Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών Ε.Μ.Π.

ΕΡΓΑΣΤΗΡΙΟ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
2015-2016

ΑΣΚΗΣΗ 3η
ΜΕΡΟΣ Α': Υλοποίηση chat

Χρυσούλα Βαρηά
ΑΜ: 03112105
Ευαγγελία-Σοφία Γεργατσούλη
ΑΜ: 03112064
Εξάμηνο:8ο

1 Chat: η πλευρά του server

Αρχικά ο server κάνει bind μια port (στην περίπτωση μας την 35001) στην οποία θα ακούει για εισερχόμενες συνδέσεις. Σε περίπτωση που ο client προσπαθήσει να συνδεθεί σε άλλη port, δεν θα τα καταφέρει, καθώς δεν θα πάρει απάντηση από τον server. Σημειώνεται ότι και ο client θα μπορούσε να κάνει bind μια port πριν συνδεθεί με τον server ώστε να συνδέεται πάντα από την ίδια, στην περίπτωση μας απλά επιλέγεται μια διαθέσιμη.
Στη συνέχεια, ο server δέχεται μια εισερχόμενη σύνδεση μέσω της accept και σε περίπτωση που δεν υπάρχει καμία, γίνεται block (σταματάει) μέχρι να έρθει κάποια. Αφού γίνει η αποδοχή κάποιας εισερχόμενης σύνδεσης ξεκινάει η συνομιλία μεταξύ server και client μέχρι να τερματιστεί η σύνδεση μεταξύ τους.

Για την ανταλλαγή και την ειδοποιήση λήψης μηνυμάτων χρησιμοποιείται η select. Συγκεκριμένα η select δέχεται δύο ομαδες από file descriptors και ενημερώνει όταν κάποιος αό αυτους είναι έτοιμος για read ή για write (ανάλογα αν είναι στους write ή στους read descriptors). Επιπλέον, η select μπλοκάρει μέχρι να υπάρξουν δεδομένα είτε στο αντίστοιχο socket (ο χρήστης λαμβάνει δεδομένα) είτε στο terminal (όπου χρήστης γράφει κάτι) είτε αν εξαντληθεί το χρονικό όριο, το οποίο έχει τεθεί μέσω της δομής timeval σε 1 sec. Όταν συμβεί κάποιο από τα παραπάνω γεγονότα εξετάζεται η αντίστοιχη περίπτωση (ανάλογα με το ποιο socket είναι set):

- Αν το newsd είναι set σημαίνει ότι ο client έστειλε δεδομένα στο server, τα οποία ο δεύτερος πρέπει να επεξεργαστεί.
- Αν το 0 είναι set σημαίνει ότι ο server έγραψε κάτι μέσω terminal, το οποίο πρέπει να σταλεί στον client.

Ο κώδικας για τον server φαίνεται αναλυτικά στην ενότητα 4.1.

2 Chat: η πλευρά του client

Στις ζητούμενες εφαρμογές έχει γίνει η παραδοχή ότι ο server επικοινωνεί με ένα client κάθε φορά, ενώ οι υπόλοιποι clients βρίσκονται σε αναμονή μέχρι να εξυπηρετηθεί ο αρχικός client και να κλείσει τη σύνδεση.

Ο client στέλνει αίτημα σύνδεσης μέσω της connect, με την οποία γίνεται η αντιστοίχιση του sd socket με την διεύθυνση sa. Η διαδικασία ειδοποίησης του client για τη λήψη μηνυμάτων είναι αντίστοιχη με αυτή που περιγράφτηκε για τη μεριά του server. Επιπλέον, για την πιθανή λήψη του σήματος $SIGSTOP$ (Ctrl+C) από τον χρήστη, υπάρχει αντίστοιχος signal handler για τον επιτυχή (και ομαλό) τερματισμό της σύνδεσης από τη μεριά του client.

Ο κώδικας για τον client φαίνεται αναλυτικά στην ενότητα 4.2.

3 Προσθήκη κρυπτογράφησης

Χρησιμοποιούμε το module cryptodev του linux για να υλοποιηθεί η κρυπτογράφηση της επικοινωνίας client-server μέσω κάποιων κλήσεων στη συσκευη /dev/crypto που παρουσιάζονται παρακάτω.
Αρχικά τα πεδία $key[], iv[]$ (key και initialization vector αντίστοιχα) της δομής data είναι προσυμφωνημένα και έχουν την ίδια τιμή τόσο από μεριά του client όσο και από αυτήν του server (χωρίς να χρειάζεται να επικοινωνήσουν για να τα καθορίσουν). Η έναρξη ενός session με τη συσκευή γίνεται με την αντίστοιχη

κλήση ioctl (CIOCGSESSION) και από τις δύο πλευρές (από τη μεριά του server γίνεται μια κλήση για κάθε νέα σύνδεση, η οποία γίνεται accept). Αντίστοιχα ο τερματισμός του session γίνεται μέσω της κλήσης ioctl(CIOCFSESSION).
Η ενημέρωση για τη λήψη μηνυμάτων και από τις δύο πλευρές γίνεται με τη χρήση της select, όπως και στην επικοινωνία χωρίς κρυπτογράφηση. Σε αυτή την περίπτωση όμως, πριν την αποστολή κάποιου μηνύματος πρέπει να γίνει κρυπτογράφηση με τη χρήση ioctl(CIOCCRYPT) (με το flag COP_ENCRYPT ενεργοποιημένο), ενώ για την ανάγνωση ενός νέου μηνύματος απαιτείται αποκρυπτογράφηση, η οποία επιτυγχάνεται πάλι με την κλήση ioctl(CIOCCRYPT) αλλά με το flag COP_DECRYPT ενεργοποιημένο.
Σημειώνεται ότι σε αντίθεση με την πρώτη περίπτωση στην οποία γινόταν αποστολή μόνο του μηνύματος, εδώ γίνεται αποστολή όλου του περιεχομένου του buffer. Αυτό είναι απαραίτητο για τη σωστή διαδικασία της κρυπτογράφησης και αποκρυπτογράφησης, ενώ το περιεχόμενο του μηνύματος σταματάει με τον τερματικό χαρακτήρα \0 (NULL).

4 Παράρτημα: Κώδικας

4.1 Η πλευρά του server

```
/*
 * socket-server.c
 * Simple TCP/IP communication using sockets
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 */

#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 <stdbool.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>

#include <arpa/inet.h>
#include <netinet/in.h>
#include <crypto/cryptodev.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "socket-common.h"
```

```
/* 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(void)
{
        char buf[DATA_SIZE];
        char addrstr[INET_ADDRSTRLEN];
        int sd, newsd, maxsd, cfd, i;
        int server_writes=0, server_reads=0;
        ssize_t n;
        socklen_t len;
        fd_set read_fd_set;
        struct sockaddr_in sa;
        struct timeval tmv;
        struct session_op sess;
        struct crypt_op cryp;
        struct {
                unsigned char   in[DATA_SIZE],
                                encrypted[DATA_SIZE],
                                decrypted[DATA_SIZE],
                                iv[BLOCK_SIZE],
                                key[KEY_SIZE];
        } data;

        /* Make sure a broken connection doesn't kill us */
        signal(SIGPIPE, SIG_IGN);

        /* Create TCP/IP socket, used as main chat channel */
        if ((sd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
                perror("socket");
                exit(1);
        }
        fprintf(stderr, "Created␣TCP␣socket\n");
```

```
/* Bind to a well-known port */
memset(&sa, 0, sizeof(sa));
sa.sin_family = AF_INET;
sa.sin_port = htons(TCP_PORT);
sa.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(sd, (struct sockaddr *)&sa, sizeof(sa)) < 0) {
        perror("bind");
        exit(1);
}
fprintf(stderr, "Bound TCP socket to port %d\n", TCP_PORT);

/* Listen for incoming connections */
if (listen(sd, TCP_BACKLOG) < 0) {
        perror("listen");
        exit(1);
}



cfd = open("/dev/crypto", O_RDWR);
if (cfd < 0) {
        perror("open(/dev/crypto)");
        return 1;
}
/*Key must be the same for client and server. We agree on it
   before starting communicate*/
/*Same for initialization vector (iv)*/
for(i=0; i<KEY_SIZE; i++)        data.key[i]='x';
for(i=0; i<BLOCK_SIZE; i++)      data.iv[i]='y';

/* Loop forever, accept()ing connections */
for (;;) {
        fprintf(stderr, "Waiting for an incoming connection
           ...\n");

        /* Accept an incoming connection */
        len = sizeof(struct sockaddr_in);
        if ((newsd = accept(sd, (struct sockaddr *)&sa, &len)
           ) < 0) {
                perror("accept");
                exit(1);
        }
        if (!inet_ntop(AF_INET, &sa.sin_addr, addrstr, sizeof
           (addrstr))) {
                perror("could not format IP address");
                exit(1);
        }
        fprintf(stderr, "Incoming connection from %s:%d\n",
```

```
        addrstr, ntohs(sa.sin_port));


        memset(&sess, 0, sizeof(sess));
        memset(&cryp, 0, sizeof(cryp));
        /*
        * Get crypto session for AES128
        */
        sess.cipher = CRYPTO_AES_CBC;
        sess.keylen = KEY_SIZE;
        sess.key = data.key;

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


for(;;){

        tmv.tv_sec =1;
        tmv.tv_usec=0;
        FD_ZERO(&read_fd_set);
        FD_SET(0, &read_fd_set);
        FD_SET(newsd,&read_fd_set);
        maxsd=newsd;

        if(select(maxsd+1, &read_fd_set,NULL, NULL, &
           tmv)<0){
          perror ("select");
          exit(1);
        }

        /*Check if someone wrote something. It's
           either us or the other client*/
        if(FD_ISSET(0, &read_fd_set))
                server_writes=1;
        else
                server_writes=0;

        if(FD_ISSET(newsd, &read_fd_set))
                server_reads =1;
        else
                server_reads =0;

        /*In case we wrote something in stdin we need
            to send it*/
        if(server_writes){
```

```
                n=read(0, buf, sizeof(buf));
                if (n < 0) {
                        perror("Reading␣from␣command␣
                           line");
                        break;
                }
                if(n>0)
                        buf[n-1]='\0';
                else
                        buf[0]='\0';

                for(i=0;i<DATA_SIZE;i++)        data.
                   in[i] = buf[i];


                /*
                * Encrypt data.in to data.encrypted
                */
                cryp.ses = sess.ses;
                cryp.len = sizeof(data.in);
                cryp.src = data.in;
                cryp.dst = data.encrypted;
                cryp.iv = data.iv;
                cryp.op = COP_ENCRYPT;

                if (ioctl(cfd, CIOCCRYPT, &cryp)) {
                        perror("ioctl(CIOCCRYPT)");
                        return 1;
                }

                for(i=0;i<DATA_SIZE;i++)        buf[i
                   ]=data.encrypted[i];

                if (insist_write(newsd, data.
                   encrypted, DATA_SIZE) != DATA_SIZE)
                    {
                        perror("write␣to␣remote␣peer␣
                           failed");
                        break;
                }

        }

        /*In case the other client wrote something we
            need to read it*/
        if(server_reads){
```

```
                        n=read(newsd, buf, DATA_SIZE);

                        if (n <= 0) {
                                if (n < 0)
                                        perror("read from
                                           remote peer failed"
                                           );
                                else
                                        fprintf(stderr, "Peer
                                           went away\n");
                                break;
                        }

                        for(i=0;i<DATA_SIZE;i++)  data.
                           encrypted[i]=buf[i];
                        printf("Encrypted data:\n");
                        for(i=0;i<DATA_SIZE;i++)  printf("%x"
                           ,data.encrypted[i]);
                        printf("\n");

                        /*
                        * Decrypt data.encrypted to data.
                           decrypted
                        */
                        cryp.ses = sess.ses;
                        cryp.len = sizeof(data.encrypted);
                        cryp.src = data.encrypted;
                        cryp.dst = data.decrypted;
                        cryp.iv = data.iv;
                        cryp.op = COP_DECRYPT;
                        if (ioctl(cfd, CIOCCRYPT, &cryp)) {
                                perror("ioctl(CIOCCRYPT)");
                                return 1;
                        }


                        for(i=0;i<DATA_SIZE;i++)          buf[i
                           ]=data.decrypted[i];

                        fprintf(stdout, "(Decrypted data)
                           Client said:%s\n", buf);

                }


        }
```

```
                /* Make sure we don't leak open files */
                if (close(newsd) < 0)
                        perror("close");

                /* Finish crypto session */
                if (ioctl(cfd, CIOCFSESSION, &sess.ses)) {
                        perror("ioctl(CIOCFSESSION)");
                        return 1;
                }
        }


        if (close(cfd) < 0) {
                perror("close(fd)");
                return 1;
        }

        /* This will never happen */
        return 1;
}
```

4.2 Η πλευρά του client

```
/*
 * socket-client.c
 * Simple TCP/IP communication using sockets
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 */

#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 <stdbool.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>

#include <arpa/inet.h>
#include <netinet/in.h>
#include <crypto/cryptodev.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
#include "socket-common.h"


/* 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[])
{
        int sd, port, maxsd, cfd, i;
        ssize_t n;
        char buf[DATA_SIZE];
        int client_writes=0, client_reads=0;
        char *hostname;
        struct hostent *hp;
        struct sockaddr_in sa;
        fd_set read_fd_set;
        struct timeval tmv;
        struct session_op sess;
        struct crypt_op cryp;
        struct {
                unsigned char   in[DATA_SIZE],
                                encrypted[DATA_SIZE],
                                decrypted[DATA_SIZE],
                                iv[BLOCK_SIZE],
                                key[KEY_SIZE];
        } data;


        if (argc != 3) {
                fprintf(stderr, "Usage: %s hostname port\n", argv[0])
                   ;
                exit(1);
        }
```

```
hostname = argv[1];
port = atoi(argv[2]);

if(port<=0 || port>65535){
        fprintf(stderr, "%s is not a valid port number.\n",
           argv[2]);
        exit(1);
}

/* Create TCP/IP socket, used as main chat channel */
if ((sd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket");
        exit(1);
}
fprintf(stderr, "Created TCP socket\n");

/* Look up remote hostname on DNS */
if ( !(hp = gethostbyname(hostname))) {
        printf("DNS lookup failed for host %s\n", hostname);
        exit(1);
}

/* Connect to remote TCP port */
sa.sin_family = AF_INET;
sa.sin_port = htons(port);
memcpy(&sa.sin_addr.s_addr, hp->h_addr, sizeof(struct in_addr
   ));
fprintf(stderr, "Connecting to remote host... "); fflush(
   stderr);

if (connect(sd, (struct sockaddr *) &sa, sizeof(sa)) < 0) {
        perror("connect");
        exit(1);
}
fprintf(stderr, "Connected.\n");
fprintf(stderr, "To end the connection press 'Ctrl+c'\n");

/*Key must be the same for client and server. We agree on it
   before starting communicate*/
/*Same for initialization vector (iv)*/
cfd = open("/dev/crypto", O_RDWR);
if (cfd < 0) {
        perror("open(/dev/crypto)");
        return 1;
}
for(i=0;i<KEY_SIZE;i++)          data.key[i]='x';
for(i=0; i<BLOCK_SIZE; i++)      data.iv[i]='y';
```

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

/*
* Get crypto session for AES128
*/
sess.cipher = CRYPTO_AES_CBC;
sess.keylen = KEY_SIZE;
sess.key = data.key;

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

for(;;){
        tmv.tv_sec =1;
        tmv.tv_usec=0;
        FD_ZERO(&read_fd_set);
        FD_SET(0, &read_fd_set);
        FD_SET(sd,&read_fd_set);
        maxsd=sd;

        if(select(maxsd+1, &read_fd_set,NULL, NULL, &tmv)<0){
                perror ("select");
                exit(1);
        }

                /*Check if someone wrote something. It's
                   either us or the other client*/
                if(FD_ISSET(0, &read_fd_set))
                        client_writes=1;
                else
                        client_writes=0;

                if(FD_ISSET(sd, &read_fd_set))
                        client_reads =1;
                else
                        client_reads =0;


                /*In case we wrote something in stdin we need
                    to send it*/
                if(client_writes){

                        n=read(0, buf, sizeof(buf));
                        if (n < 0) {
```

```
                perror("Reading from command
                    line");
                break;
            }
            if(n>0)
                buf[n-1]='\0';
            else
                buf[0]='\0';

            for(i=0;i<DATA_SIZE;i++)          data.
                in[i] = buf[i];


            /*
            * Encrypt data.in to data.encrypted
            */
            cryp.ses = sess.ses;
            cryp.len = sizeof(data.in);
            cryp.src = data.in;
            cryp.dst = data.encrypted;
            cryp.iv = data.iv;
            cryp.op = COP_ENCRYPT;

            if (ioctl(cfd, CIOCCRYPT, &cryp)) {
                perror("ioctl(CIOCCRYPT)");
                return 1;
            }

            for(i=0;i<DATA_SIZE;i++)          buf[i
                ]=data.encrypted[i];

            if (insist_write(sd, data.encrypted,
                DATA_SIZE) != DATA_SIZE) {
                perror("write to remote peer
                    failed");
                break;
            }

        }

        /*In case the other client wrote something we
            need to read it*/
        if(client_reads){

            n=read(sd, buf, DATA_SIZE);

            if (n <= 0) {
                if (n < 0)
```

```
                                        perror("read from
                                            remote peer failed"
                                            );
                                else
                                        fprintf(stderr, "Peer
                                            went away\n");
                                break;
                        }

                        for(i=0;i<DATA_SIZE;i++)  data.
                            encrypted[i]=buf[i];
                        printf("Encrypted data:\n");
                        for(i=0;i<DATA_SIZE;i++)  printf("%x"
                            ,data.encrypted[i]);
                        printf("\n");


                        /*
                        * Decrypt data.encrypted to data.
                            decrypted
                        */
                        cryp.ses = sess.ses;
                        cryp.len = sizeof(data.encrypted);
                        cryp.src = data.encrypted;
                        cryp.dst = data.decrypted;
                        cryp.iv = data.iv;
                        cryp.op = COP_DECRYPT;
                        if (ioctl(cfd, CIOCCRYPT, &cryp)) {
                                perror("ioctl(CIOCCRYPT)");
                                return 1;
                        }


                        for(i=0;i<DATA_SIZE;i++)        buf[i
                            ]=data.decrypted[i];

                        fprintf(stdout, "(Decrypted data)
                            Server said:%s\n", buf);

                }

        }

        /*
         * Let the remote know we're not going to write anything else
            .
         * Try removing the shutdown() call and see what happens.
         */
```

```
        printf("Client exiting...\n");

        /* Finish crypto session */
        if (ioctl(cfd, CIOCFSESSION, &sess.ses)) {
                perror("ioctl(CIOCFSESSION)");
                return 1;
        }

        if (close(cfd) < 0) {
                perror("close(fd)");
                return 1;
        }

        if (shutdown(sd, SHUT_WR) < 0) {
                perror("shutdown");
                exit(1);
        }


        fprintf(stderr, "\nDone.\n");
        return 0;
}
```