Εργαστήριο Λειτουργικών Συστημάτων
Άσκηση 3 (Z1,Z2,Z3)
Ντούνης Πέτρος (03115091)
Διαμάντη Ιωάννα (03115035)

Ζητούμενο 1 : Εργαλείο chat πάνω από TCP/IP sockets.

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 Πρωτόκολλο:
→ Αξιόπιστη (ACK), αμφίδρομη μεταφορά δεδομένων.
→ Δεν διατηρεί πληροφορία για όρια του μηνύματος
→ Connection-oriented επικοινωνία (πρέπει πρώτα να έχει προηγηθεί σύνδεση)
→ PF_INET (domain) + SOCK_STREAM (type) + 0 (protocol) = TCP
→ Διαχωρισμός συνδέσεων με: {Διεύθυνση, Πόρτα πηγής}, {Διεύθυνση, Πόρτα προορισμού}.
→ Στα linux υλοποιείται από τον πυρήνα.

Client Socket:
→ int socket(int domain, int type, int protocol): δημιουργεί το socket , που θα χρησιμοποιηθεί ως κανάλι επικοινωνίας και επιστρέφει τον file descriptor.
→ int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen): 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(int fd): κλείνει το ανοιχτό αρχείο στο οποίο δείχνει ο file descriptor, ο οποίος τώρα μπορεί να επαναχρησιμοποιηθεί.

Server Socket:
→ int socket(int domain, int type, int protocol): δημιουργεί το socket , που θα χρησιμοποιηθεί ως κανάλι επικοινωνίας και επιστρέφει τον file descriptor.
→ int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen): 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 που προσδιορίζεται από το όρισμα sockfd, με μία τη διεύθυνση που δίνεται στο όρισμα *addr.
→ int listen(int sockfd, int backlog): χαρακτηρίζει το socket ως παθητικό, δηλαδή το καθιστά κατάλληλο να δεχτεί συνδέσεις. Το μέγιστο μήκος της ουράς με τις εν αναμονή εισερχόμενες συνδέσεις του socket προσδιορίζεται από το όρισμα backlog. 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).
→ int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen): 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 του sockfd, 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.

→ read() / write(): από/στον νέο file descriptor

→ close(int newsd): κλείνει το νέο socket descriptor που δημιούργησε η accept().

Χρήση της select() σε server,client:
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. Γι αυτό κάναμε χρήση της select() , η οποία διαχειρίζεται κάποια set από file descriptors. Στην περίπτωση μας έχουμε 2 set από file descriptors, το read set και το write set, δηλαδή τους file descriptors από τους οποίους διαβάζουμε και αυτούς στους οποίους γράφουμε. Κάθε file descriptor έχει ένα bit ετοιμότητας (0 ή 1) με βάση το οποίο μαθαίνουμε αν είναι έτοιμος (1). Μέσω των μακροεντολών FD_ZERO(),FD_SET() αρχικοποιούμε στο 0 τα bit όλων των file descriptors και των 2 set (FD_ZERO()), προσθέτουμε στα set τους αντίστοιχους file descriptors(FD_SET() - στην δική μας περίπτωση οι 2 file descriptors είναι και για read() και για write() οπότε μπαίνουν και οι 2 και στα 2 set). Κάνοντας κλήση της select() παρακολουθούμε πρακτικά τα bit ετοιμότητας των file descriptors των 2 set, και μόλις κάποιο από αυτά γίνει 1 (δηλαδή ο file descriptor είναι έτοιμος για διάβασμα ή ανάγνωση) η select() ξεμπλοκάρει. Στη συνέχεια με χρήση της μακροεντολής FD_ISSET() ελέγχουμε για κάθε file descriptor που των 2 set αν έχει bit 1. Αν αυτό αληθεύει τότε ο fd είναι έτοιμος για χρήση και κάνουμε την επιθυμητή κλήση συστήματος (read() ή write() αναλόγως) χωρίς να υπάρχει κίνδυνος να μπλοκάρουμε σε αυτή, ενώ κάποιος άλλος fd μπορεί να είναι ήδη έτοιμος. Έτσι πρακτικά καταφέραμε να μην πρέπει ο ένας χρήστης να περιμένει την απάντηση του άλλου για να στείλει κι άλλο μήνυμα.

Ζητούμενο 2 : Κρυπτογραφημένο chat πάνω από TCP/IP.

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 στο μέλος ses της δομής cryp και χρησιμοποιείται στις κλήσεις ioctl() (απο)κρυπτογράφησης. Επίσης για το session έχουμε ορίσει αυθαίρετα τιμές για τα μέλη key, iv , οι οποίες συμφωνούν βέβαια ανάμεσα σε server και client.
CIOCCRYPT: Η κλήση αυτή ζητά από τη συσκευή κρυπτογράφησης να (απο)κρυπτογραφήσει δεδομένα.
CIOCFSESSION: Η κλήση αυτή τερματίζει ένα session με τη συσκευή όταν πλέον δεν θέλουμε να τη χρησιμοποιήσουμε άλλο (όταν κάποιος από τους server-client αποσυνδέεται).

Ζητούμενο 3 : Υλοποίηση συσκευής cryptodev με VirtIO:

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.

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):
Η συνάρτηση αυτή εκτελείται από το ΛΣ του guest κάθε φορά που κάποια διεργασία ανοίγει (system call open()) ένα ειδικό αρχείο της εικονικής συσκευής χαρακτήρων virtio-cryptodev (π.χ. /dev/cryptodev0). Συγκεκριμένα δημιουργεί ένα ανοιχτό αρχείο (struct file) και ένα αντικείμενο τύπου struct crypto_open_file. Στο αντικείμενο αυτό, μέσω του minor number του ειδικού αρχείου, βρίσκουμε και αποθηκεύουμε την εικονική κρυπτογραφική συσκευή με την οποία συνδέεται το ανοιχτό αρχείο. Κατόπιν κλειδώνουμε το spinlock της συσκευής και δημιουργούμε 2 scatter-gather λίστες ,μία για να διαβάσει ο host τον τύπο του system call που θέλει να κάνει ο guest-εδώ open- και μία για να επιστρέψει ο host (στον guest) τον file descriptor που θα επιστραφεί από το open() που θα πραγματοποιήσει ο ίδιος. Μέσω της συνάρτησης virtqueue_add_sgs() προσθέτουμε τις δύο αυτές λίστες στην virtqueue , ενώ μέσω της virtqueue_kick() ειδοποιούμε τον host για την προσθήκη δεδομένων στην VirtQueue. Μέχρι ο host να επεξεργαστεί τα δεδομένα, ο guest κάνει busy wait με συνεχείς κλήσεις της συνάρτησης virtqueue_get_buf(). Μόλις αυτή σταματήσει να επιστρέφει NULL, ο host έχει επιστρέψει τα επεξεργασμένα δεδομένα πίσω στον guest πάλι μέσω της VirtQueue και ξεκλειδώνεται το spinlock της συσκευής, ώστε κάποια άλλη εν αναμονή διεργασία να μπορέσει να προσθέσει δεδομένα στην VirtQueue της ίδιας συσκευής χωρίς αυτά να προστίθενται παράλληλα με άλλες διεργασίες, οδηγώντας σε τυχαίο και απροσδιόριστο αποτέλεσμα. Έτσι (εφόσον το open() έγινε επιτυχώς) ο file descriptor στον guest έχει τώρα τιμή που αφορά το ανοιχτό αρχείο της πραγματικής συσκευής στον host. Ο file descriptor αποθηκεύεται στο μέλος host_fd της δομής crypto_open_file, η οποία αποθηκεύεται στα private_data του ανοιχτού αρχείου ώστε να χρησιμοποι- ηθεί για επόμενες κλήσεις συστήματος. Σε επιτυχία η συνάρτηση αυτή επιστρέφει 0 σαν file descriptor στον guest, χωρίς αυτό να επηρεάζει εφόσον ο file descriptor που χρησιμοποιείται πάντα από τους drivers του ΛΣ είναι ο host_fd.

static int crypto_chrdev_release(struct inode *inode, struct file *filp):
Η συνάρτηση αυτή καλείται από το ΛΣ του guest κάθε φορά που μία διεργασία κλείνει (system call close()) ένα ανοιχτό αρχείο της εικονικής συσκευής χαρακτήρων virtio-cryptodev. Συγκεκριμένα από τα private_data του ανοιχτού αρχείου ανακτά τη δομή crypto_open_file και συνεπώς την εικονική συσκευή με την οποία συνδέεται το ανοιχτό αρχείο,καθώς και τον πραγματικό file descriptor που έχει στο host μηχάνημα. Κλειδώνουμε το spinlock της συσκευής και δημιουργούμε τώρα 3 scatter-gather λίστες, μία για να διαβάσει ο host τον τύπο του system call που θέλει να κάνει ο guest-εδώ close()-, μία για να διαβάσει ο host τον file descriptor του ανοιχτού αρχείου που θέλουμε να κλείσουμε, καθώς και μία για να γράψει ο host την τιμή επιστροφής της κλήσης συστήματος close() και να ενημερώσει τον guest για την επιτυχία/αποτυχία της. Η διαδικασία γίνεται ακριβώς με τον ίδιο τρόπο που περιγράφηκε στην crypto_chrdev_open() και μόλις ο host επιστρέψει τα επεξεργασμένα

δεδομένα, ξεκλειδώνουμε το spinlock της συσκευής και ο guest διαβάζει την τιμή επιστροφής της close() του Host και (σε περίπτωση επιτυχίας) απελευθερώνει (kfree()) το χώρο που είχε δεσμεύσει για την δομή crypto_open_file.

static long crypto_chrdev_ioctl(struct file *filp, unsigned int cmd,unsigned long arg):
Η συνάρτηση αυτή εκτελείται από το ΛΣ του guest κάθε φορά που μία διεργασία κάνει μία κλήση συστήματος ioctl() σε κάποιο ανοιχτό αρχείο της εικονικής συσκευής χαρακτήρων (filp). Το όρισμα cmd προσδιορίζει το είδος της ioctl() κλήσης που επιθυμεί να κάνει η διεργασία, ενώ το όρισμα arg, είναι η διεύθυνση (στο χώρο χρήστη του guest) του session, του identifier ενός session ή ενός αντικειμένου τύπου crypt_op (αναλόγως το cmd) που χρησιμοποιεί η διεργασία για την κρυπτογράφηση. Για τις τρεις ioctl() κλήσεις συστήματος έχουμε 4 κοινά δεδομένα που πρέπει να περάσουμε στον host. Έτσι οι πρώτες 4 scatter-gather λίστες της VirtQueue είναι κάθε φορά ίδιες και περιέχουν : α) το είδος της κλήσης συστήματος που θέλει να κάνει ο guest - εδώ 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.
CIOCGSESSION: Στην περίπτωση που η διεργασία θέλει να ξεκινήσει ένα session με την συσκευή κρυπτογράφησης,το όρισμα arg περιέχει τη διεύθυνση ενός αντικειμένου τύπου session_op, το οποίο χρησιμοποιείται για την έναρξη του session. Η διεύθυνση αυτή όμως βρίσκεται σε χώρο χρήστη. Έτσι για να μπορέσουμε να έχουμε πρόσβαση στις τιμές του αντικειμένου από το χώρο πυρήνα δημιουργούμε ένα νέο αντικείμενο ίδιου τύπου και χρησιμοποιούμε την συνάρτηση copy_from_user(), μέσω της οποίας αντιγράφουμε τις τιμές του arg σε αυτό. Από τα δεδομένα αυτά πρέπει να μεταφερθεί στον host το όλο το αντιγραμμένο αντικείμενο, ώστε να μπορεί να αρχικοποιηθεί κατάλληλα το session. Από το αντικείμενο αυτό ο host χρειάζεται να διαβάσει το (ήδη αρχικοποιημένο) session key (κλειδί κρυπτογράφησης), καθώς και να αρχικοποιήσει (να γράψει δηλαδή) ορισμένα πεδία του (π.χ. session id). Καθώς μία scatter-gather λίστα μπορεί να οριστεί είτε για διάβασμα είτε για εγγραφή (όχι και για τα 2), εμφανίζεται η ανάγκη να δημιουργήσουμε μία νέα μεταβλητή sess_key ώστε να περάσουμε το κλειδί ξεχωριστά με μία scatter-gather λίστα ανάγνωσης , ενώ το υπόλοιπο αντικείμενο δίνεται στον host μέσω άλλης λίστας για εγγραφή.
CIOCFSESSION:
Στη περίπτωση που η διεργασία θέλει να τερματίσει ένα session με τη συσκευή κρυπτογράφησης, το όρισμα arg περιέχει τη διεύθυνση του identifier του. Για τους ίδιους λόγους που εξηγήσαμε παραπάνω , χρησιμοποιούμε την copy_from_user() για να αντιγράψουμε την τιμή του στη μεταβλητή που δημιουργήσαμε στο χώρο πυρήνα. Στη συνέχεια την προσθέτουμε στη VirtQueue μέσω μίας scatter-gather λίστας με δικαίωμα ανάγνωσης από τον host, ώστε να γνωρίζει ποιο session επιθυμούμε να τερματίσουμε.
CIOCCRYPT:
Στη περίπτωση που η διεργασία θέλει να κάνει κρυπτογράφηση/αποκρυπτογράφηση με τη συσκευή, το όρισμα arg περιέχει τη διεύθυνση σε χώρο χρήστη guest ενός αντικειμένου τύπου crypt_op, το οποίο ανάμεσα σε άλλα πεδία περιέχει και το αρχικό μήνυμα (src), το initialization vector (iv), καθώς και ένα πεδίο για την αποθήκευση του (απο)κρυπτογραφημένου μηνύματος (dst). Χρησιμοποιούμε την συνάρτηση copy_from_user() για να αντιγράψουμε την τιμή των πεδίων του, στα πεδία του νέου αντικειμένου cryp ίδιου τύπου που δημιουργούμε στο χώρο πυρήνα. Εδώ υπάρχει μια ιδιαιτερότητα καθώς κάποια από τα πεδία του (src,dst,iv) είναι δείκτες με συνέπεια η τιμή τους να είναι και αυτή διεύθυνση στο χώρο χρήστη του guest. Για το λόγο αυτό δημιουργούμε νέες μεταβλητές ξεχωριστά για τα πεδία αυτά στο χώρο πυρήνα και χρησιμοποιούμε την copy_from_user() για να έχουμε πρόσβαση στην τιμή τους. Στο host μηχάνημα χρειάζεται φυσικά να γίνει γνωστό το αρχικό μήνυμα(src) καθώς και το iv για να γίνει η (απο)κρυπτογράφηση. Επίσης προκειμένου να επιστραφεί στο guest το επεξεργασμένο μήνυμα πρέπει ο host να έχει δικαίωμα εγγραφής στη νέα μεταβλητή που δημιουργήσαμε ως αντίστοιχη της dst σε χώρο πυρήνα. Δημιουργούμε λοιπόν 4 scatter-gather λίστες, μια για ανάγνωση των υπόλοιπων πεδίων του αντικειμένου cryp, μία για ανάγνωση των αντίστοιχων src, iv που

δημιουργήσαμε και μία καθαρά για εγγραφή του αποτελέσματος στη αντίστοιχη μεταβλητή dst και επιστροφή του στον guest.

Μόλις γίνει κάποιο από τα παραπάνω (αναλόγως cmd) ,κλειδώνουμε το spinlock της συσκευής χαρακτήρων και προσθέτουμε μέσω της virtqueue_add_sgs() τις λίστες στην VirtQueue της. Μέσω της virtqueue_kick() ενημερώνουμε τον host για προσθήκη νέου στοιχείου στην ουρά, ενώ με συνεχείς κλήσεις της virtqueue_get_buf() κάνουμε busy wait , ώστε να περιμένουμε μέχρι ο host να επιστρέψει τον buffer με τα επεξεργασμένα δεδομένα στον χώρο πυρήνα του guest. Μόλις αυτό συμβεί, ξεκλειδώνουμε το spilock της συσκευής απελευθερώνοντας την για χρήση από άλλες διεργασίες και ελέγχουμε την τιμή επιστροφής retval. Σε περίπτωση αποτυχίας επιστρέφουμε -EIO σφάλμα στην διεργασία και επιστρέφουμε σε χώρο χρήστη guest. Διαφορετικά ανάλογα με το είδος ioctl() που κάναμε (τιμή cmd) κάνουμε χρήση της συνάρτησης copy_to_user() ώστε η τιμή των επεξεργασμένων από τον host μεταβλητών, που δημιουργήθηκαν και υπάρχουν σε χώρο πυρήνα, να αντιγραφεί στα αντίστοιχα πεδία του ορίσματος arg. Τέλος απελευθερώνουμε τον χώρο που δεσμεύσαμε στον χώρο πυρήνα για την αντιμετώπιση των παραπάνω δυσκολιών και επιστρέφουμε σε χώρο χρήστη.

Backend (χώρος χρήστη host μηχανήματος):

static void vq_handle_output(VirtIODevice *vdev, VirtQueue *vq):
Η συνάρτηση αυτή καλείται κάθε φορά που ο frontend οδηγός προσθέτει έναν buffer στην VirtQueue της εικονικής συσκευής χαρακτήρων. Αρχικά μέσω της συνάρτησης virtqueue_pop() παίρνει το αντικείμενο από την ουρά, το οποίο είναι τύπου VirtQueueElement και διαθέτει (ανάμεσα σε άλλα που δεν θα χρησιμοποιήσουμε) 2 πεδία με όνομα in_sg, out_sg, τα οποία είναι πίνακες και κάθε θέση τους αντιστοιχεί μία scatter-gather λίστα με τη σειρά που προστέθηκε στην ουρά από το ΛΣ του guest. Ο πίνακας in_sg αντιστοιχεί στις λίστες που προορίζονται για εγγραφή ,ενώ ο out_sg αντιστοιχεί σε αυτές που προορίζονται για ανάγνωση. Σε κάθε θέση τους περιέχουν αντικείμενα τύπου struct iovec. Το πεδίο iov_base της δομής αυτής χρησιμοποιείται για να ανακτήσουμε την διεύθυνση του αντικειμένου, με το οποίο είχε αρχικοποιηθεί η αντίστοιχη scatter-gather λίστα. Σε κάθε περίπτωση το πρώτο στοιχείο που προσθέταμε από τον guest στις λίστες ήταν ο τύπος του system call που θέλαμε να γίνει. Έτσι το πρώτο στοιχείο της out_sg, δηλαδή το out_sg[0] περιέχει τον τύπο αυτό. Μόλις ανακτήσουμε αυτή την πληροφορία στον host, ελέγχουμε το είδος της κλήσης συστήματος και πράττουμε αναλόγως.

VIRTIO_CRYPTODEV_SYSCALL_TYPE_OPEN: Στην περίπτωση αυτή η διεργασία που εκτελείται στην εικονική μηχανή επιθυμεί να ανοίξει ένα ειδικό αρχείο της εικονικής συσκευής. Από την υλοποίηση του driver στο frontend γνωρίζουμε ότι στην περίπτωση αυτή, στην πρώτη θέση του πίνακα in_sg βρίσκεται η διεύθυνση του file descriptor που θα πρέπει να χρησιμοποιηθεί για το άνοιγμα του ειδικού αρχείου και να επιστραφεί στον guest. Αναθέτουμε λοιπόν την διεύθυνση του σε έναν νέο δείκτη έστω fd και κατόπιν κάνουμε κλήση συστήματος open() στο ειδικό αρχείο /dev/crypto, το οποίο έχει δημιουργηθεί με την εισαγωγή του module για την συσκευή κρυπτογράφησης cryptodev στο host μηχάνημα. 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, καθώς βρίσκεται στην ίδια ακριβώς θέση μνήμης με τον fd.

VIRTIO_CRYPTODEV_SYSCALL_TYPE_CLOSE: Στην περίπτωση αυτή η διεργασία που εκτελείται στην εικονική μηχανή επιθυμεί να κλείσει ένα ανοιχτό αρχείο της εικονικής συσκευής. Από την υλοποίηση του driver στο frontend γνωρίζουμε ότι στην περίπτωση αυτή, στην δεύτερη θέση του πίνακα out_sg βρίσκεται η διεύθυνση του file descriptor που θα πρέπει να χρησιμοποιηθεί για το κλείσιμο του ανοιχτού αρχείου. Γνωρίζουμε επίσης ότι στην πρώτη θέση του πίνακα in_sg βρίσκεται η διεύθυνση της μεταβλητής επιστροφής, μέσω της οποίας θα ενημερωθεί ο guest για την επιτυχία/αποτυχία της κλήσης συστήματος στο host μηχάνημα. Αναθέτουμε τις δύο αυτές διευθύνσεις σε δύο μεταβλητές έστω fd,ret αντίστοιχα και εκτελούμε την

κλήση συστήματος *ret = close(*fd). Σε περίπτωση επιτυχίας η μεταβλητή ret έχει μηδενική τιμή, την οποία λαμβάνει ο guest και ενημερώνεται για την επιτυχία της κλήσης.

VIRTIO_CRYPTODEV_SYSCALL_TYPE_IOCTL: Στην περίπτωση αυτή η διεργασία που εκτελείται στην εικονική μηχανή επιθυμεί να κάνει μία κλήση συστήματος ioctl() στην εικονική συσκευή. Από την υλοποίηση του driver στο frontend γνωρίζουμε ότι στην περίπτωση αυτή, στην δεύτερη θέση του πίνακα out_sg βρίσκεται η διεύθυνση του file descriptor που αναφέρεται στο αντίστοιχο ανοιχτό αρχείο της συσκευής cryptodev. Γνωρίζουμε επίσης ότι στην τρίτη θέση του πίνακα out_sg βρίσκεται η διεύθυνση της μεταβλητής που περιγράφει τον τύπο της ioctl() κλήσης που επιθυμούμε να κάνουμε. Με βάση την τιμή της μεταβλητής αυτής διακρίνουμε τις εξής περιπτώσεις:
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 (sess) και της μεταβλητής επιστροφής ret. Εκτελούμε την κλήση συστήματος με την κλήση συστήματος *ret = ioctl(*fd,CIOCGSESSION,sess), 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 την κλήση συστήματος *ret = ioctl(*fd,CIOCFSESSION,ses), 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. Για λόγους που αναφέρθηκαν και εξηγήθηκαν παραπάνω, στην VirtQueue έχουν προστεθεί ξεχωριστά οι διευθύνσεις των διάφορων μεταβλητών. Έτσι δημιουργούμε ένα νέο αντικείμενο cryp1 ,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 της κλήσης συστήματος *ret = ioctl(*fd,CIOCCRYPT,cryp1) εκτελείται η ενέργεια που περιγράφει το πεδίο op της του αντικειμένου cryp1 (κρυπτογράφηση/αποκρυπτογράφηση) στα δεδομένα εισόδου cryp1->src και το αποτέλεσμα αποθηκεύεται στο πεδίο cryp1->dst και συνεπώς στη μεταβλητή dst που είχαμε εξ αρχής προσθέσει στην ουρά (εφόσον μοιράζονται την ίδια θέση μνήμης).

Τέλος μόλις ολοκληρωθούν οι παραπάνω ενέργειες, προσθέτουμε τα επεξεργασμένα στοιχεία στην ουρά μέσω της συνάρτησης virtqueue_push(vq, elem, 0) και ειδοποιείται ο guest για την ύπαρξη νέων δεδομένων μέσω της συνάρτησης virtio_notify(vdev, vq). Η συνάρτηση αυτή τερματίζει και η εκτέλεση συνεχίζεται από το frontend όπως περιγράφηκε στο παραπάνω.