόνο που κάνουμε είναι το add στο ανάλογο sgs virtual queue και μετά κάνουμε kick. Αναμένουμε την απάντηση του host και σύμφωνα με την οδηγία, αν επιστραφεί -1 στον host fd θέτουμε αναλόγως το return και μετά πηγαίνει στο fail για να επιστρέψει και την ανάλογη τιμή.
- `static int crypto_chrdev_release(struct inode *inode, struct file *filp)`:
  Εδώ χρησιμοποιείται σημαφόρος εφόσον υπάρχει περίπτωση να υπάρχει κοινή open μεταξύ διεργασιών. Μετά περιμένουμε, απελευθερώνουμε το σημαφόρο, κάνουμε free το ανάλογο struct.
- `static long crypto_chrdev_ioctl(struct file *filp, unsigned int cmd,unsigned long arg)`:
  Πρώτα, αρχικοποιούμε τα κοινά struct μεταξύ των διάφορων κλήσεων (host fd, ioctl cmd, host fd val) κάνοντας δυναμική δέσμευση των ανάλογων μεταβλητών στο χώρο μνήμης κλπ.
  Μετά πάμε ανά περιπτώσεις σύμφωνα με το σχήμα που είχε δοθεί:
  - Το session op το αντιγράφουμε από τον χρήστη και κάνοντας την μετατροπή σε τύπο user πρέπει να αντιγράψουμε την copy from user και το session key και τέλος να το περάσουμε στο scatter list.
  - Στο fsession το μόνο που χρειάζεται να αντιγράψουμε είναι απευθείας το arg στο ses id που έχει αρχικοποιηθεί σαν u32 και το περνάμε και αυτό στο scatter list. Τέλος, απελευθερώνουμε την ανάλογη δομή που χρησιμοποιήθηκε
  - Στην ciocrypt κάνουμε ό,τι και στο fsession απλά αντιγράφοντας το crypt op και το src και αρχικοποιούμε το dst και τέλος, περνάμε το host return value. Διεκδικούμε το σημαφόρο. Εφόσον το κάνουμε kick, περιμένουμε την απάντηση του host. Απελευθερώνουμε το σημαφόρο εφόσον ο κοινός πόρος που είναι το virtual queue σε περιπτώσεις πατέρα-παιδιού έχει αποδεσμευτεί πλέον οπότε μπορούμε να πάρουμε τα αποτελέσματα του host. Τέλος, περνάμε destination array και απελευθερώνουμε τις ανάλογες δομές
  - Στο gsession κάνουμε copy to user όλο το session op.

Backend (χώρος χρήστη host μηχανήματος):

- `static void vq_handle_output(VirtIODevice *vdev, VirtQueue *vq)`
- `VIRTIO_CRYPTODEV_SYSCALL_TYPE_IOCTL`:
  - `CIOCGSESSION`:
    Στην περίπτωση που θέλουμε να ξεκινήσουμε ένα session με την συσκευή cryptodev, διαβάζουμε την τιμή του κλειδιού κρυπτογράφησης, καθώς επίσης ανακτούμε και τις διευθύνσεις του αντικειμένου τύπου session_op () και της μεταβλητής επιστροφής ret. Εκτελούμε κλήση συστήματος η οποία σε επιτυχία επιστρέφει μηδενική τιμή και αρχικοποιεί κατάλληλα τα πεδία του αντικειμένου sess
  - `CIOCFSESSION`:
    Στην περίπτωση που θέλουμε να τερματίσουμε ένα session με την συσκευή cryptodev, διαβάζουμε την τιμή του identifier του session και ανακτούμε την διεύθυνση της μεταβλητής επιστροφής ret. Εκτελούμε κλήση συστήματος η οποία σε επιτυχία επιστρέφει μηδενική τιμή και τερματίζει το session
  - `CIOCCRYPT`:
    Στην περίπτωση που θέλουμε να κάνουμε κρυπτογράφηση/ αποκρυπτογράφηση ενός μηνύματος, αναθέτουμε αρχικά τις διευθύνσεις που περιέχουν οι in_sg, out_sg σε μεταβλητές (src,iv,cryp, dst, ret) . Οι πρώτες τρεις έχουν μεταφερθεί από τον guest στον host μόνο για ανάγνωση, ενώ η dst που προορίζεται να αποθηκεύσει το επεξεργασμένο κείμενο είναι ασφαλώς για εγγραφή. Έτσι, δημιουργούμε ένα νέο αντικείμενο τύπου crypt_op, στο οποίο αντιγράφουμε αρχικά μέσω memcpy() τα πεδία του αρχικού αντικειμένου cryp, κάνοντας κατόπιν overwrite σε αυτά που περάσαμε ξεχωριστά (src, iv, dst) με νέα ανάθεση. Έτσι όλα τα πεδία είναι κατάλληλα αρχικοποιημένα και με τα κατάλληλα δικαιώματα. Επίσης ανακτούμε την διεύθυνση της μεταβλητής επιστροφής ret και μέσω κλήσης συστήματος εκτελείται η ενέργεια που περιγράφει το πεδίο op (κρυπτογράφηση/αποκρυπτογράφηση) στα δεδομένα εισόδου και το αποτέλεσμα αποθηκεύεται στο πεδίο dst και συνεπώς στη μεταβλητή dst που είχαμε εξ αρχής προσθέσει στην ουρά.

Τέλος, μόλις ολοκληρωθούν οι παραπάνω ενέργειες, προσθέτουμε τα επεξεργασμένα στοιχεία στην ουρά μέσω της συνάρτησης virtqueue_push() και ειδοποιείται ο guest για την ύπαρξη νέων δεδομένων μέσω της συνάρτησης virtio_notify() με interrupt. Η συνάρτηση αυτή τερματίζει και η εκτέλεση συνεχίζεται από το frontend όπως περιγράφηκε στο παραπάνω.