ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

Τμήμα Πληροφορικής

ΕΠΛ 421 – Προγραμματισμός Συστημάτων

ΑΣΚΗΣΗ 4 – Ανάπτυξη Ασφαλούς Παράλληλου Εξυπηρετητή Εφαρμογών (Secure Application Server)

Διδάσκων: Δημήτρης Ζεϊναλιπούρ
Υπεύθυνος Εργαστηρίου: Παύλος Αντωνίου

Ημερομηνία Ανάθεσης: Δευτέρα 03/04/2023
Ημερομηνία Παράδοσης: Δευτέρα 24/04/2023

(να υποβληθεί ηλεκτρονικά στο Moodle)

Ι. Στόχος Άσκησης

Στόχος αυτής της εργασίας είναι η εξοικείωση με προχωρημένες τεχνικές προγραμματισμού διεργασιών, δια-διεργασιακής επικοινωνίας μέσω υποδοχών TCP/IP και πολυνηματικών εφαρμογών στη γλώσσα C. Ένας δεύτερος στόχος είναι να σας δοθεί η ευκαιρία να δουλέψετε ομαδικά για να υλοποιήσετε ένα ολοκληρωμένο σύστημα το οποίο θα κριθεί βάση της *ορθότητας* και της *δομής* του.

ΙΙ. Περιγραφή

Αντικείμενο της άσκησης είναι να αναπτύξετε ένα «πολύ-διεργασιακό» (ή «πολυνηματικό») εξυπηρετητή εφαρμογών (application server) στη βάση εξυπηρετητή ιστού (web server) πάνω από το πρωτόκολλο HTTP, ο οποίος θα υποστηρίζει ένα βασικό υποσύνολο της έκδοσης 1.1 του πρωτοκόλλου καθώς και 2 επιπλέον λειτουργίες όπως η εκτέλεση προγραμμάτων γραμμένων σε γλώσσα php και python (η υλοποίηση πρέπει να γίνει στο VM σας όπου είστε root και πρέπει να εγκαταστήσετε τα σχετικά εργαλεία – π.χ., php, python, εάν χρειάζεται). Η έκδοση 1.1 του πρωτοκόλλου HTTP περιγράφεται στο τεχνικό άρθρο Request For Comments 2616: http://www.rfc-editor.org/rfc/rfc2616.txt

Όταν χρησιμοποιούμε ένα φυλλομετρητή σελίδων (browser), όπως ο Mozilla, ο Internet Explorer, κλπ, για να περιηγηθούμε στο WWW, δίνουμε ένα URL της μορφής http://<host>/<path>. Αυτό που κάνει τότε ο φυλλομετρητής (πελάτης) είναι να συνδεθεί μέσω υποδοχών ροής (sockets) στην προκαθορισμένη θύρα (80) για την HTTP υπηρεσία του μηχανήματος <host> (εξυπηρετητής) και να υποβάλει ένα αίτημα, σε αυστηρά καθορισμένη μορφή που θα περιγραφεί στη συνέχεια, για να του αποσταλούν από το web server το αρχείο που βρίσκεται στη θέση /<path> μαζί με κάποιες συνοδευτικές μετα-πληροφορίες. Το /<path> είναι σχετικό σε σχέση με τον κατάλογο-ρίζα του ιεραρχικού συστήματος αρχείων που «σερβίρει» ο εξυπηρετητής. Στον εξυπηρετητή HTTP Apache, από προ-επιλογή, ο κατάλογος-ρίζα είναι ο /var/www/. Όταν ο φυλλομετρητής λάβει το αρχείο, το παρουσιάζει στο χρήστη με τρόπο που κρίνει πρόσφορο. Αν ο web server δεν περιμένει αιτήσεις σύνδεσης από

προγράμματα-πελάτες (clients) στην προκαθορισμένη θύρα 80, αλλά στην <port>, τότε το URL που έπρεπε να δώσουμε στο φυλλομετρητή θα ήταν http://<host>:<port>/<path>.

III. Μηνύματα HTTP

Ο πελάτης και ο εξυπηρετητής HTTP επικοινωνούν μεταξύ τους ανταλλάζοντας μηνύματα με την εξής βασική μορφή [RFC2616/§4.1]:

- Αρχικά υπάρχει μια ακριβώς γραμμή, η οποία ορίζει το είδος της λειτουργίας που σχετίζεται με το αποστελλόμενο μήνυμα. Πρόκειται για μια *γραμμή αίτησης*, αν το μήνυμα αποστέλλεται από τον πελάτη προς το web server, ή μια *γραμμή κατάστασης*, αν το μήνυμα είναι απάντηση του web server προς τον πελάτη.
- Ακολουθούν ένα πλήθος από *γραμμές επικεφαλίδας*.
- Μετά υπάρχει μια ακριβώς κενή γραμμή, η οποία διαχωρίζει τις γραμμές επικεφαλίδας από το σώμα του μηνύματος.
- Τέλος, ακολουθούν τα bytes του σώματος του μηνύματος, αν υπάρχουν.

Οι γραμμές που αναφέρονται παραπάνω χωρίζονται μεταξύ τους από την ακολουθία "\r\n", δηλαδή από δυο χαρακτήρες, τον "\r" (Carriage Return) με ASCII κωδικό 13 και τον "\n" (Line feed) με ASCII κωδικό 10.
Κάθε γραμμή επικεφαλίδας αποτελείται από το όνομα της επικεφαλίδας, το χαρακτήρα ":" και την τιμή της επικεφαλίδας [RFC2616/§4.2] όπως για παράδειγμα:

```
Content-Type: text/html
```

Αν το αποστελλόμενο μήνυμα είναι αίτηση του πελάτη προς το web server, η αρχική γραμμή του (*γραμμή αίτησης*) αποτελείται από μια λέξη η οποία καθορίζει την επιθυμητή λειτουργία (ή μέθοδο π.χ. GET, HEAD, POST, DELETE, PUT κτλ) που αιτείται ο πελάτης, ένα κενό χαρακτήρα, τη διαδρομή του αρχείου πάνω στο οποίο θα εφαρμοσθεί η μέθοδος αυτή, ένα ακόμα κενό χαρακτήρα, τη συμβολοσειρά "HTTP/" και τέλος την έκδοση του HTTP πρωτοκόλλου (για την άσκηση πάντα “1.1”) [RFC2616/§5.1]. Ακολουθεί παράδειγμα:

```
GET /mydirectory/myfile.html HTTP/1.1
```

Αν το αποστελλόμενο μήνυμα είναι απάντηση από το web server, η αρχική γραμμή του (*γραμμή κατάστασης*) αποτελείται από τη συμβολοσειρά "HTTP/" και την έκδοση του HTTP πρωτοκόλλου (για την άσκηση πάντα “1.1”), ένα κενό χαρακτήρα, τον κωδικό κατάληξης της αίτησης και τέλος μια συμβολοσειρά που είναι η λεκτική περιγραφή του κωδικού κατάληξης [RFC2616/§6.1]. Ακολουθούν παραδείγματα:

```
HTTP/1.1 200 OK
HTTP/1.1 404 Not Found
HTTP/1.1 501 Not Implemented
```

Για τις ανάγκες της άσκησης, πρέπει να μπορείτε να χειριστείτε αιτήσεις των μεθόδων **HEAD** [RFC2616/§9.4], **GET** [RFC2616/§9.3], **POST** [RFC2616/§9.5] και **DELETE** [RFC2616/§9.7]. Αν σε μια αίτηση σας ζητηθεί οποιαδήποτε άλλη μέθοδος που δεν είναι υλοποιημένη, πρέπει να απαντήσετε σε αυτή με ένα μήνυμα απάντησης που θα έχει κωδικό κατάληξης της αίτησης το 501 και περιγραφή "Not Implemented" [RFC2616/§10.5.2]. Ακολουθεί παράδειγμα:

```
HTTP/1.1 501 Not Implemented
Server: my_webserver
Connection: close
Content-Type: text/plain
Content-Length: 24

Method not implemented!
```

Η μέθοδος GET [RFC2616/§9.3] που θα υλοποιήσετε θα πρέπει να δοκιμάσει την ανάκτηση του αρχείου που της προσδιορίζει η γραμμή αίτησης μέσα από τον κατάλογο περιεχομένου του web server σας. Αν τα καταφέρει, θα πρέπει να στείλει ένα μήνυμα απάντησης με κωδικό κατάληξης της αίτησης το 200 και περιγραφή "OK" [RFC2616/§10.2.1]. Στο σώμα του μηνύματος πρέπει να βρίσκεται το περιεχόμενο του αρχείου που σας ζητήθηκε. Αν αποτύχει στην ανάκτηση του αρχείου, θα πρέπει να στείλει ένα μήνυμα απάντησης με κωδικό κατάληξης της αίτησης 404 και περιγραφή "Not Found" [RFC2616/§10.4.5]. Αν το αρχείο που ζητείται είναι τύπου (έχει επέκταση) .php ή .py τότε πρώτα θα καλείται ο αντίστοιχος interpreter μέσα από τον οποίο θα περνά το ζητούμενο αρχείο και το output του interpreter (αποτέλεσμα της εκτέλεσης του προγράμματος) θα αποστέλλεται στον πελάτη. Για να εκτελέσουμε ένα αρχείο php από το terminal χρησιμοποιούμε την εντολή *php programName.php* Για να εκτελέσουμε ένα αρχείο python από το terminal χρησιμοποιούμε την εντολή *python3 programName.py* ή *python programName.py*

Τη μέθοδο HEAD πρέπει να τη χειρίζεστε με εντελώς παρόμοιο τρόπο με αυτό για την GET, εκτός από το ότι στην απάντηση από το web server δε θα περιλαμβάνεται το περιεχόμενο του αρχείου που ζητήθηκε [RFC2616/§9.4].
Η εντολή DELETE πρέπει να τη χειρίζεστε με εντελώς παρόμοιο τρόπο με αυτό για την HEAD, με τη διαφορά ότι ο web server θα σβήνει το περιεχόμενο του αρχείου που ζητήθηκε και να επιστρέφει HTTP/1.1 200 OK ή HTTP/1.1 404 Not Found [RFC2616/§9.7].

Κατά την εξυπηρέτηση των αιτήσεων που έρχονται, ο εξυπηρετητής σας θα πρέπει να αντιλαμβάνεται και να ερμηνεύει σωστά την επικεφαλίδα Connection της αίτησης, εφ' όσον υπάρχει, με τον εξής τρόπο: Αν έχει την τιμή close, μετά την εξυπηρέτηση της αίτησης, ο εξυπηρετητής θα πρέπει να τερματίσει τη σύνδεση με τον πελάτη. Στην απάντησή του σε αυτήν την περίπτωση θα πρέπει να συμπεριλάβει τη γραμμή επικεφαλίδας "Connection: close". Διαφορετικά, αν δηλαδή η επικεφαλίδα στην αίτηση του πελάτη δεν έχει την τιμή close ή αν απουσιάζει, τότε ο web server θα πρέπει να περιμένει για νέα μηνύματα αιτήσεων μετά από την εξυπηρέτηση της τρέχουσας αίτησης. Στην απάντησή του, σε αυτή την περίπτωση, θα πρέπει να συμπεριλάβει τη γραμμή επικεφαλίδας "Connection: keep-alive" [RFC2616/§14.10], [RFC2616/§8.1]. Οποιαδήποτε άλλη επικεφαλίδα της αίτησης μπορείτε να την αγνοείτε*.

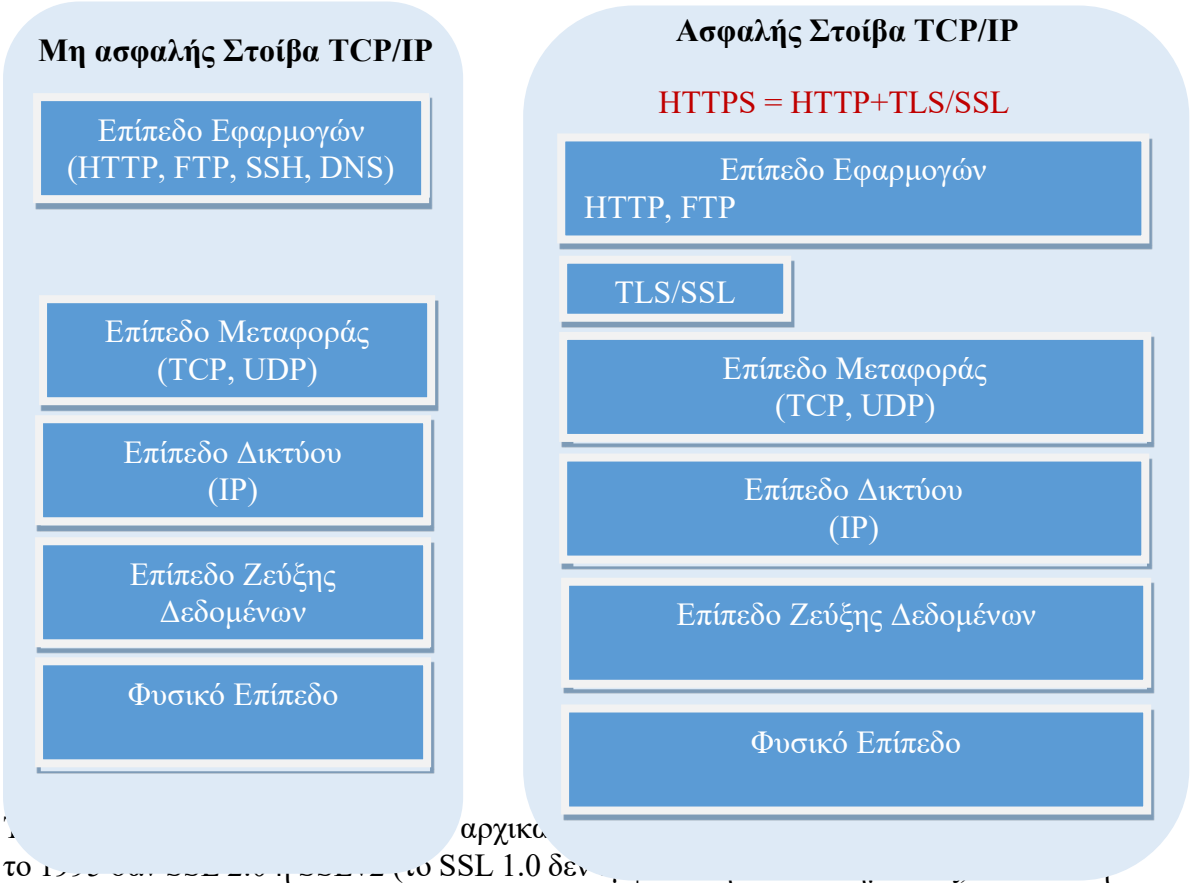
Στα μηνύματα απάντησης που στέλνει ο server σας θα πρέπει να περιλαμβάνει οπωσδήποτε τις ακόλουθες επικεφαλίδες:

* Σημειώνεται ότι στην έκδοση 1.1 του πρωτοκόλλου HTTP, στις αιτήσεις των πελατών είναι υποχρεωτική η επικεφαλίδα HOST, στη μορφή Host: <host>:<port>, όπου <host> είναι το μηχάνημα του εξυπηρετητή και <port> η θύρα στην οποία περιμένει αιτήσεις σύνδεσης, ή απλώς Host: <host>, όπου η θύρα είναι η προκαθορισμένη για το HTTP πρωτόκολλο, η 80 [RFC2616/§14.23]. Παρ' όλα αυτά, στην δική σας υλοποίηση πρέπει να αγνοήσετε αυτό τον περιορισμό γιατί αυτό θα κάνει ευκολότερη την αποσφαλμάτωση της εφαρμογής σας με telnet.

- Connection: Έχει τις τιμές close ή keep-alive, ανάλογα με το αν πρόκειται να κλείσει τη σύνδεση μετά από αυτή την απάντηση, ή αν θα περιμένει για νέες αιτήσεις [RFC2616/§14.10], [RFC2616/§8.1].
- Server: Εδώ σαν τιμή βάζετε το όνομα του εξυπηρετητή σας. [RFC2616/§14.38].
- Content-Length: Έχει σαν τιμή το μέγεθος σε bytes του σώματος του μηνύματος. Πρακτικά δηλαδή είναι το μέγεθος του αρχείου που στέλνετε [RFC2616/§14.13].
- Content-Type: Έχει σαν τιμή τον τύπο του αρχείου που επιστρέφεται, κατά MIME (*Multipurpose Internet Mail Extensions*). Για τις ανάγκες της άσκησης, μπορείτε να διαλέγετε έναν από τους τύπους text/plain (.txt, .sed, .awk, .c, .h), text/html (.html, .htm), text/x-php (.php), application/x-python-code (.py), image/jpeg (.jpeg, .jpg), image/gif (.gif) και application/pdf (.pdf), ανάλογα με την κατάληξη του αρχείου που στέλνετε. Αν η κατάληξη δεν είναι κάποια απ' αυτές, μπορείτε να στείλετε τον τύπο application/octet-stream [RFC2616/§14.17].

IV. Δημιουργία ασφαλούς καναλιού επικοινωνίας

Το πρωτόκολλο HTTP δεν προδιαγράφει ασφαλή μετάδοση εντολών και δεδομένων. Η ασφάλεια των συνδέσεων επιτυγχάνεται στη βάση των πρωτοκόλλων SSL/TLS (Secure Sockets Layer/Transport Layer Security). Ο συνδυασμός των πρωτοκόλλων HTTP και SSL/TLS δημιουργεί το HTTPS.
Τα SSL και TLS πρωτόκολλα είναι socket oriented πρωτόκολλα του επιπέδου μεταφοράς (transport layer) που χρησιμοποιούνται για την δημιουργία ασφαλούς καναλιού επικοινωνίας μεταξύ απομακρυσμένων οντοτήτων (client / server).



αρχικά
το SSL 1.0 δεν
SSL 2.0 αντικαταστάθηκε νωρίς από την έκδοση SSL 3.0 (ή SSLv3) το 1996 μετά από

ένα αριθμό τρωτών σημείων (vulnerabilities) που εμφανίστηκαν. Το TLS εμφανίστηκε το 1999 σαν μια αναβαθμισμένη, πιο ασφαλής έκδοση του SSL, στη βάση του πρωτοκόλλου SSL 3.0. Η τρέχουσα έκδοση του TLS είναι η v1.3. Η έκδοση SSL 2.0 δεν είναι διαλειτουργική με την έκδοση SSL 3.0 και η έκδοση SSL 3.0 δεν είναι διαλειτουργική με την έκδοση 1 του TLS. Τόσο το SSL 2.0 όσο και το 3.0 έχουν χαρακτηριστεί ως πεπαλαιωμένα (deprecated) από το IETF (το 2011 και το 2015, αντίστοιχα). Στην παρούσα άσκηση ο HTTP server που θα υλοποιήσετε θα υποστηρίζει τα πρωτόκολλα SSL 2.0 και SSL 3.0 (δείτε συνάρτηση SSLv23_server_method στον κώδικα του server που θα παρουσιάσουμε πιο κάτω).

Τα πρωτόκολλο SSL δημιουργεί μια ασφαλή σήραγγα (secure tunnel) μεταφοράς δεδομένων η οποία παρέχει κρυπτογράφηση δεδομένων (data encryption), πιστοποίηση ταυτότητας του server (server authentication), πιστοποίηση ταυτότητας του client (client authentication), και μηχανισμούς ακεραιότητας (μη αλλοίωσης) δεδομένων (data integrity). Εμπλέκει τη χρήση ψηφιακών πιστοποιητικών (certificates) τύπου X.509 για την πιστοποίηση ταυτότητας του client και του server. Ένα πιστοποιητικό X.509 περιέχει το δημόσιο κλειδί (public key) μιας οντότητας και την ταυτότητα της οντότητας (π.χ. όνομα κόμβου/hostname, όνομα οργανισμού, όνομα ιδιώτη) και είναι υπογεγραμμένο από κάποια αρχή πιστοποίησης (Certificate Authority, CA) ή αυτό-υπογεγραμμένο (self-signed) με το ιδιωτικό κλειδί (private key) της ίδιας της οντότητας. Και επειδή το SSL χρησιμοποιεί ψηφιακά πιστοποιητικά, απαιτεί συνεπώς την παρουσία υποδομής δημόσιου κλειδιού (Public Key Infrastructure, PKI) και τη συμμετοχή μιας αρχής πιστοποίησης (CA).

Το HTTPS πιστοποιεί την ταυτότητα ενός χρήστη (user authentication) μέσω πιστοποιητικού. Αυτή η διαδικασία διεκπεραιώνεται μέσω των πρωτοκόλλων TLS/SSL. Όταν επιχειρείται ασφαλής σύνδεση σε ένα HTTPS server, ο HTTPS client (π.χ. browser) θα ελέγξει πρώτα αν το πιστοποιητικό του server είναι αξιόπιστο (trusted). Το πιστοποιητικό θεωρείται αξιόπιστο εάν το πιστοποιητικό υπογράφηκε από μια γνωστή αρχή έκδοσης πιστοποιητικών (CA) ή εάν το πιστοποιητικό υπογράφηκε (ψηφιακά) με το ιδιωτικό κλειδί του ίδιου του server (self-signed certificated) και εγκατασταθεί με την έγκριση του χρήστη στην «αποθήκη» αξιόπιστων πιστοποιητικών (Trusted Root Certification Authorities store) του client. Ο server μπορεί επίσης να απαιτήσει από τον client να προσκομίσει το πιστοποιητικό του όταν επιχειρεί να συνδεθεί σε αυτόν. Εάν το πιστοποιητικό του client δεν έχει υπογραφεί από μια γνωστή αρχή έκδοσης πιστοποιητικών, ο server μπορεί να απαιτήσει να του αποσταλεί το πιστοποιητικού ψηφιακά υπογεγραμμένο με το ιδιωτικό κλειδί του client για να το φορτώσει στο αποθηκευτικό χώρο των αξιόπιστων πιστοποιητικών του.

Υπάρχει διαθέσιμο το δωρεάν εργαλείο OpenSSL το οποίο υλοποιεί το πρωτόκολλο TLS/SSL και θα χρησιμοποιηθεί για να δημιουργήσει τα ασφαλή κανάλια (συνδέσεις ελέγχου και συνδέσεις δεδομένων) πάνω από τα οποία θα υλοποιήσετε το πρωτόκολλο HTTP. Για περισσότερες λεπτομέρειες διαβάστε την επόμενη ενότητα.

Αντικείμενο της άσκησης είναι να αναπτύξετε ένα ασφαλή «πολυδιεργασιακό» ή «πολυνηματικό» εξυπηρετητή εφαρμογών, **Application server πάνω από το HTTPS**. Πιο συγκεκριμένα θα υλοποιήσετε το πρωτόκολλο HTTP πάνω από το πρωτόκολλο SSL/TLS. Δεν θα υλοποιήσετε το SSL/TLS αλλά θα χρησιμοποιήσετε έτοιμες συναρτήσεις του εργαλείου OpenSSL που θα αναλάβουν τις διαδικασίες πιστοποίησης ταυτότητας και κρυπτογράφησης / αποκρυπτογράφησης. Ο εξυπηρετητής θα υποστηρίζει ένα βασικό υποσύνολο των εντολών του πρωτοκόλλου HTTP όπως περιγράφηκαν πιο πάνω. Η λειτουργία του πρωτοκόλλου HTTP περιγράφεται στο

τεχνικό άρθρο Request For Comments 2616: http://tools.ietf.org/html/rfc2616. Πιο κάτω θα παρουσιάσουμε κάποια απλά παραδείγματα χρήσης του εργαλείου OpenSSL. Στις μηχανές του εργαστηρίου υπάρχει εγκατεστημένο το εργαλείο OpenSSL τόσο για χρήση από τη γραμμή εντολών (command line tool) όσο και σαν βιβλιοθήκη της γλώσσας C. Το πιο κάτω πρόγραμμα (tls_server.c) δημιουργεί ένα απλό TLS server που ακούει για συνδέσεις στη θύρα 4433. Δείτε περισσότερα πιο κάτω.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
#include <string.h>

int create_socket(int port)
{
    int s;
    struct sockaddr_in addr;

    /* set the type of connection to TCP/IP */
    addr.sin_family = AF_INET;
    /* set the server port number */
    addr.sin_port = htons(port);
    /* set our address to any interface */
    addr.sin_addr.s_addr = htonl(INADDR_ANY);

    s = socket(AF_INET, SOCK_STREAM, 0);
    if (s < 0) {
        perror("Unable to create socket");
        exit(EXIT_FAILURE);
    }

    /* bind serv information to s socket */
    if (bind(s, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
        perror("Unable to bind");
        exit(EXIT_FAILURE);
    }

    /* start listening allowing a queue of up to 1 pending connection */
    if (listen(s, 1) < 0) {
        perror("Unable to listen");
        exit(EXIT_FAILURE);
    }

    return s;
}

void init_openssl()
{
    SSL_load_error_strings();
    OpenSSL_add_ssl_algorithms();
}
```

```
void cleanup_openssl()
{
    EVP_cleanup();
}
```

```
SSL_CTX *create_context()
{
    const SSL_METHOD *method;
    SSL_CTX *ctx;

    /* The actual protocol version used will be negotiated to the
     * highest version mutually supported by the client and the server.
     * The supported protocols are SSLv3, TLSv1, TLSv1.1 and TLSv1.2.
     */
    //method = SSLv23_server_method();
    method = TLSv1_2_server_method();

    /* creates a new SSL_CTX object as framework to establish TLS/SSL or
     * DTLS enabled connections. It initializes the list of ciphers, the
     * session cache setting, the callbacks, the keys and certificates,
     * and the options to its default values
     */
    ctx = SSL_CTX_new(method);
    if (!ctx) {
        perror("Unable to create SSL context");
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }

    return ctx;
}

void configure_context(SSL_CTX *ctx)
{
    SSL_CTX_set_ecdh_auto(ctx, 1);

    /* Set the key and cert using dedicated pem files */
    if (SSL_CTX_use_certificate_file(ctx, "cert.pem", SSL_FILETYPE_PEM)
<= 0) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }

    if (SSL_CTX_use_PrivateKey_file(ctx, "key.pem", SSL_FILETYPE_PEM) <=
0 ) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }
}

int main(int argc, char **argv)
{
    int sock;
    SSL_CTX *ctx;

    /* initialize OpenSSL */
    init_openssl();

    /* setting up algorithms needed by TLS */
    ctx = create_context();

    /* specify the certificate and private key to use */
    configure_context(ctx);

    sock = create_socket(4433);

    /* Handle connections */
    while(1) {
        struct sockaddr_in addr;
        uint len = sizeof(addr);
        SSL *ssl;
```

```
        const char reply[] = "test\n";

        /* Server accepts a new connection on a socket.
         * Server extracts the first connection on the queue
         * of pending connections, create a new socket with the same
         * socket type protocol and address family as the specified
         * socket, and allocate a new file descriptor for that socket.
         */
        int client = accept(sock, (struct sockaddr*)&addr, &len);
        if (client < 0) {
            perror("Unable to accept");
            exit(EXIT_FAILURE);
        }

        /* creates a new SSL structure which is needed to hold the data
         * for a TLS/SSL connection
         */
        ssl = SSL_new(ctx);
        SSL_set_fd(ssl, client);

        /* wait for a TLS/SSL client to initiate a TLS/SSL handshake */
        if (SSL_accept(ssl) <= 0) {
            ERR_print_errors_fp(stderr);
        }
        /* if TLS/SSL handshake was successfully completed, a TLS/SSL
         * connection has been established
         */
        else {
            /* writes num bytes from the buffer reply into the
             * specified ssl connection
             */
            SSL_write(ssl, reply, strlen(reply));
        }

        /* close ssl connection */
        SSL_shutdown(ssl);
        /* free an allocated SSL structure */
        SSL_free(ssl);
        close(client);
    }

    close(sock);
    SSL_CTX_free(ctx);
    cleanup_openssl();
}
```

Όπως θα προσέξετε στη συνάρτηση configure_context απαιτείται η χρήση αρχείου που να περιέχει το ψηφιακό πιστοποιητικό (cert.pem) εξυπηρετητή και ενός άλλου αρχείου (key.pem) που να περιέχει το ιδιωτικό κλειδί του εξυπηρετητή. Μπορούμε να δημιουργήσουμε και τα 2 αυτά αρχεία μέσω του εργαλείου openssl από τη γραμμή εντολών μέσω της εντολής:

```
openssl req -newkey rsa:2048 -new -nodes -x509 -days 3650 -keyout
key.pem -out cert.pem
```

η οποία δημιουργεί το ζεύγος δημόσιου και ιδιωτικού κλειδιού με βάση τον αλγόριθμο RSA, και αποθηκεύει το δημόσιο κλειδί μέσα στο πιστοποιητικό τύπου x509 (cert.pem) το οποίο ισχύει 3650 μέρες (10 χρόνια) προτού λήξει, και το ιδιωτικό κλειδί μέσα στο αρχείο key.pem. Και τα 2 αυτά κλειδιά έχουν μέγεθος 2048 bits. Κατά τη διάρκεια της δημιουργίας του κλειδιού, το εργαλείο ζητά να δοθούν κάποιες πληροφορίες για τον ιδιοκτήτη των κλειδιών (π.χ. χώρα, επαρχία, πόλη, όνομα οργανισμού, όνομα κόμβου/hostname, email). Αν θέλετε, μπορείτε να τα αφήσετε όλα κενά.

Όταν ο πιο πάνω εξυπηρετητής δεχθεί μια αίτηση για σύνδεση και γίνει αποδεκτή (από τη συνάρτηση accept), αρχίζει η διαδικασία πιστοποίησης ταυτότητας μέσω ανταλλαγής πιστοποιητικών. Αν η πιστοποίηση ταυτότητας είναι επιτυχής, ο εξυπηρετητής στέλνει πίσω τη συμβολοσειρά test και τερματίζει τη σύνδεση.

Στη συνέχεια μπορείτε να μεταγλωττίσετε με την εντολή:

```
gcc -o tls_server tls_server.c -lssl -lcrypto
```

και να τρέξετε το πρόγραμμα με την εντολή:

```
./tls_server
```

Για να δείτε ότι τρέχει το πρόγραμμα, χρειάζεστε ένα πρόγραμμα πελάτη (client). Αυτό μπορεί να γίνει με 2 τρόπους. Ο πρώτος τρόπος είναι με το εργαλείο openssl από τη γραμμή εντολών:

```
openssl s_client -connect b103ws12.in.cs.ucy.ac.cy:4433
```

(προσέξετε να δώσετε το σωστό όνομα μηχανής πάνω στο οποίο τρέχει ο εξυπηρετητής και τη σωστή θύρα). Μετά την εκτέλεση της εντολής θα δείτε στην οθόνη τη διαδικασία ανταλλαγής πιστοποιητικών (SSL handshake and negotiation phase) και στο τέλος το μήνυμα test που στάλθηκε από τον εξυπηρετητή.

Ο δεύτερος τρόπος είναι γράφοντας ένα μικρό πρόγραμμα σε γλώσσα C (δείτε πρόγραμμα tls_client.c στο as4-supplementary.zip).

V. Ζητούμενα άσκησης

Στη δική σας άσκηση θα τροποποιήσετε τον πιο πάνω κώδικα (tls_server.c) για να δημιουργήσετε ένα ασφαλή, πολύ-διεργασιακό ή πολύ-νηματικό HTTPS server που χειρίζεται τα μηνύματα που περιεγράφηκαν στην ενότητα III. Ο HTTPS server θα πρέπει να παρέχει ασφάλεια (μέσω TLS/SSL). Για την πολύ-διεργασιακή ή πολύ-νηματική λειτουργία δείτε την ενότητα VII.

Για να δοκιμάσετε τον HTTPS server σας θα χρειαστείτε ένα HTTPS client για να ανεβάζετε αρχεία στο server ή να κατεβάζετε αρχεία από το server. Μπορείτε να γράψετε το δικό σας client σε γλώσσα C (στη βάση του tls_client) ή να δοκιμάσετε αν κάποιος εμπορικός HTTPS client (π.χ. Postman, Chrome, Firefox) μπορεί να συνδεθεί με τον server σας. Συστήνουμε το Postman με το οποίο μπορείτε να δοκιμάσετε να στείλετε οποιαδήποτε αίτηση HTTP (οι browser στέλνουν αυτόματα GET requests για να κατεβεί κάποια σελίδα, ενώ για να στείλουν οποιοδήποτε άλλη εντολή (POST, DELETE, HEAD) πρέπει να χρησιμοποιηθεί JavaScript). Στην περίπτωση που ο server σας θα μπορεί να εξυπηρετήσει εμπορικό HTTPS client, θα έχετε επιπλέον BONUS στη βαθμολογία σας. Επίσης θα πρέπει να δημιουργήσετε ένα κατάλογο με όνομα httpshome/ μέσα στον ίδιο κατάλογο που είναι το .c πρόγραμμά σας ο οποίος θα περιέχει τα αρχεία που μπορεί να «σερβίρει» ο HTTPS server και στον οποίο επίσης να μπορεί να τοποθετεί τα αρχεία που θα ανεβάζουν οι clients (το όνομα του καταλόγου αυτού μπορείτε να το βρίσκετε και μέσα στο config file – δες ενότητα VII).

VI. Παράδειγμα Εκτέλεσης

Ακολουθούν παραδείγματα αιτήσεων από ένα πελάτη (αριστερή στήλη) και οι αντίστοιχες απαντήσεις από ένα εξυπηρετητή (δεξιά στήλη). Τα παραδείγματα είναι ενδεικτικά της λειτουργίας του πρωτοκόλλου και δεν στηρίζονται σε αρχεία που υπάρχουν στον πιο πάνω δοθέντα ιστοχώρο.

```
GET /sample.html HTTP/1.1
User-Agent: My_web_browser
Host: astarti.cs.ucy.ac.cy:30000
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Server: My_test_server
Content-Length: 211
Connection: keep-alive
Content-Type: text/html

<html>
<head>
    <title>It worked!!!</title>
</head>
<body>
    <h1>Yes, It worked!!!</h1>
    Click at the image to see a
     sample text!<br>
     <a href="sample.txt">
         <img src="sample.gif">
     </a>
</body>
</html>
```

```
GET /sample.txt HTTP/1.1
User-Agent: My_web_browser
Host: astarti.cs.ucy.ac.cy:30000
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Server: My_test_server
Content-Length: 24
Connection: keep-alive
Content-Type: text/plain

Yes, this works also!!!
```

```
GET /dir/non-existent.gif HTTP/1.1
User-Agent: My_web_browser
Host: astarti.cs.ucy.ac.cy:30000
Connection: keep-alive
```

```
HTTP/1.1 404 Not Found
Server: My_test_server
Content-Length: 20
Connection: keep-alive
Content-Type: text/plain

Document not found!
```

```
OPTIONS * HTTP/1.1
User-Agent: My_web_browser
Host: astarti.cs.ucy.ac.cy:30000
Connection: close
```

```
HTTP/1.1 501 Not Implemented
Server: My_test_server
Content-Length: 24
Connection: close
Content-Type: text/plain

Method not implemented!
```

VII. Παραλληλη (πολύ-διεργασιακή ή πολύ-νηματική) λειτουργία εξυπηρετητή

Ο HTTPS server που θα υλοποιήσετε πρέπει να είναι σε θέση να εξυπηρετεί «ταυτόχρονα» πολλές αιτήσεις από πελάτες. Δεν πρέπει, δηλαδή, να τελειώσει πρώτα με την εξυπηρέτηση μιας αίτησης και μετά να δέχεται νέες. Για να το πετύχετε αυτό, θα πρέπει να εκμεταλλευτείτε τη δυνατότητα ύπαρξης πολλών διεργασιών ή νημάτων μέσα στη διεργασία του HTTPS server. Μια ιδέα θα ήταν, όταν παίρνει μια αίτηση από πελάτη, να δημιουργεί μια διεργασία ή νήμα για να την εξυπηρετήσει, ενώ η αρχική διεργασία ή νήμα να περιμένει νέες αιτήσεις. Η εξυπηρέτηση των αιτήσεων αυτών θα γίνεται από νέες διεργασίες ή νήματα που θα δημιουργεί η αρχική διεργασία ή νήμα. Φυσικά, όταν ένα νήμα τελειώνει την αποστολή του, θα πρέπει να τερματίζει. **Η προσέγγιση αυτή δεν είναι πολύ καλή**, γιατί δεν είναι ιδιαίτερα ελεγχόμενη η δημιουργία και καταστροφή νημάτων ή διεργασίων στην εφαρμογή, κάτι που μπορεί να αποβεί εξαιρετικά προβληματικό σε κάποιες περιπτώσεις.

Μια άλλη, καλύτερη, ιδέα είναι το αρχικό νήμα να δημιουργήσει ένα thread-pool ή process-pool, δηλαδή να δημιουργήσει εξ αρχής ένα σταθερό αριθμό νημάτων ή διεργασιών εργατών (που το πλήθος τους να δίνεται) και όταν υπάρχει αίτηση για εξυπηρέτηση να την αναθέτει σε κάποιο από τα νήματα ή διεργασία αυτά που δεν έχει δουλειά. Οι διεργασίες ή τα νήματα, αφού εξυπηρετήσουν ένα πελάτη, δεν τερματίζουν, αλλά μεταβαίνουν σε κατάσταση αναμονής. Φυσικά, αν δεν υπάρχει διαθέσιμο νήμα ή διεργασία, το αρχικό θα πρέπει να περιμένει μέχρι να υπάρξει, χωρίς να δέχεται νέες αιτήσεις. Συγκεκριμένα εάν υπάρξουν περισσότερες αιτήσεις από το μέγιστο αριθμό νημάτων ή διεργασιών στο pool τότε το σύστημα απορρίπτει την αίτηση κλείνοντας το socket (χωρίς να επιστρέφει οποιανδήποτε απάντηση στον πελάτη). Άλλες παραμέτρους που χρειάζεται να πάρει ο HTTPS server σας, εκτός από το πλήθος των νημάτων ή διεργασιών, είναι ο αριθμός θύρας στον οποίο θα αναμένει αιτήσεις, ο κατάλογος-ρίζα του ιεραρχικού συστήματος αρχείων που «σερβίρει» και άλλες παραμέτρους που τυχόν χρησιμοποιήσετε. Οι παράμετροι μπορεί είτε να δίδονται ως ορίσματα στο πρόγραμμά σας ή καλύτερα να βρίσκονται σε κάποιο αρχείο config.txt, το οποίο θα έχει τη δομή (παράδειγμα για νήματα):

```
# HTTPS server Configuration File

# The Number of Threads in the Threadpool
THREADS=40

# The Port number of the HTTPS server
PORT=30000

# The HOME folder of the HTTPS server
HOME=./httpshome
…
```

VIII. Ανάπτυξη Λογισμικού

Η άσκηση αυτή θα υλοποιηθεί σε ομάδες όπως έχουν αναρτηθεί στο Moodle για την παρουσίαση, των οποίων τα άτομα αναμένεται να συμβάλουν ισομερώς σε χρόνο και ουσιαστική δουλειά.

IX. Αξιολόγηση
Α) Τι πρέπει να παραδώσετε;

- **Στο Moodle:** Ενα αρχείο **https.tar.gz** το οποίο θα περιέχει:
  1. Τον πηγαίο κώδικα μαζί με το σχετικό Makefile,
  2. Ένα README.txt αρχείο οποίο θα δίδει οδηγίες χρήσης του συστήματός σας (περίπου 1 σελίδα) και
  3. Architecture (DOC ή PDF), το οποίο θα περιγράφει την αρχιτεκτονική του συστήματος, τις βασικές επιλογές στο σχεδιασμό αυτής της αρχιτεκτονικής, περιγραφή της επιπλέον λειτουργίας που αποφασίσατε να υλοποιήσετε, διάφορες δυσκολίες που αντιμετωπίσατε (~2-3 σελίδες).

Β) Κριτήρια Αξιολόγησης.

1. **Δομή Συστήματος:** Το σύστημα πρέπει να χρησιμοποιεί τεχνικές δομημένου προγραμματισμού με τη χρήση συναρτήσεων, αρχείων επικεφαλίδας (.h), πολλαπλών αρχείων για καλύτερη δομή του πηγαίου κώδικα, Makefile, αρχείων ελέγχου (unit tests) τα οποία θα ελέγχουν την ορθότητα των συστατικών (modules) του συστήματος σας ανεξάρτητα από την υπόλοιπη λειτουργία του συστήματος, διαχείριση λαθών συστήματος με την perror, έλεγχος ταυτοχρονίας νημάτων με χρήση σηματοφόρων, κτλ.

2. **Ορθότητα Λειτουργίας:** Το σύστημα θα πρέπει να διεκπεραιώνει ορθά τις λειτουργίες του συστήματος όπως αυτές περιγράφονται σε αυτή την εκφώνηση και το RFC2616. Η εκφώνηση της άσκησης δεν σας δεσμεύει για τις δυνατότητες που θα έχει ο εξυπηρετητής που θα υλοποιήσετε. **Η εκφώνηση απλά θέτει ένα ελάχιστο όριο δυνατοτήτων που θα πρέπει να υλοποιήσετε.** Αυτό είναι σκόπιμο για να σας αφήσει αρκετή ελευθερία στη λήψη πρωτοβουλιών και στην εκδήλωση δημιουργικότητας από την πλευρά σας. Μέσα από αυτή την άσκηση θέλουμε να σας δοθεί η δυνατότητα να επεξεργαστείτε από μόνοι σας ένα τεχνικό έγγραφο (RFC2616) καθώς επίσης να ανακαλύψετε νέες συναρτήσεις πέρα από αυτές που διδαχθήκατε ήδη στις διαλέξεις.

Καλή Επιτυχία !