

**Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)
Кафедра ТСиВС**

**Лабораторная работа №3 по дисциплине Мобильные системы новых
поколений на тему:
«Написание обработчика TCP трафика со стороны srsRAN4»**

Выполнил:
студент группы ИА-131
Нагорный С.Д.
Проверил:
Ахпашев Р.В.

Новосибирск 2024 г.

Теоретические сведения

ZeroMQ (также ØMQ, ZMQ, 0MQ) — высокопроизводительная асинхронная библиотека обмена сообщениями, ориентированная на использование в распределённых и параллельных вычислениях. Библиотека реализует очередь сообщений, которая может функционировать без выделенного брокера сообщений.

Обществом ZeroMQ определено как «сокеты на стероидах». Формально ZeroMQ определяются как библиотека сообщений, которая помогает разработчикам создавать распределённые и параллельные приложения. Первое, что мы должны узнать о ZeroMQ это то, что она не является традиционной системой очередей сообщений, таких как ActiveMQ, WebSphereMQ или RabbitMQ. ZeroMQ отличается. Она даёт нам инструменты для создания собственной системы очередей сообщений. Это библиотека.

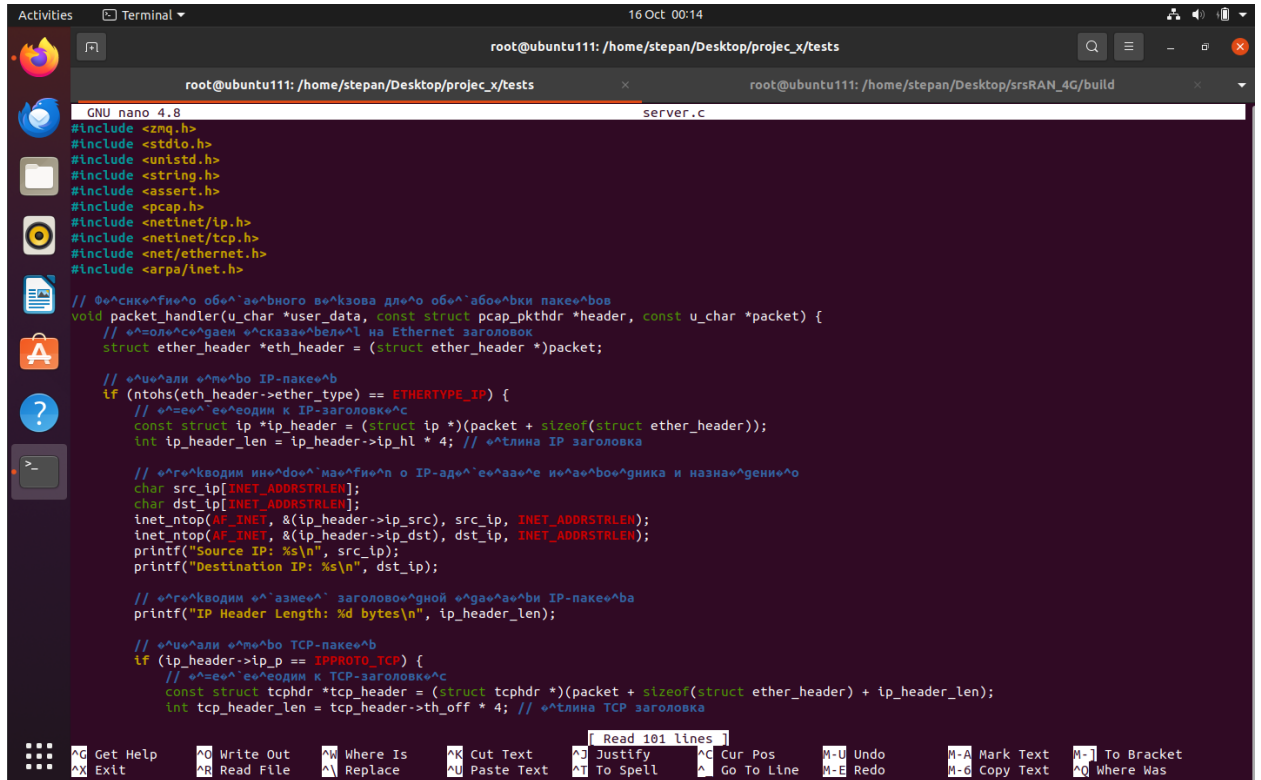
Она работает на различных архитектурах от ARM до Itanium и поддерживается более чем в 20 языках программирования.

Задание

1. Установить и подключить к проекту (обновить CMakeFile) библиотеку libpcap-dev. (<https://github.com/the-tcpdump-group/libpcap>);
 1. За основу сервера берём код из Практики №1.
2. Написать приложение-сервер с обработчиком TCP пакетов (<https://www.devdungeon.com/content/using-libpcap-c>); Обработчик пакетов должен:
 1. Выводить информацию о IP-адресах (dest, source);
 2. Выводить информацию о размере заголовочной части IP-пакета;
 3. Вывести побайтово значения Payload поля.
3. Обновить github репозиторий с сервером.
4. Оформить отчёт по выполнению каждого пункта.

Результат выполненной работы:

Заходим в su права, запускаем измененный server.c, далее запускаем srsRAN.



```
GNU nano 4.8
#include <zmq.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <assert.h>
#include <pcap.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/ether.h>
#include <arpa/inet.h>

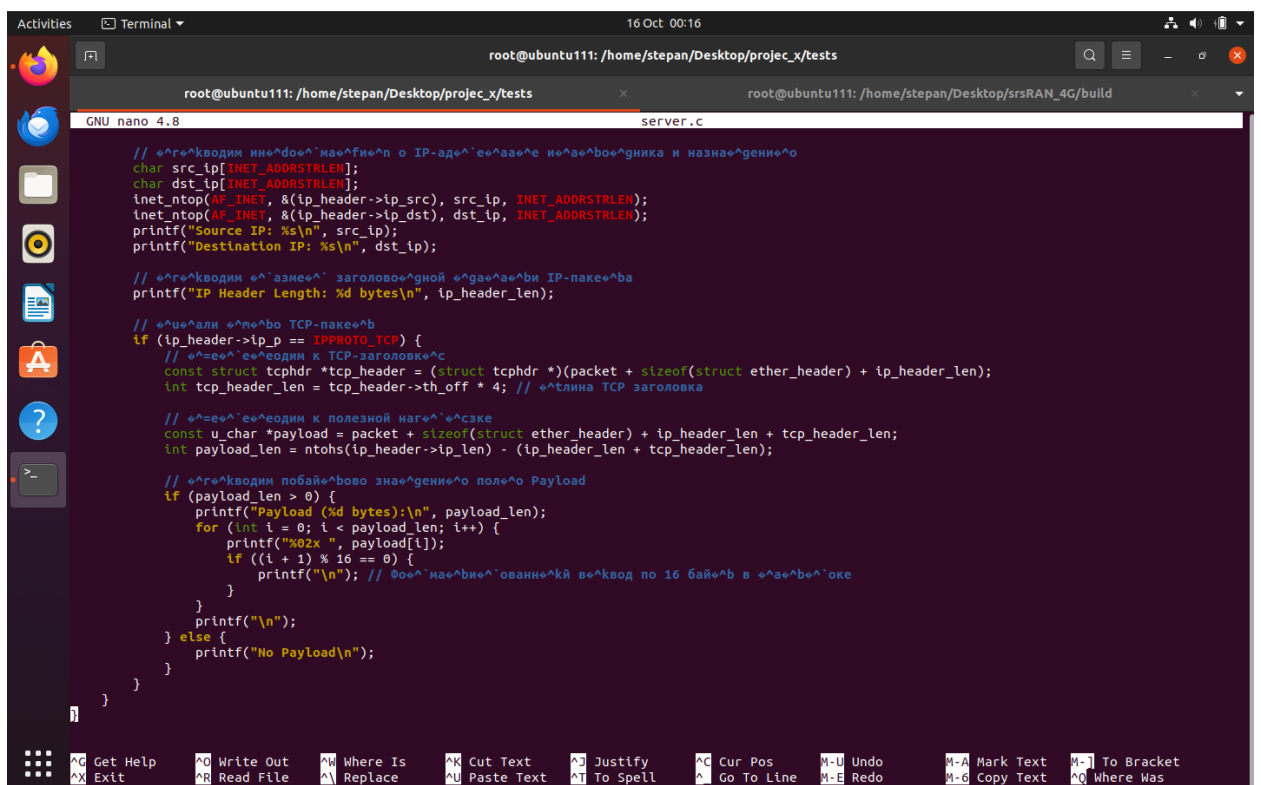
// Функция для обработки пакета
void packet_handler(u_char *user_data, const struct pcap_pkthdr *header, const u_char *packet) {
    // Проверяем, является ли пакет Ethernet-пакетом
    struct ether_header *eth_header = (struct ether_header *)packet;

    // Проверяем, является ли пакет IP-пакетом
    if (ntohs(eth_header->ether_type) == ETHERTYPE_IP) {
        // Проверяем, является ли пакет IP-пакетом
        const struct ip *ip_header = (struct ip *) (packet + sizeof(struct ether_header));
        int ip_header_len = ip_header->ihl * 4; // Длина IP заголовка

        // Проверяем, является ли пакет IP-пакетом
        char src_ip[INET_ADDRSTRLEN];
        char dst_ip[INET_ADDRSTRLEN];
        inet_ntop(AF_INET, &(ip_header->ip_src), src_ip, INET_ADDRSTRLEN);
        inet_ntop(AF_INET, &(ip_header->ip_dst), dst_ip, INET_ADDRSTRLEN);
        printf("Source IP: %s\n", src_ip);
        printf("Destination IP: %s\n", dst_ip);

        // Проверяем, является ли пакет IP-пакетом
        printf("IP Header Length: %d bytes\n", ip_header_len);

        // Проверяем, является ли пакет TCP-пакетом
        if (ip_header->ip_p == IPPROTO_TCP) {
            // Проверяем, является ли пакет TCP-пакетом
            const struct tcp_hdr *tcp_header = (struct tcp_hdr *) (packet + sizeof(struct ether_header) + ip_header_len);
            int tcp_header_len = tcp_header->th_off * 4; // Длина TCP заголовка
```

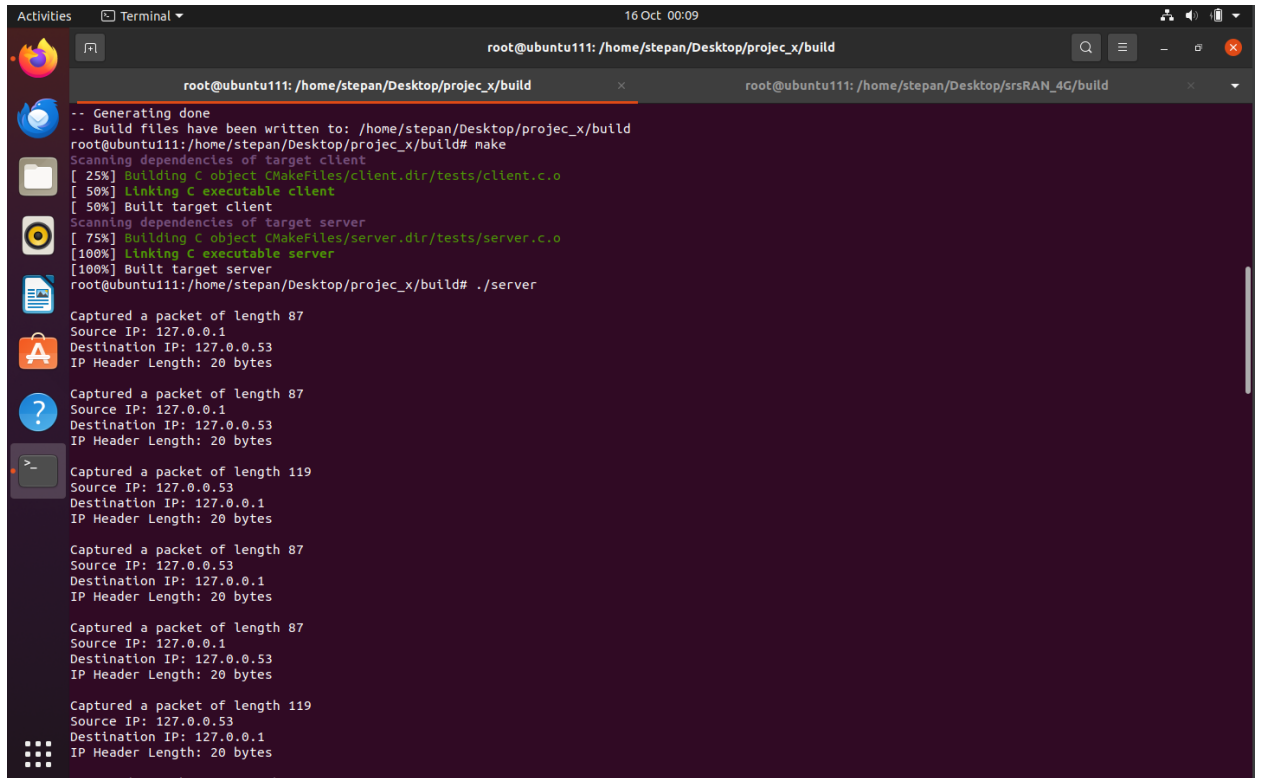


```
        // Проверяем, является ли пакет TCP-пакетом
        if (ip_header->ip_p == IPPROTO_TCP) {
            // Проверяем, является ли пакет TCP-пакетом
            const struct tcp_hdr *tcp_header = (struct tcp_hdr *) (packet + sizeof(struct ether_header) + ip_header_len);
            int tcp_header_len = tcp_header->th_off * 4; // Длина TCP заголовка

            // Проверяем, является ли пакет TCP-пакетом
            const u_char *payload = packet + sizeof(struct ether_header) + ip_header_len + tcp_header_len;
            int payload_len = ntohs(ip_header->ip_len) - (ip_header_len + tcp_header_len);

            // Проверяем, является ли пакет TCP-пакетом
            if (payload_len > 0) {
                printf("Payload (%d bytes):\n", payload_len);
                for (int i = 0; i < payload_len; i++) {
                    printf("%02x ", payload[i]);
                    if ((i + 1) % 16 == 0) {
                        printf("\n"); // Форматированный вывод по 16 байт в строке
                    }
                }
                printf("\n");
            } else {
                printf("No Payload\n");
            }
        }
    }
}
```


Результаты:



```
root@ubuntu111: /home/stepan/Desktop/projec_x/build
-- Generating done
-- Build files have been written to: /home/stepan/Desktop/projec_x/build
root@ubuntu111: /home/stepan/Desktop/projec_x/build# make
Scanning dependencies of target client
[ 25%] Building C object CMakeFiles/client.dir/tests/client.c.o
[ 50%] Linking C executable client
[ 50%] Built target client
Scanning dependencies of target server
[ 75%] Building C object CMakeFiles/server.dir/tests/server.c.o
[100%] Linking C executable server
[100%] Built target server
root@ubuntu111: /home/stepan/Desktop/projec_x/build# ./server

Captured a packet of length 87
Source IP: 127.0.0.1
Destination IP: 127.0.0.53
IP Header Length: 20 bytes

Captured a packet of length 87
Source IP: 127.0.0.1
Destination IP: 127.0.0.53
IP Header Length: 20 bytes

Captured a packet of length 119
Source IP: 127.0.0.53
Destination IP: 127.0.0.1
IP Header Length: 20 bytes

Captured a packet of length 87
Source IP: 127.0.0.53
Destination IP: 127.0.0.1
IP Header Length: 20 bytes

Captured a packet of length 87
Source IP: 127.0.0.1
Destination IP: 127.0.0.53
IP Header Length: 20 bytes

Captured a packet of length 119
Source IP: 127.0.0.53
Destination IP: 127.0.0.1
IP Header Length: 20 bytes
```

```

#include <zmq.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <assert.h>
#include <pcap.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <net/ethernet.h>
#include <arpa/inet.h>

void packet_handler(u_char *user_data, const struct pcap_pkthdr *header, const u_char *packet) {
    struct ether_header *eth_header = (struct ether_header *)packet;
    if (ntohs(eth_header->ether_type) == ETHERTYPE_IP) {
        const struct ip *ip_header = (struct ip *)(packet + sizeof(struct ether_header));
        int ip_header_len = ip_header->ip_hl * 4; // Длина IP заголовка

        char src_ip[INET_ADDRSTRLEN];
        char dst_ip[INET_ADDRSTRLEN];
        inet_ntop(AF_INET, &(ip_header->ip_src), src_ip, INET_ADDRSTRLEN);
        inet_ntop(AF_INET, &(ip_header->ip_dst), dst_ip, INET_ADDRSTRLEN);
        printf("Source IP: %s\n", src_ip);
        printf("Destination IP: %s\n", dst_ip);

        printf("IP Header Length: %d bytes\n", ip_header_len);

        if (ip_header->ip_p == IPPROTO_TCP) {
            const struct tcphdr *tcp_header = (struct tcphdr *)(packet + sizeof(struct ether_header) + ip_header_len);
            int tcp_header_len = tcp_header->th_off * 4; // Длина TCP заголовка

            const u_char *payload = packet + sizeof(struct ether_header) + ip_header_len + tcp_header_len;
            int payload_len = ntohs(ip_header->ip_len) - (ip_header_len + tcp_header_len);
            if (payload_len > 0) {
                printf("Payload (%d bytes):\n", payload_len);
                for (int i = 0; i < payload_len; i++) {
                    printf("%02x ", payload[i]);

```

```

        if ((i + 1) % 16 == 0) {
            printf("\n"); // Форматированный вывод по 16 байт в строке
        }
    }

    printf("\n");
} else {
    printf("No Payload\n");
}
}
}

}

int main (void) {
    void *context = zmq_ctx_new();

    void *responder = zmq_socket(context, ZMQ_REP);
    int rc = zmq_bind(responder, "tcp://*:5555");
    assert(rc == 0);

    char errbuf[PCAP_ERRBUF_SIZE];

    pcap_t *handle = pcap_open_live("lo", BUFSIZ, 1, 1000, errbuf); // Используем интерфейс lo для
    тестирования

    if (handle == NULL) {
        fprintf(stderr, "Couldn't open device: %s\n", errbuf);
        return 2;
    }

    while (1) {
        char buffer[10];

        zmq_recv(responder, buffer, 10, 0);
        printf("\n");

        struct pcap_pkthdr header;
        const u_char *packet = pcap_next(handle, &header);
        if (packet != NULL) {
            printf("Captured a packet of length %d\n", header.len);

            packet_handler(NULL, &header, packet); // Обработываем пакет
        }
    }
}

```

```

    sleep(1); // Имитируем работу
    zmq_send(responder, "World", 5, 0);
}
pcap_close(handle);
zmq_close(responder);
zmq_ctx_destroy(context);

return 0;
}

```

«server.c»

```

cmake_minimum_required(VERSION 3.10)

# Установите имя проекта
project(Client_Server)

set(CMAKE_C_STANDARD 99)

# Найдите библиотеки pcap
find_package(PkgConfig REQUIRED)
pkg_check_modules(PCAP REQUIRED libpcap)

pkg_check_modules(ZMQ REQUIRED libzmq)
pkg_check_modules(CZMQ REQUIRED libczmq)

# Создайте исполняемые файлы для сервера и клиента
add_executable(server tests/server.c)
add_executable(client tests/client.c)

# Свяжите библиотеки с исполняемыми файлами
target_link_libraries(server ${ZMQ_LIBRARIES} ${CZMQ_LIBRARIES} ${PCAP_LIBRARIES})
target_link_libraries(client ${ZMQ_LIBRARIES} ${CZMQ_LIBRARIES} ${PCAP_LIBRARIES})

# Включите директории заголовков
target_include_directories(server PUBLIC

```



```
    ${ZMQ_INCLUDE_DIRS}
    ${CZMQ_INCLUDE_DIRS}
    ${PCAP_INCLUDE_DIRS}
)

target_include_directories(client PUBLIC
    ${ZMQ_INCLUDE_DIRS}
    ${CZMQ_INCLUDE_DIRS}
    ${PCAP_INCLUDE_DIRS}
)
```

«CMakeList.txt»

QR-код на репозиторий



«commit “lab3”»