Aluno: Pedro Henrique da Silva Souza
RGA: 2015.1903.015-0
Data: 30/04/2017

Trabalho 02

1)
a) Para “localhost”: 13 chamadas
b) Para “resultadosdigitais.com.br”: 44 chamadas
c) Para “apache.org”: 64 chamadas
d) Para “jera.com.br”: 18 chamadas

2) Arquivo code-03.c

```
/* hw2-simple-client.c: program to connect to web server. */
/* compile with: gcc -Wall -o hw2-simple-client hw2-simple-client.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <netdb.h>

/* maximum size of a printed address -- if not defined, we define it here */
#ifndef INET6_ADDRSTRLEN
  #define INET6_ADDRSTRLEN 46
#endif /* INET6_ADDRSTRLEN */

#define BUFSIZE 1000

/* print a system error and exit the program */
static void error (char * s) {
  perror (s);
  exit (1);
}

static void usage (char * program) {
  printf ("usage: %s hostname [port]\n", program);
  exit (1);
}

static void removeProtocol (char *s) {
  char http[] = "http://";
  char https[] = "https://";
  if( strstr(s,http) ) {
    while( (s=strstr(s,http)) )
      memmove(s,s+strlen(http),1+strlen(s+strlen(http)));

  } else {
    while( (s=strstr(s,https)) )
      memmove(s,s+strlen(https),1+strlen(s+strlen(https)));
  }
}

static char * build_request (char * hostname) {
  char *full_hostname = hostname;
  char host[100];
  char path[100];
  char method [] = "GET ";
  char request [] = " HTTP/1.1\r\nHost: ";
  char sufix [] = "\r\n\r\n";

  removeProtocol(full_hostname);
  sscanf(full_hostname, "%99[^\n^/]/%99[^\n]", host, path);

  if ( (strlen(path) < 1) ) {
   strcpy(path, "/");
  }

  /* add 1 to the total length, so we have room for the null character --
  * the null character is never sent, but is needed to make this a C string */
  int total_length = strlen(host) + strlen(path) + strlen(method) + strlen(request) +
strlen(sufix) + 1;
  char * result = malloc (total_length);

  if (result == NULL) {
    return NULL;
  }

  snprintf (result, total_length, "%s%s%s%s%s", method, path, request, host, sufix);

  printf("%s", result);

  return result;
}

/* must be executed inline, so must be defined as a macro */
#define next_loop(a, s) { if (s >= 0) close (s); a = a->ai_next; continue; }
```

```
int main (int argc, char ** argv) {
  int sockfd;
  struct addrinfo * addrs;
  struct addrinfo hints;
  char * port = "80"; /* default is to connect to the http port, port 80 */

  if ((argc != 2) && (argc != 3)) {
    usage (argv [0]);
  }

  char * hostname = argv [1];

  if (argc == 3) {
    port = argv [2];
  }

  bzero (&hints, sizeof (hints));
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;

  if (getaddrinfo (hostname, port, &hints, &addrs) != 0) {
    error ("getaddrinfo");
  }

  struct addrinfo * original_addrs = addrs;

  while (addrs != NULL) {
    char buf [BUFSIZE];
    char prt [INET6_ADDRSTRLEN] = "unable to print";
    int af = addrs->ai_family;
    struct sockaddr_in * sinp = (struct sockaddr_in *) addrs->ai_addr;
    struct sockaddr_in6 * sin6p = (struct sockaddr_in6 *) addrs->ai_addr;

    if (af == AF_INET){
      inet_ntop (af, &(sinp->sin_addr), prt, sizeof (prt));
    } else if (af == AF_INET6) {
      inet_ntop (af, &(sin6p->sin6_addr), prt, sizeof (prt));
    } else {
      printf ("unable to print address of family %d\n", af);
      next_loop (addrs, -1);
    }

    if ((sockfd = socket (af, addrs->ai_socktype, addrs->ai_protocol)) < 0) {
      perror ("socket");
      next_loop (addrs, -1);
    }

    printf ("trying to connect to address %s, port %s\n", prt, port);

    if (connect (sockfd, addrs->ai_addr, addrs->ai_addrlen) != 0) {
      perror ("connect");
      next_loop (addrs, sockfd);
    }

    printf ("connected to %s\n", prt);
    char * request = build_request (hostname);

    if (request == NULL) {
      printf ("memory allocation (malloc) failed\n");
      next_loop (addrs, sockfd);
    }
    if (send (sockfd, request, strlen (request), 0) != strlen (request)) {
      perror ("send");
      next_loop (addrs, sockfd);
    }

    free (request); /* return the malloc'd memory */
                /* sometimes causes problems, and not needed
                shutdown (sockfd, SHUT_WR); */
    int count = 0;
    while (1) {
      /* use BUFSIZE - 1 to leave room for a null character */
      int rcvd = recv (sockfd, buf, BUFSIZE - 1, 0);
      count++;
      if (rcvd <= 0) {
        break;
      }
      buf [rcvd] = '\0';
      printf ("%s", buf);
    }
    printf ("data was received in %d recv calls\n", count);
    next_loop (addrs, sockfd);
  }

  freeaddrinfo (original_addrs);

  return 0;
```

```
}
```

3) accepted a connection from 127.0.0.1
```
accepted a connection from 127.0.0.1
```

Para contagem de requisições:

```
(...)
int server_socket = socket (AF_INET, SOCK_STREAM, 0);
int requests_count = 0;
(...)
requests_count++;
printf("------> Requests Count: %d\n", requests_count);
(...)
```

4) Arquivo code-04

```
/* hw2-simple-server.c: program to send a constant HTTP 404/501 response. */
/* compile with: gcc -Wall -o hw2-simple-server hw2-simple-server.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <arpa/inet.h>
/* maximum size of a printed address -- if not defined, we define it here */
#ifndef INET6_ADDRSTRLEN
  #define INET6_ADDRSTRLEN 46
#endif /* INET6_ADDRSTRLEN */

#define BUFSIZE 1000
/* print a system error and exit the program */
static void error (char * s) {
  perror (s);
  exit (1);
}

/* terminates the buffer with a null character and: */
/* returns 0 if the first line has not yet been read */
/* returns -1 if the first line fills (overflows) the buffer */
/* returns 1 if the first line has been read and starts with GET */
/* returns 2 if the first line has been read and starts with something else */
static int parse_request (char * buf, int len, int maxlen) {
  if (len >= maxlen) /* overflow */
    return -1;
  buf [len] = '\0'; /* terminate, i.e. make it into a C string */

  if (index (buf, '\n') == NULL) /* not finished reading the first line */
    return 0;
  if (strncmp (buf, "GET", 3) == 0)
    return 1;

  return 2;
}

// Count how many digits a number have
static int countDigits(int number) {
  int count = 0;
  while(number != 0) {
    number /= 10;
    ++count;
  }
  return count;
}

// Build Content Length Header
static char *getContentLengthHeader(int bytes) {
  char header[] = "Content-Length: ";
  char sufix[] = " bytes;\n\t";

  int total_length = strlen(header) + strlen(sufix) + countDigits(bytes) + 1;

  char * content_length = malloc (total_length);

  snprintf (content_length, total_length, "%s%d%s", header, bytes, sufix);

  return content_length;
}

/* must be executed inline, so must be defined as a macro */
#define next_loop(s) { if (s >= 0) close (s); continue; }

int main (int argc, char ** argv) {
  int port = 9000; /* can be used without root privileges */
  int server_socket = socket (AF_INET, SOCK_STREAM, 0);
  int requests_count = 0;

  if (server_socket < 0)
    error ("socket");
```

```
struct sockaddr_in sin;
sin.sin_family = AF_INET;
sin.sin_port = htons (port);
sin.sin_addr.s_addr = INADDR_ANY;
struct sockaddr * sap = (struct sockaddr *) (&sin);

printf ("starting HTTP server on port %d\n", port);

if (bind (server_socket, sap, sizeof (sin)) != 0)
  error ("bind");

listen (server_socket, 10);

// Reading file content and storing into 'index_page' variable
FILE *file;
int character;
file = fopen("index.html", "r");

fseek(file, 0L, SEEK_END);
int FILE_LENGTH = ftell(file);
fseek(file, 0L, SEEK_SET);

char * index_page = malloc (FILE_LENGTH);

if (file) {
  int i = 0;
  while ((character = getc(file)) != EOF) {
    index_page[i] = character;
    i++;
  }
  fclose(file);
}

while (1) { /* infinite server loop */
  struct sockaddr_storage from;
  struct sockaddr_in * from_sinp = (struct sockaddr_in *) (&from);
  struct sockaddr * from_sap = (struct sockaddr *) (&from);

  socklen_t addrlen = sizeof (from);

  int sockfd = accept (server_socket, from_sap, &addrlen);

  if (sockfd < 0)
  error ("accept");

if (from_sap->sa_family != AF_INET) {
  printf ("accepted connection in address family %d, only %d supported\n",
  from_sap->sa_family, AF_INET);
  next_loop (sockfd);
}

char prt [INET6_ADDRSTRLEN] = "unable to print";

inet_ntop (AF_INET, &(from_sinp->sin_addr), prt, sizeof (prt));

printf ("accepted a connection from %s\n", prt);

char buf [BUFSIZE];
int received = 0;

while (parse_request (buf, received, sizeof (buf)) == 0) {
  int r = recv (sockfd, buf + received, sizeof (buf) - received, 0);

  if (r <= 0) {
    printf ("received %d\n", r);
    next_loop (sockfd);
  }

  received += r;
}

int parse = parse_request (buf, received, sizeof (buf));
if (parse == -1) { /* first line longer than buffer */
  printf ("error: first line longer than %ld\n", sizeof (buf));
  next_loop (sockfd);
}

char * code = "HTTP/1.0 404 Not Found\r\n";
if (parse == 2)
  code = "HTTP/1.0 501 Not implemented\r\n";

send (sockfd, code, strlen (code), 0);

char date_buf [BUFSIZE];
time_t now = time (NULL);
char * time_str = asctime (gmtime (&now));
```

```
    snprintf (date_buf, sizeof (date_buf), "Date: %s\n", time_str);

    /* time_str ends with \n. We replace it with \r to give \r\n */
    * (index (date_buf, '\n')) = '\r';

    send (sockfd, date_buf, strlen (date_buf), 0);

    char server_id [] = "Server: dummy HTTP server\r\n";

    // Send HEADER: Connection
    send (
      sockfd,
      "Connection: close, Server: code-04;\n\t",
      strlen ("Connection: close, Server: code-04;\n\t"),
      0
    );

    // Send HEADER: Content-Type
    send (
      sockfd,
      "Content-Type: text/html;\n\t",
      strlen ("Content-Type: text/html;\n\t"),
      0
    );

    // Send HEADER: Content-Length
    char * content_length_header = getContentLengthHeader (FILE_LENGTH);
    send (sockfd, content_length_header, strlen (content_length_header), 0);

    send (sockfd, server_id, strlen (server_id), 0);
    send (sockfd, "\r\n", 2, 0);

    // Sending page content
    write(sockfd, index_page, strlen(index_page));

    requests_count++;
    printf("------> Requests Count: %d\n", requests_count);

    shutdown (sockfd, SHUT_WR); /* not useful, since we close right away */

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