Εργαστήριο Λειτουργικών Συστημάτων

Εργαστηριακή αναφορά 3ης άσκησης

Νικόλαος Παγώνας
Νικήτας Τσίννας

Ζητούμενο 1

Εισαγωγή

Το πρώτο ζητούμενο αποσκοπεί στην δημιουργία μίας εφαρμογής συνομιλίας μεταξύ δύο υπολογιστών (peer-to-peer chat). Για τον σκοπό αυτό χρησιμοποιήθηκε το BSD Sockets API το οποίο εισαγάγαμε μέσω της εντολής `#include <sys/socket.h>`.

Δημιουργήθηκαν 2 αρχεία πηγαίου κώδικα. Το `socket-client.c` για την λειτουργία της πλευράς του πελάτη και το `socket-server.c` για την πλευρά του server. Όλα τα μηνύματα που ανταλλάσσονται μεταξύ των 2 ομοτίμων βασίζονται στο διαδικτυακό πρωτόκολλο TCP/IP και επομένως χρησιμοποιούμε και στις 2 περιπτώσεις TCP/IP sockets.

socket-server.c

Για την υλοποίηση του `socket-server.c` χρησιμοποιήθηκαν οι διαδοχικές κλήσεις συστήματος `socket()`, `bind()`, `listen()`, `accept()` οι οποίες χρησιμεύουν στην πραγματοποίηση συνδέσεων με πιθανούς clients. Ειδικότερα, πρώτα δημιουργείται η υποδοχή του server, η οποία μέσω της `bind()` συνδέεται με την διεύθυνση IP που ορίζουμε και έπειτα με την `listen()` ο server αρχίζει να ακούει για εισερχόμενες συνδέσεις, οι οποίες δεν μπορούν να ξεπεράσουν σε πλήθος τον αριθμό `TCP_BACKLOG`. Με την `accept()` μπορεί και δέχεται συνδέσεις στην οποία και μπλοκάρει αν δεν υπάρχουν άλλες που περιμένουν στην ουρά του socketfd. Μόλις τελειώσει μία συνεδρία (session), ο εξυπηρετητής μπαίνει ξανά στο loop περιμένοντας νέα σύνδεση.

socket-client.c

Από τη μεριά του πελάτη (client) χρησιμοποιείται και πάλι η κλήση `socket()` για τη δημιουργία υποδοχής και στη συνέχεια χρησιμοποιείται η κλήση `connect()` με όρισμα την διεύθυνση του server για να πραγματοποιηθεί αίτηση σύνδεσης. Όταν ξεκινήσει επιτυχώς η συνεδρία μπαίνει στο loop που αντιστοιχεί στην εφαρμογή συνομιλίας, η οποία έχει παρόμοια υλοποίηση με την πλευρά του server.

Υλοποίηση chat

Το κύριο θέμα σε αυτό το ζητούμενο είναι η συνεχής συνομιλία μεταξύ των δύο διεργασιών. Πιο συγκεκριμένα, η κάθε πλευρά πρέπει να μπορεί να στέλνει μηνύματα κατά βούληση χωρίς να είναι αναγκαστικό να μπλοκάρει περιμένοντας απάντηση από την άλλη πλευρά. Για τον σκοπό αυτό θα πρέπει να παρακολουθούνται ταυτόχρονα οι περιγραφητές αρχείων που αντιστοιχούν στην υποδοχή TCP/IP (socket) αλλά και στην γραμμή εισόδου (`stdin`). Για τον σκοπό αυτό χρησιμοποιείται η κλήση `poll()` η οποία παρακολουθεί τους 2 file descriptors ταυτόχρονα για νέα δεδομένα. Στην περίπτωση που υπάρχουν νέα δεδομένα σε κάποιον file descriptor, η `poll()` ξεμπλοκάρει. Χρησιμοποιείται ένας buffer 100 θέσεων που χρησιμεύει και στην ανάγνωση εισερχόμενων χαρακτήρων αλλά και στην αποστολή. Να σημειωθεί πως χρησιμοποιείται και η συνάρτηση `insist_write()` ώστε να διασφαλιστεί η εγγραφή όλων των εξερχόμενων δεδομένων στο socket. Τέλος, η συνεδρία μπορεί να τερματιστεί και από τις δύο πλευρές με τον ειδικό χαρακτήρα τερματισμού αρχείου.

Ζητούμενο 2

Το 2ο ζητούμενο είναι η κρυπτογράφηση του καναλιού επικοινωνίας των δύο διεργασιών, οι οποίες αλληλεπιδρούν όπως περιγράφηκε παραπάνω. Οι διαδικασία σύνδεσης και επικοινωνίας των δύο πλευρών υλοποιείται ακριβώς με τον ίδιο τρόπο με την μόνη διαφορά τώρα πως πριν σταλθεί το μήνυμα στον socket descriptor πρώτα κρυπτογραφείται μέσω της συνάρτησης `encrypt()` και αποκρυπτογραφείται πριν σταλθεί στο stdout με το που ληφθεί από το socket με την `decrypt()`.

Οι συναρτήσεις αυτές μας προσφέρονται από ειδικό module το οποίο περιέχει hardware accelarated αλγορίθμους κρυπτογράφησης και το οποίο εγκαθιστούμε στο σύστημά μας με τις εντολές `make && make install && insmod ./cryptodev.ko`. Στην συνέχεια, στους πηγαίους κώδικες της κάθε πλευράς προσθέτουμε το `#include <crypto/cryptodev.h>` για να μπορέσουμε να χρησιμοποιήσουμε τις δυνατότητες κρυπτογράφησης του module. Οι ειδικές συναρτήσεις για την κρυπτογράφηση και αποκρυπτογράφηση υλοποιούνται με κλήσεις συστήματος `ioctl()` στο ειδικό αρχείο συσκευής κρυπτογράφησης `dev/crypto`. Οι 3 κλήσεις `ioctl()` που χρησιμοποιήθηκαν στην άσκηση είναι:

CIOCGSESSION

Δημιουργεί ένα νέο session της συσκευής. Επιστρέφεται ένα `session id` το οποίο αποθηκεύεται στην δομή `cryp` και χρησιμοποιείται στις υπόλοιπες κλήσεις `ioctl()`. Επίσης, σημειώνουμε πως τα μέλη `key` και `iv` έχουν ίδιες τιμές σε client και server και επιλέχθηκαν αυθαίρετα.

CIOCCRYPT

Χρησιμοποιείται για κρυπτογράφηση ή αποκρυπτογράφηση.

CIOCFSESSION

Τερματίζει ένα session. Στην υλοποίησή μας, ένα session τερματίζεται όταν γίνει αποκοπή της σύνδεσης λόγω αποχώρησης οποιασδήποτε πλευράς.

Ζητούμενο 3

Εισαγωγή

Στο 3ο ζητούμενο υλοποιήσαμε μια εικονική συσκευή κρυπτογράφησης για εικονικές μηχανές που εκτελούνται μέσω του QEMU, έτσι ώστε να μπορούμε να εκμεταλλευτούμε την πραγματική συσκευή του host μηχανήματος. Απαιτείται δηλαδή επικοινωνία host-guest, την οποία επιτρέπει το πρότυπο VirtIO, μέσω κοινών πόρων (VirtQueues) για ανταλλαγή δεδομένων. Έτσι μια κλήση συστήματος `ioctl()` που θα έκανε μια διεργασία στη συσκευή χαρακτήρων cryptodev, εκτελείται πλέον 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. Στη συνέχεια ο hypervisor, καλεί την αντίστοιχη `ioctl()` για την πραγματική συσκευή, και επιστρέφει το αποτέλεσμα στον guest. Για τη μεταφορά εκμεταλλευόμαστε τις scatterlists (χρησιμοποιούνται για DMA μεταφορά δεδομένων), τις οποίες προσθέτουμε στην VirtQueue της συσκευής, μέσω των συναρτήσεων που προσφέρει το πρωτόκολλο `virtio_ring` για το πρότυπο VirtIO.

Η υλοποίηση του οδηγού ουσιαστικά χωρίζεται σε δύο τμήματα, στον κώδικα του guest (frontend) και στον κώδικα του host (backend).

Frontend

Εδώ χρειάζεται κυρίως προσοχή στην σωστή εφαρμογή του προτύπου VirtIO. Η βασική δυσκολία που αντιμετωπίστηκε ήταν η σωστή μεταφορά των structs από το userspace στον kernel. Συγκεκριμένα με την εντολή `copy_from_user()` κάναμε αντιγραφή ενός struct αλλά τα πεδία `key`, `iv` και `src` μας έδιναν έναν δείκτη στα δεδομένα και όχι τα ίδια τα δεδομένα. Συνεπώς χρειάστηκε να επαναληφθεί η `copy_from_user()` για τους δείκτες αυτούς. Αντίστοιχα χρειάστηκαν πολλαπλά `copy_to_user()`. Κυρίως τροποποιήσαμε το `crypto-chrdev.c`. Μόνο για τη δήλωση ενός lock, χρειάστηκε να τροποποιήσουμε το `crypto.h`.

Βασικές συναρτήσεις στο frontend

crypto_chrdev_open(inode, filp)

Εδώ δημιουργούνται ένα `struct file` και ένα `struct crypto_open_file`. Μέσω του minor number του ειδικού αρχείου, βρίσκουμε και αποθηκεύουμε την εικονική κρυπτογραφική συσκευή με την οποία συνδέεται. Κατόπιν κλειδώνουμε το `spinlock` της συσκευής και δημιουργούμε 2 scatterlists, μία για τον τύπο του επιθυμητού system call, δηλαδή `open()`, και μία για να επιστρέψει ο host (στον guest) τον file descriptor που γυρνάει η `open()`. Μέσω της `virtqueue_add_sgs()`

προσθέτουμε τις δύο αυτές λίστες στην virtqueue, ενώ μέσω της `virtqueue_kick()` ειδοποιούμε τον host για την προσθήκη των δεδομένων. Ύστερα ο guest κάνει busy wait:

```
while(virtqueue_get_buf() == NULL);
```

Μόλις βγει ο guest από το busy-wait ξεκλειδώνεται το `spinlock` της συσκευής, ώστε κάποια άλλη εν αναμονή διεργασία να μπορέσει να προσθέσει δεδομένα στην VirtQueue της ίδιας συσκευής. Έτσι ο file descriptor στον guest έχει τώρα τιμή που αφορά το ανοιχτό αρχείο της πραγματικής συσκευής στον host. Ο file descriptor αποθηκεύεται στο `host_fd`.

crypto_chrdev_release(inode, filp)

Εδώ δημιουργούμε 3 scatterlists, μία για να διαβάσει ο host τον τύπο του system call που θέλει να κάνει ο guest, δηλαδή `close()`, μία για να διαβάσει ο host τον file descriptor του αρχείου που θέλουμε να κλείσουμε, και μία για να γράψει ο host την τιμή επιστροφής της `close()`. Η διαδικασία είναι ανάλογη με αυτή της `crypto_chrdev_open()`.

crypto_chrdev_ioctl(filp, cmd, arg)

Εδώ το τι αντιπροσωπεύει το `arg` εξαρτάται από το `cmd`. Οι πρώτες 4 scatterlists της VirtQueue είναι κάθε φορά ίδιες και περιέχουν : α) το είδος του system call που θέλει να κάνει ο guest, δηλαδή `ioctl()` β) τον file descriptor `host_fd` που αφορά το ανοιχτό αρχείο της πραγματικής συσκευής χαρακτήρων στον host γ) το είδος της κλήσης συστήματος `ioctl()` που θέλει να κάνει ο guest (CIOCGSESSION, CIOCCRYPT ,CIOCFSESSION) και δ) μια μεταβλητή `retval`, η τιμή της οποίας θα ενημερώνει τον guest για την επιτυχία/αποτυχία της κλήσης συστήματος που πραγματοποιήθηκε στο host μηχάνημα. Στη συνέχεια ανάλογα με την τιμή του ορίσματος `cmd` προσθέτουμε τα απαραίτητα δεδομένα στην VirtQueue. Έχουμε τις εξής περιπτώσεις:

CIOCGESSION

Εδώ το `arg` περιέχει τη διεύθυνση ενός `struct session_op`. Η διεύθυνση αυτή όμως βρίσκεται σε user space και για αυτό χρησιμοποιούμε την `copy_from_user()`.

CIOCFSESSION

Εδώ το όρισμα `arg` δείχνει στο identifier του session. Και πάλι χρησιμοποιούμε την `copy_from_user()`. Στη συνέχεια προσθέτουμε την τιμή του στη VirtQueue με δικαίωμα ανάγνωσης από τον host, ώστε να γνωρίζει ποιο session επιθυμούμε να τερματίσουμε.

CIOCCRYPT

Εδώ το όρισμα `arg` περιέχει τη διεύθυνση ενός `struct crypt_op`. Εδώ υπάρχει μια ιδιαιτερότητα στην χρήση της `copy_from_user()` γιατί κάποια από τα πεδία του είναι δείκτες (σε χώρο χρήστη). Γι' αυτό ξαναχρησιμοποιούμε όσες φορές χρειαστεί την `copy_from_user()` για να έχουμε πρόσβαση στην τιμή τους. Δημιουργούμε λοιπόν 4 scatterlists, μια για ανάγνωση

των υπόλοιπων πεδίων του αντικειμένου cryp, μία για ανάγνωση των αντίστοιχων src, iv που δημιουργήσαμε και μία για εγγραφή του αποτελέσματος στη αντίστοιχη μεταβλητή dst και επιστροφή του στον guest.
Σε κάθε περίπτωση, προσθέτουμε μέσω της virtqueue_add_sgs() τις λίστες στην VirtQueue της και μέσω της virtqueue_kick() ενημερώνουμε τον host για προσθήκη νέου στοιχείου στην ουρά. Ύστερα κάνουμε busy wait:

```
while (virtqueue_get_buf == NULL);
```

Μόλις βγούμε από το busy wait, ξεκλειδώνουμε το spinlock της συσκευής απελευθερώνοντας την για χρήση από άλλες διεργασίες και ελέγχουμε την τιμή επιστροφής. Ύστερα, κάνουμε χρήση της συνάρτησης copy_to_user() ώστε η τιμή των επεξεργασμένων από τον host μεταβλητών να αντιγραφεί στα αντίστοιχα πεδία του arg.

Backend

Εδώ υλοποιήθηκε το εικονικό hardware. Στην ουσία το frontend κομμάτι στέλνει δεδομένα εδώ μέσω Virtqueues και στη συνέχεια το backend κομμάτι κάνει τις απαραίτητες κλήσεις συστήματος για να κάνει access στην πραγματική συσκευή του host μηχανήματος. Αφού ολοκληρωθεί αυτό, στέλνει τα νέα δεδομένα πίσω στο frontend κομμάτι. Αυτό με τη σειρά του ενημερώνει την userspace εφαρμογή. Κυρίως τροποποιήσαμε το αρχείο virtio-cryptodev.c και (για πολύ λίγες δηλώσεις) το virtio-cryptodev.h.

Βασικές συναρτήσεις στο backend

vq_handle_output(vdev, vq)

Αρχικά μέσω της συνάρτησης virtqueue_pop() παίρνουμε το αντικείμενο τύπου VirtQueueElement. Οι πίνακες in_sg, out_sg αντιστοιχούν στις λίστες που προορίζονται για εγγραφή/ανάγνωση αντίστοιχα. Σε κάθε περίπτωση το out_sg[0] περιέχει τον τύπο του επιθυμητού system call. Έχουμε τις εξής περιπτώσεις:

VIRTIO_CRYPTODEV_SYSCALL_TYPE_OPEN

Σε αυτήν την περίπτωση, στο in_sg[0] βρίσκεται η διεύθυνση του file descriptor που θα πρέπει να χρησιμοποιηθεί για το άνοιγμα του ειδικού αρχείου και να επιστραφεί στον guest. Εκτελούμε open() στο ειδικό αρχείο /dev/crypto. Σε περίπτωση επιτυχίας ο file descriptor που υπάρχει μέσα στο ΛΣ του guest (host_fd) έχει πάρει την τιμή που επέστρεψε η κλήση συστήματος open() στο host μηχάνημα.

VIRTIO_CRYPTODEV_SYSCALL_TYPE_CLOSE

Σε αυτήν την περίπτωση, στο out_sg[1] βρίσκεται η διεύθυνση του file descriptor που θα πρέπει να χρησιμοποιηθεί για το κλείσιμο, ενώ στο in_sg[0] βρίσκεται η διεύθυνση της μεταβλητής επιστροφής, μέσω της οποίας θα ενημερωθεί ο guest για την επιτυχία/αποτυχία της κλήσης συστήματος. Ανακτούμε αυτές τις τιμές και εκτελούμε την αντίστοιχη close().

VIRTIO_CRYPTODEV_SYSCALL_TYPE_IOCTL

Σε αυτήν την περίπτωση στο `out_sg[1]` βρίσκεται η διεύθυνση του file descriptor που αναφέρεται στο αντίστοιχο ανοιχτό αρχείο της συσκευής cryptodev, ενώ στο `out_sg[2]` βρίσκεται η διεύθυνση της μεταβλητής που περιγράφει τον τύπο της επιθυμητής `ioctl()` κλήσης. Έχουμε τις εξής περιπτώσεις:

CIOCGSESSION

Διαβάζουμε την τιμή του κλειδιού κρυπτογράφησης, καθώς και τις διευθύνσεις του `struct session_op sess` και της μεταβλητής επιστροφής. Εκτελούμε `*ret = ioctl(*fd,CIOCGSESSION,sess)`, η οποία σε επιτυχία επιστρέφει μηδενική τιμή και αρχικοποιεί κατάλληλα τα πεδία του αντικειμένου `sess`.

CIOCFESSION

Διαβάζουμε την τιμή του identifier του session, ανακτούμε την διεύθυνση της μεταβλητής επιστροφής και εκτελούμε την κλήση συστήματος `*ret = ioctl(*fd,CIOCFSESSION,ses)`, η οποία σε επιτυχία επιστρέφει μηδενική τιμή και τερματίζει το session.

CIOCCRYPT

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 έχει μεταφερθεί για εγγραφή. Στην VirtQueue έχουν προστεθεί ξεχωριστά οι διευθύνσεις των διαφόρων μεταβλητών. 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. Επίσης ανακτούμε την διεύθυνση της μεταβλητής επιστροφής `ret` και μέσω της κλήσης συστήματος `ioctl()` εκτελείται η ενέργεια που περιγράφει το πεδίο `op` του νέου `struct crypt_op` (κρυπτογράφηση/αποκρυπτογράφηση) στα δεδομένα εισόδου και το αποτέλεσμα αποθηκεύεται στο πεδίο `dst` του νέου `struct` και συνεπώς στη μεταβλητή `dst` που είχαμε εξ αρχής προσθέσει στην ουρά (εφόσον μοιράζονται την ίδια θέση μνήμης). Τέλος, προσθέτουμε τα επεξεργασμένα στοιχεία στην ουρά μέσω της `virtqueue_push(vq, elem, 0)` και ”ειδοποιείται” ο guest για την ύπαρξη νέων δεδομένων μέσω της συνάρτησης `virtio_notify(vdev, vq)` (η συνάρτηση αυτή δεν κάνει τίποτα αφού η υλοποίηση μας ακολουθεί σύγχρονη λογική, αλλά αν θέλουμε να ακολουθήσουμε ασύγχρονη λογική μπορούμε στο σώμα της συνάρτησης να γράψουμε τον κώδικα που ξυπνάει την αντίστοιχη διεργασία). Μετά τον τερματισμό της συνάρτησης, η εκτέλεση συνεχίζεται στο frontend όπως περιγράφηκε παραπάνω.