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

«Семейство протоколов TCP/IP. Сокеты в UNIX и работа с ними»

Выполнил студент группы 9091

\_\_\_\_\_\_\_\_\_\_\_\_\_/Лопатин Даниил Михайлович/

Подпись ФИО

Принял преподаватель

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

Подпись ФИО

Великий Новгород

2021

**Цель лабораторной работы**

Цель работы: познакомиться с семейством протоколов TCP/IP, сокетами в UNIX и научиться с ними работать.

**Исходный текст программы**

|  |
| --- |
| client.c |
| #include "time.h"  #include "stdio.h"  #include "errno.h"  #include "unistd.h"  #include "stdlib.h"  #include "stdint.h"  #include "string.h"  #include "strings.h"  #include "arpa/inet.h"  #include "sys/types.h"  #include "netinet/in.h"  #include "sys/socket.h"  #include <time.h>  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 array\_fill\_random\_value(int32\_t\* array, uint16\_t size, int32\_t min, int32\_t max)  {  for (size\_t i = 0; i < size; i++)  array[i] = min + rand() % max;  }  void print\_array\_values(int32\_t\* array, uint16\_t size)  {  for (size\_t i = 0; i < size; i++)  printf("%i ", array[i]);  printf("\n");  }  int main(int argv, char\* argc[])  {  if (argv <= 4)  {  printf("Error! Not enough arguments! Required (4): <ip-address>:<port>, array size, array min, array max\n");  return -1;  }  srand(time(NULL));  u\_int16\_t udp\_socket\_fd = 0;  char\* udp\_server\_ip = parse\_substring\_from(argc[1], ':');  char\* udp\_server\_port = parse\_substring\_from\_to(argc[1], ':', '\0');  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 UDP socket  if ((udp\_socket\_fd = socket(PF\_INET, SOCK\_DGRAM, 0)) < 0)  {  perror(NULL);  return -1;  }  client\_address.sin\_family = AF\_INET;  client\_address.sin\_port = htons(0);  client\_address.sin\_addr.s\_addr = htons(INADDR\_ANY);  if (bind(udp\_socket\_fd, (struct sockaddr\*) &client\_address, sizeof(client\_address)) < 0)  {  perror(NULL);  close(udp\_socket\_fd);  return -1;  }  printf("Try to connect to server with address: %s:%s\n", udp\_server\_ip, udp\_server\_port);  server\_address.sin\_family = AF\_INET;  server\_address.sin\_port = htons(atoi(udp\_server\_port));    if (inet\_aton(udp\_server\_ip, &server\_address.sin\_addr) == 0)  {  printf("Error! Invalid IP address!\n");  close(udp\_socket\_fd);  return -1;  }  unsigned short client\_array\_size = atoi(argc[2]);  int\* client\_array = malloc(client\_array\_size \* sizeof(int));  array\_fill\_random\_value(client\_array, client\_array\_size, atoi(argc[3]), atoi(argc[4]));  printf("==== Generate next array ====\n");  print\_array\_values(client\_array, client\_array\_size);  printf("=============================\n");  // Send client array size to server  if (sendto(udp\_socket\_fd, &client\_array\_size, sizeof(client\_array\_size), 0, (struct sockaddr\*) &server\_address, sizeof(server\_address)) < 0)  {  perror(NULL);  close(udp\_socket\_fd);  return -1;  }  // Send client array to server  clock\_t start\_time = clock();  if (sendto(udp\_socket\_fd, client\_array, sizeof(int) \* client\_array\_size, 0, (struct sockaddr\*) &server\_address, sizeof(server\_address)) < 0)  {  perror(NULL);  close(udp\_socket\_fd);  return -1;  }  // Receive sorted array from server  int received\_bytes\_from\_server = 0;  received\_bytes\_from\_server = recvfrom(udp\_socket\_fd, client\_array, sizeof(int) \* client\_array\_size, 0, (struct sockaddr\*) NULL, NULL);  clock\_t end\_time = clock();  if (received\_bytes\_from\_server < 0)  {  perror(NULL);  close(udp\_socket\_fd);  return -1;  }  printf("==== Response from server ====\n");  print\_array\_values(client\_array, client\_array\_size);  printf("Time required: %f\n", (end\_time - start\_time) / (double) CLOCKS\_PER\_SEC);  printf("==============================\n");  close(udp\_socket\_fd);  free(client\_array);  free(udp\_server\_ip);  free(udp\_server\_port);  return 0;  } |

|  |
| --- |
| server.c |
| #include "stdio.h"  #include "errno.h"  #include "unistd.h"  #include "stdlib.h"  #include "stdint.h"  #include "strings.h"  #include "arpa/inet.h"  #include "sys/types.h"  #include "netinet/in.h"  #include "sys/socket.h"  int compare\_int\_value(const void\* a, const void\* b)  {  return \*((int\*) b) - \*((int\*) a);  }  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 udp\_socket\_fd = 0;  u\_int16\_t host\_port = atoi(argc[1]);  struct sockaddr\_in server\_address;  struct sockaddr\_in client\_address;  bzero(&server\_address, sizeof(server\_address));  bzero(&client\_address, sizeof(client\_address));  server\_address.sin\_family = AF\_INET;  server\_address.sin\_port = htons(host\_port);  server\_address.sin\_addr.s\_addr = htonl(INADDR\_ANY);    // Try to create UDP socket  if ((udp\_socket\_fd = socket(PF\_INET, SOCK\_DGRAM, 0)) < 0)  {  perror(NULL);  return -1;  }  // Try to set up UDP socket  if (bind(udp\_socket\_fd, (struct sockaddr\*) &server\_address, sizeof(server\_address)) < 0)  {  printf("Error! Cant get %i port! Trying to get another!\n", host\_port);  server\_address.sin\_port = htons(0);  if (bind(udp\_socket\_fd, (struct sockaddr\*) &server\_address, sizeof(server\_address)) < 0)  {  perror(NULL);  close(udp\_socket\_fd);  return -1;  }  }  print\_server\_port(udp\_socket\_fd);  while (1)  {  int client\_received\_bytes = 0;    client\_length = sizeof(client\_length);  // Wait to get array size from client  unsigned short client\_array\_size = 0;  client\_received\_bytes = recvfrom(udp\_socket\_fd, &client\_array\_size, sizeof(client\_array\_size), 0, (struct sockaddr\*) &client\_address, &client\_length);    if (client\_received\_bytes < 0)  {  perror(NULL);  close(udp\_socket\_fd);  return -1;  }  printf("============ Incoming form ============\n");  printf("Received next array size: %i\n", client\_array\_size);  printf("Wait to receive array...\n");  // Wait to get array with "client\_array\_size" size  int\* client\_array = malloc(client\_array\_size \* sizeof(int));  client\_received\_bytes = recvfrom(udp\_socket\_fd, client\_array, sizeof(int) \* client\_array\_size, 0, (struct sockaddr\*) &client\_address, &client\_length);    if (client\_received\_bytes < 0)  {  perror(NULL);  close(udp\_socket\_fd);  return -1;  }  printf("Client array is received\n");  printf("Sorting...\n");  // Sort received array and send it back to client  qsort(client\_array, client\_array\_size, sizeof(int), compare\_int\_value);    printf("Sending back...\n");  if (sendto(udp\_socket\_fd, client\_array, sizeof(int) \* client\_array\_size, 0, (struct sockaddr\*) &client\_address, client\_length) < 0)  {  perror(NULL);  close(udp\_socket\_fd);  return -1;  }  free(client\_array);  printf("========================================\n");  }  return 0;  } |

**Результат выполнения программ**

|  |
| --- |
| Вывод в терминале клиента |
| >> ./client 0.0.0.0:8000 15 -50 100  Try to connect to server with address: 0.0.0.0:8000 ==== Generate next array ==== 6 33 -28 -16 19 12 14 23 -20 34 -15 18 0 41 46  ============================= ==== Response from server ==== 46 41 34 33 23 19 18 14 12 6 0 -15 -16 -20 -28  Time required: 0.000014 ============================== |

|  |
| --- |
| Вывод в терминале сервера |
| >> ./server 8000  Server started in 8000 port ============ Incoming form ============ Received next array size: 15 Wait to receive array... Client array is received Sorting... Sending back... ======================================== |

**Вывод**

Вывод: выполняя лабораторную работу, я познакомился с семейством протоколов TCP/IP, сокетами в UNIX и научился с ними работать.