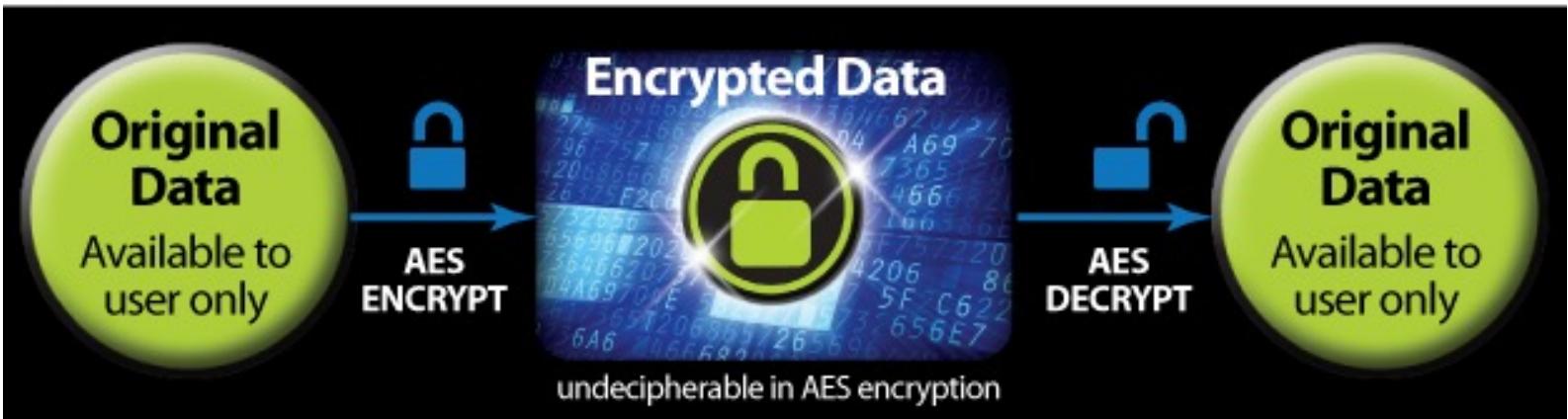


3η Εργαστηριακή Ασκηση Κρυπτογραφική συσκευή VirtIO για QEMU-KVM



Εργαστήριο Λειτουργικών Συστημάτων

Ροη : Y

Ομάδα : b04

Ον/μο : Βαβουλιώτης Γεώργιος

A.M. : 03112083

Εξάμηνο : 8

Εισαγωγή - Σκοπός Άσκησης : Αντικείμενο της παρούσας άσκησης είναι η ανάπτυξη εικονικού υλικού στο περιβάλλον εικονικοποίησης QEMU-KVM. Στη πράξη θα πρέπει να υλοποιήσω μια εικονική συσκευή κρυπτογράφισης VirtIO, η οποία πλέον θα είναι μέρος του QEMU. Ωστόσο η άσκηση αυτή έχει χωρίζεται σε 3 μέρη. Στο πρώτο μέρος καλούμαστε να υλοποιήσουμε ενα εργαλείο chat πάνω από TCP/IP sockets, το οποίο θα επιτρέπει αμφίδρομη επικοινωνία πάνω από TCP/IP με χρήση του BSD Sockets API χωρίς κρυπτογράφηση. Στο δέυτερο μέρος ουσιαστικά πρέπει να επεκτείνουμε το πρώτο μέρος, στέλνοντας πλέον κρυπτογραφημένα μηνύματα μέσω του cryptodev userspace API (/dev/crypto). Στο τρίτο μέρος της άσκησης ζητούμενο είναι η υλοποίηση εικονικής συσκευής cryptodev κάνοντας χρήση του πλαισίου VirtIO ώστε το εργαλείο που υλοποίησα στα προηγούμενα ζητήματα να μπορεί να εκτελείται μέσα σε VM κάνοντας χρήση κρυπτογραφικών επιταχυντών σε υλικό, οι οποίοι ως τώρα ήταν προσβάσιμοι μόνο από τον host.

Ζητούμενα 1 και 2

Θα πρέπει να επισημάνω ότι το πρώτο και το δεύτερο μέρος της άσκησης είναι ουσιαστικά ενα μέρος, το οποίο έχει ως ζητούμενο την υλοποίηση ενος εργαλείου chat πάνω από TCP/IP sockets, το οποίο θα επιτρέπει αμφίδρομη επικοινωνία πάνω από TCP/IP, αλλά τα data τα οποία μεταφέρονται θα είναι κρυπτογραφημένα με προσυμφωνημένο κλειδί, χρήσιμοποιώντας του cryptodev-linux API. Για το λόγο αυτό επέλεξα να σας παραθέσω τον κώδικα που υλοποιεί και τα δυο ζητούμενα και οχι ενα ξεχωριστό κώδικα για κάθε ερώτημα, αφού η μοναδική τους διαφορά είναι η κρυπτογράφηση και αποκρυπτογράφηση. Οι κώδικες **socket-server.c** και **socket-client.c** φαίνονται παρακάτω:

```

/*
* socket-client.c
* Simple TCP/IP communication using sockets
*
* Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
* Vavouliotis Giorgos <nuovocominzio@hotmail.com>
* AM : 03112083
*/

```

```

#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/ioctl.h>
#include <sys/stat.h>

#include <arpa/inet.h>
#include <netinet/in.h>

#include <crypto/cryptodev.h>

#include "socket-common.h"

/* Insist until read all the data */
ssize_t insist_read(int fd, void *buf, size_t cnt)
{
    ssize_t ret;
    size_t orig_cnt = cnt;

    while (cnt > 0) {
        ret = read(fd, buf, cnt);
        if (ret == 0) {
            printf("Server went away. Exiting...\n");
            return 0;
        }
        if (ret < 0) {
            perror("read from server failed");
            return ret;
        }
        buf += ret;
        cnt -= ret;
    }

    return orig_cnt;
}

```

```

/* 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;
}

int main(int argc, char *argv[]){
    char server_msg[256], keyboard[256], temp[256];
    char key[KEY_SIZE+1], iv[BLOCK_SIZE+1];
    char * hostname = argv[1];
    int port, sd, fdnew;
    struct hostent *hp;
    struct sockaddr_in sa;
    struct session_op sess;
    struct crypt_op cryp;
    fd_set set1, temp_set;

    /* Sets the first num bytes of the block of memory pointed by ptr to the
       specified value (interpreted as an unsigned char). */
    memset(&sess, 0, sizeof(sess));
    memset(&cryp, 0, sizeof(cryp));

    /* check if i take the right args */
    if (argc != 3) {
        fprintf(stderr, "Usage: %s hostname port\n", argv[0]);
        exit(1);
    }

    /*
     * create a socket. socket() returns as a file descriptor named sd
     * the correct thing to do is to use AF_INET in your struct sockaddr_in and
     * PF_INET in your call to socket().
     * But practically speaking, you can use AF_INET everywhere
    */
    if ((sd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket");
        exit(-1);
    }

    /* get host info ---> hostname is localhost */
    if ( !(hp = gethostbyname(hostname)) ) {
        fprintf(stderr, "DNS failed lookup for host: %s\n", hostname);
        exit(-1);
    }

    /* all the clients that i will create the will have the name localhost */
    printf("I am the client with name %s and now i am starting.\n", hp->h_name);

    /* atoi converts a string into a integer. If no valid conversion could be
       performed, it returns zero*/
    port = atoi(argv[2]);
    if ( port <= 0 ){
        printf("I have a problem with atoi. I am gonna quit!\n");
        exit(1);
    }
}

```

```

sa.sin_family = AF_INET;

/* The htons() function converts the unsigned short integer port from host
   byte order to network byte order.*/
sa.sin_port = htons(port);
/* copy to the sa.sin_addr.s_addr the hp->h_addr */
memcpy(&sa.sin_addr.s_addr, hp->h_addr, sizeof(struct in_addr));

/* Initializes the file descriptor set with name set1 to have zero bits for
   all file descriptors */
FD_ZERO(&set1);

/* connect() sundeei ena socket me fd=sd me thn dieu8unsh pou uparxei san
   2o orisma-> an exw tcp/ip tote egka8ista mia nea sundesh apo ton client*/
if (connect(sd, (struct sockaddr *) &sa, sizeof(sa)) < 0) {
    perror("connect");
    exit(-1);
}

fdnew = open("/dev/crypto", O_RDWR);
if (fdnew < 0) {
    perror("open(/dev/crypto)");
    return 1;
}

// if you want info about this open the cryptodev.h
strcpy(iv, IV);
strcpy(key, KEY);
sess.cipher = CRYPTO_AES_CBC;
sess.keylen = KEY_SIZE;
sess.key = key;

/* h ioctl() elenxei ola ta I/O operations mias suskeuhs kai ta ektelei
   analoga me to ti 8a dwsoume san orisma . Analoga me to request code pou
   dinw(2o orisma) kanei to katallhlo I/O operation.
*/
if (ioctl(fdnew, CIOCGSESSION, &sess)) {
    perror("ioctl(CIOCGSESSION)");
    return 1;
}
cryp.ses = sess.ses;
cryp.iv = iv;

/* Sets the bit for the file descriptor sd and 0 in the file descriptor set
   set1 */
FD_SET(0, &set1);
FD_SET(sd, &set1);

/* Now i am ready to read and write to fd 0 and sd */
for (;;) {
    /* this loop ends when the client gives koukis\n */

    /*
    select() and pselect() allow a program to monitor multiple file
    descriptors, waiting until one or more of the file descriptors become
    "ready" for some class of I/O operation
    */

    temp_set = set1; // fd_set variable
    /* dialogw apo thn domh fd_set kapoio to opoio einai etoimo */
    select(FD_SETSIZE, &temp_set, NULL, NULL, NULL);
    if (FD_ISSET(0, &temp_set)) {
        /* get the message from the stdin and put it to the variable
           keyboard */
        fgets(keyboard, MSG_SIZE, stdin);

```

```

/* use AES for encryption */
cryp.src = keyboard;
cryp.dst = temp;
cryp.len = sizeof(keyboard);
cryp.op = COP_ENCRYPT;
if (ioctl(fdnew, CIOCCRYPT, &cryp)) {
    perror("ioctl(CIOCCRYPT)");
    return 1;
}
/* write the message to the socket */
insist_write(sd, temp, sizeof(temp));
if (strcmp(keyboard, "koukis\n") == 0) {
    printf("Koukis killed the client(OMG)\n");
    close(sd); // close the socket
    exit(1);
}
}

if (FD_ISSET(sd, &temp_set)) {
    /* the server send a message and i have to read it */
    if (insist_read(sd, server_msg, MSG_SIZE) != MSG_SIZE) {
        perror("read");
        exit(EXIT_FAILURE);
    }
    /* time for decryption */
    cryp.src = server_msg;
    cryp.dst = temp;
    cryp.len = sizeof(server_msg);
    cryp.op = COP_DECRYPT;
    if (ioctl(fdnew, CIOCCRYPT, &cryp)) {
        perror("ioctl(CIOCCRYPT)");
        return 1;
    }
    printf("Server sent:%s", temp);
    if (strcmp(temp, "halt\n") == 0) {
        printf("Server killed the client because sent 'halt'\n");
        close(sd); // close the socket
        exit(1);
    }
}
}

if (ioctl(fdnew, CIOCFSESSION, &sess.ses)) {
    perror("ioctl(CIOCFSESSION)");
    return 1;
}
if (close(fdnew) < 0) {
    perror("close(fd)");
    return 1;
}
if (close(sd) < 0) {
    perror("close(fd)");
    return 1;
}
return 0;
}

```

```

/*
 * socket-server.c
 * Simple TCP/IP communication using sockets
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 * Vavouliotis Giorgos <nuovocominzio@hotmail.com>
 * AM : 03112083
 */

#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/ioctl.h>
#include <sys/stat.h>

#include <arpa/inet.h>
#include <netinet/in.h>

#include <crypto/cryptodev.h>

#include "socket-common.h"

#define all 100
#define minus -100

/* Insist until read all the data */
ssize_t insist_read(int fd, void *buf, size_t cnt)
{
    ssize_t ret;
    size_t orig_cnt = cnt;

    while (cnt > 0) {
        ret = read(fd, buf, cnt);
        if (ret == 0) {
            printf("Server went away. Exiting...\n");
            return 0;
        }
        if (ret < 0) {
            perror("read from server failed");
            return ret;
        }
        buf += ret;
        cnt -= ret;
    }

    return orig_cnt;
}

```

```

/* 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;
}

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

    char message[256], msg1[256], key[KEY_SIZE+1];
    char iv[BLOCK_SIZE+1], temp[256], keyboard[256], buff[256];
    int sd, fd, sd2, index, ClientArr[MAX_CLIENTS][2], clnum, i, myselect, fdnew, flag;
    struct sockaddr_in sa;
    struct session_op sess;
    struct crypt_op cryp;
    fd_set set1, temp_set;

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

    /*
     * create a socket. socket() returns as a file descriptor named sd
     * the correct thing to do is to use AF_INET in your struct sockaddr_in and
     * PF_INET in your call to socket().
     * But practically speaking, you can use AF_INET everywhere.
    */
    if ((sd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket");
        exit(EXIT_FAILURE);
    }

    memset(&sa, 0, sizeof(sa));
    sa.sin_family = AF_INET;

    /* The htons() function converts the unsigned short integer port from host
     * byte order to network byte order.*/
    sa.sin_port = htons(TCP_PORT);
    sa.sin_addr.s_addr = htonl(INADDR_ANY);

    /* Bind to a well-known port */
    if (bind(sd, (struct sockaddr *)&sa, sizeof(sa)) < 0) {
        perror("bind");
        exit(EXIT_FAILURE);
    }
    /* Listen for incoming connections */
    if (listen(sd, TCP_BACKLOG) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    fdnew = open("/dev/crypto", O_RDWR);
    if (fdnew < 0) {
        perror("open(/dev/cryptodev0)");
        return 1;
    }
}

```

```

// if you want info about this open the cryptodev.h
strcpy(iv, IV);
strcpy(key, KEY);
sess.cipher = CRYPTO_AES_CBC;
sess.keylen = KEY_SIZE;
sess.key = key;
if (ioctl(fdnew, CIOCGSESSION, &sess)) {
    perror("ioctl(CIOCGSESSION)");
    return 1;
}
cryp.ses = sess.ses;
cryp.iv = iv;

/* Initializes the file descriptor set fdset to have zero bits for all file
   descriptors */
FD_ZERO(&set1);
/* Sets the bit for the file descriptor sd and 0 in the file descriptor set
   set1 */
FD_SET(sd, &set1);
FD_SET(0, &set1);
clnum = 0;

for(;;){
    /* this loop ends when the server gives -100\n */

    /* arxikopoio to myselect se mia megalh sxetika timh h opoia den
       antistoxiei se kapoio j etsi wste na parw swsto apotelesma otan
       grapsw se ena client kai meta paw na grapsw kati sto server.
    */
    myselect = 55555;
    flag = 0;
    /*
    select() and pselect() allow a program to monitor multiple file
    descriptors, waiting until one or more of the file descriptors become
    "ready" for some class of I/O operation.
    */
    temp_set = set1;
    /* dialegw apo thn domh fd_set kapoio to opoio einai etoimo */
    select(FD_SETSIZE, &temp_set, NULL, NULL, NULL);
    for(fd=0; fd<FD_SETSIZE; fd++){
        if (FD_ISSET(fd, &temp_set)){
            if (fd == 0){
                /* server writes in stdin */
                scanf("%d",&myselect); // shows what the server have to do
                if (myselect != minus){
                    /* if i dont take -100 i should take the server's
                       message */
                    fgets(keyboard, MSG_SIZE, stdin);
                    /* use AES for encryption - i have my encrypted message
                       in temp */
                    cryp.src = keyboard;
                    cryp.dst = temp;
                    cryp.len = sizeof(keyboard);
                    cryp.op = COP_ENCRYPT;
                    if (ioctl(fdnew, CIOCCRYPT, &cryp)) {
                        perror("ioctl(CIOCCRYPT)");
                        return 1;
                    }
                }
            }
            if (myselect == minus){
                /* if server writes -100 the server is going down
                   and i send halt to clients manually */
                for (i=0; i<clnum; i++){
                    /* when server is going down i quit all the
                       clnum because i want it. */
                    cryp.src = "halt\n";

```

```

        cryp.dst = temp;
        cryp.len = sizeof(keyboard);
        cryp.op = COP_ENCRYPT;
        if (ioctl(fdnew, CIOCCRYPT, &cryp)) {
            perror("ioctl(CIOCCRYPT)");
            return 1;
        }
        insist_write(ClientArr[i][0], temp, sizeof(temp));
        close(ClientArr[i]);
    }
    printf("Server is going down and all clients are
           dead also\n");
    close(sd);
    exit(0);
}
else if (myselect == all){
    /* if server writes 100, have to write the message
       in all the clients */
    for (i = 0; i<clnum; i++)
        insist_write(ClientArr[i][0], temp, sizeof(temp));
    break;
}
else{
    if (myselect == 55555) {
        printf("I don't have client with this fd.\n");
        flag = 1 ;
    }
    else{
        printf("Server wants to send the message to
               client with fd = %d.\n", myselect);
        for (i=0; i<clnum; i++){
            /* search the client that server want
               to send the msg*/
            if (ClientArr[i][0] == myselect){
                // i find the right client
                insist_write(ClientArr[i][0],
                            temp, sizeof(temp));
                break;
            }
        }
        if (i == clnum && flag == 0){
            // i did not find the client that server wants.
            printf("I don't have client with this fd.\n");
            break;
        }
    }
}
else if (fd == sd){
    /* a client wants something */
    /* accept returns a new file descriptor in case of success
       and the old sd can be used again. */
    if ((sd2 = accept(sd, NULL, NULL)) < 0){
        perror("accept");
        exit(EXIT_FAILURE);
    }
    /* if my client list is not full enter here. */
    if (clnum < MAX_CLIENTS){
        /* Set the bit for the file descriptor sd2 */

        FD_SET(sd2, &set1);
        /* put the sd2 of the new client into the ClientArr */

        ClientArr[clnum][0] = sd2;
        ClientArr[clnum][1] = clnum+1;
    }
}

```

```

        clnum++;
        fprintf(stderr, "Client accepted. This is client %d and
                has fd = %d.\n", clnum, ClientArr[clnum-1][0]);
        strcpy(buff, "Request Accepted.
                You are client with fd = ");
        snprintf(msg1, 256, "%d.\n", ClientArr[clnum-1][0]);

        strcat(buff, msg1);
        cryp.src = buff;
        cryp.dst = temp;
        cryp.len = sizeof(buff);
        cryp.op = COP_ENCRYPT;
        if (ioctl(fdnew, CIOCCRYPT, &cryp)) {
            perror("ioctl(CIOCCRYPT)");
            return 1;
        }
        insist_write(sd2, temp, sizeof(temp));
    }

} else {
    /* if server is full of clients inform them */
    cryp.src = "Full of clients. Try later. I am sorry! \n";
    cryp.dst = temp;
    cryp.len = sizeof(keyboard);
    cryp.op = COP_ENCRYPT;
    if (ioctl(fdnew, CIOCCRYPT, &cryp)) {
        perror("ioctl(CIOCCRYPT)");
        return 1;
    }
    insist_write(sd2, temp, sizeof(temp));
    close(sd2);
}

} else if (fd > 0) {
    /* a client sent something and first i have to read it */
    if (insist_read(fd, message, MSG_SIZE) != MSG_SIZE) {
        perror("read");
        exit(EXIT_FAILURE);
    }
    cryp.src = message;
    cryp.dst = temp;
    cryp.len = sizeof(message);
    cryp.op = COP_DECRYPT;
    if (ioctl(fdnew, CIOCCRYPT, &cryp)) {
        perror("ioctl(CIOCCRYPT)");
        return 1;
    }
    /* print the message that a client sent */
    printf("Client with fd = %d sent: %s", fd, temp);
    /* if a client wants to see all the list of clients
       that server manages sends the following msg */
    if (strcmp(temp, "show list of clients\n") == 0) {
        for (i = 0; i < clnum; i++) {
            strcpy(buff, "\nClient with fd = ");
            snprintf(msg1, 256, "%d.\n", ClientArr[i][0]);

            strcat(buff, msg1);
            cryp.src = buff;
            cryp.dst = temp;
            cryp.len = sizeof(buff);
            cryp.op = COP_ENCRYPT;
            if (ioctl(fdnew, CIOCCRYPT, &cryp)) {
                perror("ioctl(CIOCCRYPT)");
                return 1;
            }
        }
    }
}

```

```

                insist_write(fd, temp, sizeof(temp));
            }
        }
        /* if a client sent koukis means that is dead and i have to
           close the socket */
        if (strcmp(temp, "koukis\n") == 0) {
            printf("Client with fd = %d is out and i will
                   close the socket!\n", fd);

            close(fd);
            // clear the client that send "koukis"
            FD_CLR(fd, &set1);
            /* drop this client from the array */
            for (index = 0; index < clnum - 1; index++)
                if (ClientArr[index] == fd) break;

            for (; index < clnum - 1; index++) {
                (ClientArr[index][0]) =
                (ClientArr[index + 1][0]);
            }
            clnum--;
        }
    }
    else printf("fd<0????\n");
}
}
if (ioctl(fdnew, CIOCFSESSION, &sess.ses)) {
    perror("ioctl(CIOCFSESSION)");
    return 1;
}
if (close(fdnew) < 0) {
    perror("close(fd)");
    return 1;
}
if (close(sd) < 0) {
    perror("close(fd)");
    return 1;
}
return 0;
}

```

Εξήγηση Κόδικα :

- **socket-client.c** : Αφού κάνω αρχικά τους απαρραίτητους ελέγχους για το αν έχω πάρει σωστά ορίσματα, με χρήση της εντολής **socket(PF_INET, SOCK_STREAM, 0)** δημιουργώ ενα TCP/IP socket. Σαν namespace(domain) στέλνω το PF_INET, το οποίο είναι ενα internet namespace και σαν τύπο επικοινωνίας(type) το SOCK_STREAM, το οποίο δηλώνει ότι η σύνδεση είναι σειριακή, αξιόπιστη και υπάρχει αμφίδρομη μεταφορά δεδομένων. Η τρίτη παράμετρος αφορά το πρωτόκολλο επικοινωνίας και του δίνω το 0 ώστε να μπει η default επιλογή(βέβαια οι πάραμετροι domain και type ουσιαστικά ορίζουν μονοσήμαντα το είδος του πρωτοκόλλου επικοινωνίας στη περίπτωση μας). Η εντολή αυτή μας επιστρέφει ενα file descriptor τον οποίο θα χρησιμοποιήσουμε για την επικοινωνία με τον server. Στη συνέχεια με χρήση της εντολής **connect(sd, (struct sockaddr *) &sa, sizeof(sa))** ο αντίστοιχος client κάνει αίτηση στο server ώστε να μπορέσει να συνδεθεί μαζί του και αναμένει την απάντηση του server(θα εξηγήσω στη συνέχεια τα κριτήρια με τα οποία ο server κάνει accept κάποιο request ενός client). Έπειτα, με χρήση της εντολής **open("/dev/crypto", O_RDWR)** ανοίγω την κρυπτογραφική συσκευή και παίρνω τον file descriptor. Στο σημείο αυτό θα πρέπει να βάλω στα πεδία των struct session_op sess, struct crypt_op cryp τις κατάλληλες παραμέτρους ώστε να μπορέσει να γίνει το encrypt και το decrypt. Η εντολή **ioctl(fdnew, CIOCGSESSION, &sess)**, η οποία είναι και η επόμενη που εκτελείται ουσιαστικά αρχίζει ενα session με την συσκευή. Η συνάρτηση **ioctl()** είναι υπεύθυνη για τον έλεγχο όλων των I/O operations της συσκευής και ανάλογα με το τι θα δώσουμε σαν δεύτερο όρισμα(request code), εκτελεί και την κατάλληλη ενέργεια. Στη συνέχεια μπαίνει ο client σε μια ατέρμονη επανάληψη στην οποία κάθε φορά η εντολή **select(FD_SETSIZE, &temp_set, NULL, NULL, NULL)**, επιλέγει εναν file descriptor από αυτούς που είναι ready. Ξέχασα να αναφέρω ότι πριν την επανάληψη έθεσα το bit στους file descriptors του stdin και του socket που δημιούργησα παραπάνω. Στο σημείο αυτό ελέγχω ποιον file descriptor ‘διάλεξε’ η **select()** και αν είναι το 0(είναι ο file desriptor του stdin) τότε παίρνω το μήνυμα που γράφει το client στο stdin και το κρυπτογραφώ χρησιμοποιώντας την εντολή **ioctl(fdnew, CIOCCRYPT, &cryp)** αφού βάλω πρώτα στα κατάλληλα πεδία του struct crypt_op τις κατάλληλες τιμές. Μετά την κρυπτογράφηση στέλνω το κρυπτογραφημένο μήνυμα στο server, γράφοντας το στο αντίστοιχο socket που έχει κάθε client για την επικοινωνία με τον server. Για να το γράψω στο socket χρησιμοποιώ την συνάρτηση **insist_write()** και οχι την απλή write() διότι θέλω να είμαι σίγουρος ότι θα γράψω όλο το μήνυμα. Τέλος ελέγχω αν το μήνυμα που έγραψε ο client είναι το **”koukis”** και αν είναι τότε ο client κλείνει το socket και τερματίζει την λειτουργία του, με κατάλληλο μήνυμα. Αν τώρα η select() επιλέξει τον file descriptor του socket που δημιούργησα, σημαίνει ότι ο server έχει στείλει ενα κρυπτογραφημένο μήνυμα στο client. Στη περίπτωση αυτή ο client διαβάζει το μήνυμα με χρήση της συνάρτησης **insist_read()** και όχι της απλής read() διότι αν χρησιμοποιούσα την απλή read() και εκείνη δεν διάβαζε όσους χαρακτήρες ήθελα τότε το πρόγραμμα μου θα έσκαγε αφού δεν θα μπορούσε να γίνει σωστή αποκρυπτογράφηση. Αντίθετα η **insist_read()** διαβάζει τους χαρακτήρες που χρειαζόμαστε για να δουλέψει σωστά. Έπειτα κάνω την αντίστροφη διαδικασία από αυτή που ανέφερα παραπάνω, δηλαδή αποκρυπτογραφώ το μήνυμα που έστειλε ο server χρησιμοποιώντας την εντολή **ioctl(fdnew, CIOCCRYPT, &cryp)** αφού βάλω πρώτα στα κατάλληλα πεδία του struct crypt_op τις κατάλληλες τιμές(στο πεδίο cryp.op βάζω το

COP_DECRYPT αυτή τη φορά για να πετύχω την αποκρυπτογράφηση). Τέλος τυπώνω το μήνυμα που έστειλε ο server χωρίς να είναι πλέον κρυπτογραφημένο και ελέγχω αν το μήνυμα αυτό είναι το **"halt"**. Αν όντως το μήνυμα είναι το **"halt"** τότε ο κάθε client παύει την λειτουργία του, κλείνοντας το αντίστοιχο socket και τυπώνοντας κατάλληλο μήνυμα. Συμπερασματικά, για να βγει ένας client από την ατέρμονη επανάληψη θα πρέπει είτε να γράψει εκείνος **"koukis"** στο stdin είτε να του έρθει **"halt"** από τον server. Στο τέλος του κώδικα με την εντολή **ioctl(fdnew, CIOCFSESSION, &sess.ses)** τερματίζεται το session με την συσκευή και υπάρχουν τα κατάλληλα **close()** για να είναι συνεπές το πρόγραμμα.

- **socket-server.c** : Στο κώδικα αυτό αρχικά γίνονται ακριβώς τα ίδια με αυτά που εξήγησα στον κώδικα του **socket-client.c**, δηλαδή έλεγχος ορισμάτων και δημιουργία ενος νέου TCP/IP socket με την εντολή **socket(PF_INET, SOCK_STREAM, 0)** με τις ίδιες παραμέτρους και για το λόγο αυτό δεν εξηγώ ξανά τον λόγο για τον οποίο τις επέλεξα. Έπειτα χρησιμοποιώ την **bind(sd, (struct sockaddr *)&sa, sizeof(sa))** για να καθορίσω την πόρτα στην οποία λαμβάνω συνδέσεις και είναι απαραίτητη η χρήση της ιδιαίτερα σε sockets με **type=SOCK_STREAM**. Στη συνέχεια χρησιμοποιώ την **listen(sd, TCP_BACKLOG)** για να ετοιμάσω τον server να δεχθεί συνδέσεις. Η δεύτερη παράμετρος με την οποία καλώ την **listen()** είναι ο αριθμός των συνδέσεων οι οποίες μπορούν να περιμένουν στην ουρά(αν η ουρά είναι γεμάτη τότε απορρίπτονται). Έπειτα, με χρήση της εντολής **open("/dev/crypto", O_RDWR)** ανοίγω την κρυπτογραφική συσκευή και παίρνω τον file descriptor. Στο σημείο αυτό θα πρέπει να βάλω στα πεδία των struct **session_op** sess, struct **crypt_op** cryp τις κατάλληλες παραμέτρους ώστε να μπορέσει να γίνει το encrypt και το decrypt. Η εντολή **ioctl(fdnew, CIOCGSESSION, &sess)**, η οποία είναι και η επόμενη που εκτελείται ουσιαστικά αρχίζει ένα session με την συσκευή. Η συνάρτηση **ioctl()** είναι υπεύθυνη για τον έλεγχο όλων των I/O operations της συσκευής και ανάλογα με το τι θα δώσουμε σαν δεύτερο όρισμα(request code), εκτελεί και την κατάλληλη ενέργεια. Στη συνέχεια μπαίνει ο client σε μια ατέρμονη επανάληψη στην οποία κάθε φορά με την εντολή **select(FD_SETSIZE, &temp_set, NULL, NULL, NULL)**, επιλέγει έναν file descriptor από αυτούς που είναι ready. Ξέχασα να αναφέρω οτι πριν την επανάληψη έθεσα το bit στους file descriptors του stdin και του socket που δημιούργησα παραπάνω. Στη συνέχεια ελέγχω αν κάποιος file descriptor είναι ready και αν βρώ κάποιον τότε κάνω τον εξής διαχωρισμό : Αν είναι ο file descriptor 0 τότε ο server έχει σκοπό να γράψει κάτι στο stdin. Αρχικά πρέπει να γράψει έναν integer ο οποίος θα υποδηλώνει σε ποιόν file descriptor(δηλαδή σε ποιόν client) θέλει να γράψει το μήνυμα. Στη συνέχεια παίρνω το μήνυμα του server και το κρυπτογραφώ με χρήση της εντολής **ioctl(fdnew, CIOCCRYPT, &cryp)** αφού βάλω πρώτα στα κατάλληλα πεδία του struct **crypt_op** τις κατάλληλες τιμές και το κρατάω στη μεταβλητή temp. Υπάρχουν δύο ειδικές περιπτώσεις σχετικά με τον ακέραιο που γράφει αρχικά ο server. Αν δώσει το **-100** τότε ο server πεθαίνει και στέλνει **"halt"** σε όλους τους client για να τερματίσουν κι αυτοί και η επικοινωνία τελειώνει. Η άλλη ειδική περίπτωση είναι να γράψει ο server το **100** ο οποίο δηλώνει οτι το μήνυμα του πρέπει να πάει σε όλους τους active clients. Αν λοιπόν δεν δώσει -100, τότε ελέγχω αν έχει δώσει το 100 και αν όντως ισχύει αυτό στέλνω το κρυπτογραφημένο μήνυμα σε όλους τους active clients. Αν καμία από τις παραπάνω ειδικές περιπτώσεις δεν ισχύει ο server αναζητά τον integer τον οποίο έδωσε μέσα στον

πίνακα με τους file descriptors και αν τον βρει στέλνει το μήνυμα στον αντίστοιχο client ενώ αν δεν το βρει γράφει στο stdin οτι δεν υπάρχει ενεργός client με το δοσμένο file descriptor και δεν στέλνει πουθενά το μήνυμα.

Αν τώρα είναι ενεργός(δηλαδή τον επιλέξει η select()) ο file descriptor του socket του server σημαίνει οτι ο client ζητά από τον server να δεχθεί το αίτημα του για σύνδεση με χρήση της **accept(sd, NULL, NULL)**. Αν τελικά ο server αποδεχθεί το request του client επιστρέφει ενα νέο socket το οποίο θα χρησιμοποιείται για να μπορούν να επικοινωνούν. Αν τώρα υπάρχει ‘χώρος’ για νέο πελάτη, ενεργοποείται ο αντίστοιχος file descriptor, αποθηκεύεται στον πίνακα που κρατάει ο server με τις πληροφορίες κάθε client και στέλνετε κρυπτογραφημένο μήνυμα στον αντίστοιχο client με τις πληροφορίες του. Αν τώρα δεν υπάρχει ‘χώρος’ για νέο πελάτη τότε στέλνετε κρυπτογραφημένο μήνυμα το οποίο λέει οτι δεν υπάρχει χώρος. Θα πρέπει να τονίσω επίσης οτι ο file descriptor του socket του server μπορεί να χρησιμοποιηθεί ξανά από την **accept()**.

Η τελευταία περίπτωση είναι να είναι ενεργός ενας άλλος file descriptor το οποίο σημαίνει οτι κάποιος από τους clients οι οποίοι έχουν γίνει accept έχει γράψει κάτι στο socket με το οποίο επικοινωνεί με τον server. Στη περίπτωση αυτή ο server διαβάζει το μήνυμα με χρήση της συνάρτησης **insist_read()** και όχι της απλής **read()** διότι αν χρησιμοποιούσα την απλή **read()** και εκείνη δεν διάβαζε όσους χαρακτήρες ήθελα τότε το πρόγραμμα μου θα έσκαγε αφού δεν θα μπορούσα να κάνω την αποκρυπτογράφηση. Αντίθετα η **insist_read()** διαβάζει τους χαρακτήρες που χρειάζονται για να γίνει σωστά το decrypt. Έπειτα αποκρυπτογραφώ το μήνυμα που έστειλε ο client χρησιμοποιώντας την εντολή **ioctl(fdnew, CIOCRYPT, &crypt)** αφού βάλω πρώτα στα κατάλληλα πεδία του struct **crypt_op** τις κατάλληλες τιμές(στο πεδίο **crypt.op** βάζω το **COP_DECRYPT** αυτή τη φορά για να πετύχω την αποκρυπτογράφηση). Έπειτα τυπώνω το μήνυμα που έστειλε ο client χωρίς να είναι πλέον κρυπτογραφημένο. Στη συνέχεια ελέγχω αν το μήνυμα που έστειλε ο client ήταν **"show list of clients"** και στη περίπτωση που ήταν αυτό στέλνω στον αντίστοιχο client μια λίστα με τα στοιχεία όλων των ενεργών clients σε κρυπτογραφημένη μορφή. Τέλος ελέγχω αν ο client έστειλε το μήνυμα **"koukis"** και αν κάτι τέτοιο έχει γίνει ο server καταλαβαίνει οτι ο client είναι έχει πεθάνει και αυτό που κάνει είναι να κλείνει το αντίστοιχο socket ώστε να μην είναι κατεύλλημένος χωρίς λόγο ο αντίστοιχος file descriptor και βγάζει και από τον πίνακα με τα στοιχεία των clients τα στοιχεία του αντίστοιχου client. Συμπερασματικά, για να βγει ενας server από την ατέρμονη επανάληψη θα πρέπει να γράψει εκείνος -100 στο stdin. Στο τέλος του κώδικα με την εντολή **ioctl(fdnew, CIOCFSESSION, &sess.ses)** τερματίζεται το session με την συσκευή και υπάρχουν τα κατάλληλα **close()** για να είναι συνεπές το πρόγραμμα. Επίσης θα πρέπει να επισημάνω οτι και στο κώδικα **socket-server.c** όπου υπήρχε ανάγκη να γράψω ή να διαβάσω κάτι από ενα socket χρησιμοποιούσα τις συναρτήσεις **insist_read()** και **insist_write()** για τους λόγους που ανέφερα και παραπάνω αλλά ιδιαίτερα στην εξήγηση του κώδικα **socket-client.c**.

Για να σας παρουσιάσω όλα όσα σας εξήγησα παραπάνω σας παραθέτω μερικά screenshots από τις λειτουργίες που επιτελεί ο κώδικας που σας δίνω :

```
gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets /final/Z2 final $ ./socket-client localhost 35001
I am the client with name localhost and now i am starting.
Server sent:Request Accepted. You are client with fd = 5.

gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets /final/Z2 final $ ./socket-client localhost 35001
I am the client with name localhost and now i am starting.
Server sent:Request Accepted. You are client with fd = 6.

gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets /final/Z2 final $ ./socket-server
```

Client accepted. This is client 1 and has fd = 5.
Client accepted. This is client 2 and has fd = 6.

```
gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets /final/Z2 final $ ./socket-client localhost 35001
I am the client with name localhost and now i am starting.
Server sent:Request Accepted. You are client with fd = 5.
hello server!

gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets /final/Z2 final $ ./socket-client localhost 35001
I am the client with name localhost and now i am starting.
Server sent:Request Accepted. You are client with fd = 6.
hello server!
```

Client accepted. This is client 1 and has fd = 5.
Client accepted. This is client 2 and has fd = 6.
Client with fd = 5 sent: hello server!
Client with fd = 6 sent: hello server!

Tuesday May 31, 15:11:46

Tuesday May 31, 15:12:02

```
gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets  
/final/Z2 final $ ./socket-client localhost 35001  
I am the client with name localhost and now i am starting.  
Server sent:Request Accepted. You are client with fd = 5.  
hello server!  
Server sent: hello world!  
Server sent: hello client
```

```
gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets  
/final/Z2 final $ ./socket-client localhost 35001  
I am the client with name localhost and now i am starting.  
Server sent:Request Accepted. You are client with fd = 6.  
hello server!  
Server sent: hello world!  
Server sent: hello client
```

```
gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets/final/Z2 final $ ./socket-server  
Client accepted. This is client 1 and has fd = 5.  
Client accepted. This is client 2 and has fd = 6.  
Client with fd = 5 sent: hello server!  
Client with fd = 6 sent: hello server!  
100 hello world!  
5 hello client  
Server wants to send the message to client with fd = 5.  
6 hello client  
Server wants to send the message to client with fd = 6.
```

```
gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets  
/final/Z2 final $ ./socket-client localhost 35001  
I am the client with name localhost and now i am starting.  
Server sent:Request Accepted. You are client with fd = 5.  
hello server!  
Server sent: hello world!  
Server sent: hello client
```

```
gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets  
/final/Z2 final $ ./socket-client localhost 35001  
I am the client with name localhost and now i am starting.  
Server sent:Request Accepted. You are client with fd = 6.  
hello server!  
Server sent: hello world!  
Server sent: hello client
```

```
gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets/final/Z2 final $ ./socket-server  
Client accepted. This is client 1 and has fd = 5.  
Client accepted. This is client 2 and has fd = 6.  
Client with fd = 5 sent: hello server!  
Client with fd = 6 sent: hello server!  
100 hello world!  
5 hello client  
Server wants to send the message to client with fd = 5.  
6 hello client  
Server wants to send the message to client with fd = 6.  
7 hello client  
Server wants to send the message to client with fd = 7.  
I don't have client with this fd.
```

```

gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets
/final/Z2 final $ ./socket-client localhost 35001
I am the client with name localhost and now i am starting.
Server sent:Request Accepted. You are client with fd = 5.
hello server!
Server sent: hello world!
Server sent: hello client
show list of clients
Server sent:
Client with fd = 5.
Server sent:
Client with fd = 6.
|

```

```

gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets/final/Z2 final $ ./socket-server
Client accepted. This is client 1 and has fd = 5.
Client accepted. This is client 2 and has fd = 6.
Client with fd = 5 sent: hello server!
Client with fd = 6 sent: hello server!
100 hello world!
5 hello client
Server wants to send the message to client with fd = 5.
6 hello client
Server wants to send the message to client with fd = 6.
7 hello client
Server wants to send the message to client with fd = 7.
I don't have client with this fd.
Client with fd = 5 sent: show list of clients
Client with fd = 6 sent: show list of clients
|

```

```

gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets
/final/Z2 final $ ./socket-client localhost 35001
I am the client with name localhost and now i am starting.
Server sent:Request Accepted. You are client with fd = 5.
hello server!
Server sent: hello world!
Server sent: hello client
show list of clients
Server sent:
Client with fd = 5.
Server sent:
Client with fd = 6.
|

```

```

gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets
/final/Z2 final $ ./socket-client localhost 35001
I am the client with name localhost and now i am starting.
Server sent:Request Accepted. You are client with fd = 6.
hello server!
Server sent: hello world!
Server sent: hello client
show list of clients
Server sent:
Client with fd = 5.
Server sent:
Client with fd = 6.
koukis
Koukis killed the client(OMG)
gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets
/final/Z2 final $ |

```

```

Client accepted. This is client 2 and has fd = 6.
Client with fd = 5 sent: hello server!
Client with fd = 6 sent: hello server!
100 hello world!
5 hello client
Server wants to send the message to client with fd = 5.
6 hello client
Server wants to send the message to client with fd = 6.
7 hello client
Server wants to send the message to client with fd = 7.
I don't have client with this fd.
Client with fd = 5 sent: show list of clients
Client with fd = 6 sent: show list of clients
Client with fd = 6 sent: koukis
Client with fd = 6 is out and i will close the socket!
6 hello client
Server wants to send the message to client with fd = 6.
I don't have client with this fd.
|

```

```

gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets
/final/Z2 final $ |

```

```

gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets
/final/Z2 final $ ./socket-client localhost 35001
I am the client with name localhost and now i am starting.
Server sent:Request Accepted. You are client with fd = 5.
hello server!
Server sent: hello world!
Server sent: hello client
show list of clients
Server sent:
Client with fd = 5.
Server sent:
Client with fd = 6.
show list of clients
Server sent:
Client with fd = 5.

```

```

gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets
/final/Z2 final $ ./socket-client localhost 35001
I am the client with name localhost and now i am starting.
Server sent:Request Accepted. You are client with fd = 6.
hello server!
Server sent: hello world!
Server sent: hello client
show list of clients
Server sent:
Client with fd = 5.
Server sent:
Client with fd = 6.
koukis
Koukis killed the client(OMG)
gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets
/final/Z2 final $ |

```

Client with fd = 5 sent: hello server!
 Client with fd = 6 sent: hello server!
 100 hello world!
 5 hello client
 Server wants to send the message to client with fd = 5.
 6 hello client
 Server wants to send the message to client with fd = 6.
 7 hello client
 Server wants to send the message to client with fd = 7.
 I don't have client with this fd.
 Client with fd = 5 sent: show list of clients
 Client with fd = 6 sent: show list of clients
 Client with fd = 6 sent: koukis
 Client with fd = 6 is out and i will close the socket!
 6 hello client
 Server wants to send the message to client with fd = 6.
 I don't have client with this fd.
 Client with fd = 5 sent: show list of clients

gvavou5@linas ~ |~Terminal-gvavou5~ |~Terminal-gvavou5~ |~Terminal-gvavou5~ | Tuesday May 31, 15:14:42

```

gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets
/final/Z2 final $ ./socket-client localhost 35001
I am the client with name localhost and now i am starting.
Server sent:Request Accepted. You are client with fd = 5.
hello server!
Server sent: hello world!
Server sent: hello client
show list of clients
Server sent:
Client with fd = 5.
Server sent:
Client with fd = 6.
show list of clients
Server sent:
Client with fd = 5.
Server sent:
Client with fd = 6.
Server sent:
Client with fd = 5.
Server sent:
Client with fd = 6.
koukis
Koukis killed the client(OMG)
gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets
/final/Z2 final $ |

```

100 hello world!
 5 hello client
 Server wants to send the message to client with fd = 5.
 6 hello client
 Server wants to send the message to client with fd = 6.
 7 hello client
 Server wants to send the message to client with fd = 7.
 I don't have client with this fd.
 Client with fd = 5 sent: show list of clients
 Client with fd = 6 sent: show list of clients
 Client with fd = 6 sent: koukis
 Client with fd = 6 is out and i will close the socket!
 6 hello client
 Server wants to send the message to client with fd = 6.
 I don't have client with this fd.
 Client with fd = 5 sent: show list of clients
 -100
 Server is going down and all clients are dead also
 gvavou5@linas ~/Desktop/3d_Oslab_Exercise/virtio-crypto-helpcode-20160422/sockets/final/Z2 final \$ |

gvavou5@linas ~ |~Terminal-gvavou5~ |~Terminal-gvavou5~ |~Terminal-gvavou5~ | Tuesday May 31, 15:14:48

Ζητούμενο 3

Στο μέρος αυτό θα σας παραθέσω αρχικά τους κώδικες τους οποίους άλλαξα και στη συνέχεια θα σας τους εξηγήσω αναλυτικά. Οι κώδικες αυτοί είναι οι εξής :

1) **crypto-chrdev.c** 2) **virtio-crypto.c** και τους παραθέτω στη συνέχεια :

```
/*
 * crypto-chrdev.c
 *
 * Implementation of character devices
 * for virtio-crypto device
 *
 * Giorgos Vavouliotis <nuovocominzio@hotmail.com>
 * 03112083
 *
 */
#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;
#define KEY_SIZE 16

#define IV      "0123456789ABCDEF"
#define KEY     "0123456789ABCDEF"
#define MSG_LEN 256

spinlock_t chdevlock;

/**
 * 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 ");

    /* i lock here because a dont want someone enter
       here and delete the device with rmmod
    */
    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, host_fd = -1;
    /* posa 8a steilw kai den 8a peiraxtoun */
    unsigned int num_in = 0;
    /* posa 8a steilw kai 8a peiraxtoun */
    unsigned int num_out = 0;
    unsigned long flags;
    unsigned int len;
    unsigned int syscall_type = VIRTIO_CRYPTO_SYSCALL_OPEN;
    struct crypto_open_file *cropenfile;
    struct crypto_device *crdev;
    /* sg list for syscall_type */
    struct scatterlist sglist_syscall_type;
    /* sg list for host_fd(i get it from the host) */
    struct scatterlist sglist_host_fd;
    /* 2 cell array with pointers to sg lists */
    struct scatterlist *sgs[2];

    debug("Entering to open");

    ret = -ENODEV;
    if ((ret = nonseekable_open(inode, filp)) < 0)
        goto fail;

    /* Associate this open file with the relevant crypto device. */
    /* Given the minor number of the inode return the crypto device that
     * owns that number. */
    crdev = get_crypto_dev_by_minor(iminor(inode));
    if (!crdev) {
        debug("Could not find crypto device with %u minor", iminor(inode));
        ret = -ENODEV;
        goto fail;
    }

    /*The memory is set to zero with kzalloc. */
    cropenfile = kzalloc(sizeof(*cropenfile), GFP_KERNEL);
    if (!cropenfile) {
        ret = -ENOMEM;
        goto fail;
    }

    /* enhmerwsh pediwn */
    cropenfile->crdev = crdev;
    cropenfile->host_fd = -1;
    filp->private_data = cropenfile;
}

```

```

/* baww sthn sg list sglist_syscall_type to syscall_type to opoio einai
   num_out dhladh den 8a peiraxtei h timh tou.
*/
sg_init_one(&sglist_syscall_type,&syscall_type,sizeof(syscall_type));
sgs[num_out++] = &sglist_syscall_type;

/* baww sthn sg list sglist_host_fd to host_fd to opoio einai
   num_in dhladh 8a allaksei h timh tou.
*/
sg_init_one(&sglist_host_fd,&host_fd,sizeof(host_fd));
sgs[num_out + num_in++] = &sglist_host_fd;

/* spinlock => lock here */
spin_lock_irqsave(&chdevlock, flags);

/* here add data to the virtqueue */
virtqueue_add_sgs(crdev->vq,sgs,num_out,num_in,&sglist_syscall_type,GFP_ATOMIC);

/* the data are placed ==> inform with kick */
virtqueue_kick(crdev->vq);

/* here i do nothing. I wait for the host to proccess the data */
while (virtqueue_get_buf(crdev->vq,&len) == NULL );

/* unlock because i have the fd of the host crypto device. */
spin_unlock_irqrestore(&chdevlock,flags); //time to unlock

debug( "The host returned fd: %d", host_fd );

/* store the fd of the host device */
cropenfile->host_fd = host_fd;

/* If host failed to open() return -ENODEV. */
if (cropenfile->host_fd == -1){
    ret = -ENODEV;
}

fail:
debug("Leaving from open");
return ret;
}

static int crypto_chrdev_release(struct inode *inode, struct file *filp)
{
    int ret = 0, host_fd = -1;
    unsigned long flags;
    unsigned int len;
    unsigned int syscall_type = VIRTIO_CRYPTO_SYSCALL_CLOSE;
    /* posa 8a steilw kai den 8a peiraxtoun */
    unsigned int num_out = 0;
    /* posa 8a steilw kai 8a peiraxtoun */
    unsigned int num_in = 0;
    struct crypto_open_file *cropenfile = filp->private_data;
    struct crypto_device *crdev = cropenfile->crdev;
    /* sg list for syscall_type */
    struct scatterlist sglist_syscall_type;
    /* sg list for host_fd(i get it from the host) */
    struct scatterlist sglist_host_fd;
    /* 2 cell array with pointers to sg lists */
    struct scatterlist *sgs[2];

    debug("Entering Release");

```

```

host_fd = copenfile->host_fd;

/* bazw sthn sg list sglist_syscall_type to syscall_type to opoio einai
   num_out dhladh den 8a peiraxtei h timh tou.
*/
sg_init_one(&sglist_syscall_type,&syscall_type,sizeof(syscall_type));
sgs[num_out++] = &sglist_syscall_type;

/* bazw sthn sg list sglist_host_fd to host_fd to opoio einai
   num_out dhladh den 8a allaksei h timh tou.
*/
sg_init_one(&sglist_host_fd,&host_fd,sizeof(host_fd));
sgs[num_out++] = &sglist_host_fd;

/* lock */
spin_lock_irqsave(&chdevlock, flags);

/* here add data to the virtqueue */
virtqueue_add_sgs(crdev->vq,sgs,num_out,num_in,&sglist_syscall_type,GFP_ATOMIC);

/* the data are placed ==> inform with kick */
virtqueue_kick(crdev->vq);

/* here i do nothing. I wait for the host to proccess the data */
while (virtqueue_get_buf(crdev->vq,&len) == NULL );

/* unlock */
spin_unlock_irqrestore(&chdevlock,flags);

/* The fd is closed. */
debug( "The host closed the fd: %d", host_fd );

kfree(copenfile);
debug("Leaving");
return ret;
}

static long crypto_chrdev_ioctl(struct file *filp, unsigned int cmd,unsigned long arg)
{
    int host_fd, host_return_val=-1;
    long ret = 0,ret_value;
    unsigned long flags;
    unsigned char output_msg[MSG_LEN], input_msg[MSG_LEN];
    unsigned char *session_key, *src,*dst,*iv;
    unsigned int num_out=0, num_in=0;
    unsigned int syscall_type = VIRTIO_CRYPTO_SYSCALL_IOCTL,len;
    u_int32_t sess_ses,ses_id,key_size;
    u_int16_t oper;
    u_int msg_len;

    /* Structs */
    struct crypto_open_file *copenfile = filp->private_data;
    struct crypto_device *crdev = copenfile->crdev;
    struct virtqueue *vq = crdev->vq;
    struct session_op sess,*session_pointer;
    struct crypt_op cryp,*cryp_pointer;

    /* All the scatterlists */
    struct scatterlist sglist_syscall_type;
    struct scatterlist sglist_fd;
    struct scatterlist sglist_request;
    struct scatterlist sglist_return_val;

```

```

/* CIOCGSESSION scatterlists */
struct scatterlist sglist_session_op;
struct scatterlist session_key_list;
struct scatterlist sglist_keysizes;

/* CIOCFSESSION scatterlist */
struct scatterlist sglist_sess_id;

/* CIOCCRYPT scatterlists */
struct scatterlist sglist_outmsg;
struct scatterlist sglist_inmsg;
struct scatterlist sglist_iv;
struct scatterlist sglist_crypto;
struct scatterlist sglist_operation;
struct scatterlist sglist_msrlen;

struct scatterlist *sgs[10];

/* Initalizations */
session_pointer = NULL;
cryp_pointer = NULL;
dst=NULL;
host_fd = cropenfile->host_fd ;

debug("Entering Ioctl");

/* baww sthn sg list sglist_syscall_type to syscall_type to opoio einai
   num_out dhladh den 8a peiraxtei h timh tou.
*/
sg_init_one(&sglist_syscall_type, &syscall_type, sizeof(syscall_type));
sgs[num_out++] = &sglist_syscall_type; //sg_out[0]

/* baww sthn sg list sglist_fd to host_fd to opoio einai
   num_out dhladh den 8a peiraxtei h timh tou.
*/
sg_init_one(&sglist_fd,&host_fd,sizeof(host_fd));
sgs[num_out++] = &sglist_fd; // sg_out[1]

/* omoia */
sg_init_one(&sglist_request,&cmd,sizeof(cmd));
sgs[num_out++] = &sglist_request;

switch (cmd) {
case CIOCGSESSION:
    /* Start a new session */
    debug("CIOCGSESSION");

    session_pointer = (struct session_op*)arg;
    /* Returns number of bytes that could not be copied. On success,
       this will be zero. */
    ret_value=copy_from_user(&sess,session_pointer,sizeof(struct
                                         session_op));

    if( ret_value != 0)
        debug("I didn't copy %lu\n bytes", ret_value);

    session_key = kmalloc(sess.keylen+1, GFP_KERNEL);

    ret_value=copy_from_user(session_key,session_pointer->key,
                           session_pointer->keylen*sizeof(unsigned char));

    if( ret_value != 0)
        debug("I didn't copy %lu\n bytes",ret_value);
}

```

```

session_key[sess.keylen]='\0';

key_size = session_pointer->keylen;

/* bazw sthn sg list sglist_session_key to session_key to opoio
   sinai num_out dhladh den 8a peiraxtei h timh tou. Omoia gemizw
   kai tis epomenes sg lists.
*/

sg_init_one(&session_key_list,session_key,sess.keylen);
sgs[num_out++] = &session_key_list;

sg_init_one(&sglist_keysizes,&key_size,sizeof(key_size));
sgs[num_out++] = &sglist_keysizes;

/* host_return_val will show if the open succeeded */

sg_init_one(&sglist_return_val,&host_return_val,sizeof(host_return_val));
sgs[num_out + num_in++] = &sglist_return_val;

sg_init_one(&sglist_session_op,&sess,sizeof(sess));
sgs[num_out + num_in++] = &sglist_session_op;

break;

case CIOCFSESSION:
    /* Close the session */
    debug("CIOCFSESSION");

    ret_value=copy_from_user(&sess_ses,(u_int32_t *)arg ,
                           sizeof(sess_ses));

    if( ret_value != 0)
        debug("I didn't copy %lu\n bytes", ret_value);

    sg_init_one(&sglist_sess_id,&sess_ses,sizeof(sess_ses));
    sgs[num_out++] = &sglist_sess_id; //sg_out[3];

    /* ret_val_host will show if the close succeeded */

sg_init_one(&sglist_return_val,&host_return_val,sizeof(host_return_val));
sgs[num_out+num_in++] = &sglist_return_val; //sg_in[0];

break;

case CIOCCRYPT:
    /* Start a crypto session */
    debug("CIOCCRYPT");

    cryp_pointer = (struct crypt_op * ) arg;

    ret_value=copy_from_user(&crypt,cryp_pointer,sizeof(struct
                           crypt_op));

    if( ret_value != 0)
        debug("I didn't copy %lu\n bytes", ret_value);

    src = kmalloc(cryp.len,GFP_KERNEL);

    ret_value = copy_from_user(src, cryp.src , cryp.len);
    if( ret_value != 0)
        debug("I didn't copy %lu\n bytes", ret_value);

```

```

    iv = kmalloc(sizeof(cryp.iv)*2, GFP_KERNEL);

    ret_value = copy_from_user(iv,cryp.iv,sizeof(cryp.iv)*2);
    if( ret_value != 0)
        debug("I didn't copy %lu\n bytes", ret_value);

    oper = cryp.op;
    dst = kmalloc(cryp.len,GFP_KERNEL);
    msg_len = cryp.len;

    sg_init_one(&sglist_crypto,&cryp,sizeof(struct crypt_op));
    sgs[num_out++] = &sglist_crypto;

    sg_init_one(&sglist_inmsg,src,cryp.len * sizeof(unsigned char));
    sgs[num_out++] = &sglist_inmsg;

    sg_init_one(&sglist_iv,iv,sizeof(cryp.iv)*2);
    sgs[num_out++] = &sglist_iv;

    sg_init_one(&sglist_operation,&oper,sizeof(oper));
    sgs[num_out++] = &sglist_operation;

    sg_init_one(&sglist_msrlen,&msg_len,sizeof(msg_len));
    sgs[num_out++]=&sglist_msrlen;

sg_init_one(&sglist_return_val,&host_return_val,sizeof(host_return_val));
    sgs[num_out+num_in++] = &sglist_return_val;

    sg_init_one(&sglist_outmsg,dst,cryp.len*sizeof(unsigned char));
    sgs[num_out + num_in++] = &sglist_outmsg;

break;

default:
    debug(" Unsupported ioctl command");

    break;
}

/* lock */
spin_lock_irqsave(&chdevlock, flags);

/* add the data to the virtqueue */
virtqueue_add_sgs(crdev->vq, sgs, num_out, num_in ,
                  &sglist_syscall_type, GFP_ATOMIC);

/* inform that the data are placed */
virtqueue_kick(crdev->vq);

/* do nothing until you get the data from the host */
while (virtqueue_get_buf(crdev->vq, &len) == NULL);
spin_unlock_irqrestore(&chdevlock, flags);

/* analoga me to cmd 8a prepei na steilw ta data ston user */
switch (cmd) {
case CIOCGSESSION:
    debug("CIOCGSESSION SEND DATA TO USER");
    ret_value = copy_to_user (&(session_pointer->ses),&(sess.ses),sizeof
                           (u_int32_t));
    if( ret_value != 0)
        debug("I didn't copy %lu\n bytes", ret_value);
    if (host_return_val < 0) {
        /* FAIL */
        return -1;
    }
}

```

```

        break;

    case CIOCFSESSION: //CLOSE
        debug("CIOCFSESSION SEND DATA TO USER");
        if (host_return_val < 0) {
            /* FAIL */
            return -1;
        }
        break;

    case CIOCCRYPT: //CRYPTO
        debug("CIOCCRYPT SEND DATA TO USER");
        if (host_return_val < 0) {
            /* FAIL */
            return -1;
        }
        ret_value = copy_to_user(cryp_pointer->dst,dst,cryp.len);
        if (ret_value != 0)
            debug("I didn't copy %lu\n bytes", ret_value);

        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");
}

```

```

/*
 * Virtio Crypto Device
 *
 * Implementation of virtio-crypto qemu backend device.
 *
 * Dimitris Siakavaras <jimsiak@cslab.ece.ntua.gr>
 * Stefanos Gerangelos <sgerag@cslab.ece.ntua.gr>
 * Vavouliotis Giorgos <nuovocominzio@hotmail.com>
 * 03112083
 */

#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>

#define KEY_SIZE 24
#define BLOCK_SIZE 16
#define MSG_SIZE 16384
#define IV      "0123456789ABCDEF"
#define KEY     "0123456789ABCDEF"

#define DEBUG_print(fmt, ...) \
    do { fprintf(stderr, fmt, ## __VA_ARGS__); } while (0)

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)
{
    DEBUG_IN();
    int ret;
    unsigned int *syscall_type;
    unsigned char *key;
    u_int32_t *session_ses, *key_size;
    u_int16_t *operation;
    u_int *messaglen;
    VirtQueueElement element;
    struct session_op *sess;
    struct crypt_op *crys;

    if (!virtqueue_pop(vq, &element)) {
        DEBUG("No item to pop from VQ");
        return;
    }

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

    /* i take the syscall type that frontend sent */
    syscall_type = element.out_sg[0].iov_base;

    switch (*syscall_type) {
    case VIRTIO_CRYPTO_SYSCALL_TYPE_OPEN:
        DEBUG("VIRTIO_CRYPTO_SYSCALL_TYPE_OPEN");
        /* Try to open the real crypto device */
        int fdo = open(CRYPTODEV_FILENAME, O_RDWR, 0);
        if (fdo < 0) {
            perror("Failed to open(" CRYPTODEV_FILENAME ")");
        }

        /* Store the result into the gather list of host_fd */
        mempcpy(element.in_sg[0].iov_base, (char *) &fdo, sizeof(fdo));

        DEBUG(" OK with opening.\n");
        break;

    case VIRTIO_CRYPTO_SYSCALL_TYPE_CLOSE:
        DEBUG("VIRTIO_CRYPTO_SYSCALL_TYPE_CLOSE");
        /* i take the fd of the device which i want to close */
        int *fdcl = element.out_sg[1].iov_base;

        if ((close(*fdcl)) < 0) {
            perror("Failed to close(" CRYPTODEV_FILENAME ")");
        }

        DEBUG(" OK with closing.\n");
        break;

    case VIRTIO_CRYPTO_SYSCALL_TYPE_IOCTL:
        DEBUG("VIRTIO_CRYPTO_SYSCALL_TYPE_IOCTL");

        /* take the fd and the target request that frontend sent */

        int *fdcr = element.out_sg[1].iov_base;
        unsigned int *request = element.out_sg[2].iov_base;

        switch (*request) {
        case CIOCGSESSION:
            /* OPEN the real crypto device */
            DEBUG("CIOCGSESSION BACKEND");

            /* take the sess to edit it */

```

```

sess = element.in_sg[1].iov_base;
/* take the key size and the key */
key = element.out_sg[3].iov_base;
key_size = element.out_sg[4].iov_base;

/* fill the proper fields to open the real cryptodev */
(*sess).cipher = CRYPTO_AES_CBC;
(*sess).keylen = *key_size;
(*sess).key     = (_u8 __user *)key;

if (ret=ioctl(*fdcr, CIOCGSESSION, sess)) {
    perror("ioctl(CIOCGSESSION)");
    return;
}

/* copy the return value to the right place of the
   element_in */
mempcpy(element.in_sg[0].iov_base, &ret, sizeof(ret));
/* copy the struct session_op to the right place of the
   element_in */
mempcpy(element.in_sg[1].iov_base, sess, sizeof(struct
                                             session_op));

/* all the input elemenets(these that i have to edit)
   are edited and placed to the right place */

break;

case CIOCFSESSION:
/* CLOSE the device */
DEBUG ("CIOCFSESSION BACKEND");

/* take the sess to edit it */
session_ses = element.out_sg[3].iov_base;

if ((ret=ioctl(*fdcr, CIOCFSESSION, session_ses))) {
    perror("ioctl(CIOCFSESSION)");
    return;
}

/* copy the return value to the right place of the
   element_in to know the frontend if close succeeded */
mempcpy(element.in_sg[0].iov_base, &ret,
         sizeof(ret));

break;

case CIOCCRYPT:
/* CRYPTO */
DEBUG ("CIOCCRYPT BACKEND");

/* take all the necessary to make the encryption/
   decryption */

/* num_out data ==> not edit */

cryp      = element.out_sg[3].iov_base;
cryp->src = element.out_sg[4].iov_base;
cryp->iv  = element.out_sg[5].iov_base;
messaglen = element.out_sg[7].iov_base;
cryp->len = *messaglen;
/* this show if have to encrypt or decrypt */
operation = element.out_sg[6].iov_base;
cryp->op = *operation;
/* num_in data ==> edit */
cryp->dst = element.in_sg[1].iov_base;

```

```

        /* classic ioctl() call */
        if (ret = ioctl(*fdcr, CIOCCRYPT, cryp)) {
                perror("ioctl(CIOCCRYPT)");
                return;
        }
        /* copy the return value to the right place of the
           element_in to know the frontend if crypt succeeded */
        mempcpy(element.in_sg[0].iov_base, &ret, sizeof(ret));

        /*copy the destination to the right place of the
           element_in to know the frontend if the result of crypt*/
        mempcpy(element.in_sg[1].iov_base, cryp->dst,cryp-
                >len*sizeof(unsigned char));

        break;

    default:
        DEBUG ("Unsupported ioctl command BACKEND");
        break;
}

default:
    DEBUG ("Unknown syscall_type");
}
virtqueue_push(vq, &element, 0);
DEBUG_print( "Notify VM" );
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)

```

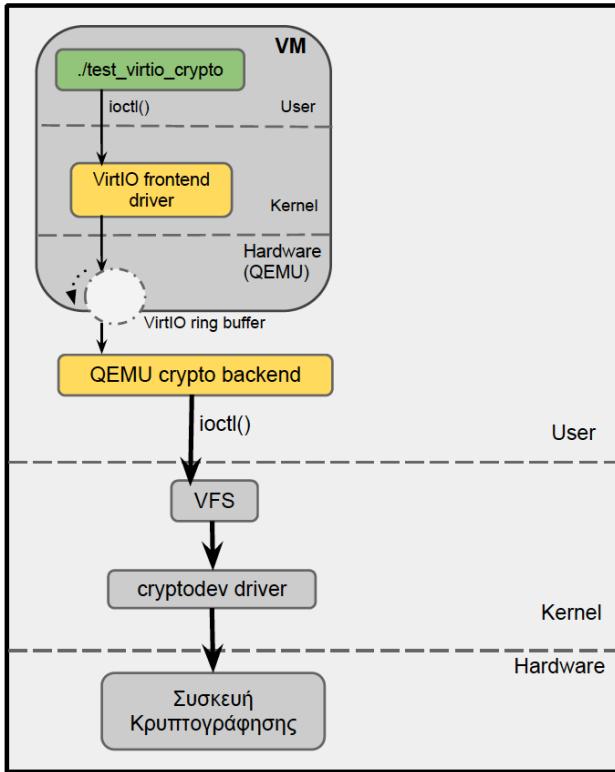
Εξήγηση Κώδικα :

- **crypto-chrdev.c (Frontend)** : Ο frontend οδηγός υπάρχει μέσα στο vm το οποίο δημιουργούμε. Για να γίνει η εισαγωγή ενος virtio driver στο πυρήνα του vm θα πρέπει να κληθεί η συνάρτηση **register_virtio_driver()**, η οποία και καλείται από την συνάρτηση **static int __init init()** που υπάρχει μέσα στο αρχείο **crypto-module.c**. Το **struct virtio_driver** έχει τα εξής πεδία : **unsigned int feature_table[]** το οποίο για την άσκηση αυτή είναι άδειο, **struct virtio_device_id id_table[]** το οποίο καθορίζει τις συσκευές τις οποίες υποστηρίζει ο οδηγός μας, **void (*remove)(struct virtio_device *)** η οποία είναι μια συνάρτηση που καλείται όταν η συσκευή θέλουμε να αποσυνδεθεί από το σύστημα και η **int (*probe)(struct virtio_device *)**, η οποία καλείται όταν βρίσκεται μια συσκευή με file descriptor ο οποίος υπάρχει στο πεδίο **id_table**. Η συνάρτηση αυτή είναι πολύ σημαντική γιατί μέσα εκεί αρχικοποιούμε όλα τα πεδία του struct της συσκευής. Συγκεκριμένα αρχικοποιούμε ένα spinlock το οποίο θα μας είναι χρήσιμο για να επιλύσουμε κάποια race conditions όταν θα είμαστε σε interrupt context, βρίσκουμε τα virtqueues με τα οποία θα επικοινωνήσει ο frontend με τον backend, αποθηκεύουμε τον minor number της συσκευής και προσθέτουμε την συσκευή στη λίστα με τις συσκευές την οποία διατηρεί ο driver. Η δουλεία που καλείται να κάνει ο ο virtio driver είναι να παίρνει τα αιτήματα ενός προγράμματος που τρέχει μέσα στο vm(πχ ioctl(), open(), close(), read()) και να τα προωθεί μέσω του backend οδηγού στο host ώστε να υλοποιηθούν τα αιτήματα και να επιστρέψει πίσω το αποτέλεσμα. Δίνει την εντύπωση οτι η δουλεία γίνεται από αυτόν αλλά το μόνο που κάνει είναι να προωθεί τα κατάλληλα δεδομένα και να λαμβάνει τα επεξεργασμένα. Κάτι τέτοιο δεν είναι τόσο απλό όσο ακούγεται γιατί χρησιμοποιούνται πολλές διαφορετικές δομές και τα λάθη δεν είναι πάντα προφανή. Ας ξεκινήσουμε από την συνάρτηση **crypto_chrdev_init()**. Αρχικά καλείται η **cdev_init()** για να αρχικοποιήσει μια char device με διαθέσιμες συναρτήσεις αυτές που ορίζονται στο **static struct file_operations crypto_chrdev_fops**. Όπως είδαμε στην 2η εργαστηριακή άσκηση αναλυτικά αρχικά με την συνάρτηση **MKDEV(CRYPTO_CHRDEV_MAJOR, 0)** παίρνεις το device number της συσκευής, μετά δεσμεύεις χώρο με την **register_chrdev_region()** και τέλος ενημερώνεις τον πυρήνα για την ύπαρξη της νέας συσκευής με την **cdev_add()**. Αν θέλετε παραπάνω εξήγηση πάνω σε αυτά παρακαλώ

δείτε την αναφορά μου για την 2η άσκηση. Τελικά οι συναρτήσεις που ορίζονται μέσα στο **static struct file_operations crypto_chrdev_fops** είναι αυτές τις οποίες ο driver μας θα πρέπει να υλοποιήσει(**crypto_chrdev_open()**, **crypto_chrdev_release()**, **crypto_chrdev_read()**, **crypto_chrdev_ioctl()**). Η **crypto_chrdev_open()** καλείται όταν ενα userspace programm(πχ test_crypto) θέλει να κάνει open() την crypto device(η οποία όμως δεν είναι πραγματική αλλά εικονική). Επομένως όταν καλείται η **crypto_chrdev_open()** αρχικά με χρήση του minor number του inode καλούμαι την **get_crypto_dev_by_minor()**, στην οποία ψάχνουμαι να βρούμε στην λίστα με τις συσκευές τη συσκευή με αυτό το minor number. Θα πρέπει να τονίσω ότι όσο ψάχνω θα πρέπει να έχω κλειδώσει, με χρήση spinlock, αφού σε περίπτωση που δεν το έκανα, τότε κάποιος θα μπορούσε να διέγραψε την συσκευή και να προέκυπτε λάθος. Αν υποθέσουμε ότι βρήκαμε την συσκευή και έχουμε πλέον ενα pointer στο **struct crypto_device** θα πρέπει να χρησιμοποιήσουμε scatter lists για να καταφέρουμε DMA μεταφορές δεδομένων από συσκευές I/O. Τ δεδομένα τα οποία θέλω να στείλω στον backend οδηγό χωρίζονται σε δυο διαφορετικές κατηγορίες, αυτά τα οποία στέλνω και θέλω να επεξεργαστούν(num_in) και αυτά τα οποία τα στέλνω για να χρησιμοποιηθούν αλλα να μην επεξεργαγαστούν(num_out). Επομένως θα χρησιμοποιήσω συνολικά (num_in + num_out) το πλήθος scatter lists και αυτό ισχύει για κάθε συνάρτηση που θα πρέπει να υλοποιήσω. Συγκεκριμένα στην **crypto_chrdev_open()** χρησιμοποιώ δυο scatter lists και στην μια βάζω το syscall_type(open) το οποίο είναι num_out και στην άλλη το host_fd(το αρχικοποιώ στο -1) το οποίο είναι num_in και θέλω να μου επιστρέψει το file descriptor του host crypto device(/dev/crypto). Η εισαγώγη τους στις scatter lists γίνεται με την χρησιμοποίησει της συνάρτησης **sg_init_one()**. Θα πρέπει να προσθέσω ότι διατηρώ και ενα πίνακα δυο θέσεων(sgs), στον οποίο κάθε κελί είναι ενας pointer στις παραπάνω scatter lists. Όταν θέλω να βάλω τα δεδομένα στην virtqueueue κλειδώνω με χρήση ενός spinlock αφού δεν θέλω να μπει κάποιος άλλος και καλώ την συνάρτηση **virtqueueue_add_sgs()** η οποία βάζει τα δεδομένα στην virtqueueue με την οποία επικοινωνεί με τον backend και για να το κάνει αυτό χρησιμοποιεί τον παραπάνω πίνακα sgs. Έπειτα με την συνάρτηση **virtqueueue_kick()** ενημερώνει τον backend ότι τα δεδομένα έχω τοποθετηθεί και μπορεί να τα επεξεργαστεί(εδω στην ουσία γίνεται trap). Στη συνέχεια ο frontend οδηγός κάνει busy wait και περιμένει ο backend να γράψει κάτι στον buffer(**virtqueueue_get_buf()**) για να ξεκλειδώσει και μετά να αποθηκεύσει το host_fd που του έστειλε ο backend. Η **crypto_chrdev_release()** λειτουργεί με τον ίδιο τρόπο με την **crypto_chrdev_open()**, δηλαδή διατηρεί δυο scatter lists και τον πίνακα sgs με την διαφορά ότι οι δύο scatter lists έχουν και οι δύο num_out δεδομένα, δηλαδή αυτή η συνάρτηση δεν στέλνει δεδομένα για επεξεργασία αλλά μόνο για να γίνει αποδέσμευση της συσκευής. Στην μια scatter list βάζω εκ νέου το syscall_type και στην άλλη το host_fd(τώρα είναι num_out) με την συνάρτηση **sg_init_one()**. Βάζω τα δεδομένα στην virtqueueue με την **virtqueueue_add_sgs()** αφού πρώτα κλειδώσω. Έπειτα κάνω busy wait και τελικά αποδεσμεύω το open file το οποίο είχα για την crypto συσκευή. Η **crypto_chrdev_read()** δεν κάνει κάτι γι αυτό και δεν έχω γράψει κάτι σ'αυτή αφού δεν μας ζητείται. Η τελευταία συνάρτηση που υλοποιώ είναι η **crypto_chrdev_ioctl()** η οποία παίρνει και ενα επιπλέον όρισμα, το οποίο είναι το request target(είναι το δεύτερο όρισμα με το οποίο καλείται και στον κώδικα μου έχει όνομα cmd). Η λογική η οποία ακολουθεί είναι πάλι η χρήση scatter lists για DMA, αλλά ανάλογα με το cmd θα πρέπει να στείλω

διαφορετικά num_in και num_out επομένως χρησιμοποιώ την δομή case-switch για να ελέγξω το όρισμα cmd αφού έχει τρεις πιθανές τιμές με τις οποίες μπορεί να κληθεί(**CIOCGSESSION**, **CIOCFSESSION**, **CIOCCRYPT**).

Όποια και αν είναι η τιμή του cmd, σίγουρα το πρώτο num_out θα είναι και πάλι το syscall_type, το δεύτερο num_out θα είναι το host_fd και το τρίτο προφανώς το cmd. Για το λόγο αυτό τα βάζω πριν το case-switch σε μια scatter list το καθένα με την συνάρτηση **sg_init_one()**. Αν τώρα cmd=**CIOCGSESSION** θέλω να αρχίσει ενα νεό session. Για το λόγο αυτό αρχικά κάνω χρήση της συνάρτησης **copy_from_user()** για να πάρω ενα pointer στο struct session_op. Ο λόγος για τον οποίο χρησιμοποιήσα την **copy_from_user()** δεν έκανα απλή ανάθεση είναι οτι υπάρχει διαφορετικό address space μεταξύ user-space και kernell-space, δηλαδή αν χρησιμοποιούσα απλή ανάθεση η διεύθυνση που έδειχνε ο pointer στο user-space θα ήταν σκουπίδια για τον kernell και θα είχα πρόβλημα. Για τον ίδιο λόγο χρησιμοποιώ και την **copy_from_user()** και σε άλλα σημεία του κώδικα. Η συνέχεια είναι η αναμενόμενη αφού χρησιμοποιώ scatter lists για να στείλω τα απαιτούμενα data(και τα num_in και τα num_out). Θεωρώ οτι δεν χρειάζεται να σας αναφέρω αναλυτικά όλες τις scatter lists αφού είναι αρκετές και μπορείτε να τις δείτε μέσα στον κώδικα. Αν τώρα cmd=**CIOCFSESSION**, η λογικη είναι η ίδια, δηλαδή χρησιμοποιώ την **copy_from_user()** για να πάρω ενα pointer στο αντίστοιχο session(ο λόγος χρησιμοποίησης της **copy_from_user()** είναι ο ίδιος με πριν) και στη συνέχεια βάζω σε μια scatter list αυτό που πήρα από την **copy_from_user()** ως num_out και σε μια άλλη το return value σαν num_in για να ελέγξω στη συνέχεια αν πέτυχε το close του session. Τέλος, αν cmd=**CIOCCRYPT** θα πρέπει να γίνει είτε κρυπτογράφηση είτε αποκρυπτογράφηση(ανάλογα με το πεδίο cryp.op). Και στη περίπτωση αυτή θα χρησιμοποιήσω την **copy_from_user()** για να πάρω τους απαιτούμενους pointers για να μπορέσω να κάνω σωστά την (απο)κρυπτογράφηση. Έπειτα βάζω σε scatter lists όλα τα δεδομένα τα οποία είναι απαραίτητα για να γίνει η (απο)κρυπτογράφηση με χρήση της **sg_init_one()** αλλά δεν τα αναφέρω αναλυτικά επειδή είναι πολλά(ειδικά τα num_out). Ενδεικτικά αναφέρω οτι πάλι σαν num_in είναι το return value για να δω αν πέτυχε η λειτουργία που έχω να κάνω και σαν δεύτερο num_in είναι το αποτέλεσμα της (απο)κρυπτογράφησης. Αφού λοιπόν έβαλα στις κατάλληλες scatter lists τα δεδομένα τα οποία πρέπει ανάλογα με το cmd, κλειδώνω με χρήση του spinlock, χρησιμοποιώ την **virtqueue_add_sgs()** για να βάλω τα data μέσα στην αντίστοιχη virtqueue και μετά κάνω busy wait μέχρι να γραφτούν data στον buffer, τσεκάροντας κάθε φορά με την συνάρτηση **virtqueue_get_buf()**. Όταν τελειώσει το busy wait, αρχικά ξεκλειδώνω και μετά θα πρέπει να βάλω τα νεα data στις κατάλληλες θέσεις ανάλογα με την τιμή του cmd. Και στις τρεις περιπτώσεις τσεκάρω την μεταβλητή ret_val_host και αν είναι αρνητική σημαίνει οτι δεν έγινε σωστά η λειτουργία και κάνω return -1. Αν τώρα cmd = **CIOCGSESSION** θα χρησιμοποιήσω την συνάρτηση **copy_to_user()** για τον ίδιο λόγο για τον οποίο χρησιμοποιήσα την **copy_from_user()**. Έτσι αντιγράφω σωστά το pointer σε struct session_op. Αν τώρα cmd = **CIOCFSESSION** δεν κάνω καμία άλλη λειτουργία εκτός απο αυτή που ανέφερα οτι γίνεται και στις τρεις περιπτώσεις. Τέλος, αν cmd = **CIOCCRYPT** χρησιμοποιώ την **copy_to_user()** για να μπορέσω να περάσω σωστά το αποτέλεσμα της (απο)κρυπτογράφησης. Όπως εξήγησα και παραπάνω η χρήση της **copy_to_user()** είναι αναγκαία λόγω του διαφορετικού address space μεταξύ user και kernell space.



- **virtio-crypto.c (Backend)** : O backend οδηγός φορτώνεται στο qemu και επικοινωνεί με τον frontend με χρήση virtqueues όπως είναι ήδη γνωστό. Όταν γίνει trap ο backend οδηγός καταλαβαίνει οτι η virtqueue έχει πλέον δεδομένα τα οποία πρέπει να τα επεξεργαστεί(το trap γίνεται όταν ο frontend καλέσει την kick). Τότε καλέίται ο handler για το γεγονός αυτό και τελικά καλέίται η συνάρτηση **vq_handle_output()** που παίρνει σαν ορίσματα την εικονική συσκευή και την virtqueue. Αυτό που κάνει αρχικά είναι να εκτελεί την **virtqueue_pop(vq, &element)** για να πάρει αυτό που έχει μέσα η virtqueue. Μετά θα πάρει το πρώτο data(1o num_out data) το οποίο είναι σε κάθε περίπτωση το syscall_type το οποίο θέλουμε να ικανοποιήσουμε. Στη συνέχεια ακολουθεί ένα case-switch για το είδος του system call. Αν είναι open τότε ο backend θα πρέπει να ανοίξει την συσκευή */dev/crypto* και να επιστρέψει τον file descriptor της. Για την ακρίβεια δεν επιστρέφει τον file descriptor αλλα αποθηκεύει τον file descriptor στην αντίστοιχη scatter list η οποία κρατά το host_fd, η οποία περιμένει να αλλάξει αφού είναι num_in. Αυτό γίνεται με χρήση της memcpys() αλλά θα μπορούσε να γίνει και με απλή ανάθεση. Αν τώρα το syscall_type είναι close τότε παίρνω τον file descriptor της συσκευής τον οποίο έστειλα με την δεύτερη scatter list και απλά καλώ την close για να κλείσω την συσκευή με αυτον τον file descriptor. Τέλος, αν το syscall_type είναι ioctl τότε παίρνω τον file descriptor της συσκευής κρυπτογράφησης και το request(το cmd που έστειλα) το οποίο θα δηλώνει το target request του ioctl(). Αν το request(το cmd που έστειλα) είναι **CIOCGSESSION** τότε παίρνω απο την ουρά όλα όσα έστειλα μέσω scatterlists και τα βάζω στα σωστά πεδία ώστε να πετύχει το **ioctl(*fd_crypt, CIOCGSESSION, sess)**. Τα πεδία δεν τα ανέφερα αναλυτικά γιατί είναι αρκετά. Στη συνέχεια γράφω πίσω στις κατάλληλες scatter lists, οι οποίες είναι μέσα στην virtqueue το return value και το session(το καθένα γράφεται στη δική του scatter list). Αν το request(το cmd που έστειλα)

είναι **CIOCFSESSION** τότε παίρνω από την virtqueue το session το οποίο θέλω να τερματίσω και εκτελώ την **ioctl(*fd_crypt, CIOCFSESSION, sess_ses)** ώστε να τερματίσω το αντίστοιχο session και τελικά γράφω την return value της εντολής αυτής στην αντίστοιχη scatter list της virtqueue για να ενημερώσω τον frontend αν τελικά πέτυχε ή οχι. Η τελευταία περίπτωση είναι το request(το cmd που έστειλα) να είναι **CIOCCRYPT** στο οποίο κάνω τα ίδια με τις παραπάνω περιπτώσεις με την διαφορά ότι παίρνω(βγάζω) από την virtqueue περισσότερα data αφού μου χρειάζονται περισσότερα για να κάνω την (απο)κρυπτογράφηση. Τα δεδομένα αυτά τα βάζω στα κατάλληλα πεδία(δεν τα αναφέρω αναλυτικά αφού μπορείτε να τα δείτε στον κώδικα) και καλώ την **ioctl(*fd_crypt, CIOCCRYPT, cryp)** για να γίνει η (απο)κρυπτογράφηση ανάλογα με το operation το οποίο έστειλα. Μετά από αυτό αντιγράφω το return value της παραπάνω εντολής στην αντίστοιχη scatter list της virtqueue και το αποτέλεσμα της (απο)κρυπτογράφησης στην άλλη scatter list ώστε να πάρει μετά ο frontend το σωστό αποτέλεσμα. Αφού λοιπόν τελειώσει και το case-switch μένει μόνο να βάλω στην virtqueue τα νέα δεδομένα με την εντολή **virtqueue_push()** και να ενημερώσω τον frontend ότι πλέον μπορεί να πάρει τα δεδομένα από την virtqueue με την εντολή **virtio_notify()**.

Παρατήρηση : Ο κώδικας **crypto-module.c** δεν σας παραδόθηκε διότι η μόνη αλλάγη που έγινε σ' αυτόν ήταν η αρχικοποίηση και η χρησιμοποίηση ενός spinlock στη συνάρτηση **virtcons_probe()**, το οποίο ήταν προφανές και γι' αυτό επέλεξα να μην σας τον παραθέσω. Για πληρότητα σας παραθέτω μόνο την συνάρτηση αυτή:

```
static int virtcons_probe(struct virtio_device *vdev)
{
    int ret = 0;
    struct crypto_device *crdev;
    debug("Entering");
    crdev = kzalloc(sizeof(*crdev), GFP_KERNEL);
    if (!crdev) {
        ret = -ENOMEM;
        goto out;
    }
    /* kanw to match anamesa stis suskeues */
    crdev->vdev = vdev;
    vdev->priv = crdev;

    crdev->vq = find_vq(vdev);
    if (!(crdev->vq)) {
        ret = -ENXIO;
        goto out;
    }

    /* Other initializations - The only change here is the spinlock
       initialization */
    spin_lock_init(&crdev->lock);
    /*Grab the next minor number and put the device in the driver's list.*/
    spin_lock_irq(&crdrvdata.lock);
    crdev->minor = crdrvdata.next_minor++;
    list_add_tail(&crdev->list, &crdrvdata.devs);
    spin_unlock_irq(&crdrvdata.lock);
    debug("Got minor = %u", crdev->minor);
    debug("Leaving");
out:
    return ret;
}
```

Ενδεχόμενη Βελτίωση Γενικής Ιδέας Άσκησης: Στο σημείο αυτό θα ήθελα να σας αναφέρω μια ιδέα μου για βελτίωση της απόδοσης του κώδικα αυτού. Η ιδέα η από ακολουθείται εδώ είναι κάθε φορά που ένα θέλουμε να γίνει μια ενέργεια, οι υπόλοιποι που ζητούν να γίνει κάτι περιμένουν τη σειρά τους, δηλαδή παρατηρείται μια σειριοποίηση όσο αφορά τα αιτήματα. Κάτι τέτοιο φαντάζει κακό αφού αν υποθέσουμε οτι υπάρχουν για παράδειγμα τρια αιτήματα για κρυπτογράφηση και το πρώτο περιέχει ενα πολύ μεγάλο κείμενο ενω τα άλλα δυο είναι πολύ μικρά θα πρέπει να μικρά να περιμένουν το πολύ μεγάλο ενω θα ήταν καλό να τελείωναν πρώτα και όχι να το περιμένουν. Για να γίνει αυτό θα έπρεπε να γίνει χρήση σεμαφόρου για το κλείδωμα και για να μπορέσω να ελέγξω πότε κάποια δεδομένα είναι έτοιμα και σε ποιόν ανήκουν, θα έπρεπε να στέλνω κάθε φορά με scatterlist το pid της διεργασίας η οποία περιμένει αποτελέσματα, δηλαδή κάθε φορά που θα τελείωνε κάποιο αίτημα θα πήγαινε και επέστρεφε το αποτέλεσμα ανάλογα με το pid. Για να γίνει πράξη η ιδέα αυτή θα έπρεπε να γράψω κώδικα μέσα στη συνάρτηση static void vq_has_data(), η οποία στην άσκηση σας δεν κάνει κάτι στην ουσία και βρίσκεται στο αρχείο crypto-module.c. Σίγουρα η ιδέα αυτή δεν είναι πλήρης αλλά νομίζω οτι είναι προς την σωστή κατεύθυνση για την απαλοιφή της σειριοποίησης που ανέφερα παραπάνω.

Για να ελέγξω οτι ο κώδικας μου είναι σωστός θα πρέπει αρχικά να βάλω στο directory `~/qemu-2.0.0` τα αρχεία που βρίσκονται στο φάκελο `qemu` του `helpcode` μαζί με το υλοποιημένο **`virtio-crypto.c`**. Στο directory αυτό έπρεπε να ανοίξω ενα terminal και να κάνω διαδοχικά `make` και `sudo make install`. Έπειτα πάω στο directory `~/utopia` και εκτελώ την εντολή **`./utopia.sh -device virtio-crypto-pci`** και μετά ανοίγω ενα άλλο terminal για να συνδεθώ στο `vm`. Αφού κάνω το `sshfs` και πάω στο φάκελο `guest`, κάνω `make` στο φάκελο αυτό, μετά με την εντολή **`insmod ./virtio_crypto.ko`** εισάγω το module και με την **`./crypto_dev_nodes.sh`** φτιάχνω τα nodes. Για να ελέγξω τη σωστή λειτουργία τρέχω την **`./test_crypto`** και το **`./test_crypto_fork`** και αν πάρω `success` σημαίνει οτι ο κώδικας μου είναι σωστός. Ενα screenshot μετά από όλη αυτή τη διαδικασία το οποίο δείχνει οτι ο κώδικας μου είναι σωστός φαίνεται παρακάτω :

[] ~Terminal~gvavou5-

~Terminal~gvavou5~

