ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εργαστήριο Λειτουργικών Συστημάτων: 3η Εργαστηριακή Άσκηση
Ομάδα 40

Ονοματεπώνυμο:	Τόφαλος Φίλιππος	Χρήστος Τσούφης
Αριθμός Μητρώου:	03117087	03117176
Ημερομηνία Επίδειξης:	22 – 01 – 2021	
Ημερομηνία Παράδοσης:	31 – 12 – 2021	

Ζήτημα Z1

Υλοποίηση του Server

Επιλέχτηκε για τον εξυπηρετητή να υλοποιηθεί ένα μοντέλο στο οποίο αυτός μπορεί να διαχειρίζεται ταυτόχρονα μέχρι ένα όριο 20 πελατών (αυτό γίνεται εφικτό διατηρώντας μια λίστα με ανοικτά αρχεία – sockets που αντιστοιχούν στον κάθε πελάτη, η εξέταση της οποίας για νέα γεγονότα στους επιμέρους file descriptors πραγματοποιείται σύγχρονα με την χρήση της συνάρτησης select). Ο ίδιος ο εξυπηρετητής δεν παρεμβαίνει στο περιεχόμενο της συζήτησης, αλλά για κάθε εισερχόμενο (ενδεχομένως κρυπτογραφημένο) μήνυμα που λαμβάνει προσθέτει, μέσω της συνάρτησης reformat_message, στην αρχή αυτού το PID του αποστολέα, ύστερα από τον ειδικό χαρακτήρα '\0' που χρησιμοποιείται για την αναγνώριση της αρχής ενός μηνύματος. Αυτή η δομή έχει αποδοθεί σχηματικά στα σχόλια πάνω από την συνάρτηση reformat_message.
Το PID έχει γνωστοποιηθεί στον εξυπηρετητή για κάθε εισερχόμενη σύνδεση με μια διαδικασία “χειραψίας” που πραγματοποιείται στην διάρκεια αυτής, όπως φαίνεται στην υλοποίηση της συνάρτησης accept_new_client. Αυτή έχει ως εξής: εφόσον μετά την select στην κύρια συνάρτηση έχει διαπιστωθεί ότι το master_socket (υπεύθυνο για τις εισερχόμενες συνδέσεις) είναι έτοιμο για ανάγνωση, δηλαδή ότι ένας νέος πελάτης αναμένει να συνδεθεί, τότε τον αποδεχόμαστε μέσω της accept και αποθηκεύουμε το ανάλογο fd στην μεταβλητή new_socket, αναμένουμε να διαβάσουμε από τον ίδιο το PID του (pid_size = read(*new_socket, pid, PID_SIZE)), και εάν αυτό ληφθεί του στέλνουμε ένα μήνυμα "ACK" με το οποίο εν τέλει επιβεβαιώνουμε την σύνδεση. Στην συνέχεια αποθηκεύουμε σε ανάλογους πίνακες (και στην ίδια διαθέσιμη θέση στον καθένα) το αναγνωριστικό του νέου socket και το PID που λάβαμε από τον πελάτη. Για την αποφυγή σεναρίων στα οποία ο πελάτης για κάποιο λόγο αδυνατεί να απαντήσει, το οποίο θα οδηγούσε σε δυσλειτουργία του εξυπηρετητή λόγω αναμονής του αναγνωριστικού, μέσω της setsockopt στην κύρια συνάρτηση έχει τεθεί ένα όριο αναμονής ενός δευτερολέπτου. Σημειώνεται ότι είναι το ίδιο εφικτό και πιο εύκολο στην υλοποίηση να στέλνει ο ίδιος ο client το PID του κάθε φορά, για κάθε μήνυμα, αλλά αυτό στην πραγματικότητα παρουσιάζει προβλήματα καθώς υποθετικά θα μπορούσε κάποιος να επηρεάσει την λειτουργία του client ώστε να στέλνει κάθε φορά διαφορετικό PID και ενδεχομένως να προσποιηθεί κάποιον άλλον χρήστη, το οποίο θα έκανε την ταυτοποίηση τους δύσκολη (όσον αφορά το σενάριο χρήσης διεργασιών στο ίδιο σύστημα).
Σε περίπτωση τώρα που μετά την select διαπιστωθεί ότι κάποιο από τα αναγνωριστικά των πελατών είναι έτοιμο προς ανάγνωση, τότε καλούμε την συνάρτηση handle_client για το συγκεκριμένο αναγνωριστικό, στην οποία συνάρτηση είτε θα συνειδητοποιήσουμε την αποσύνδεση του πελάτη, οπότε θα ενημερώσουμε μέσω της εξόδου του server για το γεγονός αυτό (και θα κλείσουμε το ανάλογο socket, μηδενίζοντας ύστερα και τις ανάλογες καταχωρήσεις στους πίνακες client_socket και client_pid), είτε θα ληφθεί ένα μήνυμα (valread) το οποίο θα αναδιαμορφωθεί

στην reformat_message (και αναλόγως με τις επιλογές μεταγλώττισης για τα προγράμματα εξυπηρετητή και πελάτη, που χρειάζεται να είναι κοινές, ενδέχεται να πραγματοποιηθεί κρυπτογράφηση του προστιθέμενου header, με το κοινό hardcoded κλειδί των δύο προγραμμάτων, όπως θα αναλυθεί στο επόμενο ζήτημα) και ύστερα θα σταλεί σε όλους τους ενεργούς πελάτες. Τα παραπάνω αποτιμούν σε έναν βαθμό τις σχεδιαστικές επιλογές που πραγματοποιήθηκαν για την λειτουργία του εξυπηρετητή. Δεν εμβαθύναμε πολύ στην ανάλυση του κώδικα εφόσον οι περισσότερες εντολές (πέρα της λειτουργίας της select που γνωστοποιήθηκε από τα ανάλογα man pages) και η χρήση τους είχε αναλυθεί στον αντίστοιχο οδηγό. Στην συνέχεια παρατίθεται και η αντίστοιχη υλοποίηση.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <errno.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/time.h>
#include "message_enc.h"

#define TRUE   1
#define FALSE  0
#define PORT 8080

#define PID_SIZE 7
#define MAX_CLIENTS 20
#define BUFFER_SIZE 1024


/*                  MESSAGE FORMAT BEFORE FORMATTING
 *
 *          -----------------------------
 *          | MESSAGE (SIZE IS MULTIPLE |
 *          | OF AES_BLOCK_SIZE         |
 *          -----------------------------
 *
 *                  MESSAGE FORMAT AFTER FORMATTING
 *
 *  <-------------------------> (AES_BLOCK_SIZE)
 *  ------------------------------------------------------------
 *  | '\0'    |  PROCESS PID | MESSAGE (SIZE IS MULTIPLE |
 *  | (1 bit) |  (15 bits)   | OF AES_BLOCK_SIZE         |
 *  ------------------------------------------------------------
 */

int reformat_message(int client_pid, char* buffer, int valread, int cfd) {
        int i;
        char pid [AES_BLOCK_SIZE];
        int enc_size = AES_BLOCK_SIZE;
        memset(pid, 0x0, sizeof(pid));

        /* Compute the pid and append the character '\0'
         * (used to identify the beginning of the message) */
        sprintf(pid, "%d", client_pid);
        for (i = AES_BLOCK_SIZE - 2; i >= 0; i--)
                pid[i+1] = pid[i];
        pid[0] = '\0';

        /* We must encrypt the added header (with the same key) so that when
         * decrypted from the the client, we get all the information from the
         * message without any extra effort */
```

```
        #if (USE_ENC == 1)
        encrypt_decrypt_message (ENCRYPT, pid, &enc_size, cfd);
        #endif

        for (i = valread-1; i >= 0; i--)
                buffer[i + AES_BLOCK_SIZE] = buffer[i];

        for (i = 0; i < AES_BLOCK_SIZE; i++)
                buffer[i] = pid[i];

        /* Return the new message size*/
        return valread + AES_BLOCK_SIZE;

}

void accept_new_client (int master_socket,
                        int client_socket[MAX_CLIENTS],
                        pid_t client_pid [MAX_CLIENTS]) {

        char *pid, *end;
        int pid_size, result, i;
        struct sockaddr_in address;
        int addrlen;
        int new_socket;

        /* Accept the new connection */
        if ((new_socket = accept(master_socket,
                                 (struct sockaddr *) &address,
                                 (socklen_t*) &addrlen)) < 0) {
                perror("accept");
                exit(EXIT_FAILURE);
        }

        pid = calloc(PID_SIZE, sizeof(char));
        pid_size = read(new_socket, pid, PID_SIZE);

        if (pid_size == 0) {
                printf("Failed to receive the client's PID, connection aborted\n");
                close(new_socket);
                return;
        }

        /* Send Acknowledgement */
        send(new_socket, "ACK", 3, 0);

        result = strtol(pid, &end, 10);

        if (end == pid) {
                printf("Could not convert the received PID");
                close(new_socket);
                return;
        }

        /* Inform about the user of the socket number */
        printf("New connection: [Socket FD :: %d ] [IP :: %s] [PID :: %d]\n",
                   new_socket, inet_ntoa(address.sin_addr), result);

        /* Add new socket to array of sockets */
        for (i = 0; i < MAX_CLIENTS; i++) {
                if( client_socket[i] == 0 ) {
                        client_socket[i] = new_socket;
                        client_pid[i] = (pid_t) result;
                        printf("Adding to list of sockets as %d\n" , i);
                        return;
                }
        }

        close(new_socket);
```

```
        printf("Cannot accept any more clients [Limit: %d clients]\n",
               MAX_CLIENTS);
}

void handle_client (int index, int cfd,
                    int client_socket[MAX_CLIENTS],
                    pid_t client_pid [MAX_CLIENTS]) {

        int valread, j, sd_r;
        char buffer[BUFFER_SIZE];
        struct sockaddr_in address;
        int addrlen;

        if ((valread = read(client_socket[index], buffer, BUFFER_SIZE)) == 0) {
                /* A client disconnected */
                /* We use getoeername() to get the client's inforamtion
                 * given his FD */
                getpeername(client_socket[index], (struct sockaddr*) &address,
                            (socklen_t*) &addrlen);
                printf("Host disconnected: [IP :: %s] [PID :: %d]\n",
                                inet_ntoa(address.sin_addr) , client_pid[index]);

                /* Close the socket and mark as 0 in list for reuse */
                close(client_socket[index]);
                client_socket[index] = 0;
                client_pid[index] = 0;
        } else {
                valread = reformat_message(client_pid[index], buffer, valread,cfd);
                for (j = 0 ; j < MAX_CLIENTS; j++)
                        if(client_socket[j] > 0)
                                send(client_socket[j], buffer , valread , 0);
        }
}

int main(int argc , char *argv[])
{
        int opt = TRUE;
        int master_socket , addrlen , new_socket , client_socket[MAX_CLIENTS],
                activity, i, j, sd, sd_r;
        pid_t client_pid [MAX_CLIENTS];
        int max_sd, cfd;
        struct sockaddr_in address;

        cfd = open("/dev/crypto", O_RDWR, 0);

        /* Set of socket descriptors */
        fd_set readfds;

        /* Initialise all the client sockets */
        for (i = 0; i < MAX_CLIENTS; i++)
                client_socket[i] = 0;

        /* Create a master socket (The one listening for new
         * connections) */
        if((master_socket = socket(AF_INET , SOCK_STREAM , 0)) == 0) {
                perror("socket failed");
                exit(EXIT_FAILURE);
        }

        /* Set master socket to allow multiple connections */
        if(setsockopt(master_socket, SOL_SOCKET, SO_REUSEADDR,
                                (char *)&opt, sizeof(opt)) < 0 ) {
                perror("setsockopt");
                exit(EXIT_FAILURE);
        }

        /* We set a time limit for the read operations in case the pid
         * authentication fails */
```

```
struct timeval tv;
tv.tv_sec = 1;
tv.tv_usec = 0;

if( setsockopt(master_socket, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv,
                        sizeof tv) < 0 ) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
}

/* Type of socket created */
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons( PORT );

/* Bind the socket to localhost port 8888 */
if (bind(master_socket, (struct sockaddr *)&address, sizeof(address))<0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
}
printf("Listener on port %d \n", PORT);

/* Specify a maximum of 3 pending connections
 * for the master socket */
if (listen(master_socket, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
}

/* Accept the incoming connection */
addrlen = sizeof(address);
puts("Waiting for connections ...");

while(TRUE) {
        /* Clear the socket set */
        FD_ZERO(&readfds);

        /* Add master socket to set */
        FD_SET(master_socket, &readfds);
        max_sd = master_socket;

        /* Add all the bound client sockets */
        for ( i = 0 ; i < MAX_CLIENTS ; i++)
        {
                sd = client_socket[i];

                if(sd > 0)
                        FD_SET(sd , &readfds);

                if(sd > max_sd)
                        max_sd = sd;
        }

        /* Wait for an activity on one of the sockets */
        /* Timeout is NULL, so we wait indefinitely */
        activity = select( max_sd + 1 , &readfds , NULL , NULL , NULL);

        /* If there was an error not caused by a signal,
         * report the incident */
        if ((activity < 0) && (errno!=EINTR))
                printf("select error");

        /* Check the master socket for incoming connections */
        if (FD_ISSET(master_socket, &readfds)) {
                accept_new_client(master_socket, client_socket,
                                  client_pid);
        }
```

```
                /* Else activity was completed beacuse of some operation
                 * regarding a client socket */
                for (i = 0; i < MAX_CLIENTS; i++)
                        if (FD_ISSET(client_socket[i], &readfds))
                                handle_client (i, cfd, client_socket, client_pid);
        }

        close(master_socket);
        close(cfd);
        return 0;
}
```

Σημείωση: Σαν βάση για το παραπάνω πρόγραμμα αξιοποιήθηκε κώδικας που πρόσφερε ένας οδηγός στο διαδίκτυο για χρήση της select προς τον σκοπό της διαχείρισης πολλαπλών πελατών μέσω του socket API, οπότε οποιαδήποτε ομοιότητα με υλοποίηση άλλων ομάδων είναι συμπτωματική και οφείλεται κατά πάσα πιθανότητα σε αυτό.

Υλοποίηση του Client

Για την υλοποίηση του προγράμματος του πελάτη επιλέχθηκε να χρησιμοποιηθεί η βιβλιοθήκη ncurses προκειμένου να κατασκευαστεί ένα "γραφικό" interface εντός του τερματικού το οποίο μοιάζει με αυτό αντίστοιχων IRC Clients. Αρκετός κώδικας στην συνέχεια (όπως πχ οι συναρτήσεις initialize_window, place_in_window και reset_input_prompt) αφορά το γραφικό περιβάλλον της εφαρμογής, αλλά συμπεριλήφθηκε για σκοπούς πληρότητας της αναφοράς. Όσον αφορά την λειτουργία του προγράμματος, σημειώνεται ότι το μέγεθος του γραφικού του περιβάλλοντος παραμένει το ίδιο και ίσο με το αρχικό σε περίπτωση μεταβολής του μεγέθους του παραθύρου φλοιού που το φιλοξενεί (μια παραδοχή που έγινε για απλούστευση του κώδικα), υπάρχει ένα όριο χαρακτήρων ανά μήνυμα ίσο με 80, επιτρέπεται μόνο η εισαγωγή χαρακτήρων και η διαγραφή από το τέλος (δηλαδή δεν υποστηρίζεται περιήγηση στο μήνυμα με τα βελάκια και τροποποίηση ενδιάμεσων χαρακτήρων), και η έξοδος από το πρόγραμμα πραγματοποιείται με το πλήκτρο F1.
Η λογική που αξιοποιεί ο κορμός του προγράμματος (ότι έχει να κάνει δηλαδή με την επικοινωνία με τον εξυπηρετητή) προκύπτει σχεδόν άμεσα από την ανάλυση στον οδηγό που δόθηκε για τους σκοπούς αυτού του ζητήματος, ενώ για ευκολία στην παράλληλη αποδοχή εισόδου από τον χρήστη και παραλαβή νέων μηνυμάτων, αξιοποιήθηκε και εδώ η συνάρτηση select με λίστα εισόδου που περιέχει το ανοικτό socket και το standard input. Σε εκτενέστερη ανάλυση του κώδικα, με την συνάρτηση connect_to_server δημιουργούμε ένα νέο socket μέσω της συνάρτησης socket() (αναφέρεται εδώ ότι αξιοποιείται το address family AF_INET και προσδιορίζεται ως πρωτόκολλο μεταφοράς το TCP), ορίζουμε ένα χρονικό περιθώριο 1 δευτερολέπτου για κάθε ανάγνωση στο socket (το οποίο προσδιορίζεται κατά κύριο λόγο για την περίπτωση αποτυχίας της διαδικασίας ταυτοποίησης, ώστε να μην κολλήσει το πρόγραμμα του πελάτη, αλλά ίσως να αποτελούσε καλύτερη σχεδιαστική επιλογή η ύστερη αναίρεση αυτής της επιλογής ή η αύξηση του ανάλογου χρονικού διαστήματος), και επιχειρούμε σύνδεση στον εξυπηρετητή στην διεύθυνση που δόθηκε ως παράμετρος κατά την εκτέλεση του προγράμματος του πελάτη. Εφόσον πραγματοποιηθεί επιτυχώς η σύνδεση, τότε στέλνουμε το pid της αντίστοιχης διεργασίας (ισοδύναμο της διαδικασίας "εγγραφής" στην υπηρεσία που παρέχει ο εξυπηρετητής), και εξετάζουμε αν λάβαμε την ζητούμενη επιβεβαίωση (μήνυμα ACK). Από την στιγμή που θα επιτευχθεί η σύνδεση, είμαστε σε θέση να λάβουμε οποιοδήποτε μήνυμα προωθεί ο εξυπηρετητής (το οποίο εμφανίζουμε στην οθόνη μέσω της place_in_window σεβόμενοι την αναδιαμόρφωση του μηνύματος που πραγματοποιεί ο εξυπηρετητής), είτε να γράψουμε οποιοδήποτε μήνυμα θέλουμε (με τους περιορισμούς που αναφέρθηκαν παραπάνω) και να το στείλουμε εν τέλει στον εξυπηρετητή με το πάτημα του πλήκτρου enter.

```c
#include <ncurses.h>
#include <stdbool.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <locale.h>
#include <ctype.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include "message_enc.h"

#define MAX_CH 80
#define PORT 8080
#define PID_SIZE 7
#define DELAY 35000
#define MIDDLE 9
#define BUFFER_SIZE 1024

#ifndef MAX
#define MAX(X,Y) ((X) > (Y) ? (X) : (Y))
#endif

#define USE_BANNER 0

/* OSX3c = OSlab eXercise 3 chat */
char *banner =
"  ___  ______  _______                \n"
" / _ \\/ ___\\ \\/ /___ /  ___   \n"
"| | | \\___ \\\\  /  |_ \\ / __| \n"
"| |_| |___) /  \\ ___) | (__     \n"
" \\___/|____/_/\\_\\____/ \\___| \n";

int msg_stack_top = USE_BANNER*6;
bool window_full = 0;

void connect_to_server(int* sockfd, char* ip) {
        pid_t client_pid;
        int pid_str_size;
        char * pid_str;
        int recv;
        char rcv[BUFFER_SIZE];
        struct sockaddr_in servaddr;

        /* Get the PID and convert it to a string
         * to be sent to the server and be used as
         * an identifier */
        client_pid = getpid();
        pid_str = calloc(PID_SIZE, sizeof(char));
        pid_str_size = sprintf(pid_str, "%d", client_pid);

        /* Create the needed socket */
        *sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (*sockfd == -1) {
                printf("socket creation failed...\n");
                exit(0);
        }
        else {
                printf("Socket successfully created..\n");
        }

        /* We set a time limit for the read operations in case the pid
         authentication fails */
        struct timeval tv;
        tv.tv_sec = 1;
        tv.tv_usec = 0;
```

```
        if( setsockopt(*sockfd, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv,
                                sizeof tv) < 0 )
        {
                perror("setsockopt");
                exit(EXIT_FAILURE);
        }

        /* Assign the server arguments to the struct
         * used as the parameter of connect */
        bzero(&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        (servaddr.sin_addr).s_addr = inet_addr(ip);
        servaddr.sin_port = htons(PORT);

        /* Attemp to connect to the requested server */
        if (connect(*sockfd, (struct sockaddr*) &servaddr, sizeof(servaddr)) != 0)
{
                printf("connection with the server failed...\n");
                exit(0);
        }
        else {
                printf("connected to the server..\n");
        }

        /* The server will be waiting for the client
         * to send his PID */
        send(*sockfd, pid_str, pid_str_size, 0);

        recv = read(*sockfd, rcv, BUFFER_SIZE);

        if (strncmp(rcv, "ACK", 3)) {
                printf("Registration must have failed\n");
                exit(EXIT_FAILURE);
        }

        free(pid_str);
}

WINDOW* initialize_window (int* max_x, int* max_y) {
        WINDOW* WIN;
        initscr();
        setlocale(LC_ALL, "");
        use_default_colors();
        raw();
        keypad(stdscr, TRUE);
        noecho();

        start_color();
        init_pair(1, -1, -1);
        init_pair(2, COLOR_RED, -1);
        init_pair(3, COLOR_CYAN, -1);

        getmaxyx(stdscr, *max_y, *max_x);
        WIN = newwin(*max_y - 3, *max_x - 3, 1, 2);
        scrollok(WIN, TRUE);

        border('|', '|', '-', '-', '+', '+', '+', '+');

        /* Print Banner */
        if (USE_BANNER) {
                wprintw(WIN, "%s", banner);
        }

        mvprintw(*max_y - 2, 2, "> ");

        refresh();
        wrefresh(WIN);
```

```
        return WIN;
}

void place_in_window (WINDOW* WIN, char buffer[], int buff_lim, int max_y) {
        int i = 0;
        if (msg_stack_top == max_y - 4 && window_full)
                scroll(WIN);

        int extracted_pid = strtol(buffer + 1, NULL, 10);
        bool is_same_pid = (extracted_pid == getpid());

        if (buffer[0] == '\0') {
                wmove(WIN, msg_stack_top, 0);
                wprintw(WIN, "[");

                wattron(WIN, COLOR_PAIR(3 - is_same_pid));

                i = 0;
                for (i = 1; i <= AES_BLOCK_SIZE; i++) {
                        if (buffer[i] == 0x00) break;
                        wprintw(WIN, "%c", buffer[i]);
                }

                wattron(WIN, COLOR_PAIR(1));
                wprintw(WIN, "]");

                wmove(WIN, msg_stack_top, MIDDLE);

                for (i = AES_BLOCK_SIZE; i < buff_lim; i++)
                        if (buffer[i] != 0x00)
                                wprintw(WIN, "%c", buffer[i]);

        } else {
                msg_stack_top--;
                for (i = 0; i < buff_lim; i++)
                        if (buffer[i] != 0x00)
                                wprintw(WIN, "%c", buffer[i]);
        }


        if (msg_stack_top != max_y - 4)
                msg_stack_top++;
        else
                window_full = true;
        wrefresh(WIN);
}

void reset_input_prompt(int* buff_ptr, int max_y, int max_x, int* x) {
        *buff_ptr = 0;

        move(max_y - 2, 0);
        clrtoeol();
        mvprintw(max_y - 2, 0, "|");
        mvprintw(max_y - 2, max_x - 1, "|");
        *x = 2;
        mvprintw(max_y - 2, *x, ">");
        move(max_y - 2, ++(*x) + 1);
        refresh();
}

int main(int argc, char *argv[]) {
        int x, y, max_x, max_y, ch, i, err;
        int msg_stack_top = 0, buff_ptr = 0;
        int out_of_lim = 0;
        char buff[BUFFER_SIZE], rcv[BUFFER_SIZE];
        int recv;
        bool exit_pending = false, server_up = true;
```

```
WINDOW* WIN;
/* Variables needed for the connection to
 * the server */
int sockfd;
struct sockaddr_in servaddr;

int cfd = -1;

if (argc == 1) {
        printf("Please specify the server address\n");
        exit(0);
}

#if (USE_ENC == 1)
/* Open the crypto device */
cfd = open("/dev/crypto", O_RDWR, 0);

if (cfd < 0) {
        err = errno;
        if (err == ENOENT)
                printf("The cryptodev module is not installed\n");
        else
                printf("file open failed: error code %d: %s\n", err,
                       strerror(err));

        exit(0);
}
#endif

/* Connect to the server */
connect_to_server(&sockfd, argv[1]);
/* Initialize everything related to ncurses */
WIN = initialize_window(&max_x, &max_y);
reset_input_prompt(&buff_ptr, max_y, max_x, &x);

y = max_y - 2;
x = 3;

fd_set fds;

struct timeval tv;
tv.tv_sec = 0;
tv.tv_usec = 500000;

int max_fd, status;

max_fd = MAX(sockfd, STDIN_FILENO);

while(!exit_pending && server_up) {
        FD_ZERO(&fds);
        FD_SET(STDIN_FILENO, &fds);
        FD_SET(sockfd, &fds);

        status = select(max_fd + 1, &fds, NULL, NULL, &tv);

        if (FD_ISSET(STDIN_FILENO, &fds)) {
                switch ((ch = getch())) {
                case KEY_F(1):
                        exit_pending = true;
                        break;
                case '\n':
                        #if (USE_ENC == 1)
                        encrypt_decrypt_message(ENCRYPT, buff,
                                                &buff_ptr, cfd);
                        #endif
                        send(sockfd, buff, buff_ptr, 0);
                        reset_input_prompt(&buff_ptr, max_y, max_x, &x);
                        break;
```

```
                        case KEY_BACKSPACE:
                                if (buff_ptr == 0)
                                        break;
                                move(y, x--);
                                printw(" ");
                                buff_ptr--;
                                move(y,x+1);
                                refresh();
                                break;
                        default:
                                if (!isprint(ch) || buff_ptr == MAX_CH)
                                        break;
                                buff[buff_ptr++] = ch;
                                move(y,++x);
                                printw("%c", ch);
                                refresh();
                                break;
                        }
                }

                if (FD_ISSET(sockfd, &fds)) {
                        recv = read(sockfd, rcv, BUFFER_SIZE);
                        if (recv == 0) {
                                server_up = false;
                                break;
                        }
                        #if (USE_ENC == 1)
                        encrypt_decrypt_message(DECRYPT, rcv, &recv, cfd);
                        #endif
                        place_in_window (WIN, rcv, recv, max_y);
                        /* Restore the cursor */
                        move(y, x + 1);
                        refresh();
                }
                usleep(DELAY);

        }

        endwin();

        if (!server_up)
                printf("Something is wrong with the server...\n");

        close(sockfd);
        close(cfd);
        return 0;
}
```

Σημείωση: Το PID της διεργασίας σίγουρα δεν αποτελεί καλό αναγνωριστικό χρήστη από την στιγμή που η ίδια εφαρμογή μπορεί να εκτελεστεί με πελάτες σε διαφορετικά μηχανήματα, ωστόσο αυτό είναι κάτι που μπορεί να εξελιχθεί εύκολα σε κάτι πρακτικότερο αξιοποιώντας ως βάση τον παραπάνω κώδικα.

Μεταγλώττιση των προγραμμάτων και εκτέλεση τους

Στην συνέχεια παρατίθεται το Makefile που δημιουργήθηκε για τους σκοπούς εύκολης μεταγλώττισης των προγραμμάτων που σχετίζονται με τον πελάτη και τον εξυπηρετητή. Παρατηρείται ότι έχουν συμπεριληφθεί και επιπλέον αρχεία προς μεταγλώττιση, συγκεκριμένα τα aes.c και message_enc.c, των οποίων η χρησιμότητα αφορά το επόμενο ζήτημα.

```
all :           client server
```

```
client :        client.o aes.o message_enc.o
                gcc -o client client.o aes.o message_enc.o -lncurses

server :        server.o aes.o message_enc.o
                gcc -o server server.o aes.o message_enc.o

server.o :      server.c message_enc.h
                gcc -c -DUSE_ENC=$(USE_ENC) server.c

client.o :      client.c message_enc.h
                gcc -c -DUSE_ENC=$(USE_ENC) client.c

aes.o :         aes.c aes.h
                gcc -c aes.c

message_enc.o : message_enc.c
                gcc -c message_enc.c

clean :
                rm *.o client server
```

Για την εκτέλεση της εντολής make είναι απαραίτητο να προσδιοριστεί η παράμετρος USE_ENC σύμφωνα με την ανάγκη χρήσης ή μη κρυπτογράφησης για τα μηνύματα. Για τους σκοπούς αυτού του ζητούμενου, αρκεί να εκτελέσουμε την εντολή:

```
make clean ; make USE_ENC=0
```

Σημειώνεται ότι αν έχει προηγηθεί μια εκτέλεση για διαφορετική τιμή της παραμέτρου USE_ENC, χωρίς να έχει εκτελεστεί ύστερα η make clean, δεν θα γίνει εκ νέου μεταγλώττιση, και για αυτό προτείνεται η παραπάνω εντολή. Σκοπός του συγκεκριμένου Makefile άλλωστε δεν αποτελεί η γρηγορότερη μεταγλώττιση, εφόσον η έκταση του κώδικα είναι μικρή, αλλά η διευκόλυνση της διαδικασίας αυτής.

Αν θέλουμε να εκτελέσουμε την εφαρμογή μας, εκτελούμε στον υπολογιστή που θέλουμε να αξιοποιήσουμε ως εξυπηρετητή το αντίστοιχο πρόγραμμα. Μετά το make, αυτό γίνεται μέσω της εντολής:

```
./server
```

Ύστερα ο κάθε πελάτης εκτελεί το αντίστοιχο πρόγραμμα με παράμετρο την διεύθυνση IP του εξυπηρετητή, και στην περίπτωση που αυτός αντιστοιχεί στο ίδιο μηχάνημα, η εντολή θα είναι:

```
./client 127.0.0.1
```

Η εφαρμογή δοκιμάστηκε με τον εξυπηρετητή να εκτελείται σε εικονικό μηχάνημα της πλατφόρμας "Ωκεανός", και η σύνδεση από έναν τοπικό υπολογιστή με χρήση της παρεχόμενος δημόσιας διεύθυνσης IP του αντίστοιχου εικονικού μηχανήματος σημειώθηκε επιτυχής.

Ζήτημα Ζ2

Αρχείο aes.h

Στην συνέχεια παρατίθεται το αρχείο επικεφαλίδας aes.h το οποίο κάνει διαθέσιμες τις συναρτήσεις του αρχείου aes.c για χρήση στο message_enc.c.

```
#ifndef AES_H
#define AES_H

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <crypto/cryptodev.h>
#include <stdint.h>

struct cryptodev_ctx {
        int cfd;
        struct session_op sess;
        uint16_t alignmask;
};

#define AES_BLOCK_SIZE  16

int aes_ctx_init(struct cryptodev_ctx* ctx, int cfd, const uint8_t *key, unsigned
int key_size);
void aes_ctx_deinit();
int aes_encrypt(struct cryptodev_ctx* ctx, const void* iv, const void* plaintext,
void* ciphertext, size_t size);
int aes_decrypt(struct cryptodev_ctx* ctx, const void* iv, const void* ciphertext,
void* plaintext, size_t size);

#endif
```

Αρχείο aes.c

Ο αλγόριθμος κρυπτογράφησης που πρόκειται να αξιοποιηθεί ακολουθεί το Advanced Encryption Standard (AES), με μέγεθος block 128 bits και χρήση Cipher Blocker Chaining. Το αρχείο αυτό, του οποίου ο κώδικας παρουσιάζεται στην συνέχεια, αποτελεί τροποποίηση ενός από τα παραδείγματα χρήσης του cryptodev module όπως αυτά παρατίθενται στο αντίστοιχο github repository. Η συνάρτηση aes_ctx_init αξιοποιείται για να αρχικοποιήσει το struct cryptodev_ctx* ctx που χρησιμοποιούν οι κλήσεις ioctl, και να εκκινήσει το κρυπτογραφικό session μέσω του ioctl command με όρισμα CIOCGSESSION.
Οι συναρτήσεις aes_encrypt και aes_decrypt πραγματοποιούν κρυπτογράφηση και αποκρυπτογράφηση του περιεχομένου στις θέσεις plaintext ή ciphertext αντιστοίχως, μεγέθους size_t size κατά προτίμηση ίσο με AES_BLOCK_SIZE, αξιοποιώντας τις παραμέτρους που έχουν δοθεί στο struct cryptodev_ctx* ctx και το αρχικό διάνυσμα const void* iv. Αυτές οι ενέργειες πραγματοποιούνται αφότου φροντίσουμε για την σωστή στοίχιση των θέσεων εισόδου εξόδου σύμφωνα με το alignment mask της παραμέτρου ctx, και εκτελώντας την εντολή ioctl με όρισμα CIOCCRYPT.
Τέλος, με την συνάρτηση aes_ctx_decrypt μέσω της οποίας καλείται και το αντίστοιχο ioctl command με όρισμα CIOCFSESSION, ολοκληρώνεται το session που αφορά το ctx.
Ενδεχομένως να μην έχουμε αντιληφθεί πλήρως την λειτουργία των παρακάτω συναρτήσεων για να τις αναλύσουμε περισσότερο στην παρούσα αναφορά, αλλά έχουν παρατεθεί ανάλογα σχόλια,

συμπεριλαμβανομένου των αρχικών, που επιχειρούν να επεξηγήσουν ορισμένα τμήματα του ακόλουθου κώδικα.

```
/*
 * Some functions to perform the Advanced
 * Encryption Algorithm on a dataset using
 * the cryptodev module
 */

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <crypto/cryptodev.h>
#include "aes.h"

#define KEY_SIZE 16

int aes_ctx_init(struct cryptodev_ctx* ctx, int cfd, const uint8_t *key, unsigned
int key_size)
{
#ifdef CIOCGSESSINFO
        struct session_info_op siop;
#endif

        memset(ctx, 0, sizeof(*ctx));
        ctx->cfd = cfd;

        /* Define the parameters for the cryptographic session
         * using the session_op field of the cryptodev_ctx
         * structure */
        ctx->sess.cipher = CRYPTO_AES_CBC;
        ctx->sess.keylen = key_size;
        ctx->sess.key = (void*)key;

        /* According to the documentation */
        /* (https://www.freebsd.org/cgi/man.cgi?query=crypto&sektion=4) */
        /* Create a new cryptographic session on a file descriptor for
         * the device; that is, a persistent object specific to the
         * chosen privacy algorithm, integrity algorithm, and keys
         * specified in sessp (here: stx->sess) */

        if (ioctl(ctx->cfd, CIOCGSESSION, &ctx->sess)) {
                perror("ioctl(CIOCGSESSION)");
                return -1;
        }

#ifdef CIOCGSESSINFO
        memset(&siop, 0, sizeof(siop));

        siop.ses = ctx->sess.ses;
        if (ioctl(ctx->cfd, CIOCGSESSINFO, &siop)) {
                perror("ioctl(CIOCGSESSINFO)");
                return -1;
        }

        if (!(siop.flags & SIOP_FLAG_KERNEL_DRIVER_ONLY)) {
                // This is not an accelerated cipher
        }

        ctx->alignmask = siop.alignmask;
#endif
        return 0;
}

void aes_ctx_deinit(struct cryptodev_ctx* ctx)
```

```
{
        /* Destroy the session specified by ctx */
        if (ioctl(ctx->cfd, CIOCFSESSION, &ctx->sess.ses)) {
                perror("ioctl(CIOCFSESSION)");
        }
}

int
aes_encrypt(struct cryptodev_ctx* ctx, const void* iv, const void* plaintext, void*
ciphertext, size_t size)
{
        struct crypt_op cryp;
        void* p;

        /* check plaintext and ciphertext alignment */
        /* https://stackoverflow.com/questions/36630960/c-aligning-memory */
        /* We assume that the alignment mask is of the form 00...0011...11
         * (zeros followed exclusively by ones). The next check
         * (p =? (p+MASK) (AND) (NOT MASK)) verifies that the bits indicated
         * by the ones in the mask are zero in p */
        if (ctx->alignmask) {
                p = (void*)(((unsigned long)plaintext + ctx->alignmask) & ~ctx-
>alignmask);
                if (plaintext != p) {
                        // fprintf(stderr, "plaintext is not aligned\n");
                        return -1;
                }

                p = (void*)(((unsigned long)ciphertext + ctx->alignmask) & ~ctx-
>alignmask);
                if (ciphertext != p) {
                        // fprintf(stderr, "ciphertext is not aligned\n");
                        return -1;
                }
        }

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

        /* Request a symmetric-key (or hash) operation */
        cryp.ses = ctx->sess.ses;
        /* The field cr_op-_len supplies the length of
         * the input buffer */
        cryp.len = size;
        /* the fields   cr_op-_src, cr_op-_dst,
         * cr_op-_iv supply the addresses of the input
         * buffer, output buffer and initialization
         * vector, respectively */
        cryp.src = (void*)plaintext;
        cryp.dst = ciphertext;
        cryp.iv = (void*)iv;
        /* To encrypt, set cr_op-_op to COP_ENCRYPT.  To decrypt,
         * set cr_op-_op to COP_DECRYPT */
        cryp.op = COP_ENCRYPT;
        /* Perform the IOCTL command */
        if (ioctl(ctx->cfd, CIOCCRYPT, &cryp)) {
                perror("ioctl(CIOCCRYPT)");
                return -1;
        }

        return 0;
}

/* The used encryption method is symmetric, so the next function
 * is almost exactly the same as the above */
int
aes_decrypt(struct cryptodev_ctx* ctx, const void* iv, const void* ciphertext,
void* plaintext, size_t size)
{
```

```
        struct crypt_op cryp;
        void* p;

        /* check plaintext and ciphertext alignment */
        if (ctx->alignmask) {
                p = (void*)(((unsigned long)plaintext + ctx->alignmask) &
                                ~ctx->alignmask);
                if (plaintext != p) {
                        // fprintf(stderr, "plaintext is not aligned\n");
                        return -1;
                }

                p = (void*)(((unsigned long)ciphertext + ctx->alignmask) &
                                ~ctx->alignmask);
                if (ciphertext != p) {
                        // fprintf(stderr, "ciphertext is not aligned\n");
                        return -1;
                }
        }

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

        /* Encrypt data.in to data.encrypted */
        cryp.ses = ctx->sess.ses;
        cryp.len = size;
        cryp.src = (void*)ciphertext;
        cryp.dst = plaintext;
        cryp.iv = (void*)iv;
        cryp.op = COP_DECRYPT;
        if (ioctl(ctx->cfd, CIOCCRYPT, &cryp)) {
                perror("ioctl(CIOCCRYPT)");
                return -1;
        }

        return 0;
}
```

Αρχείο message_enc.h

Στην συνέχεια παρατίθεται το αρχείο επικεφαλίδας message_enc.h το οποίο κάνει διαθέσιμη την συνάρτηση του αρχείου message_enc.c για χρήση στα προγράμματα τόσο του πελάτη όσο και του εξυπηρετητή.

```
#include <stdbool.h>
#include "aes.h"

#define ENCRYPT 0
#define DECRYPT 1
#define KEY_SIZE 16

void encrypt_decrypt_message (bool function, char* message, int* message_size, int
cfd);
```

Αρχείο message_enc.c

Στην συνέχεια παρατίθεται το περιεχόμενο του αρχείου το οποίο υλοποιεί την συνάρτηση για την κρυπτογράφηση και αποκρυπτογράφηση των μηνυμάτων. Αναλυτικότερα, χρησιμοποιούμε το

hardcoded κλειδί crypt_key, παραγεμίζουμε το μήνυμα με μηδενικά ώστε το μέγεθος του να είναι ακέραιο πολλαπλάσιο του AES_BLOCK_SIZE (έχουμε φροντίσει στα αντίστοιχα προγράμματα να έχει δεσμευτεί η απαιτούμενη μνήμη για αυτό το παραγέμισμα) και θεωρούμε αρχικό διάνυσμα IV μηδενικό (που δεν αποτελεί καλή πρακτική, αλλά εξυπηρετεί για απλότητα του προγράμματος μας). Ύστερα εκκινούμε το κρυπτογραφικό session, διορθώνουμε την στοίχιση της διεύθυνσης μνήμης της εισόδου μας (που ονομάζεται text και προκύπτει από την αρχική διεύθυνση raw μιας περιοχής που δεσμεύει η C, και στην οποία αντιγράφεται ανά block το μήνυμα μας για τους σκοπούς της κρυπτογράφησης) και κρυπτογραφούμε ανά block το ζητούμενο μήνυμα, γράφοντας στην θέση μνήμης που αντιστοιχεί κάθε φορά στο ανάλογο block. Τελικά, τερματίζουμε το session και ανανεώνουμε το μέγεθος του μηνύματος.

```
#include "message_enc.h"

char crypt_key[KEY_SIZE] = {0x64, 0x65, 0x65, 0x70, 0x64, 0x61, 0x70, 0x75, 0x66,
0x6c, 0x79, 0x62, 0x6c, 0x75, 0x65, 0x00};

void encrypt_decrypt_message (bool function, char* message, int* message_size, int
cfd) {
        int offset, offset_limit, i;
        char data[AES_BLOCK_SIZE];
        /* We want to ensure that we will have AES_BLOCK_SIZE
         * bytes of memory for the text we want to encrypt/decrypt,
         * even after the address alignment. So, We declare not
         * only AES_BLOCK_SIZE bytes, but also the next 16 - 1
         * bytes for the worst case */ /* (!) */
        char raw[AES_BLOCK_SIZE + 63], *text;
        char iv[AES_BLOCK_SIZE];
        struct cryptodev_ctx ctx;

        offset_limit = (*message_size)/AES_BLOCK_SIZE +
                        ((*message_size) % AES_BLOCK_SIZE > 0);

        /* Empty the remaining space to end up with
         * a message with length equal to a multiple
         * of AES_BLOCK_SIZE (necessary for the encryption)*/
        for (i = *message_size; i < offset_limit*AES_BLOCK_SIZE; i++)
                message[i] = 0;

        /* We assume a zeroed initial vector for simplicity */
        memset(iv, 0x0, sizeof(iv));

        /* Initialize the cryptographic session */
        aes_ctx_init(&ctx, cfd, crypt_key, KEY_SIZE);

        if (ctx.alignmask)
                /* If needed, we fix the address alignment according to
                 * the given alignment mask */
                text = (char *)(((unsigned long)raw + ctx.alignmask)
                        & ~ctx.alignmask);
        else
                text = raw;

        for (offset = 0; offset < offset_limit; offset++) {
                /* We store the data we want to encrypt at the aligned address */
                memcpy(text, message + offset*AES_BLOCK_SIZE, AES_BLOCK_SIZE);
                if (function == ENCRYPT)
                        aes_encrypt(&ctx, iv, text, message +
                                    offset*AES_BLOCK_SIZE, AES_BLOCK_SIZE);
                else
                        aes_decrypt(&ctx, iv, text, message +
                                    offset*AES_BLOCK_SIZE, AES_BLOCK_SIZE);
        }
```

```
        /* End the session */
        aes_ctx_deinit(&ctx);

        /* Renew the message size */
        *message_size = offset_limit * AES_BLOCK_SIZE;
}
```

Η αφομοίωση της συνάρτησης αυτής στα προγράμματα του ζητουμένου Z1 είναι σχετικά απλή υπόθεση, και φαίνεται στα σημεία του αντίστοιχου κώδικα τα οποία εμπερικλείονται από τα directives #if (USE_ENC == 1) και #endif. Επίσης, εφόσον η μέθοδος κρυπτογράφησης είναι συμμετρική, για οικονομία στον κώδικα η ίδια συνάρτηση κάνει τόσο την κρυπτογράφηση όσο και την αποκρυτπογράφηση, βάσει του δοσμένου ορίσματος bool function. Σημειώνεται ξανά για τους σκοπούς αυτού του ζητουμένου, ότι ένα μειονέκτημα της εφαρμογής μας είναι ότι αν χρησιμοποιεί κρυπτογράφηση ο client, υποχρεωτικά χρειάζεται να χρησιμοποιεί και ο server, εφόσον έχουμε επιλέξει για λόγους ασφαλείας να κρυπτογραφείται και το PID που προσθέτει ο εξυπηρετητής, και στην μεριά του πελάτη η αποκρυπτογράφηση γίνεται για όλο το μήνυμα.

Η ορθή λειτουργία του προγράμματος μας εξετάζεται ακριβώς όπως υποδείχθηκε στο προηγούμενο ζήτημα, μόνο που εν προκειμένω χρησιμοποιούμε την εντολή:

```
make clean ; make USE_ENC=1
```

Ζήτημα Ζ3

Συμπλήρωση του αρχείου crypto.h (Frontend του οδηγού)

Η μόνη προσθήκη που χρειάζεται να πραγματοποιηθεί στην επικεφαλίδα υπό εξέταση, είναι η δήλωση ενός σημαφόρου lock με τον οποίον θα επιτευχθεί ο αμοιβαίος αποκλεισμός των διεργασιών που προσπαθούν να αξιοποιήσουν ταυτόχρονα το ίδιο αρχείο του οδηγού μας. Η επιλογή σημαφόρου έναντι spinlock δικαιολογείται από το γεγονός ότι οι συναρτήσεις του οδηγού χαρακτήρων υπό εξέταση (δηλαδή όσες συναρτήσεις έχουν υλοποιηθεί στο αρχείο crypto-chrdev.c) καλούνται μόνο σε process context, οπότε είναι εφικτό να μεταβεί η διεργασία που εκτελεί τον οδηγό σε κατάσταση αναμονής χωρίς πρόβλημα. Αν είχε επιλεγεί μια ασύγχρονη υλοποίηση του οδηγού, θα χρειαζόμασταν επιπλέον μια ουρά αναμονής για τις διεργασίες που αναμένουν να λάβουν απάντηση από το backend του οδηγού (ενδεχομένως στην δομή crypto_open_file).

```
/**
 * Device info.
 **/
struct crypto_device {
        /* Next crypto device in the list, head is in the crdrvdata struct
*/
        struct list_head list;

        /* The virtio device we are associated with. */
        struct virtio_device *vdev;

        struct virtqueue *vq;
        /* !! Lock !! */
        struct semaphore lock;

        /* The minor number of the device. */
        unsigned int minor;
};
```

Συμπλήρωση του αρχείου crypto-chrdev.c (Frontend του οδηγού)

Συνάρτηση crypto_chrdev_open

Στην συνέχεια παρουσιάζεται η συνάρτηση crypto_chrdev_open, ύστερα των συμπληρώσεων που πραγματοποιήθηκαν:

```
static int crypto_chrdev_open(struct inode *inode, struct file *filp)
{
        int ret = 0;
        int err;
        unsigned int len;
        struct crypto_open_file *crof;
        struct crypto_device *crdev;
        unsigned int *syscall_type;
        int *host_fd;
        /* !! */
        int num_out = 0;
        int num_in = 0;
```

```
        struct scatterlist syscall_type_sg, host_fd_sg, *sgs[2];
        struct virtqueue *vq;

        debug("Entering");

        syscall_type = kzalloc(sizeof(*syscall_type), GFP_KERNEL);
        *syscall_type = VIRTIO_CRYPTODEV_SYSCALL_OPEN;
        host_fd = kzalloc(sizeof(*host_fd), GFP_KERNEL);
        *host_fd = -1;

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

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

        crof = kzalloc(sizeof(*crof), GFP_KERNEL);
        if (!crof) {
                ret = -ENOMEM;
                goto fail;
        }
        crof->crdev = crdev;
        crof->host_fd = -1;
        filp->private_data = crof;

        /**
         * We need two sg lists, one for syscall_type and one to get the
         * file descriptor from the host.
         **/
        /* !! */
        vq = crdev->vq;
        sg_init_one(&syscall_type_sg, syscall_type, sizeof(*syscall_type));
        sgs[num_out++] = &syscall_type_sg;

        sg_init_one(&host_fd_sg, host_fd, sizeof(*host_fd));
        sgs[num_out + num_in++] = &host_fd_sg;

        /**
         * Wait for the host to process our data.
         **/
        /* !! */
        if (down_interruptible(&crdev->lock))
                return -ERESTARTSYS;

        err = virtqueue_add_sgs(vq, sgs, num_out, num_in,
                                &syscall_type_sg, GFP_ATOMIC);
        virtqueue_kick(vq);
        while (virtqueue_get_buf(vq, &len) == NULL)
                /* do nothing */;
        up(&crdev->lock);

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

        crof->host_fd = *host_fd;
fail:
```

```
        debug("Leaving");
        return ret;
}
```

Συγκεκριμένα, αφότου έχουμε βρει το crypto_device που αντιστοιχεί στο minor number του δοθέντος inode (μέσω της εντολής crdev = get_crypto_dev_by_minor(iminor(inode)) – η αρχικοποίηση του crypto_device έχει πραγματοποιηθεί κατά την εισαγωγή του module μέσω της συνάρτησης virtcons_probe του crypto-module.c), λαμβάνουμε την virtqueue του συγκεκριμένου crypto-device (μέσω της εντολής vq = crdev→vq). Στην συνέχεια θέλουμε να εισαγάγουμε στον πίνακα sgs[] τα αντικείμενα (που αναλόγως την κλήση ενδέχεται να είναι κάποια μεταβλητή ή ένα struct) που αφορούν την κλήση crypto_chrdev_open, σύμφωνα με την προτεινόμενη δομή της VirtQueue που παρατίθεται στην τελευταία σελίδα του οδηγού για το αντίστοιχο ζήτημα. Αυτά τα αντικείμενα, και ο σκοπός ύπαρξης τους στην virtqueue, αναγράφονται στον επόμενο πίνακα:

Αντικείμενο	Προορίζεται για	Σκοπός
`unsigned int syscall_type`	Ανάγνωση	Χρησιμοποιείται σε όλες τις virtqueues ανεξαρτήτως κλήσης. Χρειάζεται για να αναγνωρίσει το backend σε ποια εντολή αντιστοιχεί το virtqueue [open, close, ioctl(...CIOCGSESSION…), ioctl(...CIOCCRYPT…), ioctl(...CIOCFSESSION...)]
`int host_fd`	Εγγραφή	Το backend πρόκειται να ανοίξει το αρχείο /dev/crypto, και θα αντιστοιχηθεί ένας file descriptor που αφορά το νέο session. Το frontend θέλει να γνωρίζει αυτό το fd ώστε να αποτελεί σημείο αναφοράς για οποιοδήποτε άλλο έτοιμα προς το backend που αφορά αυτό το session.

Ο τρόπος με τον οποίο εισάγουμε κάθε αντικείμενο στο ανάλογο virtqueue είναι τετριμμένος και προκύπτει άμεσα από τον κώδικα που είχε δοθεί ως παράδειγμα στην συνάρτηση crypto_chrdev_ioctl. Συγκεκριμένα:
1) δηλώνουμε έναν δείκτη για το αντικείμενο που μας ενδιαφέρει, για παράδειγμα `int *host_fd`, καθώς επίσης και ένα struct scatterlist για αυτήν, στο παράδειγμα `struct scatterlist host_fd_sg`
2) δεσμεύουμε δυναμικά όσο χώρο χρειαζόμαστε για αυτό το αντικείμενο μέσω της kzalloc, στο προηγούμενο παράδειγμα έχουμε `host_fd = kzalloc(sizeof(*host_fd), GFP_KERNEL)`
3) κάνουμε κάποια ανάθεση στο περιεχόμενο της αντίστοιχης θέσης μνήμης αν αυτό χρειάζεται. Στο παράδειγμα, κάνουμε την ανάθεση `*host_fd = -1` καθώς ύστερα θέλουμε να εξετάσουμε αν το open έγινε επιτυχώς από την μεριά του backend και η τιμή του host_fd έχει αλλάξει
4) χρησιμοποιούμε την συνάρτηση sg_init_one για να αρχικοποιήσουμε μια μεμονωμένη καταχώρηση τύπου struct scatterlist που αφορά το συγκεκριμένο αντικείμενο, δίνοντας ως όρισμα αυτής την θέση μνήμης του αντικειμένου τύπου struct scatterlist και τον δείκτη του πρώτου βήματος, καθώς και το μέγεθος του αντικειμένου. Στο παράδειγμα μας, αυτό γίνεται με την εντολή `sg_init_one(&syscall_type_sg, syscall_type, sizeof(*syscall_type))`
5) δίνουμε στην επόμενη διαθέσιμη θέση του πίνακα sgs (ο οποίος αποτελεί όρισμα της `virtqueue_add_sgs` και δείχνει τι πρόκειται να εισαχθεί στην virtqueue) την θέση μνήμης της προηγούμενης καταχώρησης, το οποίο στο παράδειγμα που αναλύουμε γίνεται με την εντολή `sgs[num_out + num_in++] = &host_fd_sg`. Προσέχουμε ώστε πρώτα να εισάγουμε όλες τις καταχωρήσεις προς ανάγνωση, και ύστερα όλες τις καταχωρήσεις προς εγγραφή, κρατώντας

λογαριασμό του πλήθους καταχωρήσεων που αφορούν κάθε κατηγορία, το οποίο εν προκειμένω επιτυγχάνουμε μέσω των num_out και num_in.

Η ίδια διαδικασία ισχύει ακριβώς όπως περιγράφτηκε για όλες τις καταχωρήσεις που χρειάζεται να κάνουμε για τις επιμέρους κλήσεις, με προφανή διακύμανση από περίπτωση σε περίπτωση στο τρίτο βήμα, το οποίο εξαρτάται έντονα από τον σκοπό που εξυπηρετεί η συγκεκριμένη καταχώρηση.

Στην συνέχεια εκτελούμε το επόμενο τμήμα κώδικα, το οποίο είναι κοινό και για τις τρεις συναρτήσεις του fronted οδηγού που θέλουμε να συμπληρώσουμε (crypto_chrdev_open, crypto_chrdev_release και crypto_chrdev_ioctl):

```
        if (down_interruptible(&crdev->lock))
                return -ERESTARTSYS;

        err = virtqueue_add_sgs(vq, sgs, num_out, num_in,
                                &syscall_type_sg, GFP_ATOMIC);
        virtqueue_kick(vq);
        while (virtqueue_get_buf(vq, &len) == NULL)
                /* do nothing */;
        up(&crdev->lock);
```

Χρησιμοποιείται ο σημαφόρος που προσθέσαμε στην επικεφαλίδα προηγουμένως, εφόσον πρόκειται να μεταβληθεί η κοινόχρηστη (υπό ορισμένες συνθήκες) virtqueue. Την εννοούμε ως κοινόχρηστη καθώς όποια διεργασία έχει ανοίξει το ίδιο αρχείο για ανάγνωση (δηλαδή αρχείο του οδηγού με το ίδιο minor number, και κατά συνέπεια ίδιο crypto_device, δηλαδή κοινό virtqueue) συναγωνίζεται με τις υπόλοιπες αντίστοιχες διεργασίες για το ίδιο virtqueue. Συνεπώς, οι τρεις εντολές που αφορούν την συνδρομή του virtqueue στην λειτουργία του οδηγού (virtqueue_add_sgs για την προσθήκη των καταχωρήσεων, virtqueue_kick για την προώθηση της virtqueue στο backend και την virtqueue_get_buf που αποτελεί την συνθήκη για την ενεργό αναμονή και σηματοδοτεί την ολοκλήρωση της ενέργειας που αντιστοιχεί στο frontend) αποτελούν κρίσιμο τμήμα κώδικα και πρέπει να προστατευτεί από κάποιο κλείδωμα.

Εφόσον ολοκληρωθεί το παραπάνω τμήμα κώδικα, έχουμε λάβει απάντηση από το backend και απομένει να εξετάσουμε τις εγγραφές που έκανε στις ανάλογες καταχωρήσεις. Αυτό το στάδιο είναι διαφορετικό για κάθε κλήση, και εν προκειμένω αυτό που χρειάζεται να γίνει είναι να ελεγχθεί ότι έγινε επιτυχώς το άνοιγμα του αρχείου (διαφορετικά το host_fd θα παραμείνει -1 και πρέπει να επιστρέψουμε –ENODEV), και σε περίπτωση που αυτό ισχύει, να ανανεώσουμε τον πεδίο host_fd του struct crypto_open_file (που αποτελεί wrapper του αντίστοιχου crypto_device) που έχει εκχωρηθεί στο πεδίο private_data του αντίστοιχου file pointer, σύμφωνα με την τιμή που επέστρεψε το frontend.

Συνάρτηση crypto_chrdev_release

Στην συνέχεια παρουσιάζεται η υλοποίηση της συνάρτησης crypto_chrdev_release:

```
static int crypto_chrdev_release(struct inode *inode, struct file *filp)
{
        int ret = 0;
        struct crypto_open_file *crof = filp->private_data;
        struct crypto_device *crdev = crof->crdev;
        unsigned int *syscall_type;
```

```
        /* !! */
        int num_out = 0;
        int num_in = 0;
        int *host_fd;
        int err, len;

        struct scatterlist syscall_type_sg, host_fd_sg, *sgs[2];
        struct virtqueue *vq;
        /* -- */

        debug("Entering");

        syscall_type = kzalloc(sizeof(*syscall_type), GFP_KERNEL);
        *syscall_type = VIRTIO_CRYPTODEV_SYSCALL_CLOSE;

        /**
         * Send data to the host.
         **/
        /* !! */
        host_fd = kzalloc(sizeof(*host_fd), GFP_KERNEL);
        *host_fd = crof->host_fd;

        vq = crdev->vq;
        sg_init_one(&syscall_type_sg, syscall_type, sizeof(*syscall_type));
        sgs[num_out++] = &syscall_type_sg;

        sg_init_one(&host_fd_sg, host_fd, sizeof(*host_fd));
        sgs[num_out++] = &host_fd_sg;

        if (down_interruptible(&crdev->lock))
                return -ERESTARTSYS;

        err = virtqueue_add_sgs(vq, sgs, num_out, num_in,
                                &syscall_type_sg, GFP_ATOMIC);

        virtqueue_kick(vq);

        /**
         * Wait for the host to process our data.
         **/
        /* !! */
        while (virtqueue_get_buf(vq, &len) == NULL)
                /* do nothing */;

        up(&crdev->lock);

        kfree(crof);
        debug("Leaving");
        return ret;

}
```

Από εδώ και στο εξής θεωρείται γνωστή η διαδικασία συμπλήρωσης της virtqueue και αναμονής των αποτελεσμάτων από το backend, για αυτό θα αναλυθεί μόνο ο σκοπός και η αρχικοποίηση των καταχωρήσεων της virtqueue, αλλά και το πως διαχειρίζεται η κάθε συνάρτηση στην συνέχεια τις εγγραφές που πραγματοποίησε το backend μέρος του οδηγού.

Αντικείμενο	Προορίζεται για	Σκοπός
`unsigned int syscall_type`	Ανάγνωση	Αναγνώριση της ζητούμενης εντολής
`int host_fd`	Ανάγνωση	Ο file descriptor του host για το οποίο θέλουμε να πραγματοποιηθεί η κλήση close στο backend

Η μόνη αρχικοποίηση που απαιτείται είναι αυτή του host_fd, το οποίο πρέπει να λάβει το host_fd που έχουμε αποθηκεύσει στο πεδίο private_data του αντίστοιχου file pointer. Εφόσον δεν υπάρχει καταχώρηση προς εγγραφή, το μόνο που μένει είναι να αποδεσμεύσουμε τον χώρο μνήμης που αντιστοιχεί στην δομή struct crypto_open_file του πεδίου private_data.

Συνάρτηση crypto_chrdev_ioctl

Στην συνέχεια παρουσιάζεται η υλοποίηση της συνάρτησης crypto_chrdev_ioctl:

```
#define later_initialized(x) x = 0
static long crypto_chrdev_ioctl(struct file *filp, unsigned int cmd,
                                unsigned long arg)
{
        long ret = 0;
        int err;
        struct crypto_open_file *crof = filp->private_data;
        struct crypto_device *crdev = crof->crdev;
        struct virtqueue *vq = crdev->vq;
        struct scatterlist syscall_type_sg, *sgs[MAX_SG_SIZE];
        unsigned int num_out, num_in, len;
#define MSG_LEN 100
        /* unsigned char *output_msg, *input_msg; */
        unsigned int *syscall_type;
        /* !! */
        int i;
        struct scatterlist host_fd_sg, ioctl_cmd_sg, host_return_val_sg;
        int *host_fd;
        unsigned int *ioctl_cmd;
        int *host_return_val;
        /* CIOCGSESSION */
        struct scatterlist session_key_sg, session_op_sg, ret_session_op_sg;
        later_initialized(unsigned char *session_key);
        later_initialized(struct session_op *session_op);
        /* CIOCFSESSION */
        struct scatterlist ses_id_sg;
        later_initialized(u32 *ses_id);
        /* CIOCCRYPT */
        struct scatterlist crypt_op_sg, src_sg, iv_sg, dst_sg;
        later_initialized(struct crypt_op *crypt_op);
        later_initialized(unsigned char *src);
        later_initialized(unsigned char *iv);
        later_initialized(unsigned char *dst);
        /* -- */

        debug("Entering");

        /**
         * Allocate all data that will be sent to the host.
         **/

        syscall_type = kzalloc(sizeof(*syscall_type), GFP_KERNEL);
        *syscall_type = VIRTIO_CRYPTODEV_SYSCALL_IOCTL;
        /* !! */
        host_fd = kzalloc(sizeof(*host_fd), GFP_KERNEL);
        *host_fd = crof->host_fd;
        ioctl_cmd = kzalloc(sizeof(*ioctl_cmd), GFP_KERNEL);
        *ioctl_cmd = cmd;
        host_return_val = kzalloc(sizeof(*host_return_val), GFP_KERNEL);

        num_out = 0;
        num_in = 0;

        /**
         *  These are common to all ioctl commands.
         **/
```

```
        /* unsigned int syscall_type */
        sg_init_one(&syscall_type_sg, syscall_type, sizeof(*syscall_type));
        sgs[num_out++] = &syscall_type_sg;
        /* !! */
        /* int host_fd */
        sg_init_one(&host_fd_sg, host_fd, sizeof(*host_fd));
        sgs[num_out++] = &host_fd_sg;
        /* unsigned int ioctl_cmd */
        sg_init_one(&ioctl_cmd_sg, ioctl_cmd, sizeof(*ioctl_cmd));
        sgs[num_out++] = &ioctl_cmd_sg;

        /**
         *  Add all the cmd specific sg lists.
         **/
        switch (cmd) {
        case CIOCGSESSION:
                debug("CIOCGSESSION");
                /* Copy the session_op struct from the user */
                session_op = kzalloc(sizeof(struct session_op), GFP_KERNEL);
                if (copy_from_user(session_op, (void __user *)arg,
                                   sizeof(struct session_op)))
                        return -EFAULT;

                /* Copy the key from the user */
                session_key = kzalloc(session_op->keylen, GFP_KERNEL);
                if (copy_from_user(session_key, (void __user *)session_op->key,
                                   session_op->keylen))
                        return -EFAULT;

                /* unsigned char session_key[] */
                sg_init_one(&session_key_sg, session_key, session_op->keylen);
                sgs[num_out++] = &session_key_sg;
                /* struct session_op session_op */
                sg_init_one(&session_op_sg, session_op, sizeof(struct session_op));
                sgs[num_out + num_in++] = &session_op_sg;

                break;

        case CIOCFSESSION:
                debug("CIOCFSESSION");
                /* Copy the ses_id from the user */
                ses_id = kzalloc(sizeof(*ses_id), GFP_KERNEL);
                if (copy_from_user(ses_id, (void __user *)arg, sizeof(*ses_id)))
                        return -EFAULT;

                /* u32 ses_id */
                sg_init_one(&ses_id_sg, ses_id, sizeof(*ses_id));
                sgs[num_out++] = &ses_id_sg;
                break;

        case CIOCCRYPT:
                debug("CIOCCRYPT");
                crypt_op = kzalloc(sizeof(struct crypt_op), GFP_KERNEL);
                if (copy_from_user(crypt_op, (void __user *)arg,
                                   sizeof(struct crypt_op)))
                        return -EFAULT;

                src = kzalloc(crypt_op->len, GFP_KERNEL);
                if (copy_from_user(src, (void __user *)crypt_op->src,
                                   crypt_op->len))
                        return -EFAULT;

                dst = kzalloc(crypt_op->len, GFP_KERNEL);

                iv = kzalloc(EALG_MAX_BLOCK_LEN, GFP_KERNEL);
                if (copy_from_user(iv, (void __user *)crypt_op->iv,
                                   EALG_MAX_BLOCK_LEN))
                        return -EFAULT;
```

```
        /* struct crypto_op crypt_op */
        sg_init_one(&crypt_op_sg, crypt_op, sizeof(struct crypt_op));
        sgs[num_out++] = &crypt_op_sg;
        /* unsigned char src[] */
        sg_init_one(&src_sg, src, crypt_op->len);
        sgs[num_out++] = &src_sg;
        /* unsigned char iv[] */
        sg_init_one(&iv_sg, iv, EALG_MAX_BLOCK_LEN);
        sgs[num_out++] = &iv_sg;
        /* unsigned char dst[] */
        sg_init_one(&dst_sg, dst, crypt_op->len);
        sgs[num_out + num_in++] = &dst_sg;

        break;

default:
        debug("Unsupported ioctl command");
        break;
}

/* int host_return_val */
sg_init_one(&host_return_val_sg, host_return_val, sizeof(host_return_val));
sgs[num_out + num_in++] = &host_return_val_sg;

/**
 * Wait for the host to process our data.
 **/
/* !! */
/* !! Lock !! */
if (down_interruptible(&crdev->lock))
        return -ERESTARTSYS;

err = virtqueue_add_sgs(vq, sgs, num_out, num_in,
                        &syscall_type_sg, GFP_ATOMIC);
virtqueue_kick(vq);
while (virtqueue_get_buf(vq, &len) == NULL)
        /* do nothing */;

up(&crdev->lock);
ret = *host_return_val;

switch (cmd) {
case CIOCGSESSION:
        debug("CIOCGSESSION");
        if (copy_to_user((void __user *)arg, session_op,
                         sizeof(struct session_op)))
                return -EFAULT;

        kfree(session_op);
        kfree(session_key);
        break;

case CIOCFSESSION:
        debug("CIOCFSESSION");
        kfree(ses_id);
        break;

case CIOCCRYPT:
        if (copy_to_user((void __user *)crypt_op->dst, dst, crypt_op->len))
                return -EFAULT;

        kfree(iv);
        kfree(dst);
        kfree(src);
        break;

default:
```

```
                debug("Unsupported ioctl command");
                break;
        }

        kfree(host_return_val);
        kfree(ioctl_cmd);
        kfree(host_fd);
        kfree(syscall_type);

        debug("Leaving");

        return ret;
}
```

Όσον αφορά τις αρχικοποιήσεις, είναι απαραίτητο να σημειωθεί ο ρόλος του later_initialized( ), το οποίο χρειάζεται για κάθε pointer μέσω του οποίου θα δεσμευθεί χώρος στην μνήμη εντός της πρώτης switch, που στην συνέχεια χρησιμοποιείται στις αντίστοιχες περιπτώσεις της δεύτερης. Αν δεν γίνει αυτό, η μεταγλώττιση δεν θα ολοκληρωθεί και θα προκύψει μήνυμα το οποίο υποστηρίζει ότι ο αντίστοιχος δείκτης ενδεχομένως να χρησιμοποιείται μη αρχικοποιημένος. Αν και δεν είναι καλή πρακτική ένας δείκτης να αρχικοποιείται σε NULL, η δομή της συνάρτησης είναι τέτοια που δεν μπορεί να δημιουργηθεί πρόβλημα εξαιτίας αυτού.

Στην συνέχεια παρουσιάζεται η δομή της virtqueue για κάθε κλήση:

Κοινά πεδία για όλες τις ioctl κλήσεις		
Αντικείμενο	Προορίζεται για	Σκοπός
`unsigned int syscall_type`	Ανάγνωση	Αναγνώριση της ζητούμενης εντολής (εν προκειμένω είναι VIRTIO_CRYPTODEV_SYSCALL_IOCTL)
`int host_fd`	Ανάγνωση	Ο file descriptor του host για το οποίο θέλουμε να πραγματοποιηθεί η αντίστοιχη κλήση ioctl στο backend
`unsigned int ioctl_cmd`	Ανάγνωση	Η συγκεκριμένη εντολή ioctl που θέλουμε να εκτελεστεί στο backend. Η τιμή του προκύπτει άμεσα από την παράμετρο unsigned int cmd της κλήσης του οδηγού στο frontend
1) Πεδία του ioctl command CIOCGSESSION		
`unsigned char session_key[]`	Ανάγνωση	Αποτελεί το κρυπτογραφικό κλειδί στο οποίο δείχνει το struct session_op που δίνεται ως όρισμα στο αντίστοιχο ioctl command. Είναι απαραίτητο να το περάσουμε ξεχωριστά καθώς ο host δεν είναι σε θέση να αποδεικτοδοτήσει την εικονική διεύθυνση του guest. Το ίδιο ισχύει για όλους τους δείκτες εντός των struct που δίνουμε στο virtqueue
`struct session_op session_op`	Εγγραφή	Είναι το struct που δέχεται το ioctl command εντός του host και πρέπει να χρησιμοποιηθεί αντιστοίχως από το backend. Προορίζεται για εγγραφή εφόσον σίγουρα χρειάζεται να

		ανανεωθεί το πεδίο sess ώστε να γνωστοποιηθεί στο frontend το αναγνωριστικό του καινούργιου session. Τα πεδία αυτού που προορίζονται για ανάγνωση έχουν διαπιστωθεί ότι διαβάζονται επιτυχώς στο backend
2) Πεδία του ioctl command CIOCCRYPT		
`struct crypt_op crypt_op`	Ανάγνωση	Δίνεται ως όρισμα της κλήσης στο frontend, και χρειάζεται αυτούσιο στο backend για την αντίστοιχη κλήση.
`unsigned char src[]`	Ανάγνωση	Αποτελεί δείκτη του struct crypto_op και πρέπει να καταχωρηθεί ξεχωριστά ο αντίστοιχος πίνακας στην virtqueue ώστε να έχει όλη την απαιτούμενη πληροφορία το frontend. Αποτελεί το περιεχόμενο προς (απο)κρυπτογράφηση.
`unsigned char iv[]`	Ανάγνωση	Και σε αυτόν τον πίνακα δείχνει το struct crypt_op. Αποτελεί το αρχικό διάνυσμα (initial vector).
`unsigned char dst[]`	Εγγραφή	Ο πίνακας στον οποίον γράφονται τα δεδομένα της (απο)κρυπτογράφησης από το backend. Εν προκειμένω τον δεσμεύουμε δυναμικά και φροντίζουμε αργότερα να αντιγράψουμε τα περιεχόμενα του (μέσω της copy_to_user) στην θέση που έχει δοθεί από τον χρήστη στο αντίστοιχο struct crypt_op.
3) Πεδία του ioctl command CIOCFSESSION		
`u32 ses_id`	Ανάγνωση	Το αναγνωριστικό του session που επιθυμούμε να τερματίσουμε, και δίνεται ως όρισμα της κλήσης στο frontend.
Κοινό πεδίο εγγραφής για όλες τις ioctl κλήσεις		
`int host_return_val`	Εγγραφή	Η τιμή που προκύπτει από την εκτέλεση του ioctl command στην μεριά του backend

Σημειώνεται ότι υποστηρίζονται με αυτό τον τρόπο ένα μόνο υποσύνολο λειτουργιών των ζητούμενων ioctl commands, καθώς εναλλακτικά θα ήταν απαραίτητο να λάβουμε υπόψη για παράδειγμα και το πεδίο mac του crypt_op, ή γενικά τα πεδία των session_op και crypt_op που αντιστοιχούν σε hash/MAC operations.

Εφόσον κάνουμε τις αρχικοποιήσεις των κοινών πεδίων, εντός της πρώτης switch συνεχίζουμε με τις απαραίτητες αρχικοποιήσεις των πεδίων για την επιλεγμένη κάθε φορά εντολή, αντιγράφοντας το ανάλογο όρισμα από τον χρήστη με την copy_from_user, αλλά και τους πίνακες στους οποίους ενδεχομένως μπορεί να δείχνουν τα πεδία ενδιαφέροντος του ορίσματος αυτού (εφόσον όπως αναφέρθηκε παραπάνω οι δείκτες αυτών των πεδίων δεν μπορούν να μεταφραστούν άμεσα από το backend). Όπως έχει αναλυθεί και στην προηγούμενη εργαστηριακή αναφορά, οι συναρτήσεις copy_from_user και copy_to_user είναι σημαντικό να χρησιμοποιηθούν έναντι της απευθείας αποδεικτοδότησης, ώστε να διασφαλιστούν ορισμένα ζητήματα ασφαλείας (είναι δηλαδή απαραίτητο να εξεταστεί η εγκυρότητα της δοθείσας διεύθυνσης σε σχέση με τον χρήστη που

επιχειρεί να την χρησιμοποιήσει, καθώς μπορεί να συμβεί κάποιο page fault που δεν μπορούμε να διαχειριστούμε σε επίπεδο πυρήνα, ή να μεταβληθεί το περιεχόμενο μιας διεύθυνσης στην οποία ο χρήστης δεν έχει δικαιοδοσία).

Μετά την πρώτη switch ορίζουμε και το τελευταίο κοινό αντικείμενο για την virtqueue, στέλνουμε τα δεδομένα στο backend μέσω του κυκλικού buffer και αναμένουμε την απάντηση του, με τον γνωστό τρόπο. Αφού λάβουμε την απάντηση αναθέτουμε την τιμή *host_return_val στην μεταβλητή ret που αναλογεί στην τιμή επιστροφής, και καταφεύγουμε εκ νέου σε περιπτωσιολογία μέσω μιας δεύτερης switch. Εντός αυτής αντιγράφουμε στον χρήστη ότι έγραψε το backend (στην περίπτωση του CIOCGSESSION όλο το struct session_op, καθώς δεν μπορούμε να γνωρίζουμε με απόλυτη σιγουριά τι άλλο ανανεώνεται πέρα του πεδίου ses, και στην περίπτωση του CIOCCRYPT αντιγράφουμε το (απο)κρυπτογραφημένο κείμενο στην διεύθυνση προορισμού που έχει δοθεί εντός του ορίσματος struct crypt_op) και απελευθερώνουμε τις θέσεις μνήμης που δεσμεύτηκαν για την συγκεκριμένη εντολή. Εν τέλει, αποδεσμεύουμε όλες τις θέσεις για τα κοινά πεδία και επιστρέφουμε.

Η λειτουργία του αρχείου crypto-module.c (Frontend του οδηγού)

Οι συναρτήσεις init και fini

Με την συνάρτηση init εκτελείται η συνάρτηση crypto_chrdev_init του αρχείου crypto-chrdev.c, μέσω της οποίας δεσμεύουμε την επιθυμητή περιοχή αριθμών συσκευών (major number ίσος με 60 και minor numbers από 0 έως 30) και τις εκχωρούμε στην συσκευή χαρακτήρων που υλοποιούμε. Στην συνέχεια εντός της init αρχικοποιούμε την λίστα συσκευών που πρόκειται να διαχειριστούμε και περιέχεται στην global δομή crdrvdata (μέσω της INIT_LIST_HEAD(&crdrvdata.devs)), καθώς και το spinlock της δομής αυτής, και τελικά καταχωρούμε τον οδηγό τύπου virtio_driver δηλώνοντας την δομή virtio_driver που περιέχει δείκτες στις συναρτήσεις του.

Η fini λειτουργεί αναιρετικά της init, δηλαδή εκτελεί την crypto_chrdev_destroy για να αποδεσμεύσει την περιοχή αριθμών της crypto_chrdev_init και προβαίνει στην κλήση της unregister_virtio_driver για τον virtio driver.

Οι συναρτήσεις virtcons_probe και virtcons_remove

Εντός της virtcons_probe (η οποία εκτελείται κάθε φορά που ο πυρήνας βρίσκει μια εικονική συσκευή που αντιστοιχεί στην δομή virtio_device_id που έχουμε δηλώσει) αρχικοποιούμε μια δομή τύπου crypto_device, την οποία ρυθμίζουμε να δείχνει στο αντίστοιχο virtio_device (και το virtio_device να δείχνει στο crypto_device) και του αντιστοιχίζουμε την virtqueue που του αναλογεί. Τέλος, αρχικοποιούμε τον σημαφόρο του crypto_device, και εκχωρούμε την συσκευή στην λίστα της δομής crdrvdata προκειμένου να μπορούμε να βρούμε στο crypto-chrdev.c το αντίστοιχο virtqueue βάσει του minor number.
Η συνάρτηση virtcons_remove αναιρεί την δουλεία της virtcons_probe (αφαιρεί την virtio συσκευή από την λίστα της δομής crdrvdata, επαναφέρει την συσκευή και διαγράφει τα virtqueues που της αναλογούν).

Οι συναρτήσεις find_vq και vq_has_data

Η συνάρτηση find_vq είναι υπεύθυνη για να επιστρέψει ένα virtqueue για το virtio_device που της παρέχεται ως όρισμα. Συνδέει αυτό το virtqueue με την συνάρτηση vq_has_data που εκτελείται κάθε φορά που το αντίστοιχο virtqueue ενημερώνεται από το backend και λαμβάνουμε ειδοποίηση από αυτό μέσω διακοπής. Αυτή λοιπόν η συνάρτηση αποτελεί το κλειδί για μια ασύγχρονη υλοποίηση, στην οποία αντί για ενεργή αναμονή θα χρησιμοποιούσαμε στα αντίστοιχα σημεία του

αρχείου crypto-chrdev.c την συνάρτηση wait_event_interruptible (με κάποια συνθήκη που μπορεί να πιστοποιήσει ότι ο buffer που ανανεώθηκε αντιστοιχεί πράγματι σε αυτόν της διεργασίας που εκτελείται), και εντός της vq_has_data θα χρησιμοποιούσαμε την wake_up_interruptible για την ουρά διεργασιών του crypto_open_file στο οποίο αντιστοιχίζεται η virtqueue.

Συμπλήρωση του αρχείου virtio-cryptodev.c (Backend του οδηγού)

Στην συνέχεια παρατίθεται η συμπληρωμένη συνάρτηση vq_handle_output του αντίστοιχου αρχείου του backend:

```
static void vq_handle_output(VirtIODevice *vdev, VirtQueue *vq)
{
    VirtQueueElement *elem;
    unsigned int *syscall_type;
    /* !! */
    int *host_fd;
    int *host_return_val;

    DEBUG_IN();

    elem = virtqueue_pop(vq, sizeof(VirtQueueElement));
    if (!elem) {
        DEBUG("No item to pop from VQ :(");
        return;
    }

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

    syscall_type = elem->out_sg[0].iov_base;
    switch (*syscall_type) {
    case VIRTIO_CRYPTODEV_SYSCALL_TYPE_OPEN:
        DEBUG("VIRTIO_CRYPTODEV_SYSCALL_TYPE_OPEN");
        /* !! */
        host_fd = elem->in_sg[0].iov_base;
        *host_fd = open("/dev/crypto", O_RDWR, 0);
        break;

    case VIRTIO_CRYPTODEV_SYSCALL_TYPE_CLOSE:
        DEBUG("VIRTIO_CRYPTODEV_SYSCALL_TYPE_CLOSE");
        /* !! */
        host_fd = elem->out_sg[1].iov_base;
        close(*host_fd);
        break;

    case VIRTIO_CRYPTODEV_SYSCALL_TYPE_IOCTL:
        DEBUG("VIRTIO_CRYPTODEV_SYSCALL_TYPE_IOCTL");
        host_fd = elem->out_sg[1].iov_base;
        int *ioctl_cmd = elem->out_sg[2].iov_base;

        switch (*ioctl_cmd) {
        case CIOCGSESSION:
            DEBUG("CIOCGSESSION");

            struct session_op *sess = elem->in_sg[0].iov_base;
            unsigned char* restore_key_address = sess->key;

            sess->key = elem->out_sg[3].iov_base;

            host_return_val = elem->in_sg[1].iov_base;
            *host_return_val = ioctl(*host_fd, CIOCGSESSION, sess);
            sess->key = restore_key_address;
            break;
```

```
        case CIOCFSESSION:
            DEBUG("CIOCFSESSION");
            uint32_t *ses_id = elem->out_sg[3].iov_base;
            host_return_val = elem->in_sg[0].iov_base;
            *host_return_val = ioctl(*host_fd, CIOCFSESSION, ses_id);
            break;

        case CIOCCRYPT:
            DEBUG("CIOCCRYPT");

            struct crypt_op *crypt_op = malloc(sizeof(struct crypt_op));
            memcpy(crypt_op, elem->out_sg[3].iov_base, sizeof(struct crypt_op));
            crypt_op->src = elem->out_sg[4].iov_base;
            crypt_op->iv = elem->out_sg[5].iov_base;
            crypt_op->dst = elem->in_sg[0].iov_base;

            host_return_val = elem->in_sg[1].iov_base;
            *host_return_val = ioctl(*host_fd, CIOCCRYPT, crypt_op);

            free(crypt_op);
            break;

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

    default:
        DEBUG("Unknown syscall_type");
        break;
    }

    virtqueue_push(vq, elem, 0);
    virtio_notify(vdev, vq);
    g_free(elem);
}
```

Από το στοιχείο `VirtQueueElement *elem` που παίρνουμε από τον κυκλικό buffer, λαμβάνουμε τις διευθύνσεις των αντικειμένων εξόδου (όσα δηλαδή είχαν δηλωθεί για ανάγνωση) μέσω της ανάθεσης με rvalue το `elem->out_sg[idx].iov_base` όπου idx ο αύξων αριθμός που αντιστοιχεί στην σειρά εισαγωγής του ανάλογου αντικειμένου (στην ομάδα των αντικειμένων προς ανάγνωση), ενώ λαμβάνουμε τις διευθύνσεις των αντικειμένων προς εγγραφή με την ανάθεση με rvalue το `elem->in_sg[idx].iov_base` αντιστοίχως (όπου ο δείκτης idx αναλογεί στην σειρά εισαγωγής του αντικειμένου στην ομάδα των αντικειμένων προς εγγραφή).

Στην περίπτωση `VIRTIO_CRYPTODEV_SYSCALL_TYPE_OPEN` αρκεί να κάνουμε open το αρχείο /dev/crypto και να αποθηκεύσουμε το προκύπτον file descriptor στην θέση μνήμης `elem→in_sg[0].iov_base` (κατά αναλογία με το τι έχουμε επιλέξει να μεταφέρουμε μέσω του virtqueue για κάθε κλήση, όπως παρουσιάστηκε αναλυτικά στο frontend κομμάτι).

Στην περίπτωση `VIRTIO_CRYPTODEV_SYSCALL_TYPE_CLOSE` αρκεί να λάβουμε τον file descriptor του host για τον οποίο επιθυμούμε το κλείσιμο, και να καλέσουμε την close με όρισμα αυτό.

Για την περίπτωση `VIRTIO_CRYPTODEV_SYSCALL_TYPE_IOCTL` εξετάζουμε ξεχωριστά το κάθε ioctl command:
1) `CIOCGSESSION` : Αρχικά αλλάζουμε τον δείκτη key του αντίστοιχου struct session_op ώστε να δείχνει στην διεύθυνση του πίνακα του κλειδιού που περάσαμε μέσω της virtqueue. Φροντίζουμε ωστόσο να αποθηκεύσουμε την αρχική τιμή του και να την επαναφέρουμε ύστερα ώστε όταν

κάνουμε copy_to_user στον frontend οδηγό να μην έχει αλλοιωθεί ο δείκτης που είχε δοθεί από τον χρήστη. Εκτελούμε την ioctl εντολή, αποθηκεύουμε το αποτέλεσμα της εκτέλεσης στην αντίστοιχη καταχώρηση προς εγγραφή της virtuqueue, και επαναφέρουμε τον δείκτη key.

2) `CIOCFSESSION` : Σε αυτήν την περίπτωση, αρκεί να εκτελέσουμε το αντίστοιχο ioctl command με όρισμα την θέση μνήμης της καταχώρησης ses_id που δόθηκε μέσω του virtqueue, και να αποθηκεύσουμε την τιμή επιστροφής της κλήσης στην θέση host_return_val.

3) `CIOCCRYPT` : Για να μην χρειαστεί να ανακτήσουμε έναν έναν τους δείκτες του struct crypt_op, φτιάχνουμε ένα αντίγραφο αυτού μέσω της memcpy και ενημερώνουμε τους δείκτες αυτού ώστε να αντιστοιχούν στις θέσεις των πινάκων που δόθηκαν μέσω της virtqueue. Ύστερα καλούμε το αντίστοιχο ioctl command με όρισμα το αντίγραφο που επεξεργαστήκαμε, και εφόσον ολοκληρωθεί και αποθηκεύσουμε την τιμή επιστροφής του κατά τα γνωστά, αποδεσμεύουμε τον χώρο που καταλαμβάνει το αντίγραφο.

Εφόσον ολοκληρωθεί η κλήση που ζητήθηκε από το frontend, ανανεώνουμε τον κυκλικό buffer με τα αποτελέσματα του backend και εκτελούμε την virtio_notify προκειμένου να ενημερωθεί το frontend για την διεκπεραίωση της εργασίας που είχε να αναλάβει το backend.

Σημείωση: Έχουν πραγματοποιηθεί κάποιες αλλαγές σε σχέση με την λύση που παρουσιάστηκε στην εξέταση, μεταξύ των οποίων η διόρθωση της αρχικής παράληψης να επαναφέρουμε τον δείκτη στο CIOCGSESSION, ενώ σε μία περίπτωση στο frontend η εντολή διεκδίκησης του σεμαφόρου είχε τοποθετηθεί από παραδρομή μια εντολή πιο κάτω.

Διαδικασία μεταγλώττισης και δοκιμής λειτουργίας του οδηγού

Πρώτο βήμα μας είναι να εγκαταστήσουμε το cryptodev στον host:

```
[HOST] cd /path/to/cryptodev/cryptodev-linux
[HOST] make
[HOST] sudo insmod cryptodev.ko
```

Για την συμπλήρωση του οδηγού, χρειάζεται αρχικά να εφαρμόσουμε το patch που διατίθεται με τον βοηθητικό κώδικα της εργασίας, ώστε να προστεθούν στο qemu οι απαραίτητες για το backend μέρος του οδηγού εντολές, και τα απαραίτητα αρχεία. Αυτό γίνεται με τις ακόλουθες εντολές:

```
[HOST] cd path/to/qemu
[HOST] patch -p1 < path/to/virtio-cryptodev/qemu-3.0.0_helpcode.patch
```

Εφόσον γίνουν οι προσθήκες και αλλαγές στον κώδικα του qemu, όπως αυτές επιβάλλονται από το αντίστοιχο .patch αρχείο, πρέπει να μεταγλωττίσουμε το qemu, με τις ακόλουθες εντολές (υποθέτοντας ότι δεν έχει προηγηθεί κάποια παλιότερη μεταγλώττιση):

```
[HOST] ./configure --prefix=./build --enable-kvm --target-list=x86_64-softmmu
[HOST] make -j 12
[HOST] cp x86_64-softmmu/qemu-system-x86_64 build/bin
```

Η τελευταία εντολή αποδείχθηκε σημαντική για την συνέχεια της εργασίας, καθώς ύστερα από την μεταγλώττιση δεν ενημερωνόταν ο φάκελος ./build/bin ώστε να συμπεριλαμβάνει το παραγόμενο εκτελέσιμο qemu-system-x86_64, το οποίο είναι απαραίτητο για την λειτουργία του utopia.sh. Κάθε φορά που πραγματοποιούμε μια αλλαγή στο backend μέρος του οδηγού (δηλαδή στο αρχείο /path/to/qemu/hw/char/virtio-cryptodev.c) πρέπει να τερματίσουμε το utopia (σε περίπτωση που βρίσκεται σε λειτουργία), να επαναλάβουμε τις δύο τελευταίες εντολές, και να συνεχίσουμε με όλα τα επόμενα βήματα. Η διαδικασία προετοιμασίας για το utopia θεωρείται γνωστή και δεν θα αναλυθεί εκ νέου στην παρούσα αναφορά. Εκτελούμε το utopia ως εξής:

```
[HOST] cd path/to/utopia
[HOST] ./utopia.sh -device virtio-cryptodev-pci
[HOST] ssh -p 22223 root@localhost
```

Μπορούμε να επαναλάβουμε την παράμετρο "-device virtio-cryptodev-pci" τόσες φορές όσες συσκευές cryptodev είναι επιθυμητό να έχουμε. Με την παραπάνω εντολή θα έχουμε διαθέσιμη μόνο την /dev/cryptodev0. Εντός της εικονικής μηχανής, επεξεργαζόμαστε τα αρχεία crypto.h (για την προσθήκη του σεμαφόρου) και crypto-chrdev.c ώστε να αναπτύξουμε το frontend μέρος, και δοκιμάζουμε τον οδηγό ως εξής:

```
[GUEST] # Θεωρούμε ότι βρισκόμαστε ήδη στον κατάλογο με τα αρχεία του frontend
[GUEST] make
[GUEST] ./crypto_dev_nodes.sh
[GUEST] insmod virtio_crypto.ko
[GUEST] ./test_crypto /dev/cryptodev0 # Εξέταση της σωστής λειτουργίας του οδηγού
[GUEST] ./test_fork_crypto /dev/cryptodev0 # Έλεγχος σωστής χρήσης του κλειδώματος
```

Επιπροσθέτως, μπορούμε να εξετάσουμε και την σωστή λειτουργία κάποιας εφαρμογής που αξιοποιεί το cryptodev module (όπως αυτή των προηγούμενων δύο ζητημάτων), αφότου εκτελέσουμε πρώτα την επόμενη εντολή:

```
[GUEST] ln /dev/cryptodev0 /dev/crypto
```