



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ  
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ  
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ  
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ  
<http://www.cslab.ece.ntua.gr>

## Εργαστήριο Λειτουργικών Συστημάτων 8ο εξάμηνο, Ακαδημαϊκή περίοδος 2017–2018

### Οδηγός εργαστηριακής άσκησης Συσκευή κρυπτογράφησης για QEMU-KVM

Εργαστήριο Υπολογιστικών Συστημάτων Ε.Μ.Π.  
[os-lab@lists.cslab.ece.ntua.gr](mailto:os-lab@lists.cslab.ece.ntua.gr)

Απρίλιος 2018

## Περιεχόμενα

|     |  |    |
|-----|--|----|
| 1   | Γενικά περί εικονικοποίησης (Virtualization)           | 3  |
| 2   | Σκοπός της εργαστηριακής άσκησης                       | 3  |
| 3   | Από που να αρχίσω;                                     | 4  |
| 4   | Προδιαγραφή οδηγού συσκευής virtio-crypto              | 4  |
| 5   | Το πρότυπο virtio                                      | 6  |
| 6   | Προσθήκη εικονικής συσκευής στο QEMU                   | 8  |
| 6.1 | Το μοντέλο συσκευών του QEMU . . . . .                 | 8  |
| 6.2 | Προσθήκη νέας εικονικής συσκευής . . . . .             | 9  |
| 6.3 | Προσθήκη συσκευής virtio-crypto-pci . . . . .          | 10 |
| 7   | Frontend Driver - Πυρήνας Guest                        | 10 |
| 8   | Προτεινόμενη δομή των VirtQueues του virtio-crypto-pci | 11 |

Επιμέλεια: Δ. Σιακαβάρας, Στ. Γεράγγελος, Ευ. Κούκης

## 1 Γενικά περί εικονικοποίησης (Virtualization)

Όπως έχουμε αναφέρει ήδη από τη δεύτερη άσκηση με τον όρο εικονικοποίηση (virtualization) αναφερόμαστε στη δημιουργία ενός εικονικού μηχανήματος, το οποίο συμπεριφέρεται σαν ένα πραγματικό μηχάνημα. Για τους σκοπούς της δεύτερης άσκησης χρησιμοποιήσαμε το QEMU για να προσομοιώσουμε πλήρως την λειτουργία ενός συστήματος. Αυτή η τεχνική εικονικοποίησης ονομάζεται πλήρης εικονικοποίηση (full virtualization). Μία άλλη τεχνική είναι η παραεικονικοποίηση (paravirtualization). Τα βασικότερα χαρακτηριστικά αυτών των δύο τεχνικών αναλύονται παρακάτω:

- **Πλήρης εικονικοποίηση:** Το βασικότερο χαρακτηριστικό αυτής της τεχνικής είναι ότι το λειτουργικό σύστημα μπορεί να τρέξει μέσα στην εικονική μηχανή χωρίς καμία τροποποίηση, ακριβώς όπως τρέχει και στο πραγματικό μηχάνημα. Η εικονική μηχανή δεν ξέρει ότι τρέχει σε εικονικό περιβάλλον και ότι το υλικό της δεν είναι πραγματικό αλλά προσομοιώνεται μέσω κατάλληλου λογισμικού. Το κυριότερο πλεονέκτημα είναι η ευκολία στη δημιουργία εικονικών μηχανών, αλλά το γεγονός ότι τα πάντα προσομοιώνονται μέσω λογισμικού καθιστά αυτήν την τεχνική αργή.
- **Παραεικονικοποίηση:** Σε αυτή την περίπτωση η εικονική μηχανή έχει επίγνωση ότι τρέχει σε εικονικό περιβάλλον και συνεργάζεται με τον host ώστε να επιτευχθεί καλύτερη επίδοση. Γνωρίζει ότι το υλικό της δεν είναι πραγματικό. Για το λόγο αυτό δε μπορεί να χρησιμοποιηθεί ο ίδιος πυρήνας του ΛΣ που εκτελείται στο πραγματικό μηχάνημα. Χρειάζεται να γίνουν κάποιες τροποποιήσεις ώστε ένα ΛΣ να εκτελεστεί σε μία εικονική μηχανή που χρησιμοποιεί παραεικονικοποίηση. Ωστόσο η επίδοση είναι αρκετά καλύτερη σε σχέση με την πλήρη εικονικοποίηση.

## 2 Σκοπός της εργαστηριακής άσκησης

Στο πλαίσιο της παρούσας εργαστηριακής άσκησης καλείστε να δημιουργήσετε μία εικονική συσκευή κρυπτογράφησης για εικονικές μηχανές που εκτελούνται από το QEMU. Η συσκευή αυτή θα εκτελεί ένα υποσύνολο των λειτουργιών που προσφέρει η συσκευή `cryptodev` που χρησιμοποιήσατε για το πρώτο μέρος της άσκησης.

Η πρώτη επιλογή που έχετε για τη δημιουργία αυτής της συσκευής είναι η πλήρης προσομοίωση των αλγορίθμων κρυπτογράφησης σε επίπεδο λογισμικού και η ενσωμάτωση τους σε μία εικονική συσκευή του QEMU. Αυτός ο τρόπος εκτός του ότι θα έχει ως αποτέλεσμα πολύ αργή απόκριση της συσκευής σας, απαιτεί και πολύ

κόπο καθώς θα πρέπει να είστε άριστοι γνώστες των λειτουργιών κρυπτογράφησης. Θα ήταν χρήσιμο να μπορούσαν διεργασίες που εκτελούνται στην εικονική μηχανή να έχουν πρόσβαση στο υλικό του host (για παράδειγμα στην συσκευή `cryptodev`). Χρειαζόμαστε λοιπόν έναν τρόπο για να μιλάει η εικονική μηχανή απευθείας με την πραγματική συσκευή. Αυτός ο τρόπος είναι η χρήση της τεχνικής της παραεικονικοποίησης και του προτύπου `virtio` το οποίο αναλύεται σε επόμενη ενότητα.

### 3 Από που να αρχίσω;

Στη σελίδα του μαθήματος σας δίνεται βοηθητικός κώδικας για να ξεκινήσετε να δουλεύετε τα ζητούμενα της άσκησης. Στο φάκελο `guest` θα βρείτε τον κώδικα από όπου θα ξεκινήσετε για την υλοποίηση του frontend μέρους του οδηγού σας. Η μεταγλώττιση και εγκατάσταση του γίνονται μέσα στην εικονική μηχανή.

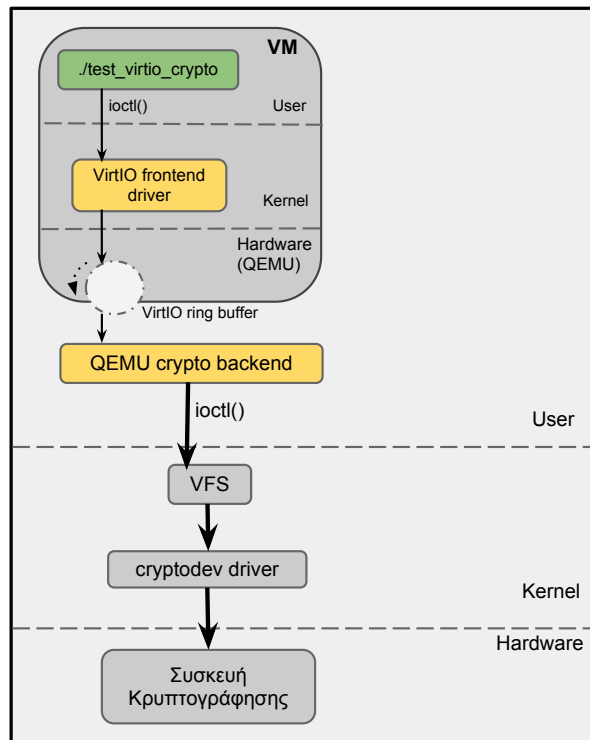
Στον φάκελο `qemu/` βρίσκονται τα αρχεία που αφορούν το backend μέρος του οδηγού σας και τα οποία θα πρέπει να αντιγράψετε στον φάκελο του πηγαίου κώδικα του QEMU. Η μεταγλώττιση τους γίνεται μαζί με την μεταγλώττιση του QEMU και η διαδικασία είναι γνωστή από τη δεύτερη άσκηση.

### 4 Προδιαγραφή οδηγού συσκευής `virtio-crypto`

Ο σκοπός αυτής της εργαστηριακής άσκησης είναι η δημιουργία ενός οδηγού συσκευής ο οποίος θα επιτρέπει τη χρήση της κρυπτογραφικής συσκευής που χρησιμοποιήσατε στο πρώτο μέρος της άσκησης μέσα σε εικονικές μηχανές. Θα μπορούσαμε να υλοποιήσουμε τον αλγόριθμο κρυπτογράφησης σε λογισμικό και να κάνουμε έτσι προσομοίωση της συσκευής, αλλά θέλουμε να εκμεταλλευτούμε την πραγματική συσκευή που υπάρχει στο μηχάνημα ώστε να έχουμε υψηλότερη επίδοση.

Το πρότυπο `virtio`, το οποίο αναλύεται σε επόμενη ενότητα, επιτρέπει την αποδοτική επικοινωνία του ΛΣ που εκτελείται μέσα στην εικονική μηχανή με κώδικα που εκτελείται στον host, μέσω του ορισμού κοινών πόρων για ανταλλαγή δεδομένων. Αυτοί οι κοινοί πόροι ονομάζονται `VirtQueues`.

Στο πρώτο μέρος η κρυπτογράφηση/αποκρυπτογράφηση γινόταν με κατάλληλες κλήσεις `ioctl()` στην συσκευή `/dev/crypto`. Τώρα θα πρέπει οι κλήσεις `ioctl()` να μεταφέρονται από τον guest στον hypervisor μέσω των `virtqueues`, ο hypervisor καλεί την αντίστοιχη `ioctl()` για την πραγματική συσκευή και επιστρέφει το αποτέλεσμα στον guest πάλι μέσω των `virtqueues`. Αυτή η αλληλουχία κλήσεων φαίνεται και στο σχήμα 1



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

Τα ζητούμενα της άσκησης περιλαμβάνουν την υποστήριξη μόνο ενός μέρους από τις λειτουργίες της συσκευής `/dev/crypto`. Συγκεκριμένα, η υλοποίησή σας θα πρέπει να υποστηρίζει τέσσερις κλήσεις `ioctl`, (i) `CIOCGSESSION`, (ii) `CIOCCRYPT` για κρυπτογράφηση και αποκρυπτογράφηση και τέλος (iii) `CIOCFSESSION`.

Από το σχήμα 1 γίνεται προφανής ο διαχωρισμός του οδηγού σε δύο μέρη, backend και frontend. Το πρώτο μέρος εκτελείται στο χώρο χρήστη του host, ενώ το δεύτερο στο χώρο πυρήνα της εικονικής μηχανής. Τα δύο μέρη επικοινωνούν μεταξύ τους μέσω των `VirtQueues`. Εσείς καλείστε να συμπληρώσετε και τα δύο μέρη του οδηγού. Η χρήση της συσκευής virtio αποτελείται από τα παρακάτω βήματα:

1. Ο χρήστης εκκινεί μία εικονική μηχανή με χρήση του userspace προγράμματος QEMU

```
user@host-machine:~/utopia$ ./utopia.sh \
> -device virtio-crypto-pci
```

2. Στον guest έχει αναγνωριστεί η συσκευή PCI και μπορούμε να την δούμε με

την εντολή `lsrpci`, η έξοδος της οποίας περιλαμβάνει μία γραμμή παρόμοια με `00:05.0 Communication controller: Red Hat, Inc Device 1013`

3. Με το script `crypto_dev_nodes.sh` δημιουργούνται τα αρχεία που θα χρησιμοποιηθούν ως συσκευές κρυπτογράφησης/αποκρυπτογράφησης.
4. Η μεταγλώττιση και εισαγωγή του `module` που δημιουργήσατε θα πρέπει να επιτρέπει στο χρήστη την χρήση αυτών των αρχείων συσκευών χαρακτήρων σαν να είναι φυσικές κρυπτογραφικές συσκευές.

## 5 Το πρότυπο virtio

Η τεχνική της παραεικονικοποίησης όπως αναφέρθηκε στην προηγούμενη ενότητα απαιτεί τροποποιήσεις στον πυρήνα του λειτουργικού συστήματος που θα χρησιμοποιηθεί. Συγκεκριμένα, απαιτούνται αλλαγές στους οδηγούς των συσκευών ώστε να επικοινωνούν απευθείας με τον `host`. Αντίστοιχα, η πλατφόρμα εικονικοποίησης, δηλαδή το λογισμικό που εκτελείται στον `host` για να προσομοιώσει τη λειτουργία της εικονική μηχανής <sup>1</sup>, πρέπει να υλοποιεί το κατάλληλο εικονικό υλικό ώστε να μπορεί να δέχεται και να απαντάει σε αιτήματα από τον `guest`. Υπάρχουν πολλές πλατφόρμες εικονικοποίησης όπως το QEMU, και κάθε μία χρησιμοποιεί διαφορετικούς `paravirtualized` οδηγούς συσκευών. Το γεγονός αυτό επιβάλλει και διαφορετικούς οδηγούς στον πυρήνα του λειτουργικού που θα εκτελεστεί στην εικονική μηχανή. Η ανάγκη για ένα κοινό πρότυπο `paravirtualized` οδηγών συσκευών οδήγησε στο `virtio`[1].

Το `virtio` ορίζει ένα αφαιρετικό επίπεδο μέσω του οποίου ο `backend` οδηγός επικοινωνεί με τον `frontend` μέσω της ανταλλαγής κατάλληλων `buffer` δεδομένων. Πιο συγκεκριμένα, στο [1] ορίζεται η δομή `Virtqueue` η οποία προδιαγράφει μία ουρά στην οποία ο `frontend` οδηγός τοποθετεί `buffers` για επεξεργασία, τους οποίους λαμβάνει ο `backend` οδηγός και όταν τελειώσει την επεξεργασία τους ενημερώνει το `frontend`.

Οι λειτουργίες που πρέπει να υποστηρίζει κάθε υλοποίηση `Virtqueue` από τη μεριά του `frontend` οδηγού είναι:

- Προσθήκη `buffer` στην ουρά.
- Ενημέρωση του `backend` οδηγού για νέο `buffer`.
- Παραλαβή επεξεργασμένων(από το `backend`) `buffer` από την ουρά.

---

<sup>1</sup> Από εδώ και στο εξής θα αναφερόμαστε σε αυτό ως `hypervisor`.

- Ενεργοποίηση/απενεργοποίηση των ενημερώσεων για παραλαβή απάντησης από τον backend οδηγό.

Το virtio\_ring αποτελεί ένα πρωτόκολλο μεταφοράς για το πρότυπο virtio, η υλοποίηση του οποίου υπάρχει ενσωματωμένη στον πυρήνα του linux, αλλά και στον κώδικα του QEMU. Οι λεπτομέρειες υλοποίησης αναλύονται στο [1]. Παραδείγματα έτοιμων virtio οδηγών για συσκευές μπλοκ και δικτύου που χρησιμοποιούν το virtio\_ring αποτελούν οι virtio\_blk και virtio\_net. Επίσης, υπάρχει ο οδηγός virtio\_console για τη δημιουργία σειριακών θυρών για μεταφορά δεδομένων μεταξύ του host και των εικονικών μηχανών. Τέλος, ο οδηγός virtio\_balloon επιτρέπει την δυναμική αυξομείωση της μνήμης που χρησιμοποιείται από κάποια εικονική μηχανή. Οι παραπάνω οδηγοί έχουν ενσωματωθεί στον κώδικα του πυρήνα του Linux οπότε μπορείτε να τους εξερευνήσετε μέσω του γνωστού εργαλείου LXR.

Η διεπαφή που προσφέρεται από το virtio\_ring στον πυρήνα του linux και θα χρησιμοποιήσετε είναι οι παρακάτω συναρτήσεις:

- `int virtqueue_add_sgs(struct virtqueue *vq, struct scatterlist *sgs[], unsigned int out_sgs, unsigned int in_sgs, void *data, gfp_t gfp);`  
Προσθέτει δεδομένα στην VirtQueue χρησιμοποιώντας λίστες scatter-gather οι οποίες χρησιμοποιούνται στον πυρήνα του linux για DMA μεταφορές δεδομένων από συσκευές I/O.
- `void virtqueue_kick(struct virtqueue *vq);`  
Ενημερώνει τον Host για την προσθήκη δεδομένων στην VirtQueue.
- `void *virtqueue_get_buf(struct virtqueue *vq, unsigned int *len);`  
;   
Επιστρέφει έναν απομονωτή από την VirtQueue. Επιστρέφει μόνο απομονωτές τους οποίους έχει επεξεργασθεί προηγουμένως το backend μέρος του οδηγού.

Η υλοποίηση των παραπάνω συναρτήσεων, είναι διαθέσιμη στο αρχείο `drivers/virtio/virtio_ring.c` στον πηγαίο κώδικα του linux.

Όσον αφορά το QEMU και το backend μέρος του οδηγού σας η διεπαφή του virtio\_ring αποτελείται από τις παρακάτω συναρτήσεις:

- `int virtqueue_pop(VirtQueue *vq, VirtQueueElement *elem);`  
Λήψη δεδομένων από την VirtQueue.
- `void virtqueue_push(VirtQueue *vq, const VirtQueueElement *elem, unsigned int len);`  
;   
Προσθήκη ενός στοιχείου στην VirtQueue.

- `void virtio_notify(VirtIODevice *vdev, VirtQueue *vq);`  
Ενημέρωση του frontend μέρους, μέσω `interrupt`.

Οι παραπάνω συναρτήσεις και ό,τι αφορά την υλοποίηση του `virtio_ring` από την μεριά του QEMU είναι διαθέσιμο στο αρχείο `hw/virtio/virtio.c` του πηγαίου κώδικα του QEMU.

## 6 Προσθήκη εικονικής συσκευής στο QEMU

Σε αυτήν την ενότητα θα δούμε πως μπορούμε να δημιουργήσουμε μία νέα εικονική συσκευή από την πλευρά του QEMU. Επίσης θα δούμε πως μπορείτε να χρησιμοποιήσετε τον έτοιμο κώδικα στο QEMU για την προσθήκη της δικής σας `virtio` συσκευής η οποία θα φαίνεται στην εικονική μηχανή σαν μία συσκευή PCI.

### 6.1 Το μοντέλο συσκευών του QEMU

Όπως έχουμε αναφέρει, το υλικό των εικονικών μηχανών που τρέχουν μέσω του QEMU δεν είναι πραγματικό αλλά προσομοιώνεται μέσω λογισμικού. Το QEMU έχει ένα σύνολο από προκαθορισμένες συσκευές οι οποίες προσαρτώνται σε κάθε εικονική μηχανή που εκτελείται. Επιπλέον συσκευές μπορούν να προστεθούν μέσω ορισμάτων στην γραμμή εντολών. Πιο συγκεκριμένα με το όρισμα `-device`. Για παράδειγμα η παρακάτω εντολή

```
user@host-machine:~/utopia$ <qemu executable and other options>
> -chardev socket,id=sensors0,host=cerberus.cs.lab.ece.ntua.gr,port=49152,ipv4
> -device isa-serial,chardev=sensors0,id=serial0
```

θα προσθέσει στην εικονική μηχανή μία σειριακή θύρα (`/dev/ttyS0`) η οποία θα είναι συνδεδεμένη μέσω `socket` με την θύρα 49152 του `cerberus`. Αυτός είναι ο τρόπος με τον οποίο βλέπατε τους αισθητήρες στην εικονική μηχανή σας στη δεύτερη εργαστηριακή άσκηση.

Ο κώδικας του QEMU παρόλο που έχει γραφτεί σε C ακολουθεί αντικειμενοστραφή λογική, όπως και ο κώδικας του πυρήνα του Linux. Κάθε συσκευή αποτελείται από δύο δομές, η μία προσδιορίζει στοιχεία σχετικά με την κλάση της ενώ η άλλη στοιχεία που αφορούν την κατάσταση ενός αντικειμένου σε μία συγκεκριμένη φάση εκτέλεσης της εικονικής μηχανής.

Η ιεραρχία που ακολουθείται στο μοντέλο συσκευών του QEMU φαίνεται στο σχήμα 2. Μαζί με το όνομα του τύπου της συσκευής σε κάθε τετράγωνο αναφέρονται οι δύο δομές με τις οποίες αναπαρίσταται ο τύπος στον κώδικα του QEMU καθώς και



το αρχείο στο οποίο ορίζονται. Ο γενικότερος τύπος αντικειμένου είναι ο `TYPE_OBJECT` και ο γενικότερος τύπος συσκευής ο `TYPE_DEVICE`. Όλες οι υπόλοιπες συσκευές είναι υποκλάσεις της. Ο τύπος `TYPE_PCI_DEVICE` αντιπροσωπεύει όλες τις PCI συσκευές.

Εξίσου σημαντικό κομμάτι του μοντέλου συσκευών είναι οι δίαυλοι (buses). Η δεντρική δομή που ακολουθείται όσον αφορά τη δημιουργία των δίαυλων φαίνεται στο σχήμα 3. Κάθε συσκευή συνδέεται σε κάποιον δίαυλο του συστήματος, για παράδειγμα όλες οι PCI συσκευές συνδέονται σε έναν δίαυλο τύπου `TYPE_PCI_BUS`.

## 6.2 Προσθήκη νέας εικονικής συσκευής

Κάθε νέα συσκευή που δημιουργείται δηλώνεται μέσω της μακροεντολής `type_init` η οποία ορίζεται στο `<module.h>`. Η μακροεντολή παίρνει σαν όρισμα έναν δείκτη σε συνάρτηση `void (*fn)(void)`, μέσα στην οποία καλείται η `type_register_static`. Η τελευταία παίρνει σαν όρισμα μία δομή τύπου `TypeInfo` (`<include/qom/object.h>`) που περιλαμβάνει τα εξής πεδία:

- `char *name;`  
Το όνομα του νέου τύπου. Αντιστοιχεί στο όρισμα `-device` που δίνεται στο QEMU για την προσθήκη της συσκευής στην εικονική μηχανή.
- `char *parent;`  
Ο πρόγονος αυτού του τύπου. Ο νέος τύπος κληρονομεί όλα τα πεδία του προγόνου του. Όπως αναφέρθηκε προηγουμένως όλες οι συσκευές είναι απόγονοι του τύπου `TYPE_DEVICE`.
- `size_t instance_size;`  
Το μέγεθος της δομής με την οποία αναπαρίσταται η κατάσταση μίας συσκευής του συγκεκριμένου τύπου.
- `void (*class_init)(ObjectClass *klass, void *data);`  
Η συνάρτηση που καλείται κατά την αρχικοποίηση του τύπου συσκευής. Όταν κληθεί αυτή η συνάρτηση από το QEMU, όλοι οι τύποι συσκευών που είναι πρόγονοι του συγκεκριμένου τύπου έχουν αρχικοποιηθεί, οπότε εδώ μπορούμε να παρακάμψουμε (override) όποιες συναρτήσεις θέλουμε.

## 6.3 Προσθήκη συσκευής virtio-crypto-pci

Στο αρχείο `hw/virtio/virtio-pci.c` ορίζονται οι συσκευές `virtio-net-pci`, `virtio-blk-pci`, `virtio-serial-pci`, `virtio-balloon-pci` και `virtio-serial-pci`. Όλες αυτές

οι συσκευές είναι συσκευές PCI, δηλαδή είναι συσκευές που συνδέονται σε ένα δίαυλο PCI. Την ίδια διαδικασία χρησιμοποιούμε για να ορίσουμε τη νέα συσκευή `virtio-crypto-pci`. Συγκεκριμένα, παρέχεται η δομή `VirtIOPCIProxy` η οποία αναλαμβάνει να συνδέσει μία συσκευή `virtio` με τον δίαυλο PCI του συστήματος.

Η δομή `VirtCrypto` που ορίζεται στο αρχείο `include/hw/virtio/virtio-crypto.h` του βοηθητικού κώδικα είναι η δομή που αναπαριστά τη συσκευή σας από τη μεριά του QEMU. Η αρχικοποίηση της γίνεται στη συνάρτηση `virtio_crypto_realize`. Με τη συνάρτηση `virtio_init` δημιουργείται μία `virtio` συσκευή. Η συνάρτηση `virtio_add_queue(struct VirtIODevice *vdev, int queue_size, void (*handle_output)(VirtIODevice *, VirtQueue *));` που ορίζεται στο `include/hw/virtio/virtio.h` δημιουργεί μία `virtqueue` και την προσθέτει στη λίστα με τις `virtqueues` της συσκευής `vdev`. Η συνάρτηση `vq_handle_output` είναι η συνάρτηση που καλείται όταν ο frontend οδηγός έχει προσθέσει ένα buffer στην ουρά.

## 7 Frontend Driver - Πυρήνας Guest

Από τη δεύτερη άσκηση έχετε αποκτήσει μία εμπειρία σχετικά με την προσθήκη οδηγών συσκευών στον πυρήνα του Linux. Η διαδικασία που πρέπει να ακολουθήσετε για την προσθήκη του frontend οδηγού δε διαφέρει πολύ από τα βήματα που εκτελέσατε για την προσθήκη του οδηγού για τους αισθητήρες της δεύτερης εργαστηριακής άσκησης.

Για την εισαγωγή ενός `virtio` οδηγού στον πυρήνα του Linux απαιτείται μία κλήση της συνάρτησης `register_virtio_driver(struct virtio_driver *)`. Η δομή `virtio_driver` περιέχει τα εξής πεδία που πρέπει να αρχικοποιηθούν:

- `unsigned int feature_table[];`  
Σε αυτόν τον πίνακα ορίζονται διάφοροι παράμετροι που καθορίζουν λειτουργίες που υποστηρίζει ο οδηγός. Αντίστοιχος πίνακας πρέπει να ορίζεται και στον backend οδηγό ώστε να υπάρχει συγχρονισμός ως προς τα είδη των λειτουργιών που παρέχονται από μία συσκευή. Η συσκευή σας δεν έχει ανάγκη από κάποιο χαρακτηριστικό οπότε ο πίνακας αυτός δε θα περιέχει τίποτα.
- `struct virtio_device_id id_table[];`  
Εδώ καθορίζουμε τις συσκευές τις οποίες υποστηρίζει ο οδηγός μας. Ο πίνακας αποτελείται από ζεύγη αναγνωριστικών συσκευών και αναγνωριστικών κατασκευαστών. Οι αριθμοί αυτοί πρέπει να είναι σε αντιστοιχία με τα αναγνωριστικά των συσκευών που δίνουμε από την μεριά του QEMU.

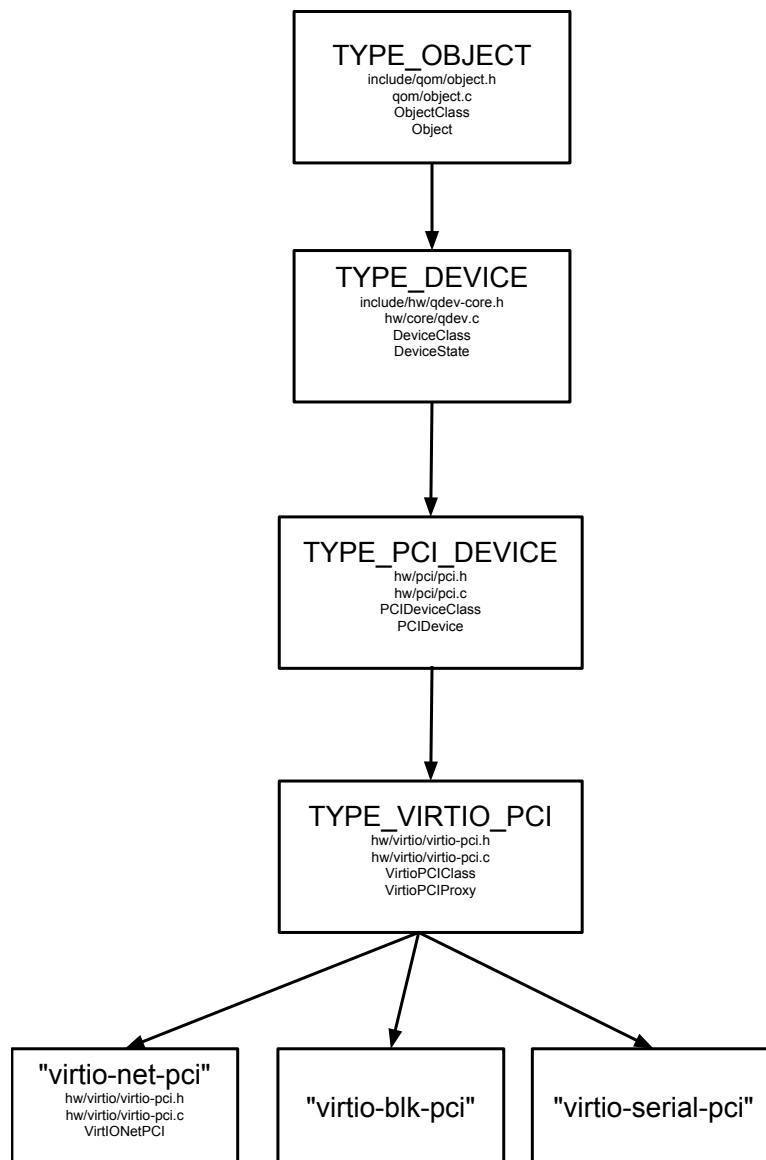
- `int (*probe)(struct virtio_device *)`;  
Αυτή είναι η συνάρτηση που θα κληθεί από τον πυρήνα του Linux μόλις εντοπιστεί μία συσκευή της οποίας το αναγνωριστικό υπάρχει στον πίνακα `id_table`. Μέσα στη συνάρτηση αυτή θα πρέπει να αρχικοποιήσουμε όλα τα απαραίτητα πεδία για τη συσκευή μας. Για παράδειγμα πρέπει να βρείτε τα `virtqueues` μέσω των οποίων θα επικοινωνήσετε με τον backend οδηγό.
- `void (*remove)(struct virtio_device *)`;  
Η συνάρτηση αυτή καλείται όταν η συσκευή αποσυνδέεται από το σύστημα.

## 8 Προτεινόμενη δομή των VirtQueues του virtio-crypto-pci

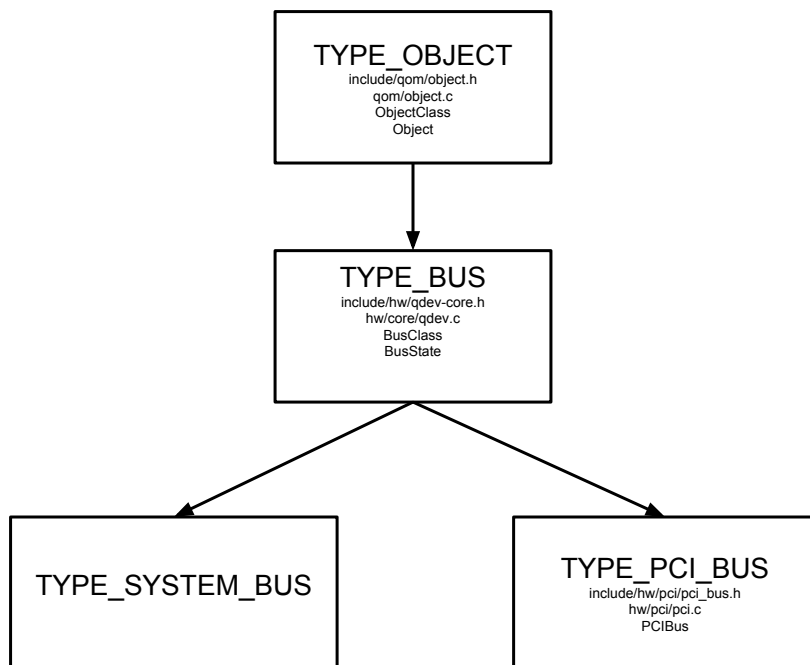
Στο σχήμα 4 φαίνεται η δομή που σας προτείνουμε για την Virtqueue της συσκευής `virtio-crypto-pci` για κάθε κλήση συστήματος που θα υλοποιεί ο οδηγός σας.

### Αναφορές

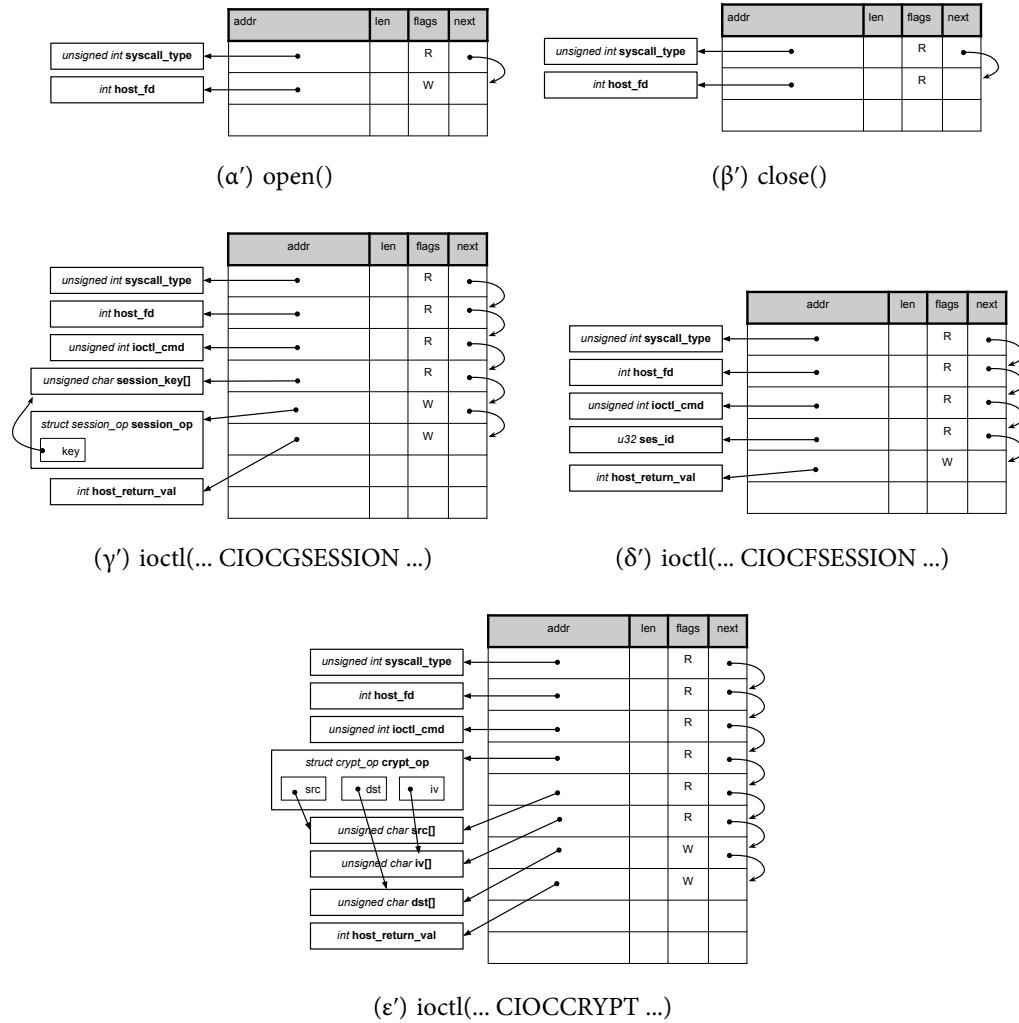
- [1] Rusty Russell, *virtio: towards a de-facto standard for virtual I/O devices*. SIGOPS Oper. Syst. Rev., 2008.
- [2] *Linux Device Drivers*, Jonathan Corbet, Alessandro Rubin, Greg Kroah-Hartman, O'Reilly Media, 3rd Edition, 2005, <http://lwn.net/Kernel/LDD3/>



Σχήμα 2: Η ιεραρχία τύπων συσκευών στο μοντέλο του QEMU.



Σχήμα 3:



Σχήμα 4: Η προτεινόμενη δομή της Virtqueue που χρησιμοποιεί η συσκευή virtio-crypto-pci για κάθε κλήση συστήματος.