|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**ОТЧЕТ**

*к лабораторной работе №6*

*По курсу: «Операционные системы»*

Студент ИУ7-66Б

Дедич Б.Б

Преподаватель

Рязанова Н.Ю.

*Москва, 2021 г.*

1. СОКЕТ В ФАЙЛОВОМ ПРОСТРАНСТВЕ

В процессе-сервере с помощью вызова socket() создается сокет семейства AF\_UNIX с типом SOCK\_DGRAM. С помощью системного вызова bind() происходит связка сокета с локальным адресом. Сервер блокируется на функции recv() и ждет сообщения от процессов клиентов.

В процессе-клиенте создается сокет семейства AF\_UNIX с типом SOCK\_DGRAM с помощью системного вызова socket(). С помощью функции sendto() отправляется сообщение процессу-серверу.

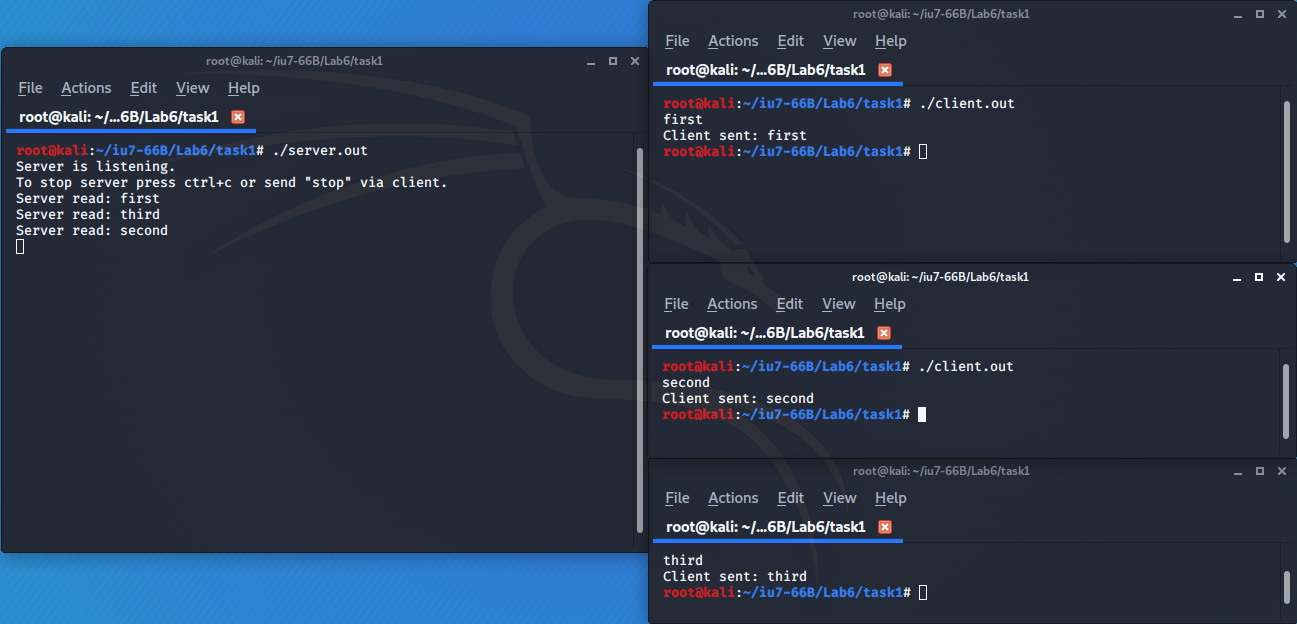
**Текст программы сервера:**

|  |
| --- |
| **#include <sys/types.h>**  #include <sys/socket.h>  #include <string.h>  #include <stdlib.h>  #include <stdio.h>  #include <unistd.h>  #include <signal.h>  #define SOCK\_NAME "mysocket.s"  int sock\_fd;  void close\_sock(int sock\_fd, char \*name)  {  close(sock\_fd);  unlink(name);  }  void sigint\_handler(int signum)  {  close\_sock(sock\_fd, SOCK\_NAME);  printf("\nSocket was closed due to ctrl+c!\n");  printf("Server will be stopped.\n");  exit(0);  }  int main()  {  sock\_fd = socket(AF\_UNIX, SOCK\_DGRAM, 0);  if (sock\_fd < 0)  {  perror("socket failed");  return EXIT\_FAILURE;  }    struct sockaddr srvr\_name;  srvr\_name.sa\_family = AF\_UNIX;  strcpy(srvr\_name.sa\_data, SOCK\_NAME);  if(bind(sock\_fd,&srvr\_name, strlen(srvr\_name.sa\_data)+sizeof(srvr\_name.sa\_family)) < 0)  {  perror("bind failed");  return EXIT\_FAILURE;  }    signal(SIGINT, sigint\_handler);  printf("Server is listening.\nTo stop server press ctrl+c or send \"stop\" via client.\n");    char buf[100];  while (strcmp(buf, "stop"))  {  int bytes = recv(sock\_fd, buf, sizeof(buf), 0);  if (bytes <= 0)  {  perror("recv failed");  close\_sock(sock\_fd, SOCK\_NAME);  return EXIT\_FAILURE;  }  buf[bytes] = 0;  printf("Server read: %s\n", buf);  }    printf("Server stopped listening\n");  close\_sock(sock\_fd, SOCK\_NAME);  printf("Socket closed\n");  return 0;  } |

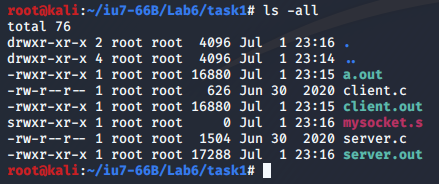
**Текст программы клиента:**

|  |
| --- |
| #include <sys/types.h>  #include <sys/socket.h>  #include <string.h>  #include <stdlib.h>  #include <stdio.h>  #define SOCK\_NAME "mysocket.s"  int main()  {  int sock\_fd = socket(AF\_UNIX, SOCK\_DGRAM, 0);  if (sock\_fd < 0)  {  perror("socket failed");  return EXIT\_FAILURE;  }    struct sockaddr srvr\_name;  srvr\_name.sa\_family = AF\_UNIX;  strcpy(srvr\_name.sa\_data, SOCK\_NAME);    char buf[100];  scanf("%99s", buf);  sendto(sock\_fd, buf,strlen(buf), 0, &srvr\_name, strlen(srvr\_name.sa\_data) + sizeof(srvr\_name.sa\_family));    printf("Client sent: %s\n", buf);  return 0;  } |

**Результат работы программы**



Создание сокета



1. СЕТЕВОЙ СОКЕТ

В процессе-сервере с помощью вызова socket() создается сокет семейства AF\_INET с типом SOCK\_STREAM. С помощью системного вызова bind() происходит связка сокета с адресом, прописанным в SOCKET\_ADDRESS. С помощью вызова listen() сокету сообщается, что должны приниматься новые соединения. На каждой итерации цикла создается новый набор дескрипторов set. В него заносятся сокет сервера и сокеты клиентов с помощью функции FD\_SET. После этого сервер блокируется на вызове функции select(), она возвращает управление, если хотя бы один из проверяемых сокетов готов к выполнению соответствующей операции. После выхода из блокировки, проверяется наличие новых соединений. При наличии таковых вызывается функция connectHandler(). В этой функции с помощью accept() принимается новое соединение, а также создается сокет, который записывается в массив файловых дескрипторов. Затем происходит обход массива дескрипторов, и, если дескриптор находится в наборе дескрипторов, то запускается функция clientHandeler(). В ней осуществляется считывание с помощью recv() и вывод сообщения от клиента. Если recv() возвращает ноль, то соединение было сброшено. В таком случае выводится сообщение о закрытии сокета.

В процессе-клиенте создается сокет семейста AF\_INET с типом SOCK\_STREAM с помощью системного вызова socket(). С помощью функции gethostbyname() доменный адрес преобразуется в сетевой и с его помощью можно установить соединение, используя функцию connect(). Затем происходит отправка сообщений серверу.

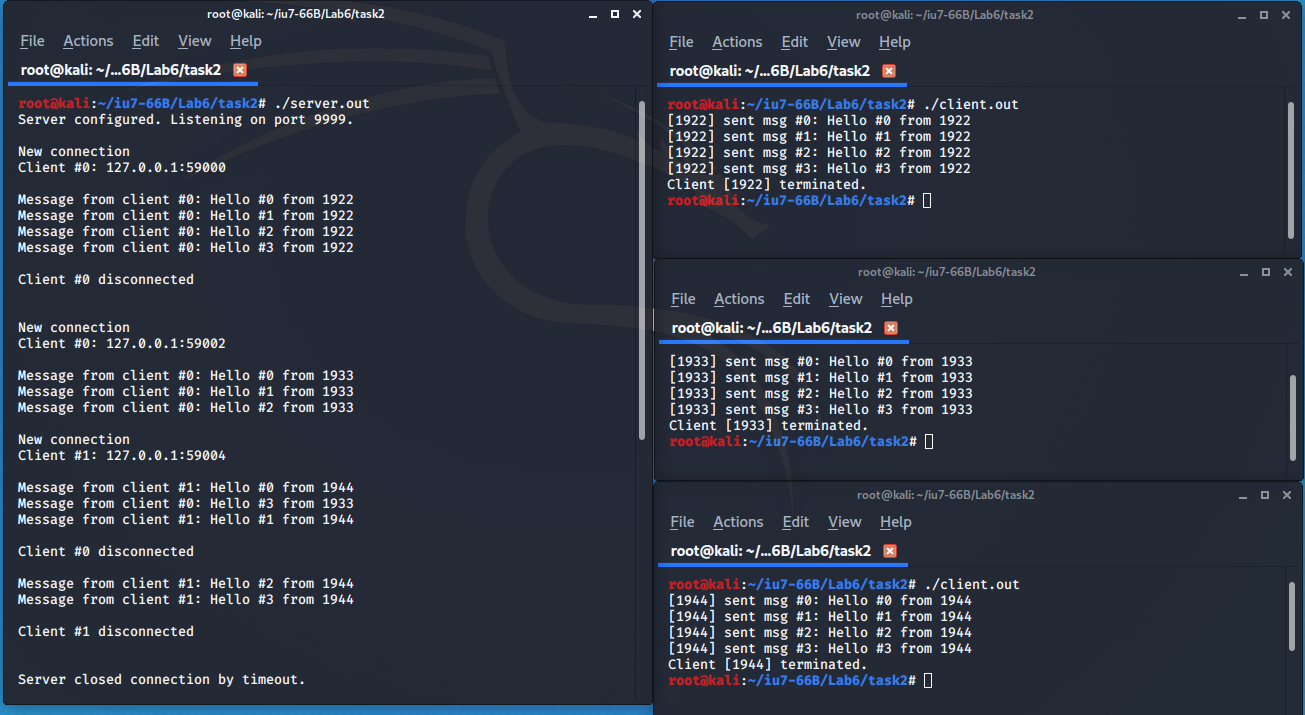
**Текст программы сервера:**

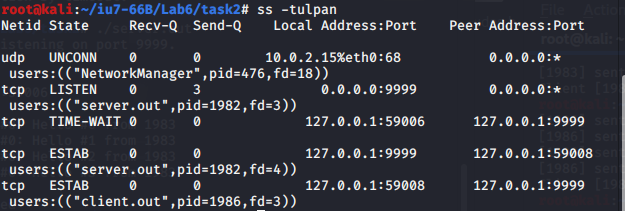
|  |
| --- |
| #include "includes.h"  #include <signal.h>  #define MAX\_CLIENTS 10  int clients[MAX\_CLIENTS] = { 0 };  void newConnectionHandler(unsigned int fd)  {  struct sockaddr\_in addr;  int addrSize = sizeof(addr);  int incom = accept(fd, (struct sockaddr\*) &addr, (socklen\_t\*) &addrSize);  if (incom < 0)  {  perror("accept failed");  exit(EXIT\_FAILURE);  }  int i;  for (i = 0; i < MAX\_CLIENTS; i++)  {  if (clients[i] == 0)  {  clients[i] = incom;  break;  }  }    printf("\nNew connection\nClient #%d: %s:%d\n\n",  i, inet\_ntoa(addr.sin\_addr), ntohs(addr.sin\_port));  }  void clientHandler(unsigned int fd, unsigned int client\_id)  {  char msg[MSG\_LEN];  memset(msg, 0, MSG\_LEN);  struct sockaddr\_in addr;  int addrSize = sizeof(addr);  int recvSize = recv(fd, msg, MSG\_LEN, 0);  if (recvSize == 0)  {  getpeername(fd, (struct sockaddr\*) &addr, (socklen\_t\*) &addrSize);  printf("\nClient #%d disconnected\n\n", client\_id);  close(fd);  clients[client\_id] = 0;  }  else  {  msg[recvSize] = '\0';  printf("Message from client #%d: %s\n", client\_id, msg);  }  }  int main(void)  {  // Establishing connection  int sock = socket(AF\_INET, SOCK\_STREAM, 0);  if (sock < 0)  {  perror("socket failed\n");  return EXIT\_FAILURE;  }  struct sockaddr\_in addr;  addr.sin\_family = AF\_INET;  addr.sin\_port = htons(SOCK\_PORT);  addr.sin\_addr.s\_addr = INADDR\_ANY;  if (bind(sock, (struct sockaddr\*) &addr, sizeof(addr)) < 0)  {  perror("bind failed\n");  return EXIT\_FAILURE;  }  if (listen(sock, 3) < 0)  {  perror("listen failed ");  return EXIT\_FAILURE;  }    // Handling requests  printf("Server configured. Listening on port %d.\n", SOCK\_PORT);  while (1)  {  // Fill sockets  int max\_fd = sock;    fd\_set set;  FD\_ZERO(&set);  FD\_SET(sock, &set);  for (int i = 0; i < MAX\_CLIENTS; i++)  {  if (clients[i] > 0)  FD\_SET(clients[i], &set);  max\_fd = (clients[i] > max\_fd) ? (clients[i]) : (max\_fd);  }    // Wait for event in one of sockets  struct timeval timeout = {15, 0}; // 15 sec  int select\_ret = select(max\_fd + 1, &set, NULL, NULL, &timeout);  if (select\_ret == 0)  {  printf("\nServer closed connection by timeout.\n\n");  return 0;  }  else if (select\_ret < 0)  {  perror("select failed");  return EXIT\_FAILURE;  }    // Checking for updates  // Connections  if (FD\_ISSET(sock, &set))  newConnectionHandler(sock);  // Messages  for (int i = 0; i < MAX\_CLIENTS; i++)  {  int fd = clients[i];  if ((fd > 0) && FD\_ISSET(fd, &set))  clientHandler(fd, i);  }  }  return 0;  } |

**Текст программы клиента:**

|  |
| --- |
| **#include "includes.h"**  int main(void)  {  // Establishing connection  int sock = socket(AF\_INET, SOCK\_STREAM, 0);  if (sock < 0)  {  perror("socket failed\n");  return sock;  }  struct hostent\* host = gethostbyname(SOCK\_ADDR);  if (!host)  {  perror("gethostbyname failed\n ");  return EXIT\_FAILURE;  }  struct sockaddr\_in addr;  addr.sin\_family = AF\_INET;  addr.sin\_port = htons(SOCK\_PORT);  addr.sin\_addr = \*((struct in\_addr\*) host->h\_addr\_list[0]);  if (connect(sock, (struct sockaddr\*) &addr, sizeof(addr)) < 0)  {  perror("connect failed\n");  return EXIT\_FAILURE;  }  // Sending messages  int pid = getpid();    char msg[MSG\_LEN];  for (int i = 0; i < 4; i++)  {  memset(msg, 0, MSG\_LEN);  sprintf(msg, "Hello #%d from %d", i, pid);    if (send(sock, msg, strlen(msg), 0) < 0)  {  perror("send failed: ");  return EXIT\_FAILURE;  }    printf("[%d] sent msg #%d: %s\n", pid, i, msg);  sleep(3);  }  printf("Client [%d] terminated.\n", pid);  return 0;  } |

**Результат работы программы**

****

Порты, 1 клиент посылает сообщение