**ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ**

**ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ KΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ**



Εργαστήριο Υπολογιστικών Συστημάτων

**Άσκηση 3:** *Κρυπτογραφική συσκευή VirtIO για QEMU-KVM*

**Ημερομηνία:** *24/06/2016*

**Ομάδα cslabb06**

***Πετρόπουλος-Τράκας Ευθύμης (031 11525)***

***Ροδίτης-Κουτσαντώνης Ορέστης (031 11052)***

**3η Άσκηση**

Αντικείμενο της παρούσας άσκησης είναι η ανάπτυξη εικονικού υλικού στο περιβάλλον εικονικοποίησης QEMU-KVM.

Στο πλαίσιο της άσκησης, θα σχεδιάσετε και θα υλοποίησετε εικονική κρυπτογραφική συσκευή VirtIO, η οποία θα αποτελεί πλέον μέρος του QEMU.

Η εικονική κρυπτογραφική συσκευή θα επιτρέπει σε διεργασίες που εκτελούνται μέσα στην εικονική μηχανή να έχουν πρόσβαση σε πραγματική κρυπτογραφική συσκευή του host, τύπου cryptodev-linux, με χρήση τεχνικής παρα-εικονικοποίησης (paravirtualization).

Η συσκευή cryptodev-linux επιτρέπει σε εφαρμογές να έχουν πρόσβαση σε επιταχυντές υλικού για κρυπτογραφία (hardware crypto accelerators), κάνοντας κλήσεις σε ειδικό αρχείο /dev/crypto. Περισσότερα για τον τρόπο πρόσβασης σε συσκευές cryptodev-linux και το προσφερόμενο API μπορείτε να διαβάσετε στον οδηγό προγραμματισμού cryptodev-linux που σας δίνεται, καθώς και στο <http://cryptodev-linux.org/>.

Το σενάριο χρήσης του οδηγού σας, με το οποίο και θα επαληθεύσετε τη σωστή λειτουργία του, είναι η υλοποίηση ενός εργαλείου για κρυπτογραφημένη επικοινωνία (encrypted chat) πάνω από TCP/IP sockets, όπως φαίνεται στο Σχήμα 1.

**Ζητούμενα της Άσκησης**

Τα ζητούμενα της άσκησης, σε χρονολογική σειρά είναι:

***Ζ1: Εργαλείο chat πάνω από TCP/IP sockets***

Το εργαλείο αυτό θα επιτρέπει αμφίδρομη επικοινωνία πάνω από TCP/IP, με χρήση του BSD Sockets API, χωρίς κρυπτογράφηση των μηνυμάτων που ανταλλάσσονται.

***Ζ2: Κρυπτογραφημένο chat πάνω από TCP/IP***

Επέκταση του εργαλείου chat, ώστε τα δεδομένα που μεταφέρονται πάνω από TCP/IP να είναι κρυπτογραφημένα με προσυμφωνημένο κλειδί. Χρή- ση του cryptodev-linux API, από userspace, για την κρυπτογράφηση των δεδομένων.

***Ζ3: Υλοποίηση συσκευής cryptodev με VirtIO***

Σχεδίαση και υλοποίηση εικονικής συσκευής cryptodev, με χρήση του πλαι- σίου VirtIO, έτσι ώστε το προηγούμενο εργαλείο να μπορεί να εκτελείται μέσα σε VM κάνοντας χρήση κρυπτογραφικών επιταχυντών σε υλικό, οι οποίοι ως τώρα ήταν προσβάσιμοι μόνο από τον host. Το εργαλείο πρέπει να εκτελείται ακριβώς με το ίδιο API, αλλάζοντας μόνο τη συσκευή την οποία χρησιμοποιεί για κρυπτογράφηση των δεδομένων.

**Ακολουθεί σύντομη επεξήγηση των επιλογών που κάναμε όσον αφορά τις προσθήκες / αλλαγές κώδικα ώστε ο οδηγός και το chat να δουλέψουν σωστά.**

**Υλοποίηση Chat μέσω BSD Sockets, με τη χρήση του μηχανισμού POLL.**

Για το κομμάτι της υλοποίσης του chat, χρησιμοποιήσαμε, όπως μας ζητήθηκε, BSD Sockets. Ακολούθησαμε το πρότυπο server - client, δηλαδή μια διεργασία δεσμεύει μια TCP πορτα (bind), στην οποία ακούει (listen) και δέχεται συνδέσεις (accept) από πιθανούς clients μέσω TCP Sockets. Όταν λάβει μια αίτηση για σύνδεση, τότε βάζει στην ουρά (backlog) οποιεσδήποτε άλλες συνδέσεις πιθανόν έρθουν μέχρι να τελειώσει με την ήδη ανοικτή σύνδεση, με την οποία στη συνέχεια ανταλλάσει μηνύματα, αφού πρώτα κρυπτογραφήσει τα εξερχόμενα και αποκρυπτογραφήσει τα εισερχόμενα.

Αντίστοιχα, ο client δημιουργεί ένα TCP Socket, το οποίο και συνδέει στην προκαθόρισμένη πόρτα, στην οποία ακού ει ο server. Αφού συνδεθεί, στέλνει ένα προκαθορισμένο μήνυμα, για να “καταλάβει” ο χειριστής του server ότι κάποιος συνδέθηκε. Στη συνέχεια και σε αντιστοίχιση με τις λειτουργίες του server, ο client κρυπτογραφεί τα εξερχόμενα μηνύματα και αποκρυπτογραφεί τα εισερχόμενα.

Η κρυπτογράφηση και αποκρυπτογράφηση των μηνυμάτων γίνεται με τη χρήση της συσκευής /dev/cryptodev0, για την οποία υλοποιήσαμε οδηγό για πρόωθηση των αιτημάτων στην συσκευή /dev/crypto του host μηχανήματος για επιτάχυνση από το υλικό της διαδικασίας της κρυπτογράφησης (λεπτομέρειες για τον οδηγό ακολουθούν έπειτα). Χρησιμοποιείται ένα προσυμφωνημένο κλειδί για την διαδικασία (απο)κρυπτογράφησης, το οποίο έχει δοθεί και στις δύο πλευρές κατά τη διάρκεια της μεταγλώτισσης, αλλά θα μπορούσε απλά να ανταλλάσεται ένα τυχαία παραγόμενο κλειδί, σαν πρώτο μήνυμα (αναγνωριστικό από τον client) αποκρυπτογραφημένο και από εκεί κι έπειτα να επικοινωνούν με κρυπτογράφηση τα δύο άκρα. Αυτό όμως είναι κάτι που δεν ζητείται για την άσκηση και ξεφεύγει του ζητούμενου.

Για να επιτύχουμε την ασύγχρονη επικοινωνία μεταξύ των sockets του Client και του Server χρησιμοποιήσαμε τον μηχανισμό POLL, ο οποίος με την χρήση των κατάλληλων δικών του structs, στα οποία δηλώνουμε σε ποιούς file descriptors θέλουμε να δράσε και σε ποια eventsι, μας ειδοποιεί ασύγχρονα όποτε έχουμε κάποια δεδομένα έτοιμα προς διάβασμα στον αντίστοιχο file descriptor. Με αυτόν τον τρόπο επιτυγχάνουμε ασύγχρονη επικοινωνία, χωρίς τη χρήση threads ή διεργασιών-παιδιών, με χρήση fork. Επιλέξαμε τον μηχανισμό POLL έναντι του select, καθώς έχει καλύτερη κλιμάκωση, ταχύτητα και πιο σύγχρονο API, επομένως και πιο φιλικό για τον προγραμματιστή.

Επίσης φτιάξαμε μια συνάρτηση για ασφαλή ανάγνωση του μηνύματος από τα sockets, η οποία είναι η *insist\_read* στο αρχείο *socket-common.h* η οποία φροντίζει να διαβάζει το σωστό μέγεθος από bytes από το socket, ώστε να μπορεί να γίνεται ορθά η αποκρυπτογράφηση των δεδομένων, καθώς ο αλγόριθμος κρυπτογράφησης που επιλέξαμε (AES128) επιστρέφει πάντα συγκεκριμένο αριθμό από bytes μετά τη διαδικασία κρυπτογράφησης. Έτσι, έπρεπε να είμαστε σίγουροι ότι έχουμε λάβει όλα τα bytes πρωτού ξεκινήσουμε την αντίστροφη διαδικασία, της αποκρυπτογράφησης.

**Υλοποίηση του driver (από τη μεριά του guest μηχανήματος):**

Το πρώτο αρχείο που αλλάξαμε ήταν το *crypto-chrdev.c*.

Στη συνάρτηση *crypto\_chrdev\_open* δημιουργήσαμε έναν πίνακα από *scatter-gather structs*, με ένα struct που περιείχε τον δείκτη προς τον αριθμό του operation για το open, και ένα struct που περιείχε τον δείκτη προς το *host\_fd*, δηλαδή το αποτέλεσμα που θα παίρναμε από τον host, το οποίο είναι ο file descriptor του cryptodev αρχείου που άνοιξε ο virtio οδηγός από την πλευρά του QEMU. Ο εν λόγω πίνακας, που σε όλα τα σημεία του κώδικα έχει το όνομα sgs, λειτουργεί σαν *scatter-gather list*, για την επικοινωνία με τον virtio οδηγό. Στη συνέχεια τοποθετήσαμε το sgs στο virtqueue του character device μας, κάναμε kick σε αυτό και λάβαμε τα δεδομένα που περιμέναμε αφού πρώτα εκτελέσουμε ενεργό αναμονή, καλώντας συνεχώς την *virtqueue\_get\_buf()* μέχρι να επιστρέψει τα δεδομένα που θέλουμε.

Στη συνάρτηση *crypto\_chrdev\_release()* κάναμε παρόμοια δουλειά, παρ’όλα αυτά σε αυτή την περίτπωση είχαμε δεδομένα μόνο προς αποστολή, τον αριθμό του operation και τον file descriptor του αρχείου που αντιστοιχεί στο ανοιχτό από τη μεριά του host αρχείου. Αν και δεν είχαμε δεδομένα να λάβουμε, εντούτοις μετά την αποστολή των δεδομένων μέσω του virtqueue εκτελέσαμε πάλι μια επανάληψη πάνω στο *virtqueue\_get\_buf()*, προκειμένου η εκτέλεση στον οδηγό του guest να συνεχιστεί μόνον όταν το backend κομμάτι τελειώσει τη δουλειά του.

Στη συνάρτηση *crypto\_chrdev\_ioctl()* έπρεπε να φροντίσουμε ώστε να εκτελεστεί σωστά κάθε μία από τις 3 δυνατές λειτουργίες που μας ζητούνταν να προσφέρει ο οδηγός μας. Τα κοινά δεδομένα που θα πρέπει να αποσταλούν μέσω του virtqueue σε κάθε περίπτωση ήταν ο αριθμός του operation για το ioctl, ο file descriptor του ανοιχτού host αρχείου, και ο αριθμός για το συγκεκριμένο command που θα εκτελούσε η κλήση ioctl. Οπότε πριν από οτιδήποτε άλλο προσθέταμε στην sgs στοιχεία με buffers προς αυτά τα δεδομένα.

Στην περίπτωση της λειτουργίας CIOGSESSION, γνωρίζουμε ότι το όρισμα arg που δεχόμαστε είναι ένας δείκτης προς ένα *struct session\_op* στοιχείο. Σκοπός μας είναι, προκειμένου να περάσουμε ολόκληρη την πληροφορία που περιέχεται στο όρισμα αυτό, να δημιουργήσουμε ένα πλήρες αντίγραφο του εν λόγω *struct session\_op*, και των απλών στοιχείων του και των δεδομένων που περιέχονται στους buffers του. Ο λόγος είναι ότι τα δεδομένα του arg βρίσκονται εξ ολοκλήρου στο userspace, και θέλουμε ένα πλήρες αντίγραφό τους σε kernelspace, προκειμένου να μπορέσουμε να τα στείλουμε στο virtio-crypto μέσω του virtqueue. Έτσι και κάνουμε, δημιουργούμε μια τοπική μεταβλητή τύποτ struct session\_op, αντιγράφουμε σε αυτή όλα τα στοιχεία της αρχικής μεταβλητής, έπειτα κάνουμε malloc (kzalloc) στον buffer *key* του νέου στοιχείου με το κατάλληλο μέγεθος, και αντιγράφουμε τα περιεχόμενα του key του struct που μας περάστηκε ως όρισμα, στον buffer *key* του νέου μας στοιχείου. Στη συνέχεια βάζουμε σε scatter gather structs ξεχωριστά τον buffer προς το *key* του νέου *session\_op struct*, τον δείκτη προς το ίδιο το struct και τον δείκτη προς το *host\_return\_val*, τα στέλνουμε μέσω του virtqueue μέσω του *virtqueue\_kick()*, αφού τα κάνουμε add σε αυτό, και δεχόμαστε μετά από ενεργό αναμονή την τιμή του *host\_return\_val*. Το *host\_return\_val* είναι σε αυτή την περίπτωση ο έγκυρος αριθμός session id που πρέπει να βρίσκεται στο μέλος *ses* τόσο του *session\_op struct* που μας ενδιαφέρει, όσο και στο *crypt\_op struct*. Εδώ να σημειώσουμε ότι κάθε περίπτωση από τις CIOGSESSION, CIOFSESSION και CIOCRYPT, η ομάδα εντολών που αναλαμβάνει να στείλει τα δεδομένα στο virtqueue και να παραλάβει από αυτό τα αποτελέσματα είναι ξεχωριστή, καθώς κάθε μία από τις λειτουργίες έχει ειδικές ανάγκες όσον αφορά τον χειρισμό της τιμής επιστροφής και δε θα μπορούσαμε να βάλουμε εντελώς κοινό κώδικα που να κάνει αυτή τη δουλειά για όλες. Επίσης, κάθε μία από αυτές τις ομάδες εντολών εκτελείται εξ’ ολοκλήρου ενώ έχει αποκτήσει το κλείδωμα spinlock του struct crypto\_device που αντιστοιχεί στο ανοιχτό αρχείο, ώστε να μην μπλέκονται πολλές διεργασίας και υπάρχουν πολλά δεδομένα προς αποστολή και λήψη στο virtqueue την ίδια στιγμή.

Στην περίπτωση του CIOFSESSION, κάνοντας παρόμοια δουλειά με παραπάνω, στείλαμε μόνο το session id ως επιπλέον πληροφορία, καθώς μόνο αυτό χρειαζόταν ώστε να λήξουμε τη συνεδρία. Λαμβάνουμε έπειτα μετά από ενεργό αναμονή το *host\_return\_va*l, δεν χρησιμοποιούμε όμως κάπου αυτή την τιμή. Το session\_id το λάβαμε αυτή τη φορά μέσω του arg, το οποίο ήταν δείκτης προς unsigned int.

Τέλος φροντίσαμε για τη σωστή λειτουργία του CIOCRYPT. Μέσω του arg λαμβάνουμε δείκτη προς ένα *struct crypt\_op*. Αναλόγως προς την περίπτωση του CIOGSESSION, αντιγράφουμε τα στοιχεία και τα δεδομένα των buffers του αρχικού struct σε ένα νέο struct που βρίσκεται εξ’ ολοκλήρου στο kernelspace του guest. Στέλνουμε στη συνέχεια το struct, και ξεχωριστά κάθε buffer (στον buffer dst κάναμε μόνο malloc, καθώς θα γεμίσει με δεδομένα μετά την εκτέλεση της λειτουργίας) και στέλνουμε τα δεδομένα μέσω του virtqueue. Μετά την εκτέλεση των απαραίτητων λειτουργιών από το virtio-crypto, ο buffer *dst* στο kernelspace έχει τα δεδομένα που ζητούνται, και αντιγράφουμε τα δεδομένα αυτά στον buffer *dst* του αρχικού userspace struct.

Το επόμενο αρχείο που αλλάξαμε ήταν το *crypto.h*.

Σε αυτό αλλάξαμε την υλοποίση του *struct crypto\_device* ώστε να περιέχει μηχανισμό υλοποίησης κλειδώματος, για προστασία από παράλληλες προσβάσεις στον ίδιο file descriptor. Επιλέξαμε *spinlock\_t* για την ευκολία υλοποίησης. Θα μπορούσαμε για καλύτερη απόδοση να χρησιμοποιήσουμε σημαφόρους, αλλά δεν θα ήταν μεγάλη η διαφορά, καθώς τα δεδομένα που στέλνονται είναι πολύ μικρού μεγέθους, οπότε η διαδικασία του active waiting δεν διαρκεί μεγάλο χρονικό διάστημα.

**Υλοποίηση του driver (από τη μεριά του host μηχανήματος):**

Το αρχείο που αλλάξαμε για τις αλλαγές στον guest ήταν το virtio-crypto.c

Σε αυτό το αρχείο καλούμασταν να συμπληρώσουμε τη συνάρτηση *vq\_handle\_output* η οποία και διαχειρίζεται τα δεδομένα που στέλνει ο kernelspace του guest μηχανήματος και να τα προωθήσει στον kernel του host μηχανήματος. Για το λόγο αυτόν, ανάλογα με την εντολή που ερχόταν από το guest μηχάνημα εκτελούσαμε τις αντίστοιχες ενέργειες σε αντιστοίχηση με τη διαδικασία και τις δομές που περιγράφονται παραπάνω στο κομμάτι του guest μηχανήματος, και τα αρχείο *crypto-chrdev.c*.

Αρχικά στην περίπτωση της κλησης συστήματος OPEN, ανοίγαμε το αρχείο /dev/crypto και επιστρέφαμε τον file descriptor που μας έδινε η κλήση στην οpen, βάζοντας την τιμή αυτή στον δείκτη που έδειχνε τα δεδομένα στο χώρο πυρήνα του guest μηχανήματος.

Αντίστοιχα στην περίπτωση του CLOSE, κλείναμε το αρχείο του οποίου μας δινόταν ο file descriptor.

Στην περίπτωση της κλήσης της IOCTL, παίρναμε περιπτώσεις αντίστοιχες με αυτές στο guest μηχάνημα. Σε όλες τις περιπτώσεις είχαμε δείκτες προς τις δομές του kernelspace του guest μηχανήματος και φτιάχναμε τοπικές μεταβλητές (όπου χρειαζόταν) στο userspace του host, στα οποία αντιστοιχίζαμε τα πεδία που επιθυμούσαμε. Έπειτα καλούσαμε την αντίστοιχη εντολή με τη χρήση της IOCTL κι επιστρέφαμε τα δεδομένα αντιγράφοντας από την τοπική μεταβλητή στον δείκτη που έδειχνε τις έγκειρες θέσεις μνήμης για τον guest. Να αναφέρουμε ειδικά, ότι για πεδία δομών τα οποία είναι δείκτες, όπως π.χ. το src, το dst και το iv, έπρεπε να κάνουμε κάποια εξτρά βήματα για να γράφουμε στις σωστές θέσεις μνήμης. Αυτό το κάναμε αντιστοιχίζοντας τα πεδία αυτά με πραγματικές θέσεις μνήμης για το host μηχάνημα, με τη χρήση των δομών *elem.in\_sg* και *elem.out\_sg*.

Επίσης σε ορισμένες περιπτώσεις (π.χ. το dst πεδίο της crypto\_op) έπρεπε να γίνει malloc πρωτού καλέσουμε την IOCTL, ώστε να έχει το θεμιτό μέγεθος για τη λειτουργία της (από)κρυπτογράφησης. Στη συνέχεια αντιγράφαμε τα δεδομένα από την τοπική μεταβλητή στη θέση μνήμης που έδειχνε ο δείκτης για την dst, με τη χρήση της memcpy.

**Άσκηση 3: Υλοποίηση:**

**sockets/socket-server.c**

*/\**

*\* socket-server.c*

*\* Simple TCP/IP communication using sockets*

*\**

*\* Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>*

*\*/*

*#include <stdio.h>*

*#include <errno.h>*

*#include <ctype.h>*

*#include <string.h>*

*#include <stdlib.h>*

*#include <signal.h>*

*#include <unistd.h>*

*#include <netdb.h>*

*#include <fcntl.h>*

*#include <sys/time.h>*

*#include <sys/types.h>*

*#include <sys/socket.h>*

*#include <sys/stat.h>*

*#include <arpa/inet.h>*

*#include <netinet/in.h>*

*#include <crypto/cryptodev.h>*

*#include <sys/ioctl.h>*

*#include <poll.h>*

*#include "socket-common.h"*

*/\* Convert a buffer to upercase \*/*

*void toupper\_buf(char \* buf, size\_t n) {*

*size\_t i;*

*for (i = 0; i < n; i++)*

*buf[i] = toupper(buf[i]);*

*}*

*int main(void) {*

*char buf[256], iv[16];*

*char addrstr[INET\_ADDRSTRLEN];*

*int sd, newsd;*

*ssize\_t n;*

*socklen\_t len;*

*struct sockaddr\_in sa;*

*/\* Make sure a broken connection doesn't kill us \*/*

*signal(SIGPIPE, SIG\_IGN);*

*/\* Create TCP/IP socket, used as main chat channel \*/*

*if ((sd = socket(PF\_INET, SOCK\_STREAM, 0)) < 0) {*

*perror("socket");*

*exit(1);*

*}*

*fprintf(stderr, "Created TCP socket\n");*

*/\* Bind to a well-known port \*/*

*memset( & sa, 0, sizeof(sa));*

*sa.sin\_family = AF\_INET;*

*sa.sin\_port = htons(TCP\_PORT);*

*sa.sin\_addr.s\_addr = htonl(INADDR\_ANY);*

*if (bind(sd, (struct sockaddr \* ) & sa, sizeof(sa)) < 0) {*

*perror("bind");*

*exit(1);*

*}*

*fprintf(stderr, "Bound TCP socket to port %d\n", TCP\_PORT);*

*/\* Listen for incoming connections \*/*

*if (listen(sd, TCP\_BACKLOG) < 0) {*

*perror("listen");*

*exit(1);*

*}*

*/\* Loop forever, accept()ing connections \*/*

*for (;;) {*

*fprintf(stderr, "Waiting for an incoming connection...\n");*

*/\* Accept an incoming connection \*/*

*len = sizeof(struct sockaddr\_in);*

*if ((newsd = accept(sd, (struct sockaddr \* ) & sa, & len)) < 0) {*

*perror("accept");*

*exit(1);*

*}*

*if (!inet\_ntop(AF\_INET, & sa.sin\_addr, addrstr, sizeof(addrstr))) {*

*perror("could not format IP address");*

*exit(1);*

*}*

*fprintf(stderr, "Incoming connection from %s:%d\n",*

*addrstr, ntohs(sa.sin\_port));*

*/\* We break out of the loop when the remote peer goes away \*/*

*int cfd = open("/dev/cryptodev0", O\_RDWR);*

*struct session\_op sess;*

*struct crypt\_op crypt;*

*memset( & sess, 0, sizeof(sess));*

*memset( & crypt, 0, sizeof(crypt));*

*memset(iv, 0x0, sizeof(iv));*

*sess.cipher = CRYPTO\_AES\_CBC;*

*sess.keylen = 16;*

*sess.key = (unsigned char \* ) CRYPTOKEY;*

*ioctl(cfd, CIOCGSESSION, & sess);*

*crypt.ses = sess.ses;*

*char inbuf[256];*

*char tmpbuff[256];*

*struct pollfd ufds[2];*

*int rv;*

*// Socket*

*ufds[0].fd = newsd;*

*ufds[0].events = POLLIN;*

*// STDIN*

*ufds[1].fd = 0;*

*ufds[1].events = POLLIN;*

*for (;;) {*

*rv = poll(ufds, 2, 3500);*

*bzero(buf, sizeof(buf));*

*bzero(inbuf, sizeof(inbuf));*

*bzero(tmpbuff, sizeof(tmpbuff));*

*if (rv == -1) {*

*perror("poll"); // error occurred in poll()*

*}*

*else if (rv == 0) {*

*#ifdef DEBUG*

*printf("Timeout occurred! No data after 3.5 seconds.\n");*

*#endif*

*}*

*else {*

*// check for events on Socket:*

*if (ufds[0].revents & POLLIN) {*

*n = insist\_read(newsd, buf, sizeof(buf));*

*// DECRYPT*

*crypt.len = sizeof(buf);*

*crypt.src = (unsigned char \* ) buf;*

*crypt.dst = (unsigned char \* ) tmpbuff;*

*crypt.iv = (unsigned char \* ) iv;*

*crypt.op = COP\_DECRYPT;*

*ioctl(cfd, CIOCCRYPT, & crypt);*

*// DECRYPT*

*if (n <= 0) {*

*if (n < 0)*

*perror("read from remote peer failed");*

*else*

*fprintf(stderr, "Peer went away\n");*

*break;*

*}*

*fprintf(stdout, "CLIENT: %s", tmpbuff);*

*fflush(stdout);*

*}*

*// check for events on STDIN:*

*if (ufds[1].revents & POLLIN) {*

*n = read(0, inbuf, sizeof(inbuf));*

*inbuf[sizeof(inbuf) - 1] = '\0';*

*fflush(stdin);*

*// ENCRYPT*

*crypt.len = sizeof(inbuf);*

*crypt.src = (unsigned char \* ) inbuf;*

*crypt.dst = (unsigned char \* ) tmpbuff;*

*crypt.iv = (unsigned char \* ) iv;*

*crypt.op = COP\_ENCRYPT;*

*ioctl(cfd, CIOCCRYPT, & crypt);*

*// ENCRYPT*

*if (insist\_write(newsd, tmpbuff, sizeof(tmpbuff)) != sizeof(tmpbuff)) {*

*perror("write to remote peer failed");*

*break;*

*}*

*}*

*}*

*}*

*/\* Make sure we don't leak open files \*/*

*if (close(newsd) < 0)*

*perror("close");*

*ioctl(cfd, CIOCFSESSION, crypt.ses);*

*close(cfd);*

*}*

*/\* This will never happen \*/*

*return 1;*

*}*

**sockets/socket-client.c**

*/\**

*\* socket-client.c*

*\* Simple TCP/IP communication using sockets*

*\**

*\* Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>*

*\*/*

*#include <stdio.h>*

*#include <errno.h>*

*#include <ctype.h>*

*#include <string.h>*

*#include <stdlib.h>*

*#include <signal.h>*

*#include <unistd.h>*

*#include <netdb.h>*

*#include <fcntl.h>*

*#include <sys/time.h>*

*#include <sys/types.h>*

*#include <sys/socket.h>*

*#include <sys/stat.h>*

*#include <arpa/inet.h>*

*#include <netinet/in.h>*

*#include <crypto/cryptodev.h>*

*#include <sys/ioctl.h>*

*#include <poll.h>*

*#include "socket-common.h"*

*int main(int argc, char \*argv[])*

*{*

*int sd, port;*

*ssize\_t n;*

*char buf[256], tmpbuff[256], iv[16];*

*char \*hostname;*

*struct hostent \*hp;*

*struct sockaddr\_in sa;*

*int cfd = open("/dev/cryptodev0", O\_RDWR);*

*struct session\_op sess;*

*struct crypt\_op crypt;*

*memset(&sess, 0, sizeof(sess));*

*memset(&crypt, 0, sizeof(crypt));*

*memset(iv, 0x0, sizeof(iv));*

*sess.cipher = CRYPTO\_AES\_CBC;*

*sess.keylen = 16;*

*sess.key = (unsigned char \*) CRYPTOKEY;*

*ioctl(cfd, CIOCGSESSION, &sess);*

*crypt.ses = sess.ses;*

*if (argc != 3) {*

*fprintf(stderr, "Usage: %s hostname port\n", argv[0]);*

*exit(1);*

*}*

*hostname = argv[1];*

*port = atoi(argv[2]); /\* Needs better error checking \*/*

*/\* Create TCP/IP socket, used as main chat channel \*/*

*if ((sd = socket(PF\_INET, SOCK\_STREAM, 0)) < 0) {*

*perror("socket");*

*exit(1);*

*}*

*fprintf(stderr, "Created TCP socket\n");*

*/\* Look up remote hostname on DNS \*/*

*if ( !(hp = gethostbyname(hostname))) {*

*printf("DNS lookup failed for host %s\n", hostname);*

*exit(1);*

*}*

*/\* Connect to remote TCP port \*/*

*sa.sin\_family = AF\_INET;*

*sa.sin\_port = htons(port);*

*memcpy(&sa.sin\_addr.s\_addr, hp->h\_addr, sizeof(struct in\_addr));*

*fprintf(stderr, "Connecting to remote host... "); fflush(stderr);*

*if (connect(sd, (struct sockaddr \*) &sa, sizeof(sa)) < 0) {*

*perror("connect");*

*exit(1);*

*}*

*fprintf(stderr, "Connected.\n");*

*/\* Be careful with buffer overruns, ensure NUL-termination \*/*

*strncpy(buf, HELLO\_THERE, sizeof(buf));*

*buf[sizeof(buf) - 1] = '\0';*

*// ENCRYPT*

*crypt.len = sizeof(buf);*

*crypt.src = (unsigned char \*) buf;*

*crypt.dst = (unsigned char \*) tmpbuff;*

*crypt.iv = (unsigned char \*) iv;*

*crypt.op = COP\_ENCRYPT;*

*ioctl(cfd, CIOCCRYPT, &crypt);*

*// ENCRYPT*

*#ifdef DEBUG*

*print\_encrypted(tmpbuff, sizeof(tmpbuff));*

*#endif*

*/\* Say something... \*/*

*if (insist\_write(sd, tmpbuff, sizeof(tmpbuff)) != sizeof(tmpbuff)) {*

*perror("write");*

*exit(1);*

*}*

*printf("CLIENT: %s\n", buf);*

*fflush(stdout);*

*/\* Read answer and write it to standard output \*/*

*char inbuf[256];*

*struct pollfd ufds[2];*

*int rv;*

*// Socket*

*ufds[0].fd = sd;*

*ufds[0].events = POLLIN;*

*// STDIN*

*ufds[1].fd = 0;*

*ufds[1].events = POLLIN;*

*for (;;) {*

*rv = poll(ufds, 2, 3500);*

*bzero(buf, sizeof(buf));*

*bzero(inbuf, sizeof(inbuf));*

*bzero(tmpbuff, sizeof(tmpbuff));*

*if (rv == -1) {*

*perror("poll"); // error occurred in poll()*

*}*

*else if (rv == 0) {*

*#ifdef DEBUG*

*printf("Timeout occurred! No data after 3.5 seconds.\n");*

*#endif*

*}*

*else {*

*// check for events on Socket:*

*if (ufds[0].revents & POLLIN) {*

*n = insist\_read(sd, buf, sizeof(buf));*

*// DECRYPT*

*crypt.len = sizeof(buf);*

*crypt.src = (unsigned char \* ) buf;*

*crypt.dst = (unsigned char \* ) tmpbuff;*

*crypt.iv = (unsigned char \* ) iv;*

*crypt.op = COP\_DECRYPT;*

*ioctl(cfd, CIOCCRYPT, & crypt);*

*// DECRYPT*

*if (n <= 0) {*

*if (n < 0)*

*perror("read from remote peer failed");*

*else*

*fprintf(stderr, "Peer went away\n");*

*break;*

*}*

*fprintf(stdout, "Server: %s", tmpbuff);*

*fflush(stdout);*

*}*

*// check for events on STDIN:*

*if (ufds[1].revents & POLLIN) {*

*n = read(0, inbuf, sizeof(inbuf));*

*inbuf[sizeof(inbuf) - 1] = '\0';*

*fflush(stdin);*

*// ENCRYPT*

*crypt.len = sizeof(inbuf);*

*crypt.src = (unsigned char \* ) inbuf;*

*crypt.dst = (unsigned char \* ) tmpbuff;*

*crypt.iv = (unsigned char \* ) iv;*

*crypt.op = COP\_ENCRYPT;*

*ioctl(cfd, CIOCCRYPT, & crypt);*

*// ENCRYPT*

*if (insist\_write(sd, tmpbuff, sizeof(tmpbuff)) != sizeof(tmpbuff)) {*

*perror("write to remote peer failed");*

*break;*

*}*

*}*

*}*

*}*

*fprintf(stderr, "\nDone.\n");*

*ioctl(cfd, CIOCFSESSION, crypt.ses);*

*close(cfd);*

*return 0;*

*}*

**sockets/socket-common.h**

*/\**

*\* socket-common.h*

*\**

*\* Simple TCP/IP communication using sockets*

*\**

*\* Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>*

*\*/*

*#ifndef \_SOCKET\_COMMON\_H*

*#define \_SOCKET\_COMMON\_H*

*/\* Compile-time options \*/*

*#define TCP\_PORT 35001*

*#define TCP\_BACKLOG 5*

*#define CRYPTOKEY "\_Such\_key\_w0w\_12"*

*#define HELLO\_THERE "Initial Connection from peer!\n"*

*#include <stdio.h>*

*#include <errno.h>*

*#include <ctype.h>*

*#include <stdlib.h>*

*#include <sys/types.h>*

*#include <sys/socket.h>*

*void print\_encrypted(char \*data, int len) {*

*printf("\nEncrypted data:\n");*

*int i;*

*for (i = 0; i < len; i++) {*

*printf("%x", data[i]);*

*}*

*printf("\n");*

*}*

*void print\_decrypted(char \*data, int len) {*

*printf("\nDecrypted data:\n");*

*int i;*

*for (i = 0; i < len; i++) {*

*printf("%c", data[i]);*

*}*

*printf("\n");*

*}*

*/\* Insist until all of the data has been written \*/*

*ssize\_t insist\_write(int fd, const void \*buf, size\_t cnt)*

*{*

*ssize\_t ret;*

*size\_t orig\_cnt = cnt;*

*while (cnt > 0) {*

*ret = write(fd, buf, cnt);*

*if (ret < 0)*

*return ret;*

*buf += ret;*

*cnt -= ret;*

*}*

*return orig\_cnt;*

*}*

*ssize\_t insist\_read(int fd, const void \*buf, size\_t cnt)*

*{*

*ssize\_t ret;*

*size\_t orig\_cnt = cnt;*

*while (cnt > 0) {*

*ret = read(fd, buf, cnt);*

*if (ret < 0)*

*return ret;*

*buf += ret;*

*cnt -= ret;*

*}*

*return orig\_cnt;*

*}*

*#endif /\* \_SOCKET\_COMMON\_H \*/*

**virtio-crupto/guest/crypto.h**

*#ifndef \_CRYPTO\_H*

*#define \_CRYPTO\_H*

*#define VIRTIO\_CRYPTO\_BLOCK\_SIZE 16*

*#define VIRTIO\_CRYPTO\_SYSCALL\_OPEN 0*

*#define VIRTIO\_CRYPTO\_SYSCALL\_CLOSE 1*

*#define VIRTIO\_CRYPTO\_SYSCALL\_IOCTL 2*

*/\* The Virtio ID for virtio crypto ports \*/*

*#define VIRTIO\_ID\_CRYPTO 13*

*/\*\**

*\* Global driver data.*

*\*\*/*

*struct crypto\_driver\_data {*

*/\* The list of the devices we are handling. \*/*

*struct list\_head devs;*

*/\* The minor number that we give to the next device. \*/*

*unsigned int next\_minor;*

*spinlock\_t lock;*

*};*

*extern struct crypto\_driver\_data crdrvdata;*

*/\*\**

*\* Device info.*

*\*\*/*

*struct crypto\_device {*

*/\* Next crypto device in the list, head is in the crdrvdata struct \*/*

*struct list\_head list;*

*/\* The virtio device we are associated with. \*/*

*struct virtio\_device \*vdev;*

*struct virtqueue \*vq;*

*spinlock\_t lock;*

*/\* The minor number of the device. \*/*

*unsigned int minor;*

*};*

*/\*\**

*\* Crypto open file.*

*\*\*/*

*struct crypto\_open\_file {*

*/\* The crypto device this open file is associated with. \*/*

*struct crypto\_device \*crdev;*

*/\* The fd that this device has on the Host. \*/*

*int host\_fd;*

*};*

*#endif*

**virtio-crupto/qemu/hw/char/virtio-crypto.c**

*/\**

*\* Virtio Crypto Device*

*\**

*\* Implementation of virtio-crypto qemu backend device.*

*\**

*\* Dimitris Siakavaras <jimsiak@cslab.ece.ntua.gr>*

*\* Stefanos Gerangelos <sgerag@cslab.ece.ntua.gr>*

*\**

*\*/*

*#include <qemu/iov.h>*

*#include "hw/virtio/virtio-serial.h"*

*#include "hw/virtio/virtio-crypto.h"*

*#include <sys/types.h>*

*#include <sys/stat.h>*

*#include <fcntl.h>*

*#include <sys/ioctl.h>*

*#include <crypto/cryptodev.h>*

*static uint32\_t get\_features(VirtIODevice \*vdev, uint32\_t features)*

*{*

*DEBUG\_IN();*

*return features;*

*}*

*static void get\_config(VirtIODevice \*vdev, uint8\_t \*config\_data)*

*{*

*DEBUG\_IN();*

*}*

*static void set\_config(VirtIODevice \*vdev, const uint8\_t \*config\_data)*

*{*

*DEBUG\_IN();*

*}*

*static void set\_status(VirtIODevice \*vdev, uint8\_t status)*

*{*

*DEBUG\_IN();*

*}*

*static void vser\_reset(VirtIODevice \*vdev)*

*{*

*DEBUG\_IN();*

*}*

*static void vq\_handle\_output(VirtIODevice \*vdev, VirtQueue \*vq)*

*{*

*VirtQueueElement elem;*

*unsigned int \*syscall\_type;*

*int \*cfd, \*host\_return\_val;*

*char ds[100];*

*DEBUG\_IN();*

*if (!virtqueue\_pop(vq, &elem)) {*

*DEBUG("No item to pop from VQ :(");*

*return;*

*}*

*DEBUG("I have got an item from VQ :)");*

*syscall\_type = elem.out\_sg[0].iov\_base;*

*switch (\*syscall\_type) {*

*case VIRTIO\_CRYPTO\_SYSCALL\_TYPE\_OPEN:*

*DEBUG("VIRTIO\_CRYPTO\_SYSCALL\_TYPE\_OPEN");*

*cfd = elem.in\_sg[0].iov\_base;*

*if ( (\*cfd = open("/dev/crypto", O\_RDWR)) < 0) {*

*DEBUG("Unable to open /dev/crypto file.\n");*

*}*

*sprintf(ds,"OPEN: This is the fd i GIVE: %d",\*cfd);*

*DEBUG(ds);*

*break;*

*case VIRTIO\_CRYPTO\_SYSCALL\_TYPE\_CLOSE:*

*DEBUG("VIRTIO\_CRYPTO\_SYSCALL\_TYPE\_CLOSE");*

*cfd = elem.out\_sg[1].iov\_base;*

*close(\*cfd);*

*sprintf(ds,"CLOSE: This is the fd i GET: %d",\*cfd);*

*DEBUG(ds);*

*break;*

*case VIRTIO\_CRYPTO\_SYSCALL\_TYPE\_IOCTL:*

*DEBUG("VIRTIO\_CRYPTO\_SYSCALL\_TYPE\_IOCTL");*

*unsigned int \*cmd;*

*struct session\_op \*sess\_p;*

*struct crypt\_op \*crypt\_p;*

*uint32\_t \*ses\_id;*

*cfd = elem.out\_sg[1].iov\_base;*

*cmd = elem.out\_sg[2].iov\_base;*

*sprintf(ds,"IOCTL: This is the fd i GET: %d",\*cfd);*

*DEBUG(ds);*

*switch(\*cmd) {*

*case CIOCGSESSION:*

*DEBUG("VIRTIO\_CRYPTO\_SYSCALL\_TYPE\_IOCTL\_CIOCGSESSION");*

*sess\_p = elem.in\_sg[0].iov\_base;*

*sess\_p->key = elem.out\_sg[3].iov\_base;*

*host\_return\_val = elem.in\_sg[1].iov\_base;*

*\*host\_return\_val = ioctl(\*cfd, CIOCGSESSION, sess\_p);*

*break;*

*case CIOCFSESSION:*

*DEBUG("VIRTIO\_CRYPTO\_SYSCALL\_TYPE\_IOCTL\_CIOCFSESSION");*

*ses\_id = elem.out\_sg[3].iov\_base;*

*host\_return\_val = elem.in\_sg[0].iov\_base;*

*\*host\_return\_val = ioctl(\*cfd, CIOCFSESSION, ses\_id);*

*break;*

*case CIOCCRYPT:*

*DEBUG("VIRTIO\_CRYPTO\_SYSCALL\_TYPE\_IOCTL\_CIOCCRYPT");*

*crypt\_p = elem.out\_sg[3].iov\_base;*

*crypt\_p->src = elem.out\_sg[4].iov\_base;*

*crypt\_p->iv = elem.out\_sg[5].iov\_base;*

*crypt\_p->dst = malloc (sizeof(unsigned char) \* crypt\_p->len);*

*unsigned char \*input\_msg = elem.in\_sg[0].iov\_base;*

*host\_return\_val = elem.in\_sg[1].iov\_base;*

*\*host\_return\_val = ioctl(\*cfd, CIOCCRYPT, crypt\_p);*

*memcpy(input\_msg, crypt\_p->dst, crypt\_p->len);*

*break;*

*}*

*break;*

*default:*

*DEBUG("Unknown syscall\_type");*

*}*

*virtqueue\_push(vq, &elem, 0);*

*virtio\_notify(vdev, vq);*

*}*

*static void virtio\_crypto\_realize(DeviceState \*dev, Error \*\*errp)*

*{*

*VirtIODevice \*vdev = VIRTIO\_DEVICE(dev);*

*DEBUG\_IN();*

*virtio\_init(vdev, "virtio-crypto", 13, 0);*

*virtio\_add\_queue(vdev, 128, vq\_handle\_output);*

*}*

*static void virtio\_crypto\_unrealize(DeviceState \*dev, Error \*\*errp)*

*{*

*DEBUG\_IN();*

*}*

*static Property virtio\_crypto\_properties[] = {*

*DEFINE\_PROP\_END\_OF\_LIST(),*

*};*

*static void virtio\_crypto\_class\_init(ObjectClass \*klass, void \*data)*

*{*

*DeviceClass \*dc = DEVICE\_CLASS(klass);*

*VirtioDeviceClass \*k = VIRTIO\_DEVICE\_CLASS(klass);*

*DEBUG\_IN();*

*dc->props = virtio\_crypto\_properties;*

*set\_bit(DEVICE\_CATEGORY\_INPUT, dc->categories);*

*k->realize = virtio\_crypto\_realize;*

*k->unrealize = virtio\_crypto\_unrealize;*

*k->get\_features = get\_features;*

*k->get\_config = get\_config;*

*k->set\_config = set\_config;*

*k->set\_status = set\_status;*

*k->reset = vser\_reset;*

*}*

*static const TypeInfo virtio\_crypto\_info = {*

*.name = TYPE\_VIRTIO\_CRYPTO,*

*.parent = TYPE\_VIRTIO\_DEVICE,*

*.instance\_size = sizeof(VirtCrypto),*

*.class\_init = virtio\_crypto\_class\_init,*

*};*

*static void virtio\_crypto\_register\_types(void)*

*{*

*type\_register\_static(&virtio\_crypto\_info);*

*}*

*type\_init(virtio\_crypto\_register\_types)*

**virtio-crupto/guest/crypto-chrdev.c**

*/\**

*\* crypto-chrdev.c*

*\**

*\* Implementation of character devices*

*\* for virtio-crypto device*

*\**

*\* Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>*

*\* Dimitris Siakavaras <jimsiak@cslab.ece.ntua.gr>*

*\* Stefanos Gerangelos <sgerag@cslab.ece.ntua.gr>*

*\**

*\*/*

*#include <linux/cdev.h>*

*#include <linux/poll.h>*

*#include <linux/sched.h>*

*#include <linux/module.h>*

*#include <linux/wait.h>*

*#include <linux/virtio.h>*

*#include <linux/virtio\_config.h>*

*#include "crypto.h"*

*#include "crypto-chrdev.h"*

*#include "debug.h"*

*#include "cryptodev.h"*

*/\**

*\* Global data*

*\*/*

*struct cdev crypto\_chrdev\_cdev;*

*/\*\**

*\* Given the minor number of the inode return the crypto device*

*\* that owns that number.*

*\*\*/*

*static struct crypto\_device \*get\_crypto\_dev\_by\_minor(unsigned int minor)*

*{*

*struct crypto\_device \*crdev;*

*unsigned long flags;*

*debug("Entering");*

*spin\_lock\_irqsave(&crdrvdata.lock, flags);*

*list\_for\_each\_entry(crdev, &crdrvdata.devs, list) {*

*if (crdev->minor == minor)*

*goto out;*

*}*

*crdev = NULL;*

*out:*

*spin\_unlock\_irqrestore(&crdrvdata.lock, flags);*

*debug("Leaving");*

*return crdev;*

*}*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\* Implementation of file operations*

*\* for the Crypto character device*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*static int crypto\_chrdev\_open(struct inode \*inode, struct file \*filp)*

*{*

*int ret = 0;*

*int err;*

*unsigned int len, num\_in = 0, num\_out = 0;*

*struct crypto\_open\_file \*crof;*

*struct crypto\_device \*crdev;*

*unsigned int syscall\_type = VIRTIO\_CRYPTO\_SYSCALL\_OPEN;*

*int host\_fd = -1;*

*struct scatterlist syscall\_type\_sg, host\_fd\_sg, \*sgs[2];*

*debug("Entering");*

*ret = -ENODEV;*

*if ((ret = nonseekable\_open(inode, filp)) < 0)*

*goto fail;*

*/\* Associate this open file with the relevant crypto device. \*/*

*crdev = get\_crypto\_dev\_by\_minor(iminor(inode));*

*spin\_lock\_init(&crdev->lock);*

*if (!crdev) {*

*debug("Could not find crypto device with %u minor",*

*iminor(inode));*

*ret = -ENODEV;*

*goto fail;*

*}*

*crof = kzalloc(sizeof(\*crof), GFP\_KERNEL);*

*if (!crof) {*

*ret = -ENOMEM;*

*goto fail;*

*}*

*crof->crdev = crdev;*

*crof->host\_fd = -1;*

*filp->private\_data = crof;*

*/\*\**

*\* We need two sg lists, one for syscall\_type and one to get the*

*\* file descriptor from the host.*

*\*\*/*

*sg\_init\_one(&syscall\_type\_sg, &syscall\_type, sizeof(syscall\_type));*

*sgs[num\_out++] = &syscall\_type\_sg;*

*sg\_init\_one(&host\_fd\_sg, &host\_fd, sizeof(host\_fd));*

*sgs[num\_out + num\_in++] = &host\_fd\_sg;*

*spin\_lock(&crdev->lock);*

*virtqueue\_add\_sgs(crdev->vq, sgs, num\_out, num\_in, &syscall\_type\_sg , GFP\_ATOMIC);*

*virtqueue\_kick(crdev->vq);*

*/\*\**

*\* Wait for the host to process our data.*

*\*\*/*

*while (virtqueue\_get\_buf(crdev->vq, &len) == NULL)*

*{ ; } /\* do nothing \*/*

*spin\_unlock(&crdev->lock);*

*crof->host\_fd = host\_fd;*

*/\* If host failed to open() return -ENODEV. \*/*

*if(crof->host\_fd < 0) {*

*ret = -ENODEV;*

*}*

*fail:*

*debug("Leaving");*

*return ret;*

*}*

*static int crypto\_chrdev\_release(struct inode \*inode, struct file \*filp)*

*{*

*int ret = 0;*

*struct crypto\_open\_file \*crof = filp->private\_data;*

*struct crypto\_device \*crdev = crof->crdev;*

*unsigned int syscall\_type = VIRTIO\_CRYPTO\_SYSCALL\_CLOSE,*

*num\_in = 0, num\_out = 0,len;*

*int host\_fd = -1;*

*struct scatterlist syscall\_type\_sg, host\_fd\_sg, \*sgs[2];*

*debug("Entering");*

*/\*\**

*\* Send data to the host.*

*\*\*/*

*host\_fd = crof->host\_fd;*

*sg\_init\_one(&syscall\_type\_sg, &syscall\_type, sizeof(syscall\_type));*

*sgs[num\_out++] = &syscall\_type\_sg;*

*sg\_init\_one(&host\_fd\_sg, &host\_fd, sizeof(host\_fd));*

*sgs[num\_out++] = &host\_fd\_sg;*

*spin\_lock(&crdev->lock);*

*virtqueue\_add\_sgs(crdev->vq, sgs, num\_out, num\_in, &syscall\_type\_sg , GFP\_ATOMIC);*

*virtqueue\_kick(crdev->vq);*

*/\*\**

*\* Wait for the host to process our data.*

*\*\*/*

*while (virtqueue\_get\_buf(crdev->vq, &len) == NULL)*

*{ ; } /\* do nothing \*/*

*spin\_unlock(&crdev->lock);*

*kfree(crof);*

*debug("Leaving");*

*return ret;*

*}*

*static long crypto\_chrdev\_ioctl(struct file \*filp, unsigned int cmd, unsigned long arg) {*

*long ret = 0;*

*int err, host\_return\_val,host\_fd,\_cpy;*

*struct crypto\_open\_file \*crof = filp->private\_data;*

*struct crypto\_device \*crdev = crof->crdev;*

*struct virtqueue \*vq = crdev->vq;*

*struct scatterlist syscall\_type\_sg, host\_fd\_sg, cmd\_sg, \*sgs[8];*

*#define MSG\_LEN 64*

*unsigned char output\_msg[MSG\_LEN], input\_msg[MSG\_LEN];*

*unsigned int num\_out, num\_in,*

*syscall\_type = VIRTIO\_CRYPTO\_SYSCALL\_IOCTL,*

*len;*

*debug("Entering");*

*num\_out = 0;*

*num\_in = 0;*

*host\_fd = crof->host\_fd;*

*/\*\**

*\* These are common to all ioctl commands.*

*\*\*/*

*sg\_init\_one(&syscall\_type\_sg, &syscall\_type, sizeof(syscall\_type));*

*sg\_init\_one(&host\_fd\_sg, &host\_fd, sizeof(host\_fd));*

*sg\_init\_one(&cmd\_sg, &cmd, sizeof(cmd));*

*sgs[num\_out++ + num\_in] = &syscall\_type\_sg;*

*sgs[num\_out++ + num\_in] = &host\_fd\_sg;*

*sgs[num\_out++ + num\_in] = &cmd\_sg;*

*/\*\**

*\* Add all the cmd specific sg lists.*

*\*\*/*

*switch (cmd) {*

*struct session\_op sess\_s, \*sess\_p;*

*struct crypt\_op crypt\_s, \*crypt\_p;*

*struct scatterlist sess\_sg, sess\_key\_sg, host\_return\_val\_sg, sess\_id\_sg,*

*crypt\_sg,crypt\_src\_sg,crypt\_iv\_sg,crypt\_dst\_sg;*

*struct scatterlist output\_msg\_sg, input\_msg\_sg;*

*uint32\_t \*ses\_id;*

*int i;*

*char \* dst;*

*case CIOCGSESSION:*

*debug("CIOCGSESSION");*

*sess\_p = (struct session\_op \*) arg;*

*sess\_s = \*sess\_p;*

*sess\_s.key = kzalloc(sess\_s.keylen \* sizeof(unsigned char), GFP\_KERNEL);*

*\_cpy = copy\_from\_user(sess\_s.key, sess\_p->key, sess\_s.keylen \* sizeof(unsigned char));*

*sg\_init\_one(&sess\_sg, &sess\_s, sizeof(sess\_s));*

*sg\_init\_one(&sess\_key\_sg, sess\_s.key, sess\_s.keylen \* sizeof(unsigned char));*

*sg\_init\_one(&host\_return\_val\_sg, &host\_return\_val, sizeof(host\_return\_val));*

*sgs[num\_out++ + num\_in] = &sess\_key\_sg;*

*sgs[num\_out + num\_in++] = &sess\_sg;*

*sgs[num\_out + num\_in++] = &host\_return\_val\_sg;*

*spin\_lock(&crdev->lock);*

*err = virtqueue\_add\_sgs(crdev->vq, sgs, num\_out, num\_in,*

*&syscall\_type\_sg, GFP\_ATOMIC);*

*virtqueue\_kick(crdev->vq);*

*while (virtqueue\_get\_buf(crdev->vq, &len) == NULL)*

*/\* do nothing \*/;*

*spin\_unlock(&crdev->lock);*

*sess\_p->ses = sess\_s.ses;*

*break;*

*case CIOCFSESSION:*

*debug("CIOCFSESSION");*

*ses\_id = (uint32\_t \*) arg;*

*sg\_init\_one(&sess\_id\_sg, ses\_id, sizeof(uint32\_t));*

*sg\_init\_one(&host\_return\_val\_sg, &host\_return\_val, sizeof(host\_return\_val));*

*sgs[num\_out++ + num\_in] = &sess\_id\_sg;*

*sgs[num\_out + num\_in++] = &host\_return\_val\_sg;*

*spin\_lock(&crdev->lock);*

*err = virtqueue\_add\_sgs(crdev->vq, sgs, num\_out, num\_in,*

*&syscall\_type\_sg, GFP\_ATOMIC);*

*virtqueue\_kick(crdev->vq);*

*while (virtqueue\_get\_buf(crdev->vq, &len) == NULL)*

*/\* do nothing \*/;*

*spin\_unlock(&crdev->lock);*

*break;*

*case CIOCCRYPT:*

*debug("CIOCCRYPT");*

*crypt\_p = (struct crypt\_op \*) arg;*

*crypt\_s = \*crypt\_p;*

*crypt\_s.src = kzalloc(crypt\_s.len, GFP\_KERNEL);*

*crypt\_s.dst = kzalloc(crypt\_s.len, GFP\_KERNEL);*

*dst = kzalloc(crypt\_s.len, GFP\_KERNEL);*

*crypt\_s.iv = kzalloc(16, GFP\_KERNEL);*

*\_cpy = copy\_from\_user(crypt\_s.src, crypt\_p->src, crypt\_s.len);*

*\_cpy = copy\_from\_user(crypt\_s.dst, crypt\_p->dst, crypt\_s.len);*

*\_cpy = copy\_from\_user(crypt\_s.iv, crypt\_p->iv, 16);*

*sg\_init\_one(&crypt\_sg, &crypt\_s, sizeof(crypt\_s));*

*sg\_init\_one(&crypt\_src\_sg, crypt\_s.src, crypt\_s.len);*

*sg\_init\_one(&crypt\_iv\_sg, crypt\_s.iv, 16);*

*sg\_init\_one(&crypt\_dst\_sg, dst, crypt\_s.len);*

*sg\_init\_one(&host\_return\_val\_sg, &host\_return\_val, sizeof(host\_return\_val));*

*sgs[num\_out++ + num\_in] = &crypt\_sg;*

*sgs[num\_out++ + num\_in] = &crypt\_src\_sg;*

*sgs[num\_out++ + num\_in] = &crypt\_iv\_sg;*

*sgs[num\_out + num\_in++] = &crypt\_dst\_sg;*

*sgs[num\_out + num\_in++] = &host\_return\_val\_sg;*

*spin\_lock(&crdev->lock);*

*err = virtqueue\_add\_sgs(crdev->vq, sgs, num\_out, num\_in, &syscall\_type\_sg, GFP\_ATOMIC);*

*virtqueue\_kick(crdev->vq);*

*while (virtqueue\_get\_buf(crdev->vq, &len) == NULL)*

*/\* do nothing \*/;*

*spin\_unlock(&crdev->lock);*

*\_cpy = copy\_to\_user(crypt\_p->dst, dst, crypt\_s.len);*

*break;*

*default:*

*debug("Unsupported ioctl command");*

*break;*

*}*

*debug("Leaving");*

*return ret;*

*}*

*static ssize\_t crypto\_chrdev\_read(struct file \*filp, char \_\_user \*usrbuf,*

*size\_t cnt, loff\_t \*f\_pos)*

*{*

*debug("Entering");*

*debug("Leaving");*

*return -EINVAL;*

*}*

*static struct file\_operations crypto\_chrdev\_fops =*

*{*

*.owner = THIS\_MODULE,*

*.open = crypto\_chrdev\_open,*

*.release = crypto\_chrdev\_release,*

*.read = crypto\_chrdev\_read,*

*.unlocked\_ioctl = crypto\_chrdev\_ioctl,*

*};*

*int crypto\_chrdev\_init(void)*

*{*

*int ret;*

*dev\_t dev\_no;*

*unsigned int crypto\_minor\_cnt = CRYPTO\_NR\_DEVICES;*

*debug("Initializing character device...");*

*cdev\_init(&crypto\_chrdev\_cdev, &crypto\_chrdev\_fops);*

*crypto\_chrdev\_cdev.owner = THIS\_MODULE;*

*dev\_no = MKDEV(CRYPTO\_CHRDEV\_MAJOR, 0);*

*ret = register\_chrdev\_region(dev\_no, crypto\_minor\_cnt, "crypto\_devs");*

*if (ret < 0) {*

*debug("failed to register region, ret = %d", ret);*

*goto out;*

*}*

*ret = cdev\_add(&crypto\_chrdev\_cdev, dev\_no, crypto\_minor\_cnt);*

*if (ret < 0) {*

*debug("failed to add character device");*

*goto out\_with\_chrdev\_region;*

*}*

*debug("Completed successfully");*

*return 0;*

*out\_with\_chrdev\_region:*

*unregister\_chrdev\_region(dev\_no, crypto\_minor\_cnt);*

*out:*

*return ret;*

*}*

*void crypto\_chrdev\_destroy(void)*

*{*

*dev\_t dev\_no;*

*unsigned int crypto\_minor\_cnt = CRYPTO\_NR\_DEVICES;*

*debug("entering");*

*dev\_no = MKDEV(CRYPTO\_CHRDEV\_MAJOR, 0);*

*cdev\_del(&crypto\_chrdev\_cdev);*

*unregister\_chrdev\_region(dev\_no, crypto\_minor\_cnt);*

*debug("leaving");*

*}*

**Σημαντική σημείωση:** *Όλα τα παραπάνω αρχεία μπορείτε να τα βρείτε, σε περίπτωση ανακρίβειας, παράλειψης ή άλλης αμφιβολίας, εαν έρθετε σε επικοινωνία μαζί μας:*

* ***timos@email.com***
* ***orestarod@gmail.com***