# Практическая работа №3.

1. Программа myRPC-client реализует функционал удалённого вызова команд и получения соответствующих результатов от сервера. Пользователь задаёт параметры командной строки, такие как команда для выполнения, IP-адрес сервера, порт и тип соединения (потоковый или датаграммный сокет). После этого myRPC-client устанавливает соединение с сервером, отправляет команду, идентифицируя также текущего пользователя, и ожидает получения ответа. Поддерживаются два типа подключения: потоковое (TCP) и датаграммное (UDP). В зависимости от выбранного типа соединения, программа либо инициирует устойчивое соединение с сервером (TCP), либо использует одноразовые сообщения (UDP). Потоковые сокеты (TCP) обеспечивают надёжность передачи данных и контроль целостности, в то время как датаграммные сокеты (UDP) обладают высокой производительностью и минимальной задержкой, что делает их подходящими для менее критичных по надежности операций. При успешном подключении и отправке команды сервер выполняет её и возвращает результат выполнения, который затем отображается пользователю.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <getopt.h>  #include <pwd.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <sys/socket.h>  #include "libmysyslog.h" // Подключение библиотеки логирования  #define BUFFER\_SIZE 1024  void print\_help() {  printf("Usage: myRPC-client [OPTIONS]\n");  printf("Options:\n");  printf(" -c, --command \"bash\_command\" Command to execute\n");  printf(" -h, --host \"ip\_addr\" Server IP address\n");  printf(" -p, --port PORT Server port\n");  printf(" -s, --stream Use stream socket\n");  printf(" -d, --dgram Use datagram socket\n");  printf(" --help Display this help and exit\n");  }  int main(int argc, char \*argv[]) {  char \*command = NULL;  char \*server\_ip = NULL;  int port = 0;  int use\_stream = 1; // Default to stream socket  int opt;  static struct option long\_options[] = {  {"command", required\_argument, 0, 'c'},  {"host", required\_argument, 0, 'h'},  {"port", required\_argument, 0, 'p'},  {"stream", no\_argument, 0, 's'},  {"dgram", no\_argument, 0, 'd'},  {"help", no\_argument, 0, 0 },  {0, 0, 0, 0}  };  int option\_index = 0;  while ((opt = getopt\_long(argc, argv, "c:h:p:sd", long\_options, &option\_index)) != -1) {  switch (opt) {  case 'c':  command = optarg;  break;  case 'h':  server\_ip = optarg;  break;  case 'p':  port = atoi(optarg);  break;  case 's':  use\_stream = 1;  break;  case 'd':  use\_stream = 0;  break;  case 0:  print\_help();  return 0;  default:  print\_help();  return 1;  }  }  if (!command || !server\_ip || !port) {  fprintf(stderr, "Error: Missing required arguments\n");  print\_help();  return 1;  }  // Получение имени пользователя  struct passwd \*pw = getpwuid(getuid());  char \*username = pw->pw\_name;  // Подготовка запроса  char request[BUFFER\_SIZE];  snprintf(request, BUFFER\_SIZE, "%s: %s", username, command);  // Логирование действия  mysyslog("Connecting to the server...", INFO, 0, 0, "/var/log/myrpc.log");  // Создание сокета  int sockfd;  if (use\_stream) {  sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  } else {  sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  }  if (sockfd < 0) {  mysyslog("Socket creation failed", ERROR, 0, 0, "/var/log/myrpc.log");  perror("Socket creation failed");  return 1;  }  // Настройка адреса сервера  struct sockaddr\_in servaddr;  memset(&servaddr, 0, sizeof(servaddr));  servaddr.sin\_family = AF\_INET;  servaddr.sin\_port = htons(port);  inet\_pton(AF\_INET, server\_ip, &servaddr.sin\_addr);  if (use\_stream) {  // Подключение к серверу  if (connect(sockfd, (struct sockaddr\*)&servaddr, sizeof(servaddr)) < 0) {  mysyslog("Connection failed", ERROR, 0, 0, "/var/log/myrpc.log");  perror("Connection failed");  close(sockfd);  return 1;  }  // Логирование успешного подключения  mysyslog("Successfully connected to server", INFO, 0, 0, "/var/log/myrpc.log");  // Отправка запроса  send(sockfd, request, strlen(request), 0);  // Получение ответа  char response[BUFFER\_SIZE];  int n = recv(sockfd, response, BUFFER\_SIZE, 0);  response[n] = '\0';  printf("Server response: %s\n", response);  // Логирование результата  mysyslog("Received server response", INFO, 0, 0, "/var/log/myrpc.log");  } else {  // Отправка запроса  sendto(sockfd, request, strlen(request), 0, (struct sockaddr\*)&servaddr, sizeof(servaddr));  // Получение ответа  char response[BUFFER\_SIZE];  socklen\_t len = sizeof(servaddr);  int n = recvfrom(sockfd, response, BUFFER\_SIZE, 0, (struct sockaddr\*)&servaddr, &len);  response[n] = '\0';  printf("Server response: %s\n", response);  // Логирование результата  mysyslog("Received server response", INFO, 0, 0, "/var/log/myrpc.log");  }  close(sockfd);  return 0;  } |

Листинг 1. myrpc-client

2. myRPC-server представляет собой демон, который запускается на серверной стороне и ожидает подключения от клиентов. Сервер считывает настройки из конфигурационного файла, такие как номер порта и тип сокета (потоковый или датаграммный). После установления соединения с клиентом сервер принимает команды и осуществляет проверку прав доступа пользователя, используя заранее заданный файл с перечнем разрешённых пользователей. Это позволяет ограничить круг лиц, имеющих доступ к удалённому выполнению. После успешной проверки сервер выполняет команду, записывает результат выполнения во временные файлы (stdout и stderr) для дальнейшего анализа и обработки, а затем возвращает результаты клиенту.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <signal.h>  #include <pwd.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <sys/socket.h>  #include "config\_parser.h"  #include "libmysyslog.h"  #define BUFFER\_SIZE 1024  volatile sig\_atomic\_t stop;  void handle\_signal(int sig) {  stop = 1;  }  int user\_allowed(const char \*username) {  FILE \*file = fopen("/etc/myRPC/users.conf", "r");  if (!file) {  mysyslog("Failed to open users.conf", ERROR, 0, 0, "/var/log/myrpc.log");  perror("Failed to open users.conf");  return 0;  }  char line[256];  int allowed = 0;  while (fgets(line, sizeof(line), file)) {  line[strcspn(line, "\n")] = '\0';  if (line[0] == '#' || strlen(line) == 0)  continue;  if (strcmp(line, username) == 0) {  allowed = 1;  break;  }  }  fclose(file);  return allowed;  }  void execute\_command(const char \*command, char \*stdout\_file, char \*stderr\_file) {  char cmd[BUFFER\_SIZE];  snprintf(cmd, BUFFER\_SIZE, "%s >%s 2>%s", command, stdout\_file, stderr\_file);  system(cmd);  }  int main() {  signal(SIGINT, handle\_signal);  signal(SIGTERM, handle\_signal);  Config config = parse\_config("/etc/myRPC/myRPC.conf");  int port = config.port;  int use\_stream = strcmp(config.socket\_type, "stream") == 0;  mysyslog("Server starting...", INFO, 0, 0, "/var/log/myrpc.log");  // Создание сокета  int sockfd;  if (use\_stream) {  sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  } else {  sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  }  if (sockfd < 0) {  mysyslog("Socket creation failed", ERROR, 0, 0, "/var/log/myrpc.log");  perror("Socket creation failed");  return 1;  }  struct sockaddr\_in servaddr, cliaddr;  socklen\_t len;  memset(&servaddr, 0, sizeof(servaddr));  servaddr.sin\_family = AF\_INET;  servaddr.sin\_addr.s\_addr = INADDR\_ANY;  servaddr.sin\_port = htons(port);  if (bind(sockfd, (struct sockaddr\*)&servaddr, sizeof(servaddr)) < 0) {  mysyslog("Bind failed", ERROR, 0, 0, "/var/log/myrpc.log");  perror("Bind failed");  close(sockfd);  return 1;  }  if (use\_stream) {  listen(sockfd, 5);  mysyslog("Server listening (stream)", INFO, 0, 0, "/var/log/myrpc.log");  } else {  mysyslog("Server listening (datagram)", INFO, 0, 0, "/var/log/myrpc.log");  }  while (!stop) {  char buffer[BUFFER\_SIZE];  int n;  if (use\_stream) {  len = sizeof(cliaddr);  int connfd = accept(sockfd, (struct sockaddr\*)&cliaddr, &len);  if (connfd < 0) {  mysyslog("Accept failed", ERROR, 0, 0, "/var/log/myrpc.log");  perror("Accept failed");  continue;  }  n = recv(connfd, buffer, BUFFER\_SIZE, 0);  if (n <= 0) {  close(connfd);  continue;  }  buffer[n] = '\0';  mysyslog("Received request", INFO, 0, 0, "/var/log/myrpc.log");  char \*username = strtok(buffer, ":");  char \*command = strtok(NULL, "");  if (command) {  while (\*command == ' ')  command++;  }  char response[BUFFER\_SIZE];  if (user\_allowed(username)) {  mysyslog("User allowed", INFO, 0, 0, "/var/log/myrpc.log");  char stdout\_file[] = "/tmp/myRPC\_XXXXXX.stdout";  char stderr\_file[] = "/tmp/myRPC\_XXXXXX.stderr";  mkstemp(stdout\_file);  mkstemp(stderr\_file);  execute\_command(command, stdout\_file, stderr\_file);  // Чтение вывода команды  FILE \*f = fopen(stdout\_file, "r");  if (f) {  size\_t read\_bytes = fread(response, 1, BUFFER\_SIZE, f);  response[read\_bytes] = '\0';  fclose(f);  mysyslog("Command executed successfully", INFO, 0, 0, "/var/log/myrpc.log");  } else {  strcpy(response, "Error reading stdout file");  mysyslog("Error reading stdout file", ERROR, 0, 0, "/var/log/myrpc.log");  }  remove(stdout\_file);  remove(stderr\_file);  } else {  snprintf(response, BUFFER\_SIZE, "1: User '%s' is not allowed", username);  mysyslog("User not allowed", WARN, 0, 0, "/var/log/myrpc.log");  }  send(connfd, response, strlen(response), 0);  close(connfd);  } else {  len = sizeof(cliaddr);  n = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0, (struct sockaddr\*)&cliaddr, &len);  if (n <= 0) {  continue;  }  buffer[n] = '\0';  mysyslog("Received request", INFO, 0, 0, "/var/log/myrpc.log");  // Обработка запроса  char \*username = strtok(buffer, ":");  char \*command = strtok(NULL, "");  if (command) {  while (\*command == ' ')  command++;  }  char response[BUFFER\_SIZE];  if (user\_allowed(username)) {  mysyslog("User allowed", INFO, 0, 0, "/var/log/myrpc.log");  char stdout\_file[] = "/tmp/myRPC\_XXXXXX.stdout";  char stderr\_file[] = "/tmp/myRPC\_XXXXXX.stderr";  mkstemp(stdout\_file);  mkstemp(stderr\_file);  execute\_command(command, stdout\_file, stderr\_file);  // Чтение вывода команды  FILE \*f = fopen(stdout\_file, "r");  if (f) {  size\_t read\_bytes = fread(response, 1, BUFFER\_SIZE, f);  response[read\_bytes] = '\0';  fclose(f);  mysyslog("Command executed successfully", INFO, 0, 0, "/var/log/myrpc.log");  } else {  strcpy(response, "Error reading stdout file");  mysyslog("Error reading stdout file", ERROR, 0, 0, "/var/log/myrpc.log");  }  remove(stdout\_file);  remove(stderr\_file);  } else {  snprintf(response, BUFFER\_SIZE, "1: User '%s' is not allowed", username);  mysyslog("User not allowed", WARN, 0, 0, "/var/log/myrpc.log");  }  sendto(sockfd, response, strlen(response), 0, (struct sockaddr\*)&cliaddr, len);  }  }  close(sockfd);  mysyslog("Server stopped", INFO, 0, 0, "/var/log/myrpc.log");  return 0;  } |

Листинг 2. myRPC-server

3. Для автоматизации сборки компонентов myRPC-client и myRPC-server были разработаны соответствующие Makefile, которые обеспечивают компиляцию, сборку и создание бинарных файлов. Makefile упрощают процесс разработки и позволяют легко управлять зависимостями и процессом сборки. Оба компонента имеют свои Makefile, которые выполняют следующие задачи:

Makefile для myRPC-client содержит правила для компиляции исходного кода клиента в исполняемый файл. Основные этапы сборки включают:

* Компиляция исходного файла client.c с помощью компилятора GCC.
* Генерация исполняемого файла myRPC-client.
* Очистка артефактов сборки, таких как объектные файлы.

|  |
| --- |
| # Makefile for myRPC-client  CC := gcc  CFLAGS := -Wall -g -I../libmysyslog  LDFLAGS := -L../libmysyslog -lmysyslog  SRC\_DIR := src  OBJ\_DIR := obj  BIN\_DIR := bin  TARGET := $(BIN\_DIR)/myrpc-client  SRC := $(wildcard $(SRC\_DIR)/\*.c)  OBJ := $(patsubst $(SRC\_DIR)/%.c,$(OBJ\_DIR)/%.o,$(SRC))  .PHONY: all clean deb  all: $(TARGET)  $(TARGET): $(OBJ)  @mkdir -p $(BIN\_DIR)  $(CC) -o $@ $^ $(LDFLAGS)  $(OBJ\_DIR)/%.o: $(SRC\_DIR)/%.c  @mkdir -p $(OBJ\_DIR)  $(CC) $(CFLAGS) -c $< -o $@  clean:  rm -rf $(OBJ\_DIR) $(BIN\_DIR) \*.deb  deb: clean all  mkdir -p deb\_pkg/usr/bin  mkdir -p deb\_pkg/DEBIAN  install -m 755 $(TARGET) deb\_pkg/usr/bin/  echo "Package: myrpc-client" > deb\_pkg/DEBIAN/control  echo "Version: 1.0-1" >> deb\_pkg/DEBIAN/control  echo "Section: net" >> deb\_pkg/DEBIAN/control  echo "Priority: optional" >> deb\_pkg/DEBIAN/control  echo "Architecture: amd64" >> deb\_pkg/DEBIAN/control  echo "Maintainer: Your Name <you@example.com>" >> deb\_pkg/DEBIAN/control  echo "Description: myRPC Client Application" >> deb\_pkg/DEBIAN/control  # Build the package  fakeroot dpkg-deb --build deb\_pkg ../deb/myrpc-client\_1.0-1\_amd64.deb  # Clean up temporary files  rm -rf deb\_pkg |

Листинг 3. myRPC-client Makefile

Makefile для myRPC-server выполняет аналогичные задачи, но также включает компиляцию вспомогательных файлов, таких как config\_parser.c, необходимого для чтения конфигурационного файла сервера. Основные задачи Makefile для myRPC-server:

* Компиляция исходного кода server.c и вспомогательных файлов.
* Линковка объектных файлов для создания исполняемого файла myRPC-server.
* Очистка артефактов сборки.

|  |
| --- |
| # Makefile for myRPC-server  CC := gcc  CFLAGS := -Wall -g -I../libmysyslog  LDFLAGS := -L../libmysyslog -lmysyslog  SRC\_DIR := src  OBJ\_DIR := obj  BIN\_DIR := bin  TARGET := $(BIN\_DIR)/myrpc-server  SRC := $(wildcard $(SRC\_DIR)/\*.c)  OBJ := $(patsubst $(SRC\_DIR)/%.c,$(OBJ\_DIR)/%.o,$(SRC))  .PHONY: all clean deb  all: $(TARGET)  $(TARGET): $(OBJ)  @mkdir -p $(BIN\_DIR)  $(CC) -o $@ $^ $(LDFLAGS)  $(OBJ\_DIR)/%.o: $(SRC\_DIR)/%.c  @mkdir -p $(OBJ\_DIR)  $(CC) $(CFLAGS) -c $< -o $@  clean:  rm -rf $(OBJ\_DIR) $(BIN\_DIR) \*.deb  deb: clean all  mkdir -p deb\_pkg/usr/bin  mkdir -p deb\_pkg/DEBIAN  install -m 755 $(TARGET) deb\_pkg/usr/bin/  echo "Package: myrpc-server" > deb\_pkg/DEBIAN/control  echo "Version: 1.0-1" >> deb\_pkg/DEBIAN/control  echo "Section: net" >> deb\_pkg/DEBIAN/control  echo "Priority: optional" >> deb\_pkg/DEBIAN/control  echo "Architecture: amd64" >> deb\_pkg/DEBIAN/control  echo "Maintainer: Your Name <you@example.com>" >> deb\_pkg/DEBIAN/control  echo "Description: myRPC Server Application" >> deb\_pkg/DEBIAN/control  # Build the package  fakeroot dpkg-deb --build deb\_pkg ../deb/myrpc-server\_1.0-1\_amd64.deb  # Clean up temporary files  rm -rf deb\_pkg |

Листинг 4. myRPC-server Makefile

Makefile для сборки всего проекта сразу:

|  |
| --- |
| # Top-level Makefile  SUBDIRS := libmysyslog myRPC-server myRPC-client  .PHONY: all clean deb  all:  @for dir in $(SUBDIRS); do \  $(MAKE) -C $$dir || exit 1; \  done  clean:  @for dir in $(SUBDIRS); do \  $(MAKE) -C $$dir clean; \  done  deb: all  @mkdir -p deb  $(MAKE) -C libmysyslog deb  $(MAKE) -C myRPC-server deb  $(MAKE) -C myRPC-client deb |

Листинг 5. myRPC общий Makefile

4. Создадим README файл с описанием использования программ

|  |
| --- |
| # Руководство по сборке и использованию myRPC  ## Описание  Проект `myRPC` представляет собой механизм удалённого вызова команд, состоящий из клиентского и серверного приложений. Включает следующие компоненты:  - \*\*myRPC-client\*\* — консольное приложение для отправки команд на сервер.  - \*\*myRPC-server\*\* — серверное приложение (демон), принимающее команды и выполняющее их.  Проект разработан для взаимодействия с удалённым сервером с использованием сокетов, поддерживая как потоковые (TCP), так и датаграммные (UDP) соединения.  ## Системные требования  - Компилятор GCC  - Утилита `make`  - Поддержка разработки deb-пакетов (`dpkg-deb`)  ## Сборка проекта  Для сборки `myRPC-client` и `myRPC-server` используйте Makefile, который автоматизирует процесс компиляции и создания deb-пакетов.  ### Сборка myRPC-client  1. Перейдите в директорию с исходным кодом `myRPC-client`.  2. Выполните команду:  ```sh  make  ```  Это создаст исполняемый файл `myRPC-client` в текущей директории.  3. Для создания deb-пакета выполните команду:  ```sh  make deb  ```  Это создаст deb-пакет в директории `deb/myrpc-client/`.  4. Чтобы очистить директорию от скомпилированных файлов, используйте команду:  ```sh  make clean  ```  ### Сборка myRPC-server  1. Перейдите в директорию с исходным кодом `myRPC-server`.  2. Выполните команду:  ```sh  make  ```  Это создаст исполняемый файл `myRPC-server` в текущей директории.  3. Для создания deb-пакета выполните команду:  ```sh  make deb  ```  Это создаст deb-пакет в директории `deb/myrpc-server/`.  4. Чтобы очистить директорию от скомпилированных файлов, используйте команду:  ```sh  make clean  ```  ## Установка и настройка  После сборки и создания deb-пакетов можно установить клиент и сервер с помощью команды `dpkg`:  ```sh  sudo dpkg -i deb/myrpc-client.deb  sudo dpkg -i deb/myrpc-server.deb  ```  ### Настройка сервера  1. Создайте конфигурационный файл `/etc/myRPC/myRPC.conf` для настройки параметров сервера. Пример файла:  ```conf  # Порт, на котором будет слушать сервер  port = 1234  # Тип сокета: stream или dgram  socket\_type = stream  ```  2. Создайте файл пользователей `/etc/myRPC/users.conf`, в котором указаны разрешённые пользователи:  ```conf  # Список разрешённых пользователей  user1  user2  ```  ## Использование  ### myRPC-client  Клиентское приложение `myRPC-client` позволяет отправлять команды на сервер для выполнения. Примеры использования:  1. Отправка команды с использованием потокового сокета (TCP):  ```sh  myRPC-client -c "ls -la" -h "192.168.0.1" -p 1234 -s  ```  Здесь:  - `-c` — команда для выполнения на сервере.  - `-h` — IP-адрес сервера.  - `-p` — порт, на котором работает сервер.  - `-s` — использование потокового сокета.  2. Отправка команды с использованием датаграммного сокета (UDP):  ```sh  myRPC-client -c "date" -h "192.168.0.1" -p 1234 -d  ```  Здесь флаг `-d` указывает на использование датаграммного сокета.  ### myRPC-server  Серверное приложение `myRPC-server` запускается на сервере и ожидает подключения от клиентов. Для запуска сервера выполните команду:  ```sh  sudo myRPC-server  ```  ## Логирование и отладка  Сервер `myRPC-server` использует библиотеку `libmysyslog` для ведения журнала событий. Все события записываются в файл `/var/log/myrpc.log`. Логи содержат информацию о подключениях клиентов, выполнении команд и возможных ошибках. |

Листинг 6. README файл репозитория

5. Демонстрация установки и использования программ используя утилиту apt на машинах под управлением Ubuntu.

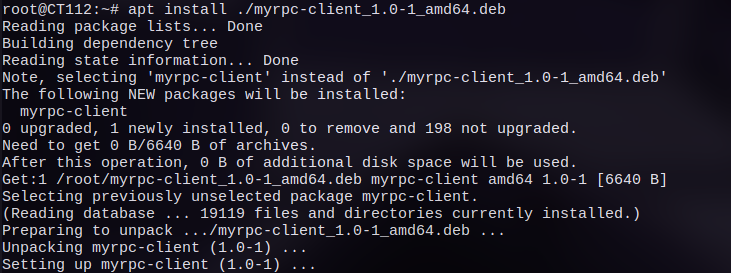


Рис. 2. Установка сервера.

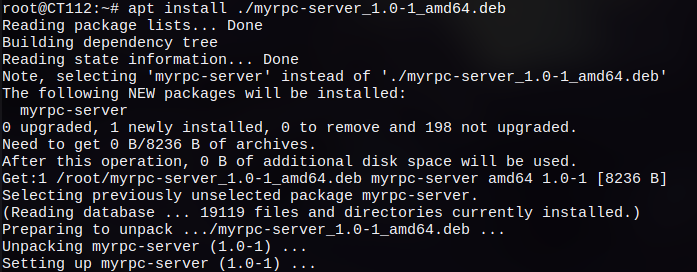


Рис. 2. Установка сервера.

Отправим запрос от пользователя neko и от пользователя root используя sudo.

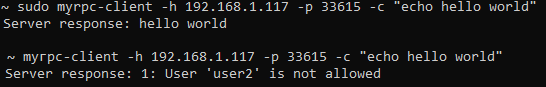


Рис. 3. Работа клиента.

Просмотрев содержимое файла /var/log/myrpc.log удостоверимся в работе системы логирования на клиенте.



Рис. 4. Логи клиента.

Просмотрев содержимое файла /var/log/myrpc.log удостоверимся в работе системы логирования на клиенте.



Рис. 5. Логи сервера.