ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΗΜΙΟΥΡΓΙΑ ΔΙΕΡΓΑΣΙΑΣ WEB-SERVER ΜΕ ΝΗΜΑΤΑ (THREADS) ΣΕ LINUX

ΚΑΛΟΠΗΤΑΣ ΚΩΝΣΤΑΝΤΙΝΟΣ - 4190

ΣΒΕΝΤΖΟΥΡΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ - 4086

Επιβλέπων Καθηγητής: Μηνάς Δασυγένης

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ ΞΑΝΘΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΚΟΠΟΣ ΤΗΣ ΕΡΓΑΣΙΑΣ

Να προγραμματίσετε σε περιβάλλον linux μια απλοποιημένη διεργασία webserver. Αυτή η διεργασία θα δημιουργεί έναν αριθμό από threads τα οποία είναι worker threads. Όταν έρθει μια αίτηση στην θύρα 80, τότε το master thread διαλέγει ένα thread για να προωθήσει το request το οποίο στέλνει ότι αρχεία του έχουν ζητηθεί. Π.χ. Αν γίνει το request "GET INDEX.HTML" τότε θα στέλνει το αρχείο INDEX.HTML . Τα worker threads θα πρέπει να έχουν μια στοιχειώδη κατανόηση του πρωτοκόλλου HTTP (να αγνοούνται επιλογές ή εντολές εκτός από GET) ώστε να μπορεί να συνδεθούμε με κάποιο Browser και να δούμε μια σελίδα.

Ο ΚΩΔΙΚΑΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ

```
/*
  (c) Copyright OS-Project_2007
        Sventzouris Konstantinos - Kalopitas Konstantinos
        Dimokritous University Of Thrace !!!  :-)

Web-Server με νήματα
*/

/*********************************/

#include <sys/socket.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/time.h>
#include <errno.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

/*********************************/

#ifndef PG_HELPER_H
#define PG_HELPER_H

void    Error_Quit(char const * msg);
```

```
int     Trim      (char * buffer);
int     StrUpper  (char * buffer);
void    CleanURL  (char * buffer);
ssize_t Readline  (int sockd, void *vptr, size_t maxlen);
ssize_t Writeline (int sockd, const void *vptr, size_t n);

#define LISTENQ         (1024)
#endif

/*********************************/

#ifndef PG_SERVREQ_H
#define PG_SERVREQ_H

int Service_Request(int conn);

#endif

/*********************************/

#ifndef PG_REQHEAD_H
#define PG_REQHEAD_H

/*  Μέθοδοι τις οποίες χρειάζεται ο χρήστης  */

enum Req_Method { GET, HEAD, UNSUPPORTED };
enum Req_Type   { SIMPLE, FULL };

struct ReqInfo {
   enum Req_Method method;
   enum Req_Type   type;
   char           *referer;
   char           *useragent;
   char           *resource;
   int             status;
};

#define MAX_REQ_LINE        (1024)

int  Parse_HTTP_Header(char * buffer, struct ReqInfo * reqinfo);
int  Get_Request      (int conn, struct ReqInfo * reqinfo);
void InitReqInfo      (struct ReqInfo * reqinfo);
void FreeReqInfo      (struct ReqInfo * reqinfo);

#endif

/*********************************/

#ifndef PG_RESOURCE_H
```

```
#define PG_RESOURCE_H

int Return_Resource (int conn, int resource, struct ReqInfo * reqinfo);
int Check_Resource  (struct ReqInfo * reqinfo);
int Return_Error_Msg(int conn, struct ReqInfo * reqinfo);

#endif

/*********************************/

#ifndef PG_RESPHEAD_H
#define PG_RESPHEAD_H

int Output_HTTP_Headers(int conn, struct ReqInfo * reqinfo);

#endif

/*********************************/

/* Μήνυμα λάθους πριν την έξοδο  */

void Error_Quit(char const * msg) {
   fprintf(stderr, "WEBSERV: %s\n", msg);
   exit(EXIT_FAILURE);
}

 /*  Διαβάζει μια γραμμή   */

ssize_t Readline(int sockd, void *vptr, size_t maxlen) {
   ssize_t n, rc;
   char    c, *buffer;

   buffer = vptr;

   for ( n = 1; n < maxlen; n++ ) {

         if ( (rc = read(sockd, &c, 1)) == 1 ) {
            *buffer++ = c;
            if ( c == '\n' )
                  break;
         }
         else if ( rc == 0 ) {
            if ( n == 1 )
                  return 0;
            else
                  break;
         }
         else {
            if ( errno == EINTR )
```

```
                continue;
            Error_Quit("Error in Readline()");
        }
    }

    *buffer = 0;
    return n;
}


/*  Γράφει μια γραμμή   */

ssize_t Writeline(int sockd, const void *vptr, size_t n) {
    size_t      nleft;
    ssize_t     nwritten;
    const char *buffer;

    buffer = vptr;
    nleft  = n;

    while ( nleft > 0 ) {
        if ( (nwritten = write(sockd, buffer, nleft)) <= 0 ) {
            if ( errno == EINTR )
                nwritten = 0;
            else
                Error_Quit("Error in Writeline()");
        }
        nleft  -= nwritten;
        buffer += nwritten;
    }

    return n;
}

/*  Αφαιρεί τα κενά  */

int Trim(char * buffer) {
    int n = strlen(buffer) - 1;

    while ( !isalnum(buffer[n]) && n >= 0 )
        buffer[n--] = '\0';

    return 0;
}


/*  Μετατροπή ενός string σε κεφαλαία  */

int StrUpper(char * buffer) {
    while ( *buffer ) {
```

```
        *buffer = toupper(*buffer);
        ++buffer;
    }
    return 0;
}

/*  Αφαίρεση κωδικοποιημένων διευθύνσεων από το string  */

void CleanURL(char * buffer) {
    char asciinum[3] = {0};
    int i = 0, c;

    while ( buffer[i] ) {
        if ( buffer[i] == '+' )
            buffer[i] = ' ';
        else if ( buffer[i] == '%' ) {
            asciinum[0] = buffer[i+1];
            asciinum[1] = buffer[i+2];
            buffer[i] = strtol(asciinum, NULL, 16);
            c = i+1;
            do {
                buffer[c] = buffer[c+2];
            } while ( buffer[2+(c++)] );
        }
        ++i;
    }
}

/*******************************/

int Parse_HTTP_Header(char * buffer, struct ReqInfo * reqinfo) {

    static int first_header = 1;
    char      *temp;
    char      *endptr;
    int       len;


    if ( first_header == 1 ) {

        /*  Αν το first_header είναι 0, είναι η πρώτη γραμμή της  HTTP κλήσης  */


        /*  Απάντηση σε κλήσεις  GET και HEAD  */

        if ( !strncmp(buffer, "GET ", 4) ) {
            reqinfo->method = GET;
            buffer += 4;
        }
        else if ( !strncmp(buffer, "HEAD ", 5) ) {
```

```
            reqinfo->method = HEAD;
            buffer += 5;
        }
        else {
            reqinfo->method = UNSUPPORTED;
            reqinfo->status = 501;
            return -1;
        }

        while ( *buffer && isspace(*buffer) )
            buffer++;

        endptr = strchr(buffer, ' ');
        if ( endptr == NULL )
            len = strlen(buffer);
        else
            len = endptr - buffer;
        if ( len == 0 ) {
            reqinfo->status = 400;
            return -1;
        }

        reqinfo->resource = calloc(len + 1, sizeof(char));
        strncpy(reqinfo->resource, buffer, len);

        /*  Δοκιμή αν υπάρχει πληροφορία της έκδοσης του  HTTP  */

        if ( strstr(buffer, "HTTP/") )
            reqinfo->type = FULL;
        else
            reqinfo->type = SIMPLE;

        first_header = 0;
        return 0;
    }

    endptr = strchr(buffer, ':');
    if ( endptr == NULL ) {
        reqinfo->status = 400;
        return -1;
    }

    temp = calloc( (endptr - buffer) + 1, sizeof(char) );
    strncpy(temp, buffer, (endptr - buffer));
    StrUpper(temp);


    /*  Αύξηση του buffer για να δείχνει στη νέα τιμή  και επιστροφή αν είναι 0   */

    buffer = endptr + 1;
```

```
    while ( *buffer && isspace(*buffer) )
            ++buffer;
    if ( *buffer == '\0' )
            return 0;

    if ( !strcmp(temp, "USER-AGENT") ) {
                reqinfo->useragent = malloc( strlen(buffer) + 1 );
                strcpy(reqinfo->useragent, buffer);
    }
    else if ( !strcmp(temp, "REFERER") ) {
                reqinfo->referer = malloc( strlen(buffer) + 1 );
                strcpy(reqinfo->referer, buffer);
    }

    free(temp);
    return 0;
}

int Get_Request(int conn, struct ReqInfo * reqinfo) {

    char   buffer[MAX_REQ_LINE] = {0};
    int    rval;
    fd_set fds;
    struct timeval tv;

    /*  Θέτουμε το  timeout ίσο με  5 sec   */

    tv.tv_sec  = 5;
    tv.tv_usec = 0;

    do {

            FD_ZERO(&fds);
            FD_SET (conn, &fds);

            /*  Περιμένει  μέχρι το timeout για να δει αν η είσοδος είναι έτοιμη   */

            rval = select(conn + 1, &fds, NULL, NULL, &tv);

            if ( rval < 0 ) {
               Error_Quit("Error calling select() in get_request()");
            }
            else if ( rval == 0 ) {

               /*  Η είσοδος δεν είναι έτοιμη μετά το timeout αρά επιστροφή λάθους  */

               return -1;

            }
            else {
```

```
            /* Έχουμε μια γραμμή σε αναμονή και αρά τη  διαβάζουμε  */

            Readline(conn, buffer, MAX_REQ_LINE - 1);
            Trim(buffer);

            if ( buffer[0] == '\0' )
                  break;

            if ( Parse_HTTP_Header(buffer, reqinfo) )
                  break;
          }
   } while ( reqinfo->type != SIMPLE );

   return 0;
}

void InitReqInfo(struct ReqInfo * reqinfo) {
   reqinfo->useragent = NULL;
   reqinfo->referer   = NULL;
   reqinfo->resource  = NULL;
   reqinfo->method    = UNSUPPORTED;
   reqinfo->status    = 200;
}
/*  Ελευθερώνει μνήμη  */

void FreeReqInfo(struct ReqInfo * reqinfo) {
   if ( reqinfo->useragent )
         free(reqinfo->useragent);
   if ( reqinfo->referer )
         free(reqinfo->referer);
   if ( reqinfo->resource )
         free(reqinfo->resource);
}

/*********************************/

/*  Καθορισμός του ριζικού καταλόγου που χρησιμοποιεί ο Server
   Στην περίπτωση μας χρησιμοποιήσαμε αυτόν τον κατάλογο : "/home/OS-
project/Web-Server" */

static char server_root[1000] = "/home/OS-project/Web-Server";

int Return_Resource(int conn, int resource, struct ReqInfo * reqinfo) {

   char c;
   int  i;

   while ( (i = read(resource, &c, 1)) ) {
         if ( i < 0 )
            Error_Quit("Error reading from file.");
```

```
        if ( write(conn, &c, 1) < 1 )
            Error_Quit("Error sending file.");
    }

    return 0;
}

int Check_Resource(struct ReqInfo * reqinfo) {

    CleanURL(reqinfo->resource);

    strcat(server_root, reqinfo->resource);
    return open(server_root, O_RDONLY);
}

/*  Επιστροφή μηνύματος λάθους */

int Return_Error_Msg(int conn, struct ReqInfo * reqinfo) {

    char buffer[100];

    sprintf(buffer, "<HTML>\n<HEAD>\n<TITLE>Server Error %d</TITLE>\n"
                    "</HEAD>\n\n", reqinfo->status);
    Writeline(conn, buffer, strlen(buffer));

    sprintf(buffer, "<BODY>\n<H1>Server Error %d</H1>\n", reqinfo->status);
    Writeline(conn, buffer, strlen(buffer));

    sprintf(buffer, "<P>The request could not be completed.</P>\n"
                    "</BODY>\n</HTML>\n");
    Writeline(conn, buffer, strlen(buffer));

    return 0;

}

/*******************************/

int Output_HTTP_Headers(int conn, struct ReqInfo * reqinfo) {

    char buffer[100];

    sprintf(buffer, "HTTP/1.0 %d OK\r\n", reqinfo->status);
    Writeline(conn, buffer, strlen(buffer));

    Writeline(conn, "Server: PGWebServ v0.1\r\n", 24);
    Writeline(conn, "Content-Type: text/html\r\n", 25);
    Writeline(conn, "\r\n", 2);

    return 0;
```

```
}

/*******************************/

int Service_Request(int conn) {

   struct ReqInfo  reqinfo;
   int             resource = 0;

   InitReqInfo(&reqinfo);

   if ( Get_Request(conn, &reqinfo) < 0 )
         return -1;

   if ( reqinfo.status == 200 )
         if ( (resource = Check_Resource(&reqinfo)) < 0 ) {
            if ( errno == EACCES )
                  reqinfo.status = 401;
            else
                  reqinfo.status = 404;
         }

   if ( reqinfo.type == FULL )
         Output_HTTP_Headers(conn, &reqinfo);

   if ( reqinfo.status == 200 ) {
         if ( Return_Resource(conn, resource, &reqinfo) )
            Error_Quit("Something wrong returning resource.");
   }
   else
         Return_Error_Msg(conn, &reqinfo);

   if ( resource > 0 )
         if ( close(resource) < 0 )
            Error_Quit("Error closing resource.");
   FreeReqInfo(&reqinfo);

   return 0;
}

/*******************************/

/* Καθορίζουμε το port που να ακούει ο Server */

#define SERVER_PORT           (80)

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

   int    listener, conn;
   pid_t  pid;
```

```
struct sockaddr_in servaddr;

/* Δημιουργία Thread  */

if ( (listener = socket(AF_INET, SOCK_STREAM, 0)) < 0 )
      Error_Quit("Couldn't create listening socket.");

memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family      = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port        = htons(SERVER_PORT);

/*  Δέσμευση του thread  */

if ( bind(listener, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0 )
      Error_Quit("Couldn't bind listening thread.");

/*  Κάνουμε ένα Thread να είναι Listening Thread   */

if ( listen(listener, LISTENQ) < 0 )
      Error_Quit("Call to listen failed.");

/*  Επανάληψη της αποδοχής των συνδέσεων   */

while ( 1 ) {

      /*  Αναμονή για μια σύνδεση   */

      if ( (conn = accept(listener, NULL, NULL)) < 0 )
         Error_Quit("Error calling accept()");

      /*  Δημιουργία ενός Worker Thread για την εξυπηρέτηση της Σύνδεσης   */

      if ( (pid = fork()) == 0 ) {

         if ( close(listener) < 0 )
               Error_Quit("Error closing listening Thread.");

         Service_Request(conn);

         if ( close(conn) < 0 )
               Error_Quit("Error closing connection Thread.");
         exit(EXIT_SUCCESS);
      }

      if ( close(conn) < 0 )
         Error_Quit("Error closing connection Thread.");

      waitpid(-1, NULL, WNOHANG);
```

```
    }

    return EXIT_FAILURE;    /*  ΛΑΘΟΣ     */
}
```

ΤΡΟΠΟΣ ΕΓΚΑΤΑΣΤΑΣΗΣ – ΕΚΤΕΛΕΣΗΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

- ΔΗΜΙΟΥΡΓΙΑ ΤΩΝ ΕΠΙΘΥΜΗΤΩΝ ΚΑΤΑΛΟΓΩΝ ΓΙΑ ΤΗ ΣΩΣΤΗ ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

- ΠΡΟΣΘΗΚΗ ΤΟΥ ΑΡΧΕΙΟΥ ΜΕ ΤΟΝ ΠΗΓΑΙΟ ΚΩΔΙΚΑ (WEBSERVER.C) ΣΤΟΝ ΑΝΤΙΣΤΟΙΧΟ ΚΑΤΑΛΟΓΟ

- ΕΚΤΕΛΕΣΗ ΤΟΥ COMPILER (gcc) ΜΕ ΕΞΟΔΟ ΤΟ ΕΠΙΘΥΜΗΤΟ ΑΠΟΤΕΛΕΣΜΑ

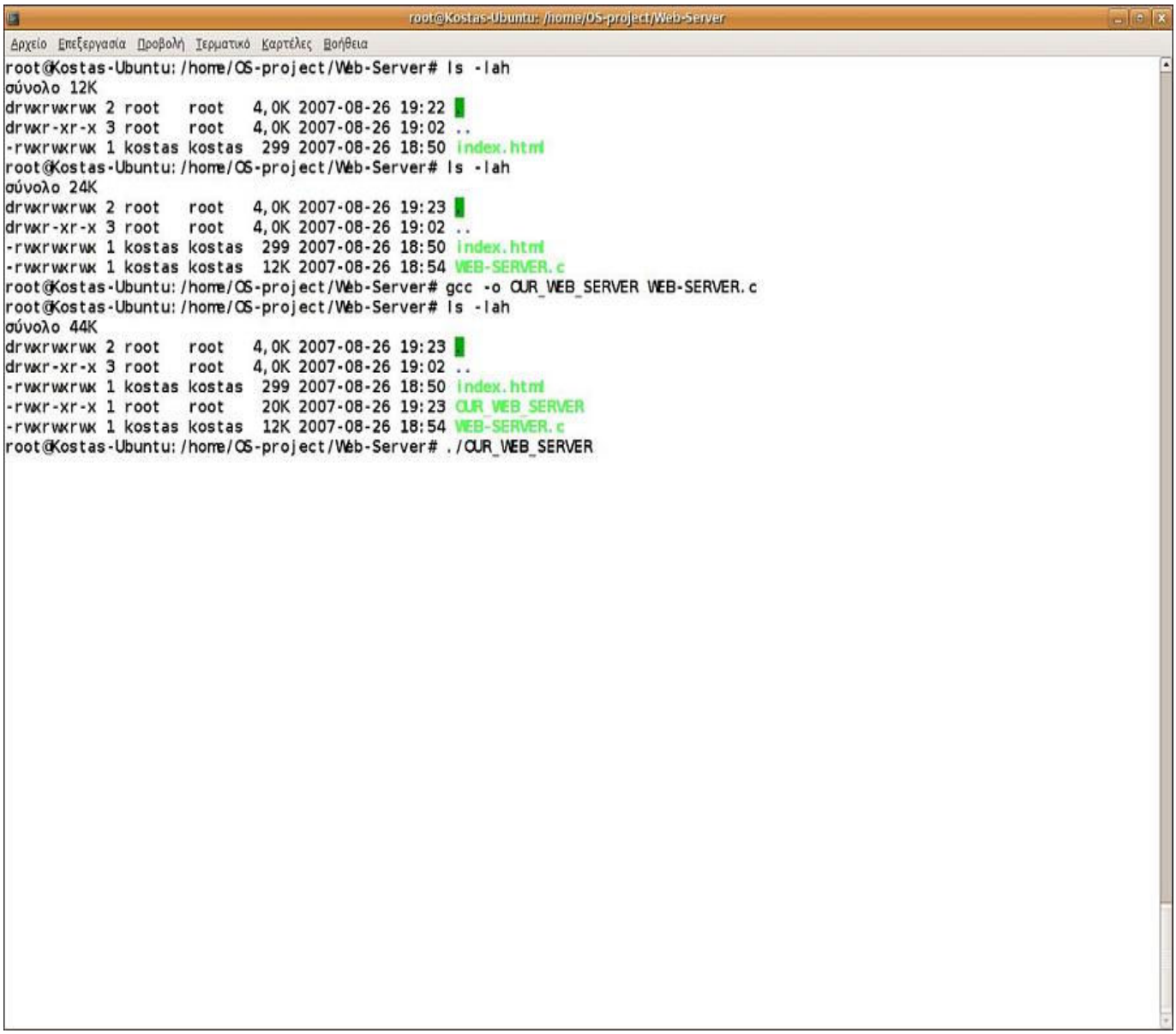
- ΕΚΤΕΛΕΣΗ ΤΟΥ ΕΚΤΕΛΕΣΙΜΟΥ ΑΡΧΕΙΟΥ OUR_WEB_SERVER

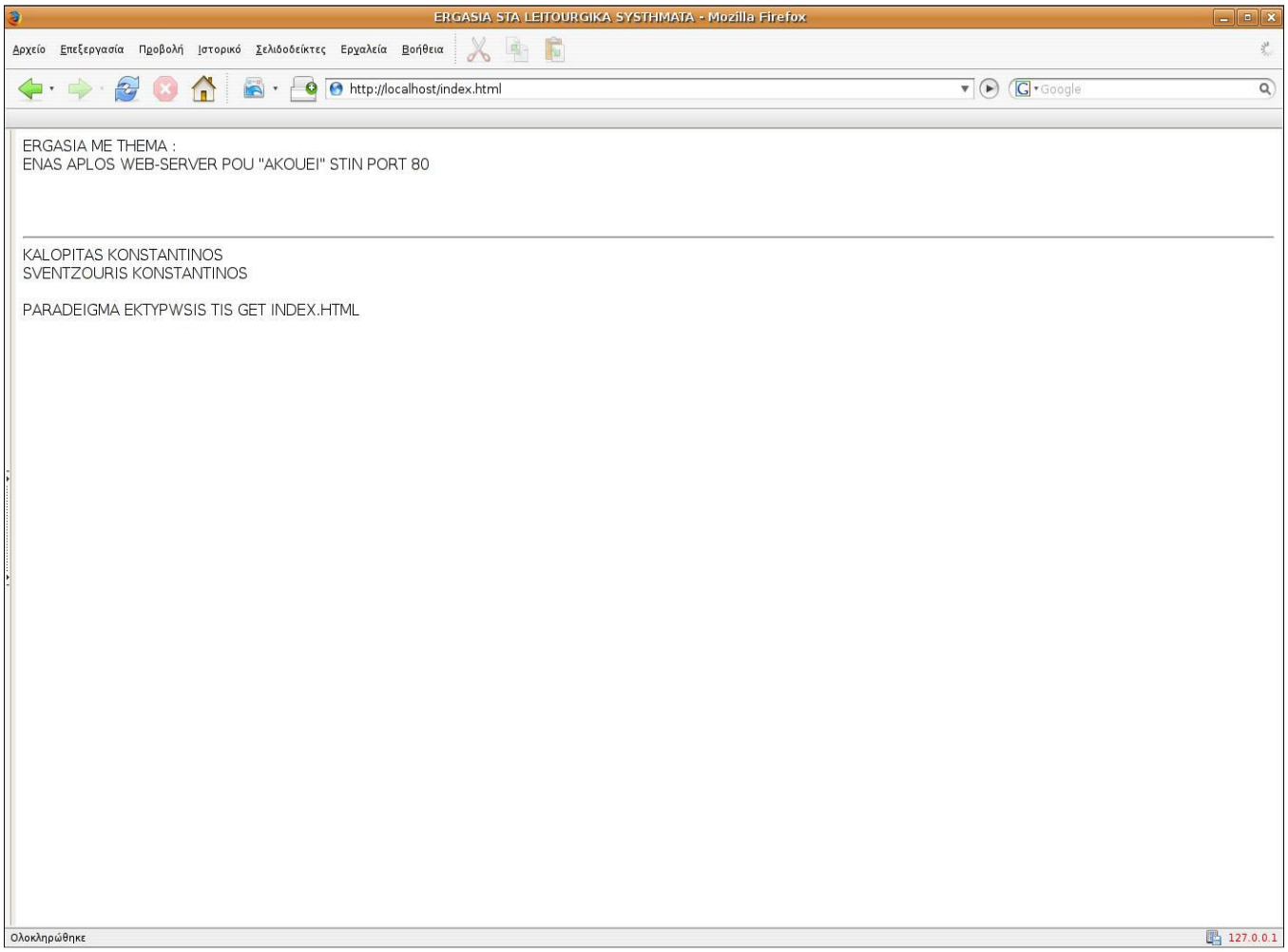
- ΕΠΙΒΕΒΑΙΩΣΗ ΛΕΙΤΟΥΡΓΙΑΣ ΤΟΥ SERVER ΜΕΣΩ ΤΟΥ BROWSER ΑΠΟ ΤΟ ΙΔΙΟ ΜΗΧΑΝΗΜΑ (localhost) ΚΑΙ ΕΠΙΣΗΣ ΣΕ ΠΡΑΓΜΑΤΙΚΕΣ ΣΥΝΘΗΚΕΣ ΔΙΚΤΥΟΥ (ΜΕΣΩ LAN - IP : 192.168.7.202:80) . Ο SERVER ΣΤΗΝ ΚΛΗΣΗ ΤΗΣ GET INDEX.HTML Η ΟΠΟΙΑ ΕΧΕΙ ΤΟΝ ΕΞΗΣ ΚΩΔΙΚΑ :

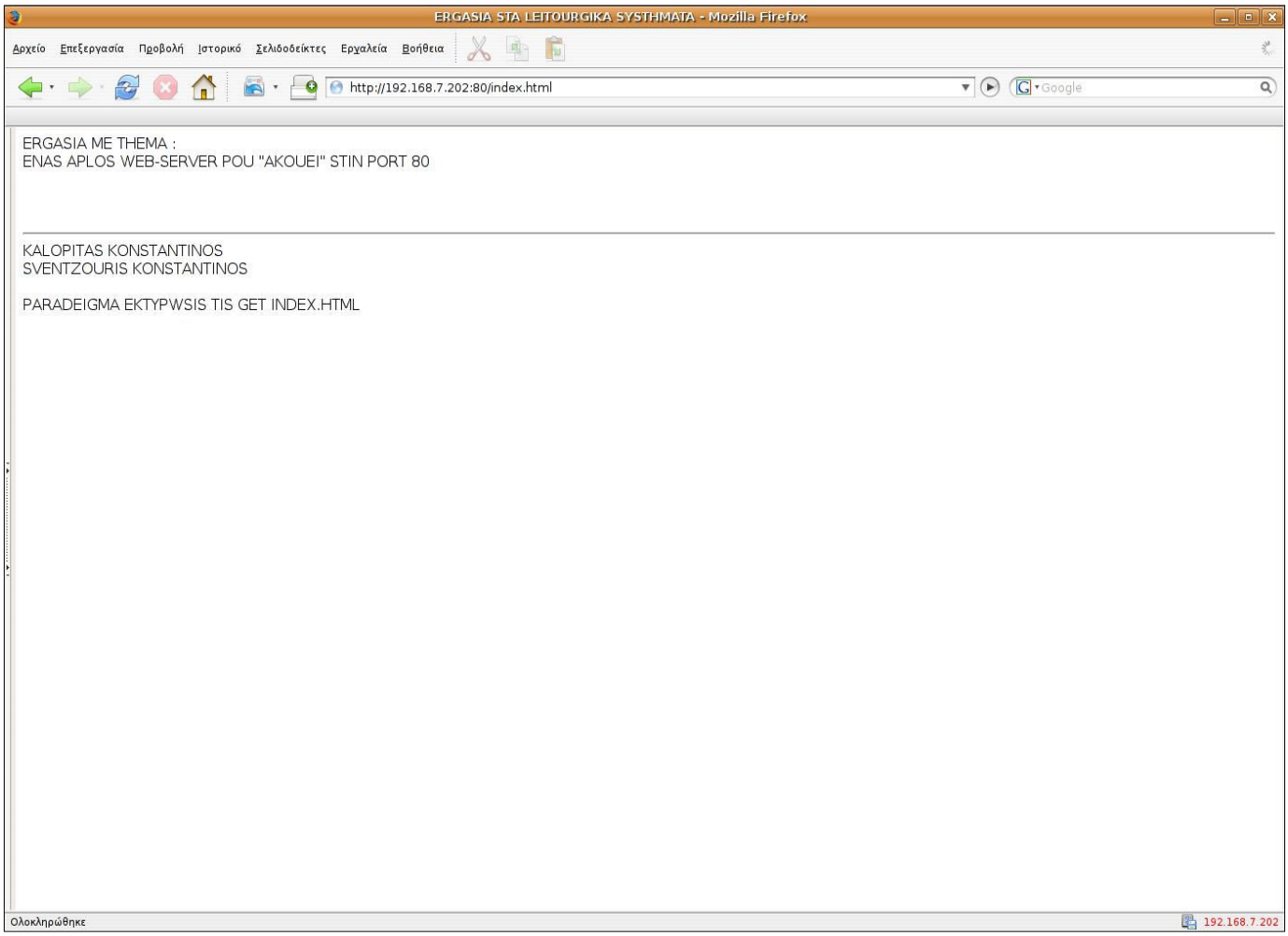
```
<HTML>
<TITLE>
ERGASIA STA LEITOURGIKA SYSTHMATA
</TITLE>
<BODY>
ERGASIA ME THEMA :<BR>
ENAS APLOS WEB-SERVER POU "AKOUEI" STIN PORT 80<BR>
<BR><BR>
<BR><HR>
KALOPITAS KONSTANTINOS<BR>
SVENTZOURIS KONSTANTINOS<BR>
<BR>
PARADEIGMA EKTYPWSIS  TIS GET INDEX.HTML<BR>
<BR>
</BODY>
</HTML>
```

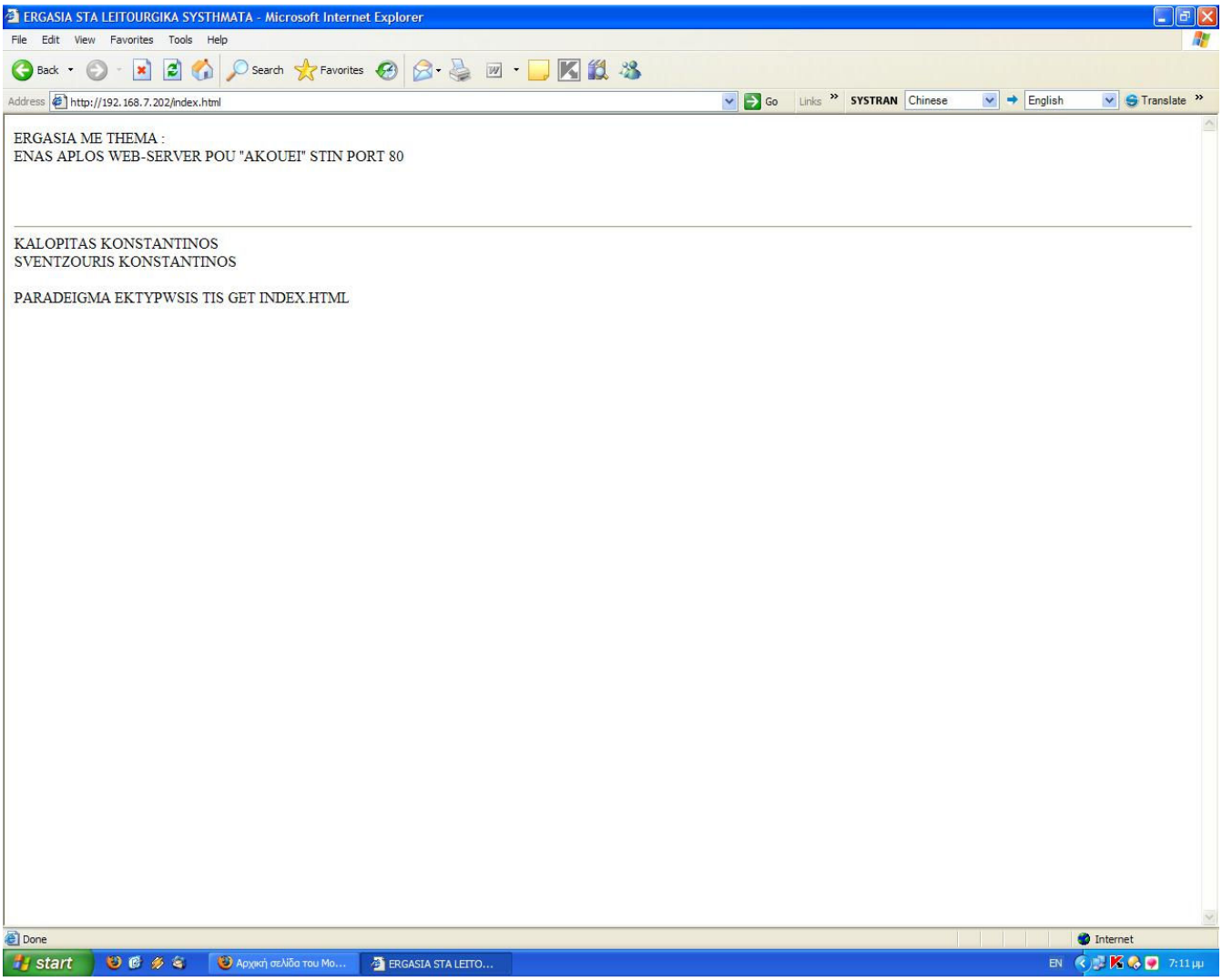
ΕΠΕΣΤΡΕΨΕ ΤΟ ΕΠΙΘΥΜΗΤΟ ΑΠΟΤΕΛΕΣΜΑ ΠΟΥ ΕΙΝΑΙ Η ΣΕΛΙΔΑ INDEX.HTML ΜΕΣΩ ΤΟΥ BROWSER .

ΑΚΟΛΟΥΘΟΥΝ ΤΑ ΑΝΑΛΟΓΑ ΣΤΙΓΜΙΟΤΥΠΑ ΟΘΟΝΗΣ :




ERGASIA STA LEITOURGIKA SYSTHMATA - Mozilla Firefox
Αρχείο Επεξεργασία Προβολή Ιστορικό Σελιδοδείκτες Εργαλεία Βοήθεια
http://localhost/index.html
Google
ERGASIA ME THEMA :
ENAS APLOS WEB-SERVER POU "AKOUEI" STIN PORT 80
KALOPITAS KONSTANTINOS
SVENTZOURIS KONSTANTINOS
PARADEIGMA EKTYPWSIS TIS GET INDEX.HTML
Ολοκληρώθηκε
127.0.0.1


ERGASIA STA LEITOURGIKA SYSTHMATA - Mozilla Firefox
Αρχείο Επεξεργασία Προβολή Ιστορικό Σελιδοδείκτες Εργαλεία Βοήθεια
http://192.168.7.202:80/index.html
Google
ERGASIA ME THEMA :
ENAS APLOS WEB-SERVER POU "AKOUEI" STIN PORT 80
KALOPITAS KONSTANTINOS
SVENTZOURIS KONSTANTINOS
PARADEIGMA EKTYPWSIS TIS GET INDEX.HTML
Ολοκληρώθηκε
192.168.7.202



ΒΙΒΛΙΟΓΡΑΦΙΑ

1. MODERN OPERATING SYSTEM – 2ND by ANDREW S. TANENBAUM
2. INTRODUCTION TO SOCKET PROGRAMMING USING C
3. INTERNET - GOOGLE