

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

Факультет безопасности информационных технологий

**Дисциплина: «Операционные системы»
ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9**

Выполнил:

Студент группы N3249

Чан Нгок Хуан

Проверил:

Савков Сергей Витальевич

Санкт-Петербург

2022г.

ЗАДАНИЕ

Лаб 9.

Одно из

1. Написать фильтр сетевых пакетов на основе `nfqqueue` и `iptables` и протестировать скорость работы
2. Протестировать работу сокетов `tcp` при различных настройках `setsockopt`

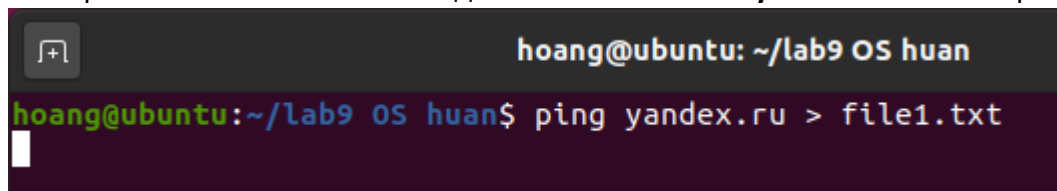
Сложный вариант (одно из)

1. Написать фильтр пакетов на основе интерфейса `netfilter`
2. Реализовать `rpc`-программу для `linux` с поддержкой аутентификации (`rpcinfo`, `rpcbind`)

I. Написать фильтр сетевых пакетов на основе nfqueue и iptables и протестировать скорость работы

- **Iptables** используется для установки, настройки и просмотра таблиц правил фильтрации IP-пакетов в ядре Linux.
- Соответственно в фильтре **iptables** все пакеты делятся на три аналогичные цепочки:
 - + **input** - обрабатывает входящие пакеты и подключения. Например, если какой-либо внешний пользователь пытается подключиться к вашему компьютеру по ssh или любой веб-сайт отправит вам свой контент по запросу браузера. Все эти пакеты попадут в эту цепочку;
 - + **forward** - эта цепочка применяется для проходящих соединений. Сюда попадают пакеты, которые отправлены на ваш компьютер, но не предназначены ему, они просто пересылаются по сети к своей цели. Как я уже говорил, такое наблюдается на маршрутизаторах или, например, если ваш компьютер раздает wifi;
 - + **output** - эта цепочка используется для исходящих пакетов и соединений. Сюда попадают пакеты, которые были созданы при попытке выполнить ping losst.ru или когда вы запускаете браузер и пытаетесь открыть любой сайт.
- Цель **NFQUEUE** используется почти так же, как цель **QUEUE**, и в основном является ее расширением. Цель **NFQUEUE** позволяет отправлять пакеты для отдельных и определенных очередей. Очередь идентифицируется 16-битным идентификатором.

- Тестировал возможность подключения хоста **yandex.ru** без фильтра



```
hoang@ubuntu: ~/lab9 OS huan
hoang@ubuntu:~/lab9 OS huan$ ping yandex.ru > file1.txt
```

- Программа фильтра сетевых пакетов на основе **nfqueue** и **iptables**

```
import netfilterqueue
import socket
import sys
from scapy.all import *
def process(pkt):
    data = pkt.get_payload()
    p = IP(data)
    p.ttl = 10
    del p.chksum
    pkt.set_verdict_modified(nfqueue.NF_ACCEPT, str(p), len(p))
nfqueue = netfilterqueue.NetfilterQueue()
nfqueue.bind(1, process)
try:
    nfqueue.run()
except:
    nfqueue.unbind()
    sys.exit(1)
```

```
hoang@ubuntu: ~/lab9 OS huan
hoang@ubuntu:~/lab9 OS huan$ gedit lab9.py
hoang@ubuntu:~/lab9 OS huan$ sudo iptables -A OUTPUT -p tcp -j NFQUEUE
[sudo] password for hoang:
hoang@ubuntu:~/lab9 OS huan$ sudo python2 lab9.py
```

Запускал фильтр и ещё раз подключить сервер **yandex.ru**

```
hoang@ubuntu: ~/lab9 OS huan
hoang@ubuntu:~/lab9 OS huan$ ping yandex.ru > file2.txt
```

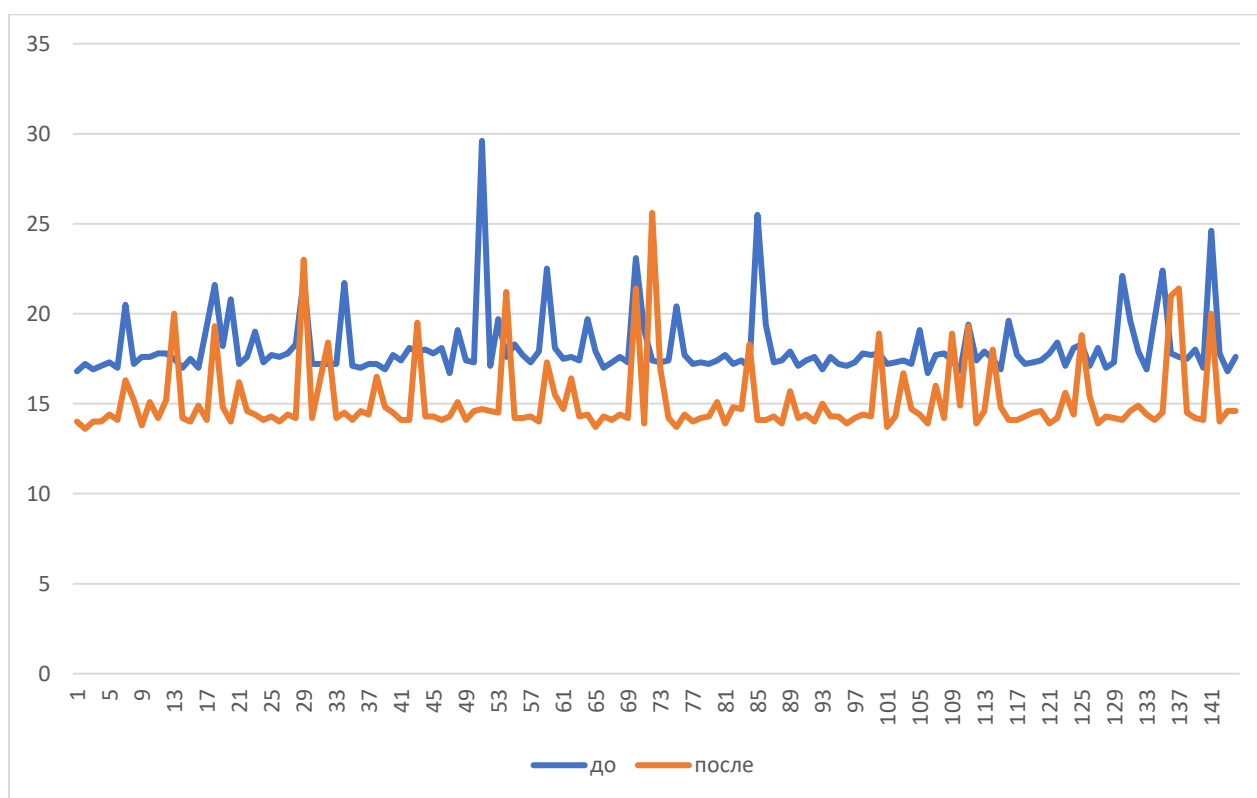


Таблица среднее значение time (Время, которое потребовалось посылке, чтобы добраться до пункта назначения и вернуться в исходное положение)

	NetFilter	Nofilter
Среднее значение (ms)	15.177	18.096

Из графика и таблицы, мы видим, что после фильтра время было более стабильным и быстрым, чем до фильтра

II. Выполнение задания (Сложный вариант) : Реализовать rpc-программу для linux с поддержкой аутентификации (rpcinfo, rpcbind)

Step 1 : Install rpcbind and check rpcinfo

```
tran@tran-virtual-machine:~$ sudo apt-get install rpcbind
[sudo] password for tran:
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package rpcbind
```

```
tran@tran-virtual-machine:~$ sudo apt install rpcbind
```

```
tran@tran-virtual-machine:~$ rpcinfo
  program version netid  address          service  owner
  100000    4    tcp6   :::0.111         portmapper superuser
  100000    3    tcp6   :::0.111         portmapper superuser
  100000    4    udp6   :::0.111         portmapper superuser
  100000    3    udp6   :::0.111         portmapper superuser
  100000    4    tcp    0.0.0.0.0.111   portmapper superuser
  100000    3    tcp    0.0.0.0.0.111   portmapper superuser
  100000    2    tcp    0.0.0.0.0.111   portmapper superuser
  100000    4    udp    0.0.0.0.0.111   portmapper superuser
  100000    3    udp    0.0.0.0.0.111   portmapper superuser
  100000    2    udp    0.0.0.0.0.111   portmapper superuser
  100000    4    local  /run/rpcbind.sock portmapper superuser
  100000    3    local  /run/rpcbind.sock portmapper superuser
```

Step 2 : Write and compile your IDL file.

```
tran@tran-virtual-machine:~$ cat> idl.x
/*The IDL File --- name IDL.x*/
/*Structure to hold the 2 values to be used in computation*/

struct values{
    float num1;
    float num2;
    char operation;
};

/*Programm, version and procedure definition*/

program COMPUTE{
    version COMPUTE_VERS{
        float ADD(values) = 1;
        float SUB(values) = 2;
        float MUL(values) = 3;
        float DIV(values) = 4;
    } =6;
} = 456123789;
```

```
tran@tran-virtual-machine:~$ rpcgen -a -C idl.x
```

Step 3 : Edit the client and server programs.

idl_client.c

```
#include "idl.h"
#include <stdio.h>
float
compute_6(char* host, float a, float b, char op)
{
    CLIENT* clnt;
    float* result_1;
    values add_6_arg;
    float* result_2;
    values sub_6_arg;
    float* result_3;
    values mul_6_arg;
    float* result_4;
    values div_6_arg;
    if (op == '+') {
        add_6_arg.num1 = a;
        add_6_arg.num2 = b;
        add_6_arg.operation = op;
        clnt = clnt_create(host, COMPUTE, COMPUTE_VERS, "udp");
        if (clnt == NULL) {
            clnt_pcreateerror(host);
            exit(1);
        }
        result_1 = add_6(&add_6_arg, clnt);
        if (result_1 == (float*)NULL) {
            clnt_perror(clnt, "call failed");
        }
        clnt_destroy(clnt);
        return (*result_1);
    }
    else if (op == '-') {
        sub_6_arg.num1 = a;
        sub_6_arg.num2 = b;
        sub_6_arg.operation = op;
        clnt = clnt_create(host, COMPUTE, COMPUTE_VERS, "udp");
        if (clnt == NULL) {
            clnt_pcreateerror(host);
            exit(1);
        }
        result_2 = sub_6(&sub_6_arg, clnt);
        if (result_2 == (float*)NULL) {
```

```

        clnt_perror(clnt, "call failed");
    }
    clnt_destroy(clnt);
    return (*result_2);
}
else if (op == '*') {
    mul_6_arg.num1 = a;
    mul_6_arg.num2 = b;
    mul_6_arg.operation = op;
    clnt = clnt_create(host, COMPUTE, COMPUTE_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror(host);
        exit(1);
    }
    result_3 = mul_6(&mul_6_arg, clnt);
    if (result_3 == (float*)NULL) {
        clnt_perror(clnt, "call failed");
    }
    clnt_destroy(clnt);
    return (*result_3);
}
else if (op == '/') {
    if (b == 0) {
        printf("You are trying to divide by zero.Please insert a valid
number.\n");
        exit(0);
    }
    else {
        div_6_arg.num1 = a;
        div_6_arg.num2 = b;
        div_6_arg.operation = op;
        clnt = clnt_create(host, COMPUTE, COMPUTE_VERS, "udp");
        if (clnt == NULL) {
            clnt_pcreateerror(host);
            exit(1);
        }
        result_4 = div_6(&div_6_arg, clnt);
        if (result_4 == (float*)NULL) {
            clnt_perror(clnt, "call failed");
        }
        clnt_destroy(clnt);
        return (*result_4);
    }
}
}
}

```

```

int login() {
    int username;
    int password;
    printf("Username: ");
    scanf("%d", &username);
    if (username == 1401) {
        printf("Password: ");
        scanf("%d", &password);
        if (password == 14012000) {
            printf("Logged in successfully\n");
            return 1;
        }
    }
    printf("Wrong username\n");
    return 0;
}

int main(int argc, char* argv[])
{
    char* host;
    float number1, number2;
    char oper;
    if (login()) {
        while (1) {
            printf("Enter the 2 numbers followed by the operation to
perform:\n");
            scanf("%f", &number1);
            scanf("%f", &number2);
            scanf("%s", &oper);
            host = argv[1];
            printf("Answer= %f\n", compute_6(host, number1, number2, oper));
        }
    }
    else {
        printf("Failed to login\n");
    }
    exit(0);
}

```


idl_server.c

```
#include "idl.h"
#include <stdio.h>
float*
add_6_svc(values* argp, struct svc_req* rqstp)
{
    static float result;
    result = argp->num1 + argp->num2;
    return &result;
}
float*
sub_6_svc(values* argp, struct svc_req* rqstp)
{
    static float result;
    result = argp->num1 - argp->num2;
    return &result;
}
float*
mul_6_svc(values* argp, struct svc_req* rqstp)
{
    static float result;
    result = argp->num1 * argp->num2;
    return &result;
}
float*
div_6_svc(values* argp, struct svc_req* rqstp)
{
    static float result;
    result = argp->num1 / argp->num2;
    return &result;
}
```

Step 4 : Compile all the files

```
tran@tran-virtual-machine:~$ make -f Makefile.idl
cc -g      -c -o idl_clnt.o idl_clnt.c
cc -g      -c -o idl_client.o idl_client.c
cc -g      -c -o idl_xdr.o idl_xdr.c
cc -g      -o idl_client idl_clnt.o idl_client.o idl_xdr.o -lnsl
cc -g      -c -o idl_svc.o idl_svc.c
cc -g      -c -o idl_server.o idl_server.c
cc -g      -o idl_server idl_svc.o idl_server.o idl_xdr.o -lnsl
```

Step 5 : Run the server and the client

```
tran@tran-virtual-machine:~$ sudo ./idl_server
```

Open another terminal and run the client there.

```
tran@tran-virtual-machine:~$ sudo ./idl_client localhost
[sudo] password for tran:
Username: 1401
Password: 14012000
Logged in successfully
Enter the 2 numbers followed by the operation to perform:
16
20
+
Answer= 36.000000
Enter the 2 numbers followed by the operation to perform:
80
36
*
Answer= 2880.000000
Enter the 2 numbers followed by the operation to perform:
100
49
-
Answer= 51.000000
Enter the 2 numbers followed by the operation to perform:
1000
250
/
Answer= 4.000000
Enter the 2 numbers followed by the operation to perform:
```

```
tran@tran-virtual-machine:~$ sudo ./idl_client localhost
[sudo] password for tran:
Username: 1400
Wrong username
Failed to login
```

Источник:

[Netfilter example in python · GitHub](#)

<http://tharikasblogs.blogspot.com/p/how-to-write-simple-rpc-programme.html>

<https://manpages.ubuntu.com/manpages/bionic/man8/rpcbind.8.html>

<https://manpages.ubuntu.com/manpages/bionic/man7/rpcinfo.7.html>

Вывод:

- Написал код питон для фильтра сетевых пакетов на основе nfqueue и iptables и протестировал скорость хоста **yandex.ru**
- После выполнения работы мы написали rpc-программу на linux с поддержкой аутентификации.