Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

 «Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Дисциплина «Сети и телекоммуникации»

**Лабораторная работа №2**

««Разработка многопоточного сервера с использованием механизма сокетов»

Выполнил студент

группы ИВТАПбд-41:

Галацков И.А.

Проверил:

Мартынов А. И.

Ульяновск, 2024 **Задание на лабораторную работу**

Разработать серверное приложение, которое осуществляет взаимодействие при помощи механизма сокетов TCP/IP. Сервер работает в локальной сети по определенному IP-адресу и порту, которые настраиваются при запуске сервера, и отвечает на запросы клиента. Формат команд, которые должен обрабатывать сервер определяется согласно варианту задания. Основное требование к серверу, это обеспечение многопоточности, т.е. серверное приложение должно быть разработано таким образом, чтобы одновременно могло обрабатывать запросы нескольких пользователей и возможность работы.

**Описание реализации**

Реализуем сервер, который принимает подключенные клиенты через сокеты и обрабатывает их запросы в многозадачном режиме с помощью потоков. Сервер поддерживает несколько команд, таких как приветствие, прощание, а также шифрование и дешифрование сообщений. В основе шифрования используется линейный конгруэнтный генератор для создания потока ключа.

Сначала описана функция hash\_password, которая хеширует строковый пароль с использованием алгоритма SHA-1. В случае, если задан неподдерживаемый алгоритм, возбуждается ошибка. Этот хеш используется для генерации ключа для шифрования или дешифрования.

Для генерации псевдослучайных битов используется линейный конгруэнтный генератор в функции linear\_congruent\_generator. Эта функция принимает начальное значение (seed) и генерирует последовательность битов длиной n, которые могут быть использованы как ключ для шифрования или дешифрования данных.

Важная часть сервера — это шифрование и дешифрование сообщений. Функция encrypt\_decrypt\_file принимает строку сообщения и начальное значение seed, затем генерирует поток ключа с помощью линейного конгруэнтного генератора. После этого она выполняет побитовую операцию XOR между каждым байтом сообщения и соответствующим битом из потока ключа. Возвращаемое значение — это зашифрованная или расшифрованная строка.

Основная часть сервера, представленная функцией handle\_client, занимается обработкой запросов от клиентов. Она ожидает получения строки данных от клиента, которая может содержать команду (например, "hello", "bye", "encrypt", "decrypt"). В зависимости от команды сервер либо отправляет приветственное или прощальное сообщение, либо выполняет шифрование или дешифрование данных. Для этих команд также проверяется корректность входных данных, и при необходимости отправляется ошибка.

Функция start\_server запускает сам сервер. Она создает сокет, привязывает его к заданному IP-адресу и порту, а затем запускает прослушивание входящих подключений. Каждый клиентский запрос обрабатывается в отдельном потоке, что позволяет серверу обслуживать несколько клиентов одновременно.

Таким образом, сервер может принимать соединения от нескольких клиентов, выполнять обработку команд, таких как приветствие, прощание, шифрование и дешифрование сообщений, и отправлять результаты обратно клиентам.

**Приложение**

import socket  
import threading  
import random  
import math  
import sys  
import hashlib  
  
def hash\_password(password, algorithm = 'sha1'):  
 password\_bytes = password.encode('utf-8')  
 try:  
 if algorithm == 'sha1' :  
 hash\_func = hashlib.new(algorithm)  
 hash\_func.update(password\_bytes)  
 return hash\_func.hexdigest()  
  
 except ValueError:  
 raise ValueError("Неподдерживаемый алгоритм хеширования")  
  
#Линийный конгруэнтный генератор  
def linear\_congruent\_generator(n, seed):  
 a = 4096  
 B = 150889  
 m = 714025  
 #X = random.randrange(m-1)  
 X = seed  
 arr = []  
 for i in range(n):  
 X = (a \* X + B) % m  
 bit = X % 2  
 arr.append(bit)  
 return arr  
#Функция шифрования/дешифрования  
def encrypt\_decrypt\_file(data, seed):  
 # Преобразуем строку в байты  
 data\_bytes = data.encode('utf-8')  
 key\_stream = linear\_congruent\_generator(len(data\_bytes), seed)  
 transformed\_data = bytes([data\_bytes[i] ^ key\_stream[i] for i in range(len(data\_bytes))])  
 # Преобразуем обратно в строку и возвращаем  
 return ''.join(chr(byte) for byte in transformed\_data)  
  
  
# Обработка клиента  
def handle\_client(conn, addr):  
 print(f"Подключен клиент {addr}")  
 try:  
 while True:  
 data = conn.recv(1024).decode('utf-8').strip()  
 if not data:  
 break  
  
 parts = data.split(' ', 2)  
 command = parts[0].lower()  
 response = ""  
  
 if command == "hello":  
 variant = parts[1]  
 response = f"hello variant {variant}"  
 # Обработка команды bye  
 elif command == "bye":  
 variant = parts[1]  
 response = f"bye variant {variant}"  
 conn.sendall((response + '\n').encode('utf-8')) # Отправить ответ клиенту  
 print(f"Клиент {addr} отключился") # Сообщение в консоль сервера  
 conn.close() # Закрыть соединение  
 break # Выйти из цикла обработки клиента  
 elif command == "encrypt":  
 try:  
  
 message, password = parts[1].split(',', 1)  
 hash\_value = hash\_password(password)  
 seed = int(hash\_value, 16)  
 response = encrypt\_decrypt\_file(message, seed)  
 except Exception as e:  
 response = f"Ошибка шифрования: {e}"  
 elif command == "decrypt":  
 try:  
 message, password = parts[1].split(',', 1)  
 hash\_value = hash\_password(password)  
 seed = int(hash\_value, 16)  
 response = encrypt\_decrypt\_file(message, seed)  
 except Exception as e:  
 response = f"Ошибка дешифрования: {e}"  
 else:  
 response = "Команда не распознана"  
  
 conn.sendall((response + '\n').encode('utf-8'))  
 finally:  
 conn.close()  
  
  
# Запуск сервера  
def start\_server(ip, port):  
 server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server.bind((ip, port))  
 server.listen(5)  
 print(f"Сервер запущен на {ip}:{port}")  
  
 while True:  
 conn, addr = server.accept()  
 thread = threading.Thread(target=handle\_client, args=(conn, addr))  
 thread.start()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start\_server("127.0.0.1", 12345)