**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №6**

**по дисциплине «Операционные системы»**

**Тема: Сокеты.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1304 |  | Чернякова В.А. |
| Преподаватель |  | Душутина Е.В. |

Санкт-Петербург

2023

**Цель работы.**

Изучить системное программирование в ОС семейства UNIX.

**Выполнение работы.**

Модель ОС:

Linux Valeriya 4.15.0-142-generic #146~16.04.1-Ubuntu SMP Tue Apr 13 09:27:15 UTC 2021 x86\_64 x86\_64 x86\_64 GNU/Linux

**Сокеты.**

Пример использования сокета – эхо сервер.

Задание: Рассмотрим пример программы - сервер прослушивает заданный порт, при запросе нового соединения создаётся новый поток для его обработки. Работа с клиентом организована как бесконечный цикл, в котором выполняется приём сообщения от клиента, вывод его на экран и пересылка обратно клиенту. Клиентская программа после установления соединения с сервером также в бесконечном цикле выполняет чтение ввода пользователя, пересылку серверу, получение работы. Для взаимодействия используются TCP сокеты, это значит, что между сервером и клиентом устанавливается логическое соединение, при этом при получении данных из сокета с помощью вызова recv, есть вероятность получить сразу несколько сообщений, или не полностью прочитать сообщение. Поэтому для установления взаимной однозначности между отосланными и принятыми данными используются функции recvFix и sendFix. Принцип их работы следующий: функция sendFix перед посылкой собственно данных посылает «заголовок» - количество байт в посылке. Функция recvFix вначале принимает этот «заголовок», и вторым вызовом recv считывает переданное количество байт. Считать ровно то, количество байт, которое указанно в аргументе функции recv, позволяет флаг MSG\_WAITALL. Если его не использовать и данных в буфере недостаточно, то будет прочитано меньшее количество.

Протестируем предложенное приложение. Для этого немного модифицируем его (каждый клиент при соединении отправит свой порядковый номер серверу). Далее напишем bash-скрипт, создающий N число клиентов.

*Server.c*

|  |
| --- |
| #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #define DEF\_PORT 8888  #define DEF\_IP "127.0.0.1"  // обработка одного клиента  //  void \*  clientHandler(void \*args)  {      int sock = (int)args;      char buf[100];      int res = 0;      for (;;)      {          bzero(buf, 100);          res = readFix(sock, buf, 100, 0);          if (res <= 0)          {              perror("Can't recv data from client, ending thread\n");              pthread\_exit(NULL);          }          printf("Some client sent: %s\n", buf);          res = sendFix(sock, buf, 0);          if (res <= 0)          {              perror("send call failed");              pthread\_exit(NULL);          }      }  }  int main(int argc, char \*\*argv)  {      int port = 0;      if (argc < 2)      {          printf("Using default port %d\n", DEF\_PORT);          port = DEF\_PORT;      }      else          port = atoi(argv[1]);      struct sockaddr\_in listenerInfo;      listenerInfo.sin\_family = AF\_INET;      listenerInfo.sin\_port = htons(port);      listenerInfo.sin\_addr.s\_addr = htonl(INADDR\_ANY);      int listener = socket(AF\_INET, SOCK\_STREAM, 0);      if (listener < 0)      {          perror("Can't create socket to listen: ");          exit(1);      }      int res = bind(listener, (struct sockaddr \*)&listenerInfo, sizeof(listenerInfo));      if (res < 0)      {          perror("Can't bind socket");          exit(1);      }      // слушаем входящие соединения      res = listen(listener, 5);      if (res)      {          perror("Error while listening:");          exit(1);      }      // основной цикл работы      for (;;)      {          int client = accept(listener, NULL, NULL);          pthread\_t thrd;          res = pthread\_create(&thrd, NULL, clientHandler, (void \*)(client));          if (res)          {              printf("Error while creating new thread\n");          }      }      return 0;  }  int readFix(int sock, char \*buf, int bufSize, int flags)  {      // читаем "заголовок" - сколько байт составляет наше сообщение      unsigned msgLength = 0;      int res = recv(sock, &msgLength, sizeof(unsigned), flags | MSG\_WAITALL);      if (res <= 0)          return res;      if (res > bufSize)      {          printf("Recieved more data, then we can store, exiting\n");          exit(1);      }      // читаем само сообщение      return recv(sock, buf, msgLength, flags | MSG\_WAITALL);  }  int sendFix(int sock, char \*buf, int flags)  {      // шлем число байт в сообщении      unsigned msgLength = strlen(buf);      int res = send(sock, &msgLength, sizeof(unsigned), flags);      if (res <= 0)          return res;      send(sock, buf, msgLength, flags);  } |

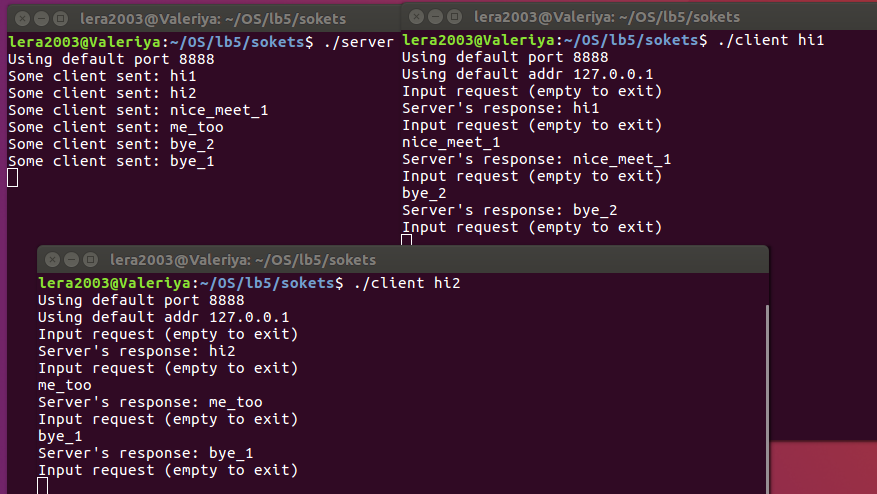
*Client.c*

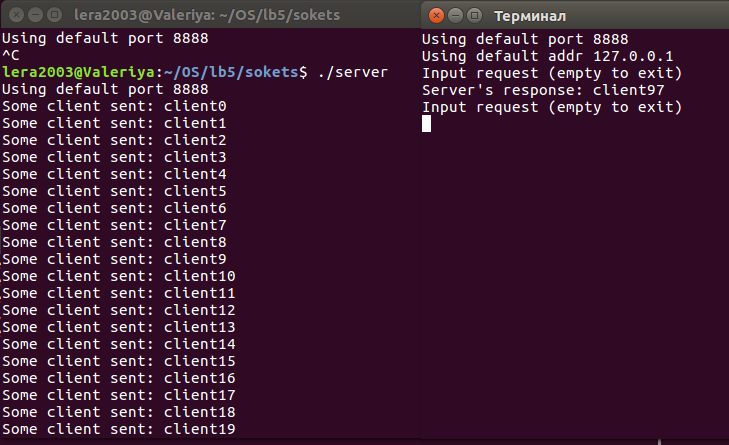
|  |
| --- |
| #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #define DEF\_PORT 8888  #define DEF\_IP "127.0.0.1"  int main(int argc, char \*\*argv)  {      char \*addr;      int port;      char \*readbuf;      printf("Using default port %d\n", DEF\_PORT);      port = DEF\_PORT;      printf("Using default addr %s\n", DEF\_IP);      addr = DEF\_IP;      // создаем сокет      struct sockaddr\_in peer;      peer.sin\_family = AF\_INET;      peer.sin\_port = htons(port);      peer.sin\_addr.s\_addr = inet\_addr(addr);      int sock = socket(AF\_INET, SOCK\_STREAM, 0);      if (sock < 0)      {          perror("Can't create socket\n");          exit(1);      }      // присоединяемся к серверу      int res = connect(sock, (struct sockaddr \*)&peer, sizeof(peer));      if (res)      {          perror("Can't connect to server:");          exit(1);      }      // основной цикл программы      char buf[100];      int first\_msg = 1;      for (;;)      {          printf("Input request (empty to exit)\n");          if (first\_msg == 0){              bzero(buf, 100);              fgets(buf, 100, stdin);              buf[strlen(buf) - 1] = '\0';          }          else{              strcpy(buf, argv[1]);              buf[strlen(buf)] = '\0';              first\_msg = 0;          }          if (strlen(buf) == 0)          {              printf("Bye-bye\n");              return 0;          }          res = sendFix(sock, buf, 0);          if (res <= 0)          {              perror("Error while sending:");              exit(1);          }          bzero(buf, 100);          res = readFix(sock, buf, 100, 0);          if (res <= 0)          {              perror("Error while receiving:");              exit(1);          }          printf("Server's response: %s\n", buf);      }      return 0;  }  int readFix(int sock, char \*buf, int bufSize, int flags)  {      // читаем "заголовок" - сколько байт составляет наше сообщение      unsigned msgLength = 0;      int res = recv(sock, &msgLength, sizeof(unsigned), flags | MSG\_WAITALL);      if (res <= 0)          return res;      if (res > bufSize)      {          printf("Recieved more data, then we can store, exiting\n");          exit(1);      }      // читаем само сообщение      return recv(sock, buf, msgLength, flags | MSG\_WAITALL);  }  int sendFix(int sock, char \*buf, int flags)  {      // число байт в сообщении      unsigned msgLength = strlen(buf);      int res = send(sock, &msgLength, sizeof(unsigned), flags);      if (res <= 0)          return res;      send(sock, buf, msgLength, flags);  } |

*Socket\_script.sh*

|  |
| --- |
| #!/bin/bash  client\_amount=$1  for (( counter=0; counter<client\_amount; counter++ ))  do      echo `gnome-terminal -- sh -c "bash -c \"./client client$counter; exec bash\""`  done |

Результат:





Как видно из скриншота, реализация с множеством клиентов не потребовала изменений.

Чтобы выполнить аналогичное взаимодействие на основе UDP, при создании сокета нужно заменить SOCK\_STREAM на SOCK\_DGRAM, функции recv() и send() на recvfrom() и sendto() соответственно.

SOCK\_STREAM - Обеспечивает создание двусторонних надежных и последовательных потоков байтов, поддерживающих соединения. Может также поддерживаться механизм внепоточных данных.

SOCK\_DGRAM - Поддерживает датаграммы (ненадежные сообщения с ограниченной длиной и не поддерживающие соединения).

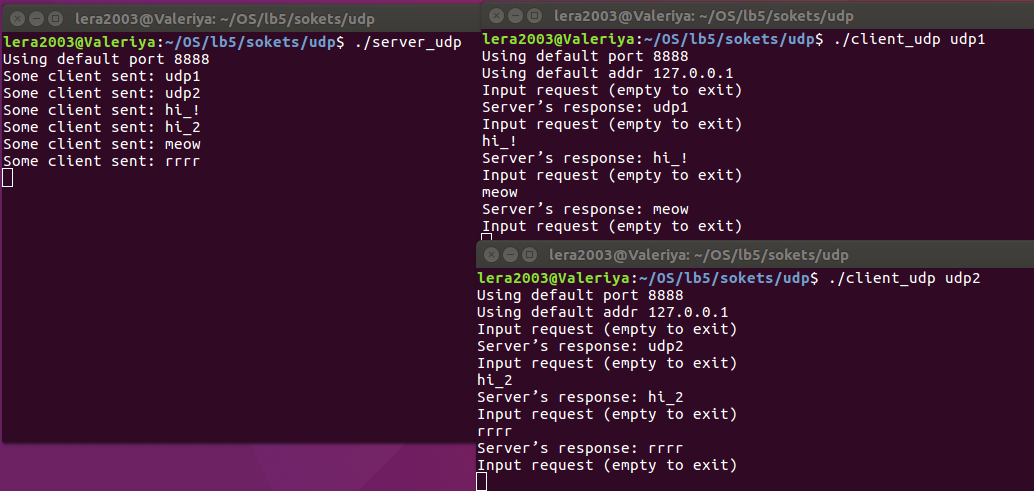
*server\_udp.c*

|  |
| --- |
| #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  // Задаем порт и адрес по умолчанию  #define DEF\_PORT 8888  #define DEF\_IP "127.0.0.1"  int main(int argc, char \*\*argv)  {      int port = 0;        if (argc < 2)      {          printf("Using default port %d\n", DEF\_PORT);          port = DEF\_PORT;      }      else          port = atoi(argv[1]);        struct sockaddr\_in listenerInfo;      // Задаем параметры для создания сокета      listenerInfo.sin\_family = AF\_INET; // Используем IPv4      listenerInfo.sin\_port = htons(port); // Задаем порт      listenerInfo.sin\_addr.s\_addr = htonl(INADDR\_ANY); // Bсе локальные интерфейсы будут слушать этот сокет      int listener = socket(AF\_INET, SOCK\_DGRAM, 0); // Создаем сокет для передачи данных по UDP      if (listener < 0) // Проверяем успешность создания сокета      {          perror("Can’t create socket to listen: ");          exit(1);      }      int res = bind(listener, (struct sockaddr \*)&listenerInfo, sizeof(listenerInfo));      // Связываем сокет с адресом и портом      if (res < 0) // Проверяем успешность связывания порта и сокета      {          perror("Can’t bind socket");          exit(1);      }      char buf[100];      struct sockaddr\_in client\_addr;      int client\_addr\_size = sizeof(client\_addr);      for (;;) // Запускаем бесконечный цикл для получения и отправки сообщений      {          bzero(buf, 100); // Обнуляем буфер          res = recvfrom(listener, buf, sizeof(buf)-1, 0, (struct sockaddr \*)&client\_addr, &client\_addr\_size);          // Принимаем сообщение от клиента и сохраняем данные в буфере          if (res <= 0) // Проверяем успешность получения сообщения от клиента          {              perror("Can’t recv data from client");              exit(1);          }          printf("Some client sent: %s\n", buf); // Выводим сообщение полученное от клиента          res = sendto(listener, buf, strlen(buf), 0, (struct sockaddr \*)&client\_addr, client\_addr\_size);          // Отправляем сообщение обратно клиенту          if (res <= 0) // Проверяем успешность отправки сообщения клиенту          {              perror("send call failed");              exit(1);          }      }      return 0;  } |

*client\_udp.c*

|  |
| --- |
| #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #define DEF\_PORT 8888  #define DEF\_IP "127.0.0.1"  int main(int argc, char \*\*argv)  {      char \*addr;      int port;      char \*readbuf;      printf("Using default port %d\n", DEF\_PORT);      port = DEF\_PORT;      printf("Using default addr %s\n", DEF\_IP);      addr = DEF\_IP;      struct sockaddr\_in peer;      peer.sin\_family = AF\_INET; // Используем IPv4      peer.sin\_port = htons(port); // Задаем порт      peer.sin\_addr.s\_addr = inet\_addr(addr);      int sock = socket(AF\_INET, SOCK\_DGRAM, 0); // Создаем сокет для передачи данных по UDP      if (sock < 0)      {          perror("Can’t create socket\n");          exit(1);      }      char buf[100];      int first\_msg = 1;      int peer\_addr\_size = sizeof(peer);      for (;;)      {          printf("Input request (empty to exit)\n");          if (first\_msg == 0){              bzero(buf, 100);              fgets(buf, 100, stdin);              buf[strlen(buf)-1] = '\0';          }          else{              strcpy(buf, argv[1]);              buf[strlen(buf)] = '\0';              first\_msg = 0;          }          if (strlen(buf) == 0)          {              printf("Bye-bye\n");              return 0;          }          // Отправляем запрос на сервер          int res = sendto(sock, buf, strlen(buf), 0, (struct sockaddr \*)&peer, sizeof(peer));          if (res <= 0)          {              perror("Error while sending:");              exit(1);          }          bzero(buf, 100);          // Получаем ответ от сервера.          res = recvfrom(sock, buf, sizeof(buf)-1, 0, (struct sockaddr \*)&peer, &peer\_addr\_size);          if (res <= 0)          {              perror("Error while receiving:");              exit(1);          }          printf("Server’s response: %s\n", buf);      }      return 0;  } |

Результат:

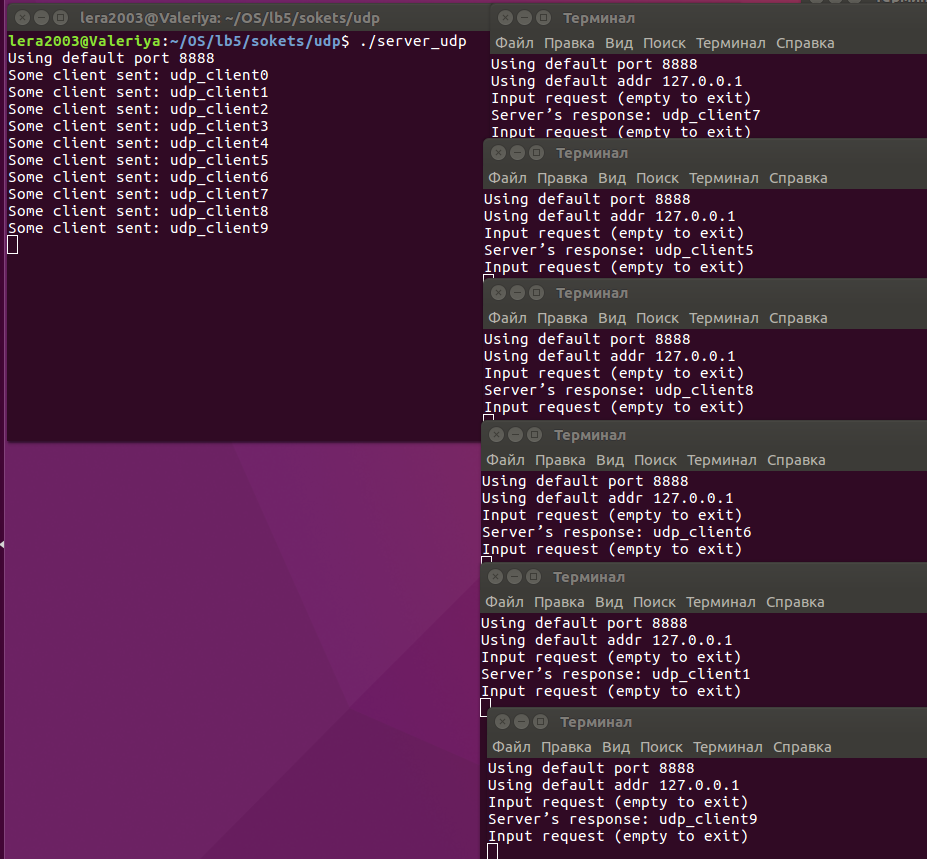


Таким образом, получилось модифицировать программу, чтобы коммуникация проходила через UDP соединение. Попробуем подключить к серверу 10 клиентов.

*script.sh*

|  |
| --- |
| #!/bin/bash  client\_amount=$1  for (( counter=0; counter<client\_amount; counter++ ))  do      echo `gnome-terminal -- sh -c "bash -c \"./client\_udp udp\_client$counter; exec bash\""`  done |

Результат:

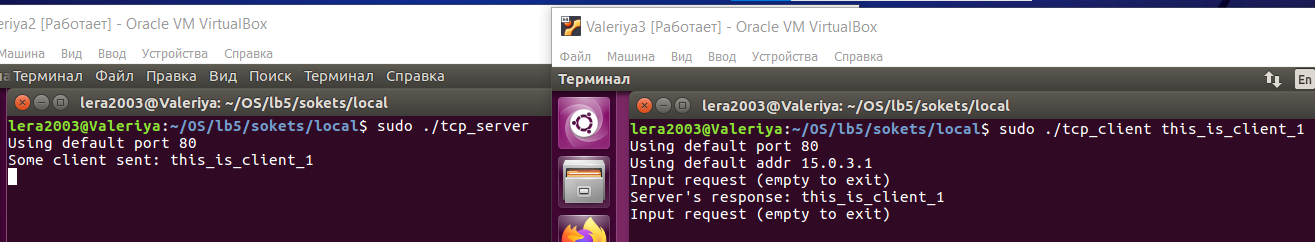


Можно сделать вывод, что между очередями сообщений и сокетами есть различия: первое – формат передачи данных. Очереди сообщений основаны на передаче структурированных сообщений, а сокеты – на передаче байтов. Кроме того, сокеты могут использовать различные протоколы (TCP или UDP), когда как очереди сообщений используют один и тот же механизм.

Повторим эксперименты в локальной сети.

Настроим на двух машинах одну сеть, IP адреса следующие: 15.0.3.1 и 15.0.3.2.

TCP:



UDP:



Задание: эксперементально покажем отличия tcp от udp.

Для нагрузки tcp в локальной сети будем компилировать как tcp\_client и запускать скрипт для многих клиентов.

Для нагрузки udp в локальной сети будем компилировать как udp\_client и тоже скрипт для множества клиентов.

Для каждого клиента отправляются сообщения вида <номер клиента><1>..<9> и так от 2х символов в сообщении до 91 символа в сообщении. Так 10 раз.

*tcpvsudp/tcp\_server.c*

|  |
| --- |
| #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #define DEF\_PORT 80  #define DEF\_IP "127.0.0.1"  // обработка одного клиента  void \*clientHandler(void \*args)  {      int sock = (int)args;      char buf[100];      int res = 0;      for (;;)      {          bzero(buf, 100);          res = readFix(sock, buf, 100, 0);          if (res <= 0)          {              perror("Can't recv data from client, ending thread\n");              pthread\_exit(NULL);          }          printf("Some client sent: %s\n", buf);          res = sendFix(sock, buf, 0);          if (res <= 0)          {              perror("send call failed");              pthread\_exit(NULL);          }      }  }  int main(int argc, char \*\*argv)  {      int port = 0;      if (argc < 2)      {          printf("Using default port %d\n", DEF\_PORT);          port = DEF\_PORT;      }      else          port = atoi(argv[1]);      struct sockaddr\_in listenerInfo;      listenerInfo.sin\_family = AF\_INET;      listenerInfo.sin\_port = htons(port);      listenerInfo.sin\_addr.s\_addr = htonl(INADDR\_ANY);      int listener = socket(AF\_INET, SOCK\_STREAM, 0);      if (listener < 0)      {          perror("Can't create socket to listen: ");          exit(1);      }      int res = bind(listener, (struct sockaddr \*)&listenerInfo, sizeof(listenerInfo));      if (res < 0)      {          perror("Can't bind socket");          exit(1);      }      // слушаем входящие соединения      res = listen(listener, 5);      if (res)      {          perror("Error while listening:");          exit(1);      }      // основной цикл работы      for (;;)      {          int client = accept(listener, NULL, NULL);          pthread\_t thrd;          res = pthread\_create(&thrd, NULL, clientHandler, (void \*)(client));          if (res)          {              printf("Error while creating new thread\n");          }      }      return 0;  }  int readFix(int sock, char \*buf, int bufSize, int flags)  {      // читаем "заголовок" - сколько байт составляет наше сообщение      unsigned msgLength = 0;      int res = recv(sock, &msgLength, sizeof(unsigned), flags | MSG\_WAITALL);      if (res <= 0)          return res;      if (res > bufSize)      {          printf("Recieved more data, then we can store, exiting\n");          exit(1);      }      // читаем само сообщение      return recv(sock, buf, msgLength, flags | MSG\_WAITALL);  }  int sendFix(int sock, char \*buf, int flags)  {      // шлем число байт в сообщении      unsigned msgLength = strlen(buf);      int res = send(sock, &msgLength, sizeof(unsigned), flags);      if (res <= 0)          return res;      send(sock, buf, msgLength, flags);  } |

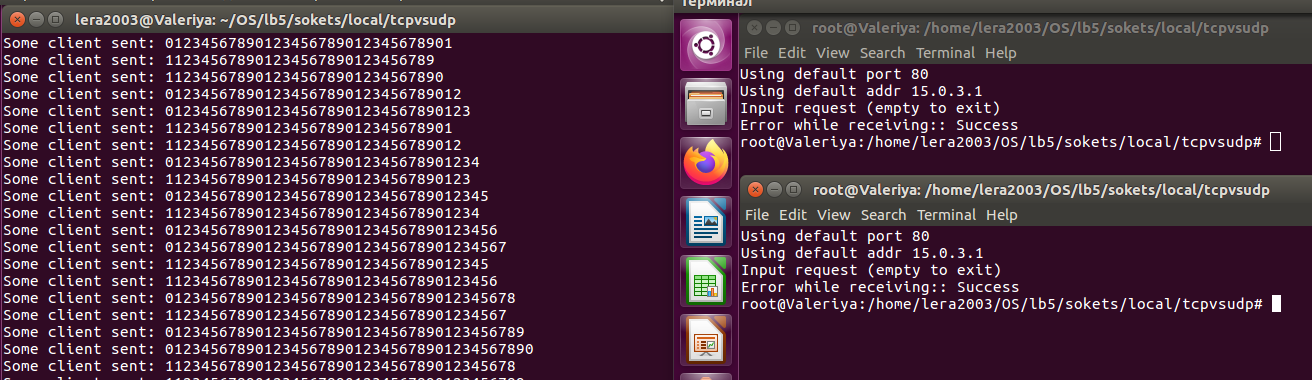
*tcpvsudp/tcp\_client.c*

|  |
| --- |
| #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #define DEF\_PORT 80  #define DEF\_IP "15.0.3.1"  int main(int argc, char \*\*argv) {      char \*addr;      int port;      char \*readbuf;      printf("Using default port %d\n", DEF\_PORT);      port = DEF\_PORT;      printf("Using default addr %s\n", DEF\_IP);      addr = DEF\_IP;      // создаем сокет      struct sockaddr\_in peer;      peer.sin\_family = AF\_INET;      peer.sin\_port = htons(port);      peer.sin\_addr.s\_addr = inet\_addr(addr);      int sock = socket(AF\_INET, SOCK\_STREAM, 0);      if (sock < 0) {          perror("Can't create socket\n");          exit(1);      }      // присоединяемся к серверу      int res = connect(sock, (struct sockaddr \*)&peer, sizeof(peer));      if (res) {          perror("Can't connect to server:");          exit(1);      }      // основной цикл программы      char buf[100];      int first\_msg = 1;      for (;;) {          printf("Input request (empty to exit)\n");          if (first\_msg == 0){               bzero(buf, 100);              fgets(buf, 100, stdin);              buf[strlen(buf) - 1] = '\0';          }          else{          for (int k = 0; k < 10;k++){              buf[0] = argv[1][6];              for (int i = 1; i < 90; i++){                  char j = i%10 + '0';                  buf[i] = j;              buf[strlen(buf)] = '\0';              first\_msg = 0;              if (strlen(buf) == 0) {              printf("Bye-bye\n");              return 0;          }          res = sendFix(sock, buf, 0);          if (res <= 0) {              perror("Error while sending:");              exit(1);          }          bzero(buf, 100);              res = readFix(sock, buf, 100, 0);              if (res <= 0) {                      perror("Error while receiving:");                      exit(1);              }              }              bzero(buf, 100);             }              //strcpy(buf, argv[1]);              //buf[strlen(buf)] = '\0';              //first\_msg = 0;          }          if (strlen(buf) == 0) {              printf("Bye-bye\n");              return 0;          }          res = sendFix(sock, buf, 0);          if (res <= 0) {              perror("Error while sending:");              exit(1);          }          bzero(buf, 100);          res = readFix(sock, buf, 100, 0);          if (res <= 0) {              perror("Error while receiving:");              exit(1);          }          printf("Server's response: %s\n", buf);      }      return 0;  }  int readFix(int sock, char \*buf, int bufSize, int flags) {      // читаем "заголовок" - сколько байт составляет наше сообщение      unsigned msgLength = 0;      int res = recv(sock, &msgLength, sizeof(unsigned), flags | MSG\_WAITALL);      if (res <= 0) return res;      if (res > bufSize) {          printf("Recieved more data, then we can store, exiting\n");          exit(1);      }      // читаем само сообщение      return recv(sock, buf, msgLength, flags | MSG\_WAITALL);  }  int sendFix(int sock, char \*buf, int flags) {      // число байт в сообщении      unsigned msgLength = strlen(buf);      int res = send(sock, &msgLength, sizeof(unsigned), flags);      if (res <= 0) return res;      send(sock, buf, msgLength, flags);  } |

*tcpvsudp/tcp\_script.sh*

|  |
| --- |
| #!/bin/bash  client\_amount=$1  for (( counter=0; counter<client\_amount; counter++ ))  do      echo `gnome-terminal -- sh -c "bash -c \"./tcp\_client client$counter; exec bash\""`  done |

Результат работы:



Выводы: клиенты отправляют сообщения параллельно, но сообщения доходят до сервера в том порядке, в котором были отправлены.

*tcpvsudp/udp\_server.c*

|  |
| --- |
| #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  // Задаем порт и адрес по умолчанию  #define DEF\_PORT 80  #define DEF\_IP "127.0.0.1"  int main(int argc, char \*\*argv)  {      int port = 0;        if (argc < 2)      {          printf("Using default port %d\n", DEF\_PORT);          port = DEF\_PORT;      }      else          port = atoi(argv[1]);        struct sockaddr\_in listenerInfo;      // Задаем параметры для создания сокета      listenerInfo.sin\_family = AF\_INET; // Используем IPv4      listenerInfo.sin\_port = htons(port); // Задаем порт      listenerInfo.sin\_addr.s\_addr = htonl(INADDR\_ANY); // Bсе локальные интерфейсы будут слушать этот сокет      int listener = socket(AF\_INET, SOCK\_DGRAM, 0); // Создаем сокет для передачи данных по UDP      if (listener < 0) // Проверяем успешность создания сокета      {          perror("Can’t create socket to listen: ");          exit(1);      }      int res = bind(listener, (struct sockaddr \*)&listenerInfo, sizeof(listenerInfo));      // Связываем сокет с адресом и портом      if (res < 0) // Проверяем успешность связывания порта и сокета      {          perror("Can’t bind socket");          exit(1);      }      char buf[100];      struct sockaddr\_in client\_addr;      int client\_addr\_size = sizeof(client\_addr);      for (;;) // Запускаем бесконечный цикл для получения и отправки сообщений      {          bzero(buf, 100); // Обнуляем буфер          res = recvfrom(listener, buf, sizeof(buf)-1, 0, (struct sockaddr \*)&client\_addr, &client\_addr\_size);          // Принимаем сообщение от клиента и сохраняем данные в буфере          if (res <= 0) // Проверяем успешность получения сообщения от клиента          {              perror("Can’t recv data from client");              exit(1);          }          printf("Some client sent: %s\n", buf); // Выводим сообщение полученное от клиента          res = sendto(listener, buf, strlen(buf), 0, (struct sockaddr \*)&client\_addr, client\_addr\_size);          // Отправляем сообщение обратно клиенту          if (res <= 0) // Проверяем успешность отправки сообщения клиенту          {              perror("send call failed");              exit(1);          }      }      return 0;  } |

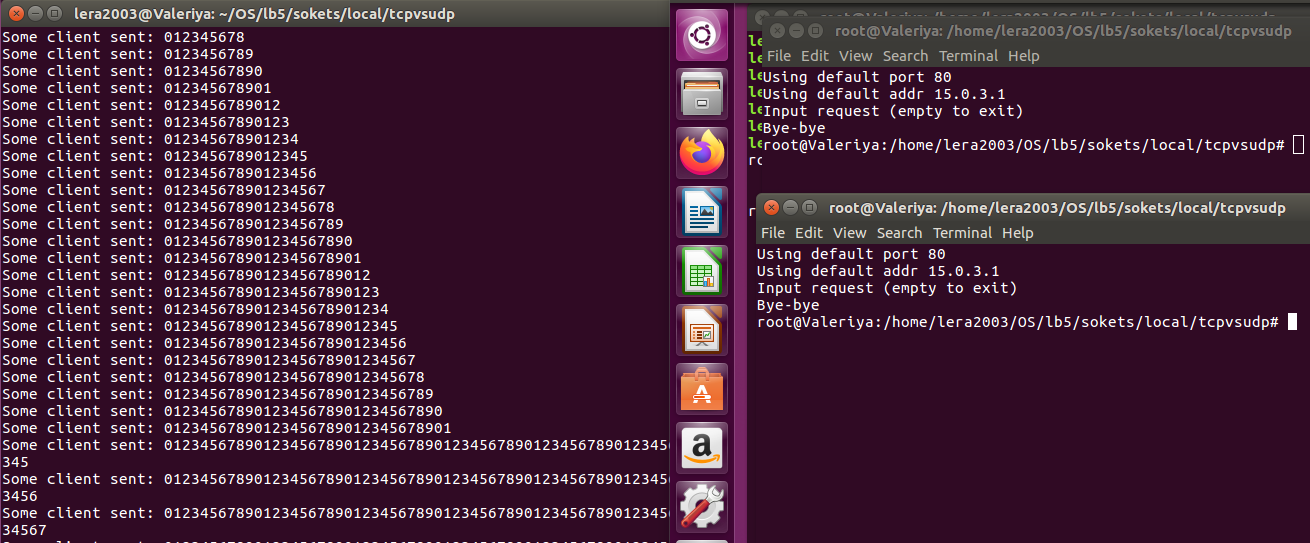
*tcpvsudp/udp\_client.c*

|  |
| --- |
| #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #define DEF\_PORT 80  #define DEF\_IP "15.0.3.1"  int main(int argc, char \*\*argv)  {      char \*addr;      int port;      char \*readbuf;      printf("Using default port %d\n", DEF\_PORT);      port = DEF\_PORT;      printf("Using default addr %s\n", DEF\_IP);      addr = DEF\_IP;      struct sockaddr\_in peer;      peer.sin\_family = AF\_INET; // Используем IPv4      peer.sin\_port = htons(port); // Задаем порт      peer.sin\_addr.s\_addr = inet\_addr(addr);      int sock = socket(AF\_INET, SOCK\_DGRAM, 0); // Создаем сокет для передачи данных по UDP      if (sock < 0)      {          perror("Can’t create socket\n");          exit(1);      }      char buf[100];      int first\_msg = 1;      int peer\_addr\_size = sizeof(peer);      for (;;)      {          printf("Input request (empty to exit)\n");          if (first\_msg == 0){              bzero(buf, 100);              fgets(buf, 100, stdin);              buf[strlen(buf)-1] = '\0';          }          else{          for (int k = 0; k < 10;k++){              buf[0] = argv[1][10];              for (int i = 1; i < 90; i++){                  char j = i%10 + '0';                  buf[i] = j;              buf[strlen(buf)] = '\0';              first\_msg = 0;              if (strlen(buf) == 0)              {                  printf("Bye-bye\n");                  return 0;              }              // Отправляем запрос на сервер              int res = sendto(sock, buf, strlen(buf), 0, (struct sockaddr \*)&peer, sizeof(peer));              if (res <= 0)              {                  perror("Error while sending:");                  exit(1);              }              }              bzero(buf, 100);             }                //strcpy(buf, argv[1]);              //buf[strlen(buf)] = '\0';              //first\_msg = 0;          }          if (strlen(buf) == 0)          {              printf("Bye-bye\n");              return 0;          }          // Отправляем запрос на сервер          int res = sendto(sock, buf, strlen(buf), 0, (struct sockaddr \*)&peer, sizeof(peer));          if (res <= 0)          {              perror("Error while sending:");              exit(1);          }          bzero(buf, 100);          // Получаем ответ от сервера.          res = recvfrom(sock, buf, sizeof(buf)-1, 0, (struct sockaddr \*)&peer, &peer\_addr\_size);          if (res <= 0)          {              perror("Error while receiving:");              exit(1);          }          printf("Server’s response: %s\n", buf);      }      return 0;  } |

*tcpvsudp/udp\_script.sh*

|  |
| --- |
| #!/bin/bash  client\_amount=$1  for (( counter=0; counter<client\_amount; counter++ ))  do      echo `gnome-terminal -- sh -c "bash -c \"./udp\_client udp\_client$counter; exec bash\""`  done |

Результат работы программы:



Выводы: Как видно, происходят скачки, это означает что часть сообщений была перемешана либо утерена.

**Вывод.**

На практике реализован пример использования сокета – эхо сервер. Написана программа-сервер, прослушивающая заданный порт, при запросе нового соединения создаётся новый поток для его обработки. Данное взаимодействие рассмотрено как на основе TCP, так и UDP.

В ходе сравнения tcp и udp выявлено, что первый является более надёжным. Все сообщения доходят без потерь.

**Список источников.**

1. «Системное программное обеспечение. Межпроцессные взаимодействия в операционных системах. Учебное пособие» Душутина Е.В.
2. Сайт [The Linux Programming Interface (man7.org)](https://man7.org/tlpi/index.html)
3. Сайт [OpenNet: Архив документации: Программирование в Linux](https://www.opennet.ru/docs/137.shtml)
4. Сайт [Ubuntu Manpage: Welcome](https://manpages.ubuntu.com/)
5. Сайт <https://www.opennet.ru/docs/RUS/linux_parallel/node7.html>