Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εργαστήριο Λειτουργικών Συστημάτων – 2η Άσκηση

Των Προπτυχιακών Φοιτητών
Αρώνη Παναγιώτη Α.Μ. 03109083
Δανασή Παναγιώτη Α.Μ. 03109004
e-mail: el09004@mail.ntua.gr
panaro32@hotmail.com

1. Μέρος Πρώτο – BSD Sockets και Cryptodev module

Στο πρώτο μέρος της άσκησης αυτής είχαμε να υλοποιήσουμε ένα chat όπου τα μηνύματα που ανταλλάσσονταν ήταν κρυπτογραφημένα. Η υλοποίηση έγινε στην μορφή του client-server. Βασιστήκαμε στο προσχέδιο του έτοιμου κώδικα που δόθηκε για την δημιουργία socket και των εργασιών που εκτελούν ο client και ο server για την σύνδεσή τους. Η διαφορά όμως στην υλοποίηση του chat ήταν ότι και ο client και ο server είχαν δύο ροές από τις οποίες μπορούσαν να διαβάσουν δεδομένα (το STDIN του καθενός και τον fd που αντιπροσωπεύει το socket), όπου σε κάθε περίπτωση, έπρεπε να ξέρουμε που υπάρχουν δεδομένα και να κάνουμε αντίστοιχα, είτε αποστολή στον άλλο υπολογιστή, είτε τύπωμα στο output μας. Για να το πετύχουμε αυτό χρησιμοποιήσαμε την select που μας επιτρέπει να παρακολουθούμε πολλαπλούς file descriptors, περιμένοντας κάποιος από αυτούς να είναι έτοιμος για την λειτουργία I/O που τον θέλουμε.
Ο κώδικας που αφορά αυτό το κομμάτι της υλοποίησης είναι ο ακόλουθος:
(ο κώδικας αυτός αφορά τον client, αλλά ακριβώς αντίστοιχος είναι και για τον server)

```
struct timeval timeout;
timeout.tv_sec  = sec;
timeout.tv_usec = msec;

fd_set set;
FD_ZERO(&set);
FD_SET(sd, &set);
FD_SET(STDIN_FILENO, &set);

/* Which file descriptor is ready? */
int returned = select(sd + 1, &set, NULL, NULL, &timeout);

if ( returned > 0 ){
    if (FD_ISSET(sd, &set)) {
        /* If server has send us data. */
        /* Decrypt and type*/
    }
    if (FD_ISSET(STDIN_FILENO, &set)) {
        /* If user has data to send. */
        /* Encrypt and send*/
    }
}
else if ( returned < 0 ) {
    perror("select failed!");
    exit(1);
}
```

Το δεύτερο πράγμα που χρειάστηκε να προσθέσουμε, ήταν η κρυπτογράφηση και αποκρυπτογράφηση των δεδομένων (στα σημεία που δείχνει ο παραπάνω κώδικας). Η υλοποίηση και εδώ βασίστηκε στον έτοιμο κώδικα που μας δόθηκε και δεν ήταν κάτι παραπάνω παρά κλήσεις συστήματος της μορφής ioctl με τα κατάλληλα ορίσματα, ανάλογα την λειτουργία που επιτελούσαμε.

Στην υλοποίηση μας, παρατηρούμε ότι από την στιγμή που θα εγκατασταθεί η σύνδεση, το κομμάτι για την αποστολή και λήψη μηνυμάτων στον client και στον server είναι ακριβώς ίδια, οπότε μία σωστότερη υλοποίηση

(αντί του copy paste που κάναμε εμείς) θα ήταν να μπει αυτό το κομμάτι σε κάποια ξεχωριστή βιβλιοθήκη, την οποία θα καλούσαμε, για ευκολότερη συντήρηση, διόρθωση λαθών και δόμηση κώδικα.

2. Μέρος Δεύτερο – Συσκευή Κρυπτογράφησης για QEMU-KVM

Στο δεύτερο μέρος της άσκησης αφορούσε την υλοποίηση μιας εικονικής συσκευής για το Qemu καθώς και του αντίστοιχου οδηγού που θα επέτρεπε την χρήση αυτής της κρυπτογραφικής συσκευής από το εικονικό μηχάνημα. Για να γίνει αυτό, χρησιμοποιήθηκε το πρότυπο VirtIO που δίνει την δυνατότητα στον guest να επικοινωνεί με τον host μέσω προκαθορισμένων δομών (VirtQueues) και να εξυπηρετεί τα αιτήματα του.
Ο κώδικας που χρειάστηκε να συμπληρώσουμε αφορούσε και το κομμάτι που έτρεχε στην πραγματική μηχανή, αλλά και τον driver για το εικονικό μηχάνημα.
Για το πραγματικό μηχάνημα, η άσκηση αφορούσε την υλοποίηση της κρυπτογραφικής συσκευής για το qemu. Στο προσχέδιο του κώδικα που μας δόθηκε συμπληρώσαμε την συνάρτηση send_buffer για την αποστολή δεδομένων στον guest, την handle_control_message που ουσιαστικά, όταν του το ζητήσει ο guest, ανοίγει και κλείνει το αρχείο "/dev/crypto" για την πραγματική κρυπτογράφηση και αποκρυπτογράφηση δεδομένων, την συνάρτηση crypto_handle_ioctl_packet που εκτελεί τις πραγματικές λειτουργίες ioctl για την κρυπτογράφηση και αποκρυπτογράφηση και τν συνάρτηση handle_output που παίρνει τα δεδομένα που μας στέλνει ο guest και καλεί την crypto_handle_ioctl_packet.
Ο κώδικας στις συγκεκριμένες συναρτήσεις αφορούσε κυρίως την παραλαβή κάποιου buffer από την ουρά ή τοποθέτηση του και ειδοποίηση του guest. Η διαφορά έγκειται μόνο στην crypto_handle_ioctl_packet, όπου ουσιαστικά γεμίζαμε την δομή struct crypt_op cryp, με τα δεδομένα που μας έχει στείλει ο guest και εκτελούσαμε την κατάλληλη ioctl κλήση, επιστρέφοντας τα αποτελέσματα στον χρήστη.

Το κομμάτι του guest αφορούσε τα εξής:

- Προσθήκη κατάλληλων locks στην δομή crypto_device για την προστασία κάθε φορά που κάνουμε access σε κοινά δεδομένα.
- Το κομμάτι προσθήκης δεδομένων στην virtqueue και ειδοποίησης του host (αντίστοιχα με αποπάνω).
- Και τέλος την υλοποίηση του crypto_ioctl που επιτελεί την παραλαβή δεδομένων από το userspace (copy_from_user), το κατάλληλο γέμισμα του buffer που θα στείλουμε (cr_data) στον host και την παραλαβή απάντησης και πέρασμα του αποτελέσματος στον χώρο χρήστη (copy_to_user).

Το μόνο κομμάτι που αξίζει να σημειώσουμε είναι τα σημεία που χρειάστηκαν τα locks. Συγκεκριμένα, έχουμε ένα σημαφόρο (crdev->sem) για τα κλειδώματα σε process context και ένα spinlock (crdev->lock) για interrupt context. Το κλείδωμα στον σημαφόρο γίνεται στην crypto_chrdev_open καθότι μόνο μία διεργασία μπορεί να ανοίξει μία συγκεκριμένη συσκευή κάθε φορά, και για λόγους συμμετρίας, βάλαμε ένα κλείδωμα και στην

crypto_chrdev_release, παρότι δεν χρειάζεται καθώς όπως είπαμε και πριν μόνο μία διεργασία μπορεί να ανοίξει μία συγκεκριμένη συσκευή κάθε φορά.

Το spinlock κλειδώνει στην in_intr, όταν ο host μας στέλνει δεδομένα και θέλουμε να πάρουμε τον buffer καθώς και στην device_has_data όπου ελέγχουμε αν έχουμε δεδομένα.