
# ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολη ΗΜ&ΜΥ
Εργαστήριο Λειτουργικών Συστημάτων
Άσκηση 2
Ακ. Έτος 2012-13

**Παπαδημητρίου Κων/νος | ΑΜ: 03108769**
**Ημερομηνία: 09/07/2013**

## Εισαγωγή

Η άσκηση αυτή χωρίζεται σε δύο μέρη. Το πρώτο αφορά τη κατασκευή ενός προγράμματος chat και την επέκτασή του ώστε να χρησιμοποιεί τη συσκευή κρυπτογράφησης cryptodev κάνοντας κατάλληλες κλήσεις συστήματος ioctl προς το ειδικό αρχείο /dev/crypto.

Για το δεύτερο μέρος της άσκησης μας ζητήθηκε να υλοποιήσουμε μία εικονική συσκευή κρυπτογράφησης για εικονικές μηχανές που εκτελούνται από το QEMU. Η συσκευή εκτελεί ένα υποσύνολο των λειτουργιών που προσφέρει η συσκευή cryptodev. Ένας τρόπος να γίνει αυτό, είναι η πλήρης προσομοίωση των αλγορίθμων κρυπτογράφησης σε λογισμικό και η ενσωμάτωσή τους σε μια συσκευή του QEMU. Κάτι τέτοιο όμως δεν είναι αποδοτικό, ενώ επιπλέον είναι σημαντικής δυσκολίας η υλοποίηση των αλγορίθμων σε λογισμικό και ένα μικρό λάθος καθιστά το σύστημα άχρηστο. Έτσι καταφεύγουμε σε ένα δεύτερο τρόπο, που είναι μία πιο άμεση πρόσβαση της εικονικής μηχανής στο υλικό του host. Για να υλοποιήσουμε αυτή τη πρόσβαση χρησιμοποιούμε το πρότυπο VirtIO.

Το VirtIO είναι ένα πρότυπο, το οποίο ορίζει αφαιρετικό επίπεδο μέσω του οποίου ο backend οδηγός επικοινωνεί με τον frontend μέσω της ανταλλαγής κατάλληλων buffer δεδομένων. Πιο συγκεκριμένα, ορίζει τη δομή Virtqueue που είναι μία ουρά στην οποία ο frontend οδηγός τοποθετεί buffers για να τα παραλάβει ο backend. Αφού γίνει η επεξεργασία τους, ο backend driver απαντάει με τη προσθήκη κατάλληλων δεδομένων στην Virtqueue.

## Μέρος 1: *Εφαρμογή chat*

Για την εφαρμογή chat, χρησιμοποιούμε internet sockets. Γίνεται επικοινωνία μεταξύ ενός server και πολλών clients. Αρχική σκέψη ήταν ο server να κάνει μόνο listen περιμένοντας καινούριες συνδέσεις και accept για να τις δεχτεί, ενώ όταν συνδεθούν περισσότεροι του ενός clients, ο server να προωθεί μηνύματα του ενός προς τους άλλους. Τελικά, και ο server μπορεί να στείλει μηνύματα προς όποιον client θέλει ή προς όλους ταυτόχρονα (broadcast).

Αν ξεκινήσουμε την εφαρμογή χωρίς παραμέτρους, θα ξεκινήσει ένας server που θα ακούει σε μια πόρτα που by default είναι η 50543. Αν δοθεί η παράμετρος -p port, η θύρα στην οποία θα ακούει ο server θα είναι η port. Για να ξεκινήσουμε την εφαρμογή ως client και να συνδεθούμε με κάποιον server στις παραμέτρους πρέπει να δώσουμε και τη διεύθυνση του server (και όπως πριν με -p port δηλώνουμε την πόρτα στην οποία θα συνδεθούμε). Έτσι για να ξεκινήσει η εφαρμογή εκτελούμε από κάποιο shell:
για server: chat [-p port]
για client: chat [-p port] address

Όταν ξεκινήσει η εφαρμογή του server, αφού δημιουργηθεί το socket στο οποίο θα ακούει για νέες συνδέσεις, το δένουμε με μια διεύθυνση (bind) και ενεργοποιούμε τις συνδέσεις (listen). Ο server στη συνέχεια περιμένει για συνδέσεις (καλεί επαναληπτικά την accept) και για κάθε νέα σύνδεση, προσθέτει το αντίστοιχο socket σε FDSET (το οποίο αρχικά περιέχει μόνο το listening socket και το 0 (stdin)). Με χρήση της select παρακολουθούμε το FDSET ώστε να αποκρίνεται ο server σε καινούριες συνδέσεις και σε δεδομένα από κάποιο client ή από το stdin.

Ο client έχει πιο απλή λειτουργία. Δημιουργεί ένα socket και συνδέεται με τον server με τη connect. Στη συνέχεια, με χρήση της select, όπως και ο server, παρακολουθεί για δεδομένα είτε από τον server είτε από το stdin.

Όταν θέλει ένας client να αποχωρήσει, αρκεί ο χρήστης να πληκτρολογήσει /quit. Παρομοίως, για να στείλει ο server σε κάποιον συγκεκριμένο client μήνυμα ο χρήστης του πρέπει να πληκτρολογήσει τον file descriptor του αντίστοιχου socket και στη συνέχεια το

μήνυμα. Αν fd=0 το μήνυμα αποστέλλεται σε όλους και αν fd=-1 ο server διώχνει όλους τους clients και τερματίζει τη λειτουργία του.

### *Επέκταση chat για χρήση cryptodev*

Στη συνέχεια, η εφαρμογή του chat επεκτάθηκε, για να χρησιμοποιεί και τη συσκευή cryptodev, ώστε να κρυπτογραφεί τα μηνύματα που αποστέλλει. Για να γίνει η κρυπτογράφηση απαιτούνται κλήσεις συστήματος ioctl προς ένα ειδικό αρχείο της συσκευής. Τη διεύθυνση του αρχείου τη δίνει ο χρήστης σαν παράμετρο όταν ξεκινάει την εφαρμογή. Όταν ξεκινάει η εφαρμογή, αφού ανοίξει το ειδικό αρχείο με την κλήση open, με κατάλληλη κλήση ioctl ανοίγει ένα καινούριο session στη συσκευή, με χρήση ενός (μέχρι στιγμής προαποφασισμένου) κλειδιού. Έπειτα, συνεχίζει κανονικά την εκτέλεσή της, με τη διαφορά, πως πριν σταλλεί και αφού ληφθεί ένα μήνυμα, γίνεται κρυπτογράφηση και αποκρυπτογράφηση αντίστοιχα.

### *Μελλοντικές επέκτασεις και διορθώσεις*

Όπως είναι αυτή τη στιγμή η εφαρμογή, έχει αρκετές ελλείψεις και λάθη. Ωστόσο, καθώς δημιουργήθηκε καθαρά για λόγους επαλήθευσης της σωστής λειτουργίας του 2ου μέρους της άσκησης δεν έδωσα μεγάλη βάση στην επέκτασή της.

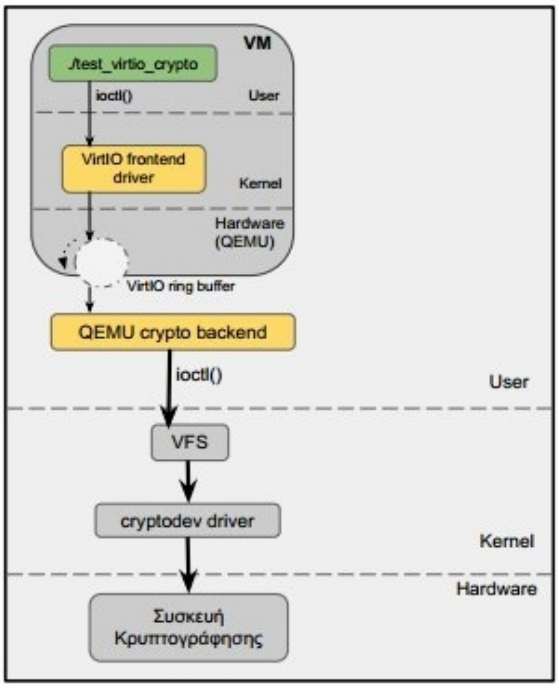
Αρχικά, θα είναι ωραίο, οι clients να έχουν τη δυνατότητα να μιλούν μεταξύ τους. Δηλαδή να μπορεί ο server να προωθεί τα μηνύματα που λαμβάνει, είτε προς όλους είτε προς μία συγκεκριμένη ομάδα από clients. Γενικά, θα είναι σωστό, να έχει ο χρήστης ένα πλήθος επιλογών για την αποστολή μηνυμάτων, όπως συμβαίνει στις περισσότερες εφαρμογές σήμερα.

Ένα πολύ σημαντικότερο κομμάτι όμως, είναι αυτό της κρυπτογράφησης. Έτσι όπως είναι τώρα η εφαρμογή, είναι εντελώς άχρηστη, καθώς το κλειδί είναι προαποφασισμένο και μπορεί ο καθένας να υποκλέψει μηνύματα και να τα αποκρυπτογραφήσει με τη χρήση του (γνωστού) κλειδιού. Έτσι θα πρέπει η ανταλλαγή των κλειδιών μεταξύ του server και των clients να γίνεται με χρήση κάποιου αλγορίθμου όπως ο rsa.

## **Μέρος 2:** *Λειτουργία του οδηγού*

Για το δεύτερο κομμάτι της άσκησης μας ζητήθηκε η υλοποίηση μίας εικονικής συσκευής κρυπτογράφησης για χρήση σε εικονικές μηχανές που εκτελούνται από το QEMU. Σκοπός της άσκησης τελικά ήταν η ανάπτυξη ενός οδηγού συσκευής που θα κάνει χρήση της συσκευής που χρησιμοποιήσαμε στο 1ο μέρος (cryptodev) μέσα σε εικονικές μηχανές. Ο οδηγός αποτελείται από 2 μέρη. Το frontend το οποίο βρίσκεται στην εικονική μηχανή (guest) και το backend το οποίο βρίσκεται στον host. Όπως είπαμε και στην εισαγωγή, τα δύο μέρη επικοινωνούν με τη χρήση του προτύπου VirtIO. Η αρχιτεκτονική που ακολουθεί ο οδηγός φαίνεται στο σχήμα 1 που ακολουθεί.

Όταν ξεκινάει λοιπόν η λειτουργία του guest και φορτωθεί ο οδηγός, μία εφαρμογή έχει πρόσβαση σε μία σειρά από ειδικά αρχεία συσκευών, στα οποία μπορεί να εκτελέσει κλήσεις συστήματος (open, close και ioctl), (σχεδόν) ακριβώς όπως θα έκανε, αν ο guest ήταν πραγματικό μηχάνημα και τα ειδικά αρχεία αντιπροσώπευαν πραγματικές συσκευές.




Σχήμα 1: Αρχιτεκτονική λογισμικού της (paravirtualized) virtio-crypto συσκευής

Μία από τις προδιαγραφές της συσκευής ήταν πως μόνο μια διεργασία, μπορεί να ανοίξει ένα ειδικό αρχείο κάθε φορά. Κάτι τέτοιο είναι εκ πρώτης όψης αρκετά εύκολο, αφού αρκεί ένας απλός έλεγχος ώστε για να δούμε αν ένα αρχείο είναι ήδη ανοιχτό. Ωστόσο, αν μια συσκευή έχει ανοίξει ένα ειδικό αρχείο και στη συνέχεια αποκήσει απογόνους, αυτοί θα έχουν πρόσβαση στη συσκευή. Σε αυτή την περίπτωση δημιουργήται πρόβλημα, καθώς όταν έρθει ένας buffer από τον host, θα υπάρχει race condition στις κοιμισμένες συσκευές και πιθανότατα τα δεδομένα θα καταλήξουν σε λάθος διεργασία. Το πρόβλημα αυτό δεν έχει επιληθεί, αν και μία σκέψη είναι να επεκταθεί η δομή των buffers ώστε να περιέχουν και το pid της διεργασίας για την οποία προορίζεται και ο frontend driver να τα προωθήσει κατάλληλα.

*Frontend:*

*Τί συμβαίνει όταν μια διεργασία κάνει κλήσεις προς τη συσκευή;*

Όταν μία διεργασία προσπαθήσει να ανοίξει ένα ειδικό αρχείο, ο driver θα βρει σε ποια (εικονική) συσκευή αντιστοιχεί, χρησιμοποιώντας το minor number του. Κάθε συσκευή αντιπροσωπεύεται στον οδηγό από μια δομή crypto_device, η οποία εκτός των άλλων περιέχει και το fd της συσκευής στον host και είναι αρχικοποιμένο σε -13 (κάτι που σημαίνει πως το αρχείο δεν έχει ανοιχθεί από άλλη διεργασία). Έτσι με έναν απλό έλεγχο, βλέπει ο οδηγός αν μπορεί να δώσει πρόσβαση στη διεργασία και αν όχι, η open επιστρέφει EMFILE. Φυσικά ο έλεγχος γίνεται με κλείδωμα, ώστε μια διεργασία να μπορεί να τον πραγματοποιήσει ατομικά. Επίσης, αν αποφασιστεί να επιτραπεί στη διεργασία να συνεχίσει, το fd γίνεται προσωρινά 0 ώστε όταν αφήσουμε το lock, η επόμενη διεργασία να πάρει EMFILE.

Τελικά, αφού ο driver διαμορφώσει έναν buffer (με τη δομή virtio_crypto_control) και τον τοποθετήσει στην αντίστοιχη Virtqueue της συσκευής, η διεργασία περιμένει στην αντίστοιχη waitqueue. Όταν έρθει η απάντηση από τον host, η διεργασία ξυπνάει και είναι

ελεύθερη να χρησιμοποιήσει τη συσκευή.

Η χρήση της συσκευής γίνεται με κλήσεις ioctl. Όταν γίνει κλήση ioctl, ο driver μεταφέρει τα δεδομένα στη δομή crypto_data την οποία διαμορφώνει σε ένα buffer (crypto_vq_buffer) που με τη σειρά του τοποθετεί στην κατάλληλη Virtqueue. Η διεργασία κοιμάται στη waitqueue μέχρι να έρθουν τα δεδομένα από τον host. Τότε ο driver μεταφέρει τα δεδομένα στο userspace.

Όταν η διεργασία κάνει close το ειδικό αρχείο, ο driver στέλνει ένα virtio_crypto_control με την αίτηση ώστε να κλείσει τη πραγματική συσκευή και επαναφέρει το fd σε -13 ώστε να μπορεί να χρησιμοποιήσει τη συσκευή μια άλλη διεργασία.

Για να στείλει έναν buffer στο backend, το frontend χρησιμοποιεί τις συναρτήσεις send_control_msg και send_buf για τις κλήσεις open/close και ioctl αντίστοιχα. Οι δύο συναρτήσεις δουλεύουν με παρόμοιο τρόπο. Για να στείλουν τους buffers στο backend χρησιμοποιούν scatter lists, οι οποίες καθιστούν δυνατή τη direct memory access. Αφού λοιπόν τοποθετήσουν τα δεδομένα στη Virtqueue (virtqueue_add_buf), στέλνουν σήμα στο backend για να τα δεχθεί και να τα επεξεργαστεί (virtqueue_kick) και περιμένουν μέχρι να ενημερωθούν για την επιτυχημένη μεταφορά.

Όταν ο host στείλει έναν buffer, προκαλεί ένα interrupt το οποίο χειρίζεται ο οδηγός με μία από τις συναρτήσεις control_in_intr ή in_intr αν πρόκειται για μήνυμα ελέγχου (open/close) ή για δεδομένα (ioctl) αντίστοιχα. Τότε συλλέγονται τα δεδομένα από τον buffer και ο driver ξυπνάει τις διεργασίες που τα περιμένουν.

*Backend:*

Όταν το backend δεχθεί μήνυμα (virtqueue_kick) από το frontend πως υπάρχει buffer προς επεξεργασία σε μία Virtqueue, τότε ανάλογα με ποια Virtqueue έχει τον buffer εκτελείται είτε η handle_output είτε η control_out. Αν λοιπόν το frontend έστειλε μήνυμα ελέγχου (τοποθέτησε buffer στην ουρά c_ovq), καλείται η control_out για να ανακτήσει τον buffer και στη συνέχεια καλεί τη handle_control_message ώστε να το χειριστεί. Αν από την άλλη το frontend έστειλε δεδομένα προς κρυπτογράφηση (τοποθέτησε buffer στην ουρά ovq), καλείται η handle_output που εξάγει τον buffer από την ουρά και με τη σειρά της καλεί την crypto_handle_ioctl_packet για να το χειριστεί.

Έτσι λοιπόν η handle_control_message, ανάλογα με το μήνυμα, εκτελεί open στην πραγματική συσκευή και απαντάει το fd στο frontend ή εκτελεί close. Για να στείλει την απάντηση στο frontend, χρησιμοποιεί την send_control_event η οποία μορφοποιεί και τοποθετεί έναν buffer στην ουρά c_ivq (virtqueue_push).

Η crypto_handle_ioctl_packet εκτελεί την αντίστοιχη κλήση ioctl που ζήτησε το frontend και στέλνει την απάντηση με τη send_buffer. Η τελευταία (χρησιμοποιώντας πάντα scatter lists) τοποθετεί τον buffer στην ουρά ivq.

Και στις δύο περιπτώσεις, οι συναρτήσεις προκαλούν interrupt στην εικονική μηχανή (κάνοντας virtio_notify στην αντίστοιχη Virtqueue), ώστε να το χειριστεί το frontend με το κατάλληλο interrupt handler.

## *Σημιώσεις για μελλοντικές επέκτασεις και διορθώσεις*

Ένα θέμα που όπως προανέφερα δεν έχει διορθωθεί ακόμα, είναι αυτό της πρόσβασης από πολλές διεργασίες στην ίδια εικονική συσκευή. Για την επίλυσή του υπάρχουν διάφορες σκέψεις, αλλά δυστυχώς καμία δεν έχει υλοποιηθεί ακόμα.

Επίσης, όταν μια διεργασία κάνει open σε ένα ειδικό αρχείο, αυτή τη στιγμή τα flags που μπορεί να περνάει δεν αποστέλονται στον host για να τα χρησιμοποιήσει στην αντίστοιχη

κλήση. Τώρα ο host εκτελεί open με O_RDWR (καθώς μόνο έτσι μπορεί η crypodev να χρησιμοποιηθεί σωστά). Για να γίνει αυτό, πρέπει να επεκταθεί η δομή virtio_crypto_control και να έχει και ένα attribute flags (τύπου int) το οποία θα αποθηκεύει το filp->f_flags.