# **1. AF\_UNIX, SOCK\_DGRAM**

## **Сервер**

| #include <signal.h> #include <stdio.h> #include <stdlib.h> #include <string.h> #include <sys/socket.h> #include <sys/un.h> #include <time.h> #include <unistd.h> #include <errno.h>  #define SOCKET\_SERVER\_NAME "server\_socket" #define BUF\_SIZE 100   static int sfd;  void cleanup\_socket(void)  {  close(sfd);  unlink(SOCKET\_SERVER\_NAME); }  void sighandler(int signum)  {  printf("\nCaught signal\n");  cleanup\_socket();  exit(0); }  int main(void)  {  if ((sfd = socket(AF\_UNIX, SOCK\_DGRAM, 0)) == -1)   {  perror("socket failed\n");  return errno;  }   struct sockaddr\_un sock\_addr;  sock\_addr.sun\_family = AF\_UNIX;  strcpy(sock\_addr.sun\_path, SOCKET\_SERVER\_NAME);    if (bind(sfd, (struct sockaddr\*)&sock\_addr, sizeof(sock\_addr)) == -1)   {  perror("bind failed\n");  cleanup\_socket();  return errno;  }   signal(SIGINT, sighandler);    char msg\_rec[BUF\_SIZE];  char msg\_send[BUF\_SIZE];    while (1)   {  printf("Waiting for client..Press Ctrl + C to stop\n");    struct sockaddr\_un client\_addr;  int addr\_size;    size\_t bytes = recvfrom(sfd, msg\_rec, BUF\_SIZE, 0, (struct sockaddr\*) &client\_addr, (socklen\_t\*)&addr\_size);  if (bytes == -1)   {  perror("recvfrom failed");  cleanup\_socket();  return errno;  }    msg\_rec[bytes] = '\0';  printf("Server received message from %s: %s\n", client\_addr.sun\_path, msg\_rec);        strcpy(msg\_send, "Message was recieved: ");  strcat(msg\_send, msg\_rec);  bytes = sendto(sfd, msg\_send, strlen(msg\_send), 0, (struct sockaddr\*) &client\_addr, addr\_size);  if (bytes <= 0)   {  printf("sendto failed");  cleanup\_socket();  return errno;  }  } } |
| --- |

## **Клиент**

| #include <stdio.h> #include <string.h> #include <sys/socket.h> #include <sys/un.h> #include <unistd.h> #include <errno.h> #include <signal.h> #include <stdlib.h>   #define SOCKET\_SERVER\_NAME "server\_socket" #define SOCKET\_CLIENT\_NAME "client\_socket\_" #define BUF\_SIZE 100   static int sfd;  void cleanup\_socket(void)  {  close(sfd);  unlink(SOCKET\_CLIENT\_NAME); }  void sighandler(int signum)  {  cleanup\_socket();  exit(0); }  int main(int argc, char\* argv[]) {  sfd = socket(AF\_UNIX, SOCK\_DGRAM, 0);  if (sfd == -1)   {  perror("socket failed");  return errno;  }     signal(SIGINT, sighandler);    char msg[BUF\_SIZE];  snprintf(msg, BUF\_SIZE, "pid\_%d", getpid());     /\* client socket \*/  struct sockaddr\_un addr\_client;  addr\_client.sun\_family = AF\_UNIX;    strcpy(addr\_client.sun\_path, SOCKET\_CLIENT\_NAME);  strcat(addr\_client.sun\_path, msg);    if (bind(sfd, (struct sockaddr\*)&addr\_client, sizeof(addr\_client)) == -1)   {  perror("bind failed");  cleanup\_socket();  return errno;  }    /\* server socket \*/  struct sockaddr\_un server\_addr;  server\_addr.sun\_family = AF\_UNIX;  strcpy(server\_addr.sun\_path, SOCKET\_SERVER\_NAME);      /\*  if (connect(sfd, (struct sockaddr \*) &server\_addr, sizeof(server\_addr)) == -1)   {  perror("connect failed");  cleanup\_socket();  return errno;  }  \*/      size\_t bytes;      //while (1)  //{  bytes = sendto(sfd, msg, strlen(msg), 0, (struct sockaddr\*) &server\_addr, sizeof(server\_addr));  if (bytes <= 0)   {  printf("sendto failed");  cleanup\_socket();  return errno;  }   /\*  if (send(sfd, msg, strlen(msg), 0) == -1)   {   perror("send failed");  return errno;  }   \*/    char servermsg[BUF\_SIZE] = {0};  bytes = recv(sfd, servermsg, sizeof(servermsg), 0);  if (bytes < 0)   {  perror("recv failed");  return errno;  }    printf("Client received message: %s\n", servermsg);  sleep(3);  //}      cleanup\_socket();  return 0; } |
| --- |

# **2. AF\_UNIX, SOCK\_STREAM**

## Сервер

| #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <sys/socket.h> #include <sys/un.h> #include <sys/types.h> #include <signal.h>  #define SOCK\_PATH "mysocket" #define BUF\_SIZE 100 #define N\_CLIENTS 5   int sock\_fd;  void del\_socket(void) {  if (close(sock\_fd) == -1) //закрытие сокета  {  return;  }  if (unlink(SOCK\_PATH) == -1) //удаление файла сокета  {  return;  } }  void sigtstp\_handler(int signum) {  printf("Catch SIGTSTP\n");  del\_socket();  exit(0); }  int main() {  int client\_sock\_fd;  struct sockaddr\_un sock\_addr;    signal(SIGTSTP, sigtstp\_handler);    sock\_fd = socket(AF\_UNIX, SOCK\_STREAM, 0);  if(sock\_fd == -1)  {  printf("socket() failed\n");  return EXIT\_FAILURE;  }   sock\_addr.sun\_family = AF\_UNIX;  strcpy(sock\_addr.sun\_path, SOCK\_PATH);   if(bind(sock\_fd, (struct sockaddr\*)&sock\_addr, strlen(sock\_addr.sun\_path) + sizeof(sock\_addr.sun\_family)) != 0)  {  printf("bind() failed\n");  del\_socket();  return EXIT\_FAILURE;  }   if(listen(sock\_fd, N\_CLIENTS) != 0 )  {  printf("listen() failed\n");  del\_socket();  return EXIT\_FAILURE;  }   printf("Listening.\nPress Ctrl + Z to stop...\n");     int bytes;  char recv\_buf[BUF\_SIZE];  char send\_buf[BUF\_SIZE];    while (1)  {  unsigned int sock\_len;  struct sockaddr\_un remote\_sock\_addr;   if((client\_sock\_fd = accept(sock\_fd, (struct sockaddr\*)&remote\_sock\_addr, &sock\_len)) == -1 )  {  printf("Error on accept() call \n");  return 1;  }    memset(recv\_buf, 0, BUF\_SIZE);  memset(send\_buf, 0, BUF\_SIZE);  bytes = recv(client\_sock\_fd, recv\_buf, BUF\_SIZE, 0);    if(bytes > 0)  {  printf("Server received: %s\n", recv\_buf);  strcpy(send\_buf, "Message recieved: ");  strcat(send\_buf, recv\_buf);   if(send(client\_sock\_fd, send\_buf, strlen(send\_buf), 0) == -1 )  {  printf("send() failed\n");  }  }  else  {  printf("recv() failed. Waiting for new connection\n");  break;  }  }    del\_socket();  return 0; } |
| --- |

## Клиент

| #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <sys/socket.h> #include <sys/un.h> #include <sys/types.h> #include <signal.h> #include <errno.h>  #define SOCK\_PATH "mysocket" #define BUF\_SIZE 100   int sock\_fd; void sigtstp\_handler(int signum) {  printf("Catch SIGTSTP\n");  close(sock\_fd);  exit(0); }   int main() {   int bytes;  struct sockaddr\_un server\_sock\_addr;  char recv\_msg[BUF\_SIZE];  char send\_msg[BUF\_SIZE];   memset(recv\_msg, 0, BUF\_SIZE);  memset(send\_msg, 0, BUF\_SIZE);  signal(SIGTSTP, sigtstp\_handler);    server\_sock\_addr.sun\_family = AF\_UNIX;  strcpy(server\_sock\_addr.sun\_path, SOCK\_PATH);  memset(send\_msg, 0, BUF\_SIZE);  sprintf(send\_msg, "pid %d", getpid());    while (1)   {  if((sock\_fd = socket(AF\_UNIX, SOCK\_STREAM, 0)) == -1 )  {  printf("socket() failed\n");  return EXIT\_FAILURE;  }    if(connect(sock\_fd, (struct sockaddr\*)&server\_sock\_addr, strlen(server\_sock\_addr.sun\_path) + sizeof(server\_sock\_addr.sun\_family)) == -1)  {  printf("connect() failed %d\n", errno);  close(sock\_fd);  return EXIT\_FAILURE;  }     if(send(sock\_fd, send\_msg, strlen(send\_msg), 0 ) == -1)  {  printf("send() failed\n");  }      memset(recv\_msg, 0, BUF\_SIZE);   if((bytes = recv(sock\_fd, recv\_msg, BUF\_SIZE, 0)) > 0 )  {  printf("Client received: %s\n", recv\_msg);  }  else  {  if(bytes < 0)  {  printf("recv() failed\n");  }  else  {  printf("Server socket closed \n");  close(sock\_fd);  break;  }   }  sleep(3);  close(sock\_fd);  }   return 0; } |
| --- |

# 3. AF\_UNIX, SOCK\_STREAM | SOCK\_NONBLOCK

## Сервер

| #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <sys/socket.h> #include <sys/un.h> #include <sys/types.h> #include <signal.h>  #define SOCK\_PATH "mysocket" #define BUF\_SIZE 100 #define N\_CLIENTS 5   int sock\_fd;  void del\_socket(void) {  if (close(sock\_fd) == -1) //закрытие сокета  {  return;  }  if (unlink(SOCK\_PATH) == -1) //удаление файла сокета  {  return;  } }  void sigtstp\_handler(int signum) {  printf("Catch SIGTSTP\n");  del\_socket();  exit(0); }  int main() {  int client\_sock\_fd;  struct sockaddr\_un sock\_addr;    signal(SIGTSTP, sigtstp\_handler);    sock\_fd = socket(AF\_UNIX, SOCK\_STREAM | SOCK\_NONBLOCK , 0);  if(sock\_fd == -1)  {  printf("socket() failed\n");  return EXIT\_FAILURE;  }   sock\_addr.sun\_family = AF\_UNIX;  strcpy(sock\_addr.sun\_path, SOCK\_PATH);   if(bind(sock\_fd, (struct sockaddr\*)&sock\_addr, strlen(sock\_addr.sun\_path) + sizeof(sock\_addr.sun\_family)) != 0)  {  printf("bind() failed\n");  del\_socket();  return EXIT\_FAILURE;  }   if(listen(sock\_fd, N\_CLIENTS) != 0 )  {  printf("listen() failed\n");  del\_socket();  return EXIT\_FAILURE;  }   printf("Listening.\nPress Ctrl + Z to stop...\n");     int bytes;  char recv\_buf[BUF\_SIZE];  char send\_buf[BUF\_SIZE];    while (1)  {  unsigned int sock\_len;  struct sockaddr\_un remote\_sock\_addr;   if((client\_sock\_fd = accept(sock\_fd, (struct sockaddr\*)&remote\_sock\_addr, &sock\_len)) == -1 )  {  continue;  //printf("Error on accept() call \n");  //return 1;  }    memset(recv\_buf, 0, BUF\_SIZE);  memset(send\_buf, 0, BUF\_SIZE);  bytes = recv(client\_sock\_fd, recv\_buf, BUF\_SIZE, 0);    if(bytes > 0)  {  printf("Server received: %s\n", recv\_buf);  strcpy(send\_buf, "Message recieved: ");  strcat(send\_buf, recv\_buf);   if(send(client\_sock\_fd, send\_buf, strlen(send\_buf), 0) == -1 )  {  printf("send() failed\n");  }  }  else  {  continue;  //printf("recv() failed. Waiting for new connection\n");  //break;  }  }    del\_socket();  return 0; } |
| --- |

## Клиент

| #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <sys/socket.h> #include <sys/un.h> #include <sys/types.h> #include <signal.h> #include <errno.h>  #define SOCK\_PATH "mysocket" #define BUF\_SIZE 100   int sock\_fd; void sigtstp\_handler(int signum) {  printf("Catch SIGTSTP\n");  close(sock\_fd);  exit(0); }   int main() {   int bytes;  struct sockaddr\_un server\_sock\_addr;  char recv\_msg[BUF\_SIZE];  char send\_msg[BUF\_SIZE];   memset(recv\_msg, 0, BUF\_SIZE);  memset(send\_msg, 0, BUF\_SIZE);  signal(SIGTSTP, sigtstp\_handler);    server\_sock\_addr.sun\_family = AF\_UNIX;  strcpy(server\_sock\_addr.sun\_path, SOCK\_PATH);  memset(send\_msg, 0, BUF\_SIZE);  sprintf(send\_msg, "pid %d", getpid());    while (1)   {  if((sock\_fd = socket(AF\_UNIX, SOCK\_STREAM | SOCK\_NONBLOCK , 0)) == -1 )  {  printf("socket() failed\n");  return EXIT\_FAILURE;  }    if(connect(sock\_fd, (struct sockaddr\*)&server\_sock\_addr, strlen(server\_sock\_addr.sun\_path) + sizeof(server\_sock\_addr.sun\_family)) == -1)  {  printf("connect() failed %d\n", errno);  close(sock\_fd);  return EXIT\_FAILURE;  }     if(send(sock\_fd, send\_msg, strlen(send\_msg), 0 ) == -1)  {  printf("send() failed\n");  }      memset(recv\_msg, 0, BUF\_SIZE);      while (1)  {  if((bytes = recv(sock\_fd, recv\_msg, BUF\_SIZE, 0)) > 0 )  {  printf("Client received: %s\n", recv\_msg);  break;  }  else  {  if(bytes < 0)  {  continue;  //printf("recv() failed\n");  }  else  {  printf("Server socket closed \n");  close(sock\_fd);  break;  }   }  }  sleep(3);  close(sock\_fd);  }   return 0; } |
| --- |

# 4. AF\_INET, SOCK\_STREAM

## Сервер

| #include <stdio.h> #include <stdlib.h> #include <errno.h> #include <strings.h> #include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <stdlib.h> #include <unistd.h> #include <fcntl.h> #include <signal.h> #include <string.h>  #define BUF\_SIZE 256 #define PORT 3425 #define MAX\_CLIENTS\_NUMB 5   int sock\_fd;   void del\_socket(void) {  if (close(sock\_fd) == -1) //закрытие сокета  {  printf("close() failed\n");  return;  } }  void sigtstp\_handler(int signum) {  printf("\nCatch SIGTSTP\n");  del\_socket();  exit(0); }    void receive\_from\_clients(int \*clients, fd\_set \*fd\_readset) {  char buf[BUF\_SIZE];  int bytes;   for (int i = 0; i < MAX\_CLIENTS\_NUMB; i++)  {  // FD\_ISSET проверяет, является ли описатель частью набора   if (FD\_ISSET(clients[i], fd\_readset))  {  // Поступили данные от клиента, читаем их  bytes = recv(clients[i], buf, BUF\_SIZE, 0);  if (bytes <= 0)  {  // Соединение разорвано, удаляем сокет из множества  // (а если клиента завершать просто так, не закрывая его сокет, то мы никуда сюда не попадем)  close(clients[i]);  clients[i] = 0;  }  else if (bytes > 0)  {  buf[bytes] = 0;  printf("Server recieved from client %d: %s\n", i, buf);   memset(buf, 0, BUF\_SIZE);  sprintf(buf, "Message recieved. You are number %d\n", i);  send(clients[i], buf, sizeof(buf), 0);  }  }  } }  int main(void) {  struct sockaddr\_in serv\_addr;  fd\_set fd\_readset;  int clients[MAX\_CLIENTS\_NUMB] = {0};    // AF\_INET - семейство сетевых сокетов  // SOCK\_STREAM - для надежного упорядоченного полнодуплексного логического соединения  // Потоковый сокет должен быть в состоянии соединения перед тем, как из него можно будет отсылать данные или принимать их.  sock\_fd = socket(AF\_INET, SOCK\_STREAM, 0);  if (sock\_fd == -1)  {  printf("socket() failed\n");  return EXIT\_FAILURE;  }    signal(SIGTSTP, sigtstp\_handler); //изменение обработчика сигнала    /\*  По умолчанию функция socket() создает блокирующий сокет (на время выполнения операции с таким сокетом ваша программа блокируется)    Чтобы сделать его не-блокирующим (и использовать select), надо использовать функцию fcntl(2) с флагом O\_NONBLOCK.  Теперь любой вызов функции read() для сокета sock\_fd будет возвращать управление сразу же.   Если на входе сокета нет данных для чтения, функция read() вернет значение EAGAIN.   \*/  fcntl(sock\_fd, F\_SETFL, O\_NONBLOCK);   serv\_addr.sin\_family = AF\_INET;  serv\_addr.sin\_addr.s\_addr = INADDR\_ANY; //сервер зарегистрируется на всех адресах той машины, на которой она выполняется.  serv\_addr.sin\_port = htons(PORT);   if (bind(sock\_fd, (struct sockaddr \*) &serv\_addr, sizeof(serv\_addr)) == -1)  {  printf("bind() failed\n");  del\_socket();  return EXIT\_FAILURE;  }   if (listen(sock\_fd, MAX\_CLIENTS\_NUMB) == -1)  {  printf("listen() failed\n");  del\_socket();  return EXIT\_FAILURE;  }     printf("Listening.\nPress Ctrl + Z to stop...\n");   while(1)  {  // initialize the descriptor set to the null set.   FD\_ZERO(&fd\_readset);  // FD\_SET - добавление дескриптора в набор  // слушающий сокет добавляется в набор дескрипторов, которые следует проверять на готовность к чтению  FD\_SET(sock\_fd, &fd\_readset);    int max\_fd = sock\_fd;  for (int i = 0; i < MAX\_CLIENTS\_NUMB; i++)  {  if (clients[i])  {  FD\_SET(clients[i], &fd\_readset);  }    if (max\_fd < clients[i])  {  max\_fd = clients[i];  }  }    /\*  select(int n, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout);  Параметры:  1 - количество проверяемых дескрипторов (на единицу больше самого большого номера описателей из всех наборов)   2, 3, 4 - наборы дескрипторов, которые следует проверять, соответственно, на готовность к чтению, записи и на наличие исключительных ситуаций.   5 - можно указать интервал времени, по прошествии которого она вернет управление в любом случае.   Возвращаемое значение:  значение больше нуля, которое соответствует числу готовых к операции дескрипторов.   А все дескрипторы, которые привели к "срабатыванию" функции, записываются в соответствующие множества.  ноль -- в случае истечения тайм-аута,   отрицательное значение при ошибке.    Сама функция select() - блокирующая, она возвращает управление, если хотя бы один из проверяемых сокетов готов к выполнению соответствующей операции.   \*/  // Ждём события в одном из сокетов  if (select(max\_fd + 1, &fd\_readset, NULL, NULL, NULL) <= 0)  {  printf("select() failed\n");  del\_socket();  return EXIT\_FAILURE;  }   // Определяем тип события и выполняем соответствующие действия  // FD\_ISSET проверяет, является ли описатель частью набора  if (FD\_ISSET(sock\_fd, &fd\_readset))  {  // Поступил новый запрос на соединение (sock\_fd - слушающий сокет)  // блокируется, пока не появится "свисток" connect (NULL - адрес и длина адреса запросившего, неважны)  int new\_sock = accept(sock\_fd, NULL, NULL);   if (new\_sock == -1)  {  printf("accept() failed\n");  del\_socket();  return EXIT\_FAILURE;  }   fcntl(new\_sock, F\_SETFL, O\_NONBLOCK);   int place\_not\_found = 1;  for (int i = 0; i < MAX\_CLIENTS\_NUMB && place\_not\_found; i++)  {  if (!clients[i])  {  clients[i] = new\_sock;  printf("New connection %d\n", i);  place\_not\_found = 0;  }  }  }    receive\_from\_clients(clients, &fd\_readset);  }  return EXIT\_SUCCESS; } |
| --- |

## Клиент

| #include <stdio.h> #include <stdlib.h> #include <errno.h> #include <strings.h> #include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <netdb.h> #include <unistd.h> #include <time.h> #include <string.h> #include <signal.h>  #define BUF\_SIZE 256 #define PORT 3425 #define SOCK\_ADDR "localhost"   int sock\_fd;   void sigtstp\_handler(int signum) {  printf("\nCatch SIGTSTP\n");  if (close(sock\_fd) == -1) //закрытие сокета  {  printf("close() failed\n");  return;  }  exit(0); }   int main(void) {  srand(time(NULL));   struct sockaddr\_in serv\_addr;  struct hostent \*host;  char message[BUF\_SIZE];      sock\_fd = socket(AF\_INET, SOCK\_STREAM, 0);  if (sock\_fd < 0)  {  printf("socket() failed\n");  return EXIT\_FAILURE;  }    signal(SIGTSTP, sigtstp\_handler); //изменение обработчика сигнала    //преобразование доменного имени сервера в его сетевой адрес.  host = gethostbyname(SOCK\_ADDR);  if (!host)  {  printf("gethostbyname() failed\n");  return EXIT\_FAILURE;  }   serv\_addr.sin\_family = AF\_INET; // Семейство адресов  //главное отличие sockaddr\_in от sockaddr\_un - наличие параметра sin\_port, предназначенного для хранения значения порта.   serv\_addr.sin\_port = htons(PORT); // Номер порта  serv\_addr.sin\_addr = \*((struct in\_addr\*) host->h\_addr\_list[0]); // IP-адрес   // Для установления активного соединения по переданному адресу.   if (connect(sock\_fd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) == -1)  {  printf("connect() failed\n");  return EXIT\_FAILURE;  }    while(1)  {  memset(message, 0, BUF\_SIZE);  sprintf(message, "pid = %d\n", getpid());   if (send(sock\_fd, message, sizeof(message), 0) == -1)  {  printf("send() failed\n");  return EXIT\_FAILURE;  }    memset(message, 0, BUF\_SIZE);  int bytes = recv(sock\_fd, message, BUF\_SIZE, 0);   if (bytes > 0)  {  message[bytes] = 0;  printf("Client recieved from server: %s\n", message);  }   sleep(1 + rand() % 3);  }   close(sock\_fd);   return EXIT\_SUCCESS; } |
| --- |