Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования "Новгородский государственный университет имени Ярослава Мудрого"

Кафедра «Информационных технологий и систем»

Дисциплина «Операционные системы»

Отчет по лабораторной работе

«Клиент-сервер. Создание Web-сервера»

Выполнил студент группы 9091

\_\_\_\_\_\_\_\_\_\_\_\_\_/Федотов Дмитрий Борисович/

Подпись ФИО

Принял преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_/Ананьев Владислав Валерьевич/

Подпись ФИО

Великий Новгород

2021

**Цель лабораторной работы**

Цель работы: научиться писать программу-клиент и прогграмму сервер.

**Исходный текст программы**

|  |
| --- |
| client.c |
| #include "stdio.h"  #include "errno.h"  #include "unistd.h"  #include "stdlib.h"  #include "string.h"  #include "strings.h"  #include "arpa/inet.h"  #include "sys/types.h"  #include "sys/socket.h"  #include "netinet/in.h"  #define MAX\_STRING\_LENGTH 512  typedef struct file\_receive  {  char file\_name[256];  size\_t file\_size;  } file\_receive\_struct;  char\* parse\_substring\_from(char\* src\_string, const char match\_char)  {  char\* first\_match = strchr(src\_string, match\_char);  size\_t bytes\_for\_malloc = first\_match - src\_string;  char\* str\_mem = malloc(bytes\_for\_malloc + 1);  memcpy(str\_mem, src\_string, bytes\_for\_malloc);  str\_mem[bytes\_for\_malloc] = '\0';  return str\_mem;  }  char\* parse\_substring\_from\_to(char\* src\_string, const char match\_char\_from, const char match\_char\_to)  {  char\* first\_match\_from = strchr(src\_string, match\_char\_from) + 1;  char\* first\_match\_to = strchr(src\_string, match\_char\_to);  size\_t bytes\_for\_malloc = first\_match\_to - first\_match\_from;  char\* str\_mem = malloc(bytes\_for\_malloc + 1);  memcpy(str\_mem, first\_match\_from, bytes\_for\_malloc);  str\_mem[bytes\_for\_malloc] = '\0';  return str\_mem;  }  void send\_client\_data\_to\_server(const u\_int16\_t tcp\_socket\_fd, const char\* client\_data)  {  uint32\_t received\_bytes = 0;  received\_bytes = write(tcp\_socket\_fd, client\_data, MAX\_STRING\_LENGTH);  if (received\_bytes < 0)  {  perror(NULL);  close(tcp\_socket\_fd);  exit(0);  }  }  int main(int argv, char\* argc[])  {  if (argv <= 1)  {  printf("Error! Not enough arguments! Required (1): <ip-address>:<port>\n");  return -1;  }  uint64\_t received\_bytes = 0;  u\_int16\_t tcp\_socket\_fd = 0;  char client\_send\_string[MAX\_STRING\_LENGTH];  char server\_recv\_string[MAX\_STRING\_LENGTH];  char\* tcp\_server\_ip = parse\_substring\_from(argc[1], ':');  char\* tcp\_server\_port = parse\_substring\_from\_to(argc[1], ':', '\0');  struct sockaddr\_in server\_address;  struct sockaddr\_in client\_address;  bzero(client\_send\_string, sizeof(client\_send\_string));  bzero(server\_recv\_string, sizeof(server\_recv\_string));  bzero(&server\_address, sizeof(server\_address));  bzero(&client\_address, sizeof(client\_address));  if ((tcp\_socket\_fd = socket(PF\_INET, SOCK\_STREAM, 0)) < 0)  {  perror(NULL);  return -1;  }  server\_address.sin\_family = AF\_INET;  server\_address.sin\_port = htons(atoi(tcp\_server\_port));  if (inet\_aton(tcp\_server\_ip, &server\_address.sin\_addr) == 0)  {  perror(NULL);  close(tcp\_socket\_fd);  return -1;  }  if (connect(tcp\_socket\_fd, (struct sockaddr\*) &server\_address, sizeof(server\_address)) < 0)  {  perror(NULL);  close(tcp\_socket\_fd);  return -1;  }  while (1)  {  printf("[COMMAND]: ");    fgets(client\_send\_string, MAX\_STRING\_LENGTH - 1, stdin);  if (strncmp(client\_send\_string, "get", 3) == 0)  {  send\_client\_data\_to\_server(tcp\_socket\_fd, client\_send\_string);  unsigned short count\_files\_to\_receive = 0;  received\_bytes = read(tcp\_socket\_fd, &count\_files\_to\_receive, sizeof(count\_files\_to\_receive));  if (received\_bytes < 0)  {  perror(NULL);  close(tcp\_socket\_fd);  return -1;  }  count\_files\_to\_receive--;  printf("Receiving %u files from server...\n", count\_files\_to\_receive);  while (count\_files\_to\_receive)  {  file\_receive\_struct file\_to\_receive;  received\_bytes = read(tcp\_socket\_fd, &file\_to\_receive, sizeof(file\_to\_receive));  if (received\_bytes < 0)  {  perror(NULL);  close(tcp\_socket\_fd);  return -1;  }  if (!file\_to\_receive.file\_size)  {  printf("%s is not exist in the server\n", file\_to\_receive.file\_name);  count\_files\_to\_receive--;  continue;  }    FILE\* file = fopen(file\_to\_receive.file\_name, "w");  char file\_buffer[file\_to\_receive.file\_size];  received\_bytes = read(tcp\_socket\_fd, &file\_buffer, file\_to\_receive.file\_size);  if (received\_bytes < 0)  {  perror(NULL);  close(tcp\_socket\_fd);  return -1;  }  fprintf(file, "%s", file\_buffer);  fclose(file);  printf("%s received %li\\%li bytes ::: DONE\n", file\_to\_receive.file\_name, received\_bytes, file\_to\_receive.file\_size);  count\_files\_to\_receive--;  }  }  else if (strncmp(client\_send\_string, "exit", 4) == 0)  {  printf("Shutting down...\n");  close(tcp\_socket\_fd);  break;  }  else if (strncmp(client\_send\_string, "date", 4) == 0)  {  send\_client\_data\_to\_server(tcp\_socket\_fd, client\_send\_string);  received\_bytes = read(tcp\_socket\_fd, server\_recv\_string, MAX\_STRING\_LENGTH);  if (received\_bytes < 0)  {  perror(NULL);  close(tcp\_socket\_fd);  return -1;  }    printf("%s", server\_recv\_string);  }  else if (strncmp(client\_send\_string, "files", 5) == 0)  {  send\_client\_data\_to\_server(tcp\_socket\_fd, client\_send\_string);  received\_bytes = read(tcp\_socket\_fd, server\_recv\_string, MAX\_STRING\_LENGTH);  if (received\_bytes < 0)  {  perror(NULL);  close(tcp\_socket\_fd);  return -1;  }  printf("%s", server\_recv\_string);  }  else if (strncmp(client\_send\_string, "clear", 5) == 0)  system("clear");  else  printf("Unknown command!\n");  }  free(tcp\_server\_ip);  free(tcp\_server\_port);  return 0;  } |

|  |
| --- |
| server.c |
| #include "time.h"  #include "stdio.h"  #include "errno.h"  #include "unistd.h"  #include "stdlib.h"  #include "string.h"  #include "dirent.h"  #include "strings.h"  #include "sys/stat.h"  #include "arpa/inet.h"  #include "sys/types.h"  #include "sys/socket.h"  #include "netinet/in.h"  #define MAX\_STRING\_LENGTH 512  #define SERVER\_FOLDER "server\_folder/"  typedef struct file\_receive  {  char file\_name[256];  size\_t file\_size;  } file\_receive\_struct;  void client\_service(const u\_int16\_t tcp\_socket\_fd, const struct sockaddr\_in\* server\_address)  {  uint32\_t received\_bytes = 0;  char client\_command[MAX\_STRING\_LENGTH];  while (1)  {  // There process working with current client  received\_bytes = read(tcp\_socket\_fd, client\_command, MAX\_STRING\_LENGTH - 1);    if (received\_bytes < 0)  {  perror(NULL);  break;  }  if (received\_bytes == 0)  {  printf("[SERVER][INFO]: One of clients break the connection with server\n");  break;  }    if (strncmp(client\_command, "get", 3) == 0)  {  unsigned short files\_count = 0;  char\* token = strtok(client\_command, " ");  char\* file\_names\_array[128];    // Parsing file names from client command  while (token)  {  unsigned short file\_name\_length = strlen(token) + 1;  file\_names\_array[files\_count] = malloc(file\_name\_length);  strcpy(file\_names\_array[files\_count], token);  file\_names\_array[files\_count][file\_name\_length - 1] = '\0';  files\_count++;  token = strtok(NULL, " ");  }  unsigned last\_char = strlen(file\_names\_array[files\_count - 1]) - 1;  file\_names\_array[files\_count - 1][last\_char] = '\0';  received\_bytes = write(tcp\_socket\_fd, &files\_count, sizeof(files\_count));  if (received\_bytes < 0)  {  printf("[SERVER][ERROR]: Cant send information to one of clients\n");  break;  }  for (int i = 1; i < files\_count; i++)  {  char file\_path[MAX\_STRING\_LENGTH] = SERVER\_FOLDER;  strcat(file\_path, file\_names\_array[i]);    FILE\* file = fopen(file\_path, "r");  file\_receive\_struct file\_to\_send;  // Get a file name  strcpy(file\_to\_send.file\_name, file\_names\_array[i]);  if (!file)  {  file\_to\_send.file\_size = 0;  received\_bytes = write(tcp\_socket\_fd, &file\_to\_send, sizeof(file\_to\_send));  if (received\_bytes < 0)  {  printf("[SERVER][ERROR]: Cant send information to one of clients\n");  break;  }    continue;  }  // Get a file size  fseek(file, 0L, SEEK\_END);  file\_to\_send.file\_size = ftell(file);  fseek(file, 0L, SEEK\_SET);  // Send to client the file information  received\_bytes = write(tcp\_socket\_fd, &file\_to\_send, sizeof(file\_to\_send));  if (received\_bytes < 0)  {  printf("[SERVER][ERROR]: Cant send information to one of clients\n");  break;  }  char file\_buffer[file\_to\_send.file\_size];  fread(file\_buffer, sizeof(char), file\_to\_send.file\_size, file);  fclose(file);  received\_bytes = write(tcp\_socket\_fd, file\_buffer, file\_to\_send.file\_size);  if (received\_bytes < 0)  {  printf("[SERVER][ERROR]: Cant send information to one of clients\n");  break;  }  }  for (int i = 0; i < files\_count; i++)  free(file\_names\_array[i]);  }  else if (strncmp(client\_command, "date", 4) == 0)  {  time\_t server\_time;  struct tm\* time\_info;  time(&server\_time);  time\_info = localtime(&server\_time);  char\* asc\_time = asctime(time\_info);  received\_bytes = write(tcp\_socket\_fd, asc\_time, strlen(asc\_time) + 1);  if (received\_bytes < 0)  {  printf("[SERVER][ERROR]: Cant send information to one of clients\n");  break;  }  }  else if (strncmp(client\_command, "files", 4) == 0)  {  char server\_answer[MAX\_STRING\_LENGTH];  memset(server\_answer, 0, sizeof(server\_answer));  server\_answer[MAX\_STRING\_LENGTH - 1] = '\0';  DIR\* directory = opendir(SERVER\_FOLDER);    if (!directory)  {  strcat(server\_answer, "Server cant open directory\n");  received\_bytes = write(tcp\_socket\_fd, server\_answer, MAX\_STRING\_LENGTH);  if (received\_bytes < 0)  {  printf("[SERVER][ERROR]: Cant send information to one of clients\n");  break;  }  continue;  }  struct dirent\* file;  struct stat file\_stat;  while((file = readdir(directory)) != NULL)  {  if (strncmp(file->d\_name, "..", 2) == 0 || strncmp(file->d\_name, ".", 1) == 0)  continue;    strcat(server\_answer, file->d\_name);  strcat(server\_answer, "\n");  }  closedir(directory);  received\_bytes = write(tcp\_socket\_fd, server\_answer, MAX\_STRING\_LENGTH);  if (received\_bytes < 0)  {  printf("[SERVER][ERROR]: Cant send information to one of clients\n");  break;  }  }  }  close(tcp\_socket\_fd);  exit(0);  }  void print\_server\_port(const u\_int16\_t tcp\_socket\_fd)  {  struct sockaddr\_in socket\_address;  socklen\_t socket\_len = sizeof(socket\_address);  getsockname(tcp\_socket\_fd, (struct sockaddr\*) &socket\_address, &socket\_len);  printf("Server started in %i port\n", ntohs(socket\_address.sin\_port));  }  int main(int argv, char\* argc[])  {  if (argv <= 1)  {  printf("Error! Not enough arguments! Required (1): port\n");  return -1;  }  uint32\_t client\_length = 0;  u\_int16\_t tcp\_socket\_fd;  u\_int16\_t tcp\_newsocket\_fd;  struct sockaddr\_in server\_address;  struct sockaddr\_in client\_address;  bzero(&server\_address, sizeof(server\_address));  bzero(&client\_address, sizeof(client\_address));  // Try to create TCP socket  if ((tcp\_socket\_fd = socket(PF\_INET, SOCK\_STREAM, 0)) < 0)  {  perror(NULL);  return -1;  }  server\_address.sin\_family = AF\_INET;  server\_address.sin\_port = htons(atoi(argc[1]));  server\_address.sin\_addr.s\_addr = htonl(INADDR\_ANY);  // Try to set up TCP socket  if (bind(tcp\_socket\_fd, (struct sockaddr\*) &server\_address, sizeof(server\_address)) < 0)  {  printf("Error! Cant get %s port! Trying to get another!\n", argc[1]);  server\_address.sin\_port = htons(0);  if (bind(tcp\_socket\_fd, (struct sockaddr\*) &server\_address, sizeof(server\_address)) < 0)  {  perror(NULL);  close(tcp\_socket\_fd);  return -1;  }  }  print\_server\_port(tcp\_socket\_fd);  if (listen(tcp\_socket\_fd, 5) < 0)  {  perror(NULL);  close(tcp\_socket\_fd);  return -1;  }  pid\_t child\_process\_server = fork();  if (child\_process\_server == -1)  {  printf("[ROOT][CRITICAL]: Server cannot be started\n");  return -1;  }  else if (child\_process\_server == 0)  {  while (1)  {  client\_length = sizeof(client\_address);  tcp\_newsocket\_fd = accept(tcp\_socket\_fd, (struct sockaddr\*) &client\_address, &client\_length);  if (tcp\_newsocket\_fd < 0)  {  perror(NULL);  close(tcp\_socket\_fd);  return -1;  }  pid\_t child\_process\_pid = fork();  if (child\_process\_pid == -1)  printf("[SERVER][ERROR]: Cant fork new process!\n");  else if (child\_process\_pid == 0)  client\_service(tcp\_newsocket\_fd, &server\_address);  else  printf("[SERVER][INFO]: New process has been started. Continue listening...\n");  }  }  else  {  printf("[ROOT][INFO]: Server has been started\n");  while (1)  {  // This is root process. It processes commands for the server  char server\_command[512];  fgets(server\_command, MAX\_STRING\_LENGTH - 1, stdin);  if (strncmp(server\_command, "exit", 4) == 0)  {  printf("[SERVER][INFO]: Server shutdown...\n");  system("killall ./server");  return 0;  }  else if (strncmp(server\_command, "date", 4) == 0)  system("date");  else if (strncmp(server\_command, "files", 5) == 0)  system("ls -l server\_folder/");  else if (strncmp(server\_command, "clear", 5) == 0)  system("clear");  else  printf("[SERVER][COMMAND]: Unknown command!\n");  }  }  return 0;  } |

**Результат выполнения программ**

|  |
| --- |
| Вывод в терминале клиента |
| >> ./client 0.0.0.0:8000  [COMMAND]: files another\_file.txt text.с [COMMAND]: date Thu May 13 23:25:29 2021 [COMMAND]: get text.c another\_text.txt Receiving 2 files from server... text.c received 77\77 bytes ::: DONE another\_text.txt is not exist in the server [COMMAND]: get text.c another\_file.txt Receiving 2 files from server... text.c received 77\77 bytes ::: DONE another\_file.txt received 22\22 bytes ::: DONE [COMMAND]: date Thu May 13 23:26:08 2021 [COMMAND]: exit Shutting down... |

|  |
| --- |
| Вывод в терминале сервера |
| >> ./server 8000  Server started in 8000 port [ROOT][INFO]: Server has been started [SERVER][INFO]: New process has been started. Continue listening... date Thu May 13 11:25:21 PM MSK 2021  files  total 8  -rw-r—r--- 1 dmitriy users 22 May 13 23:24 another\_file.txt  -rw-r—r--- 1 dmitriy users 77 May 13 23:24 text.c [SERVER][INFO]: One of clients break the connection with server exit [SERVER][INFO]: Server shutdown... Terminated |

|  |
| --- |
| Содержимое text.c у клиента |
| #include "stdio.h"  int main(void) { printf("Hello, World\n"); return 0; } |

|  |
| --- |
| Содержмиое another\_file.txt у клиента |
| This is another text |

**Вывод**

Вывод: выполняя лабораторную работу, я научился писать программу-клиент и прогграмму сервер.