Федеральное агентство по образованию

Государственное образовательное учреждение высшего профессионального образования

Нижегородский государственный архитектурно-

строительный университет (ННГАСУ)

Факультет инженерно-экологических систем и сооружений  
Кафедра информационных систем и технологий

КУРСОВАЯ РАБОТА  
по дисциплине: «Инфокоммуникационные системы и сети»

На тему: «Разработка онлайн-игры»

Выполнил студент 3 курса гр. ИС-29: Малов Д. К.

Проверил старший преподаватель: Морозов Н. С.

**Содержание**

[Введение 3](#_Toc124995005)

[Задание 4](#_Toc124995006)

[Теоретическая часть 5](#_Toc124995007)

[I. Протокол связи 5](#_Toc124995008)

[1. IP — Internet Protocol 5](#_Toc124995009)

[2. TCP/IP — Transmission Control Protocol/Internet Protocol 5](#_Toc124995010)

[3. UDP — User Datagram Protocol 5](#_Toc124995011)

[4. FTP — File Transfer Protocol 6](#_Toc124995012)

[5. DNS 6](#_Toc124995013)

[6. HTTP — HyperText Transfer Protocol 6](#_Toc124995014)

[II. IP-адресация 7](#_Toc124995015)

[III. Маршрутизация TCP/IP 8](#_Toc124995016)

[Интерфейс приложения 10](#_Toc124995017)

[Код программы 11](#_Toc124995018)

[I. Серверная часть приложения 11](#_Toc124995019)

[II. Клиентская часть приложения 13](#_Toc124995020)

[Вывод 17](#_Toc124995021)

[Список литературы 18](#_Toc124995022)

# Введение

Сейчас почти вся инфраструктура завязана на интернет-соединении, поэтому важным навыком, для становления профессионалом в области информационных систем, является умение взаимодействовать при помощи кода с другой вычислительной машиной. Для развития этого навыка мы напишем достаточно простую игру для двух человек с чатом.

## Задание

* Изучить систему маршрутизации;
* Изучить систему стека TCP/IP;
* Произвести настройку протокола транспортного уровня;
* Произвести настройку сокетов серверной и клиентской части игры;

## Теоретическая часть

## Протокол связи

### IP — Internet Protocol

Протокол передачи, который первым объединил отдельные компьютеры в единую сеть. Самый примитивный в этом списке. Он является ненадёжным, т. е. не подтверждает доставку пакетов получателю и не контролирует целостность данных. По протоколу IP передача данных осуществляется без установки соединения.

Основная задача этого протокола — маршрутизация датаграмм, т. е. определение пути следования данных по узлам сети.

Популярная версия на текущий момент — IPv4 с 32-битными адресами. Это значит, что в интернете могут хранится 4.29 млрд адресов IPv4. Число большое, но не бесконечное. Поэтому существует версия IPv6, которая поможет решить проблему переполнения адресов, ведь уникальных IPv6 будет 2 ^ 128 адресов (число с 38 знаками).

### TCP/IP — Transmission Control Protocol/Internet Protocol

Это стек протоколов TCP и IP. Первый обеспечивает и контролирует надёжную передачу данных и следит за её целостностью. Второй же отвечает за маршрутизацию для отправки данных. Протокол TCP часто используется более комплексными протоколами.

### UDP — User Datagram Protocol

Протокол, обеспечивающий передачу данных без предварительного создания соединения между ними. Этот протокол является ненадёжным. В нём пакеты могут не только не дойти, но и прийти не по порядку или вовсе продублироваться.

Основное преимущество UDP протокола заключается в скорости доставки данных. Именно поэтому чувствительные к сетевым задержкам приложения часто используют этот тип передачи данных.

### FTP — File Transfer Protocol

Протокол передачи файлов. Его использовали ещё в 1971 году — задолго до появления протокола IP. На текущий момент этим протоколом пользуются при удалённом доступе к хостингам. FTP является надёжным протоколом, поэтому гарантирует передачу данных.

Этот протокол работает по принципу клиент-серверной архитектуры. Пользователь проходит аутентификацию (хотя в отдельных случаях может подключаться анонимно) и получает доступ к файловой системе сервера.

### DNS

Это не только система доменных имён (Domain Name System), но и протокол, без которого эта система не смогла бы работать. Он позволяет клиентским компьютерам запрашивать у DNS-сервера IP-адрес какого-либо сайта, а также помогает обмениваться базами данных между серверами DNS. В работе этого протокола также используются TCP и UDP.

### HTTP — HyperText Transfer Protocol

Изначально протокол передачи HTML-документов. Сейчас же он используется для передачи произвольных данных в интернете. Он является протоколом клиент-серверного взаимодействия без сохранения промежуточного состояния. В роли клиента чаще всего выступает веб-браузер, хотя может быть и, например, поисковый робот. Для обмена информацией протокол HTTP в большинстве случаев использует TCP/IP.

HTTP имеет расширение HTTPS, которое поддерживает шифрование. Данные в нём передаются поверх криптографического протокола TLS.

## IP-адресация

В стеке TCP/IP используются три типа адресов: локальные (называемые также аппаратными), IP-адреса и символьные доменные имена.

В терминологии TCP/IP под локальным адресом понимается такой тип адреса, который используется средствами базовой технологии для доставки данных в пределах подсети, являющейся элементом составной интерсети. В разных подсетях допустимы разные сетевые технологии, разные стеки протоколов, поэтому при создании стека TCP/IP предполагалось наличие разных типов локальных адресов. Если подсетью интерсети является локальная сеть, то локальный адрес — это МАС - адрес. МАС - адрес назначается сетевым адаптерам и сетевым интерфейсам маршрутизаторов. МАС - адреса назначаются производителями оборудования и являются уникальными, так как управляются централизованно. Для всех существующих технологий локальных сетей МАС - адрес имеет формат 6 байт, например 11-AO-17-3D-BC-01.

Однако протокол IP может работать и над протоколами более высокого уровня, например над протоколом IPX или Х.25. В этом случае локальными адресами для протокола IP соответственно будут адреса IPX и Х.25. Следует учесть, что компьютер в локальной сети может иметь несколько локальных адресов даже при одном сетевом адаптере. Некоторые сетевые устройства не имеют локальных адресов. Например, к таким устройствам относятся глобальные порты маршрутизаторов, предназначенные для соединений типа «точка-точка».

IP-адреса представляют собой основной тип адресов, на основании которых сетевой уровень передает пакеты между сетями. Эти адреса состоят из 4 байт, например 109.26.17.100. IP-адрес назначается администратором во время конфигурирования компьютеров и маршрутизаторов. IP-адрес состоит из двух частей: номера сети и номера узла. Номер сети может быть выбран администратором произвольно, либо назначен по рекомендации специального подразделения Internet (Internet Network Information Center, InterNIC), если сеть должна работать как составная часть Internet.

Обычно поставщики услуг Internet получают диапазоны адресов у подразделений InterNIC, а затем распределяют их между своими абонентами. Номер узла в протоколе IP назначается независимо от локального адреса узла. Маршрутизатор по определению входит сразу в несколько сетей. Поэтому каждый порт маршрутизатора имеет собственный IP-адрес. Конечный узел также может входить в несколько IP-сетей. В этом случае компьютер должен иметь несколько IP-адресов, по числу сетевых связей.

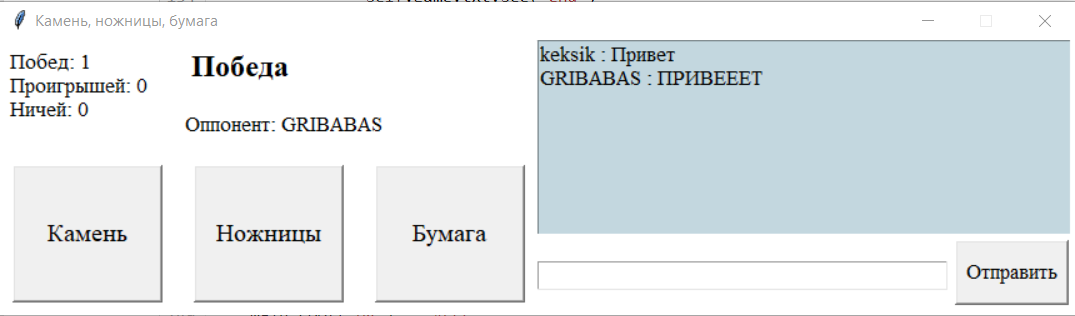
Таким образом, IP-адрес характеризует не отдельный компьютер или маршрутизатор, а одно сетевое соединение.

## Маршрутизация TCP/IP

Маршрутом называется путь, по которому пакеты пересылаются от отправителя к получателю. Маршрут определяет не полный путь, а только сегмент пути от хоста до шлюза (или от шлюза до шлюза), который может переслать пакеты целевому хосту. Список маршрутов хранится в таблице маршрутизации ядра. Описание маршрута содержит такую информацию, как список сетей, достижимых локальным хостом, и список шлюзов для отправки пакетов в удаленные сети. При получении дейтаграммы шлюз ищет в таблицах маршрутизации следующий узел ее маршрута до целевого хоста и отправляет дейтаграмму этому узлу.

В таблицу маршрутизации ядра можно добавлять несколько маршрутов к одному и тому же хосту. Процедура выбора маршрута сначала находит все маршруты, соответствующие запросу, а потом выбирает маршрут с минимальной метрикой расстояния. При наличии нескольких маршрутов одинаковой длины выбирается тот маршрут, который задан наиболее точно. Если несколько маршрутов совпадают по обоим критериям, то эти маршруты применяются по очереди.

# Интерфейс приложения



# Код программы

## Серверная часть приложения

import json  
import socket  
import threading  
from typing import Optional, Union  
  
  
class Result:  
 WIN = "win"  
 LOSE = "lose"  
 DRAW = "draw"  
  
  
class RPSServer:  
 def \_\_init\_\_(self, host, port):  
 self.host = host  
 self.port = port  
 self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.sock.bind((self.host, self.port))  
 self.clients: list[socket.socket] = []  
 self.actions: dict[socket.socket, dict[str, Union[int, str]]] = {}  
  
 def listen(self):  
 self.sock.listen(2)  
 while True:  
 client, address = self.sock.accept()  
 threading.Thread(target=self.client\_handler, args=(client,)).start()  
 self.clients.append(client)  
  
 def distribute(self, data: str, author: Optional[socket.socket] = None):  
 for client in self.clients:  
 if client == author:  
 continue  
 client.send(data.encode())  
  
 def client\_handler(self, client: socket.socket):  
 while True:  
 try:  
 data = json.loads(client.recv(1024).decode())  
 if not data:  
 continue  
  
 command = data["command"]  
 nickname = data["nickname"]  
 message = data["message"]  
 if command == "chat":  
 self.distribute(json.dumps(data), client)  
 continue  
 if command == "action":  
 self.actions[client] = {  
 "action": int(message),  
 "nickname": nickname,  
 }  
 if len(self.actions) == 2:  
 self.send\_result()  
 self.actions = {}  
 except Exception as e:  
 print(e)  
 self.clients.remove(client)  
 client.close()  
 return False  
  
 def send\_result(self):  
 user = list(self.actions.keys())[0]  
 opponent = list(self.actions.keys())[1]  
 user\_choice = self.actions[user]["action"]  
 opponent\_choice = self.actions[opponent]["action"]  
  
 user\_result = Result.LOSE  
 opponent\_result = Result.WIN  
  
 if user\_choice == opponent\_choice:  
 user\_result = Result.DRAW  
 opponent\_result = Result.DRAW  
  
 if (user\_choice + 1) % 3 == opponent\_choice:  
 user\_result = Result.WIN  
 opponent\_result = Result.LOSE  
  
 user.send(  
 json.dumps(  
 {  
 "command": "result",  
 "message": user\_result,  
 "nickname": self.actions[opponent]["nickname"],  
 }  
 ).encode()  
 )  
 opponent.send(  
 json.dumps(  
 {  
 "command": "result",  
 "message": opponent\_result,  
 "nickname": self.actions[user]["nickname"],  
 }  
 ).encode()  
 )  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 RPSServer("192.168.0.196", 9090).listen()

## Клиентская часть приложения

import json  
import random  
import threading  
import socket  
from typing import Optional  
from enum import IntEnum  
from tkinter import Tk, Frame, Button, Label, END  
import tkinter as tk  
  
  
class Action(IntEnum):  
 Rock = 0  
 Scissors = 1  
 Paper = 2  
  
  
class GameCommand:  
 def \_\_init\_\_(self, game: "Main", choice: Action):  
 self.choice = choice  
 self.game = game  
  
 def process\_button(self):  
 self.game.client.send("action", str(self.choice.value))  
  
 def \_\_call\_\_(self):  
 for btn in self.game.game\_buttons:  
 if btn["state"] == tk.DISABLED:  
 return  
 btn["state"] = tk.DISABLED  
 threading.Thread(target=self.process\_button).start()  
  
  
class Main(Frame):  
 def \_\_init\_\_(self, root, client\_: "SocketClient"):  
 super(Main, self).\_\_init\_\_(root)  
 self.client = client\_  
 client\_.game = self  
 self.root = root  
 self.opponent\_name = ""  
 self.game\_buttons = []  
 self.game\_start\_label: Optional[Label] = None  
 self.game\_data\_label: Optional[Label] = None  
 self.opponent\_label: Optional[Label] = None  
 self.entry: Optional[tk.Entry] = None  
 self.txt: Optional[tk.Text] = None  
 self.button\_font = ("Times New Roman", 15)  
 self.mini\_button\_font = ("Times New Roman", 13)  
 self.win = self.draw = self.lose = 0  
 self.start\_iu()  
  
 def game\_data\_text(self):  
 return f"Побед: {self.win}\nПроигрышей:" f" {self.lose}\nНичей: {self.draw}"  
  
 def send\_button(self, \*args):  
 input\_text = self.entry.get()  
 if not input\_text:  
 return  
 self.txt.configure(state="normal")  
 self.txt.insert(END, f"keksik : {input\_text}\n")  
 self.txt.see("end")  
 self.txt.configure(state="disabled")  
 self.entry.delete(0, END)  
 self.client.send("chat", input\_text)  
  
 def start\_iu(self):  
 self.game\_buttons = [  
 Button(self.root,text="Камень",font=self.button\_font,command=GameCommand(self, Action.Rock),),  
 Button(self.root,text="Ножницы",font=self.button\_font,command=GameCommand(self, Action.Scissors),),  
 Button(self.root,text="Бумага",font=self.button\_font,command=GameCommand(self, Action.Paper),),  
 ]  
  
 self.game\_buttons[0].place(x=10, y=100, width=120, height=110)  
 self.game\_buttons[1].place(x=155, y=100, width=120, height=110)  
 self.game\_buttons[2].place(x=300, y=100, width=120, height=110)  
 self.master.bind("<Return>", self.send\_button)  
  
 self.game\_start\_label = Label(self.root,text="Начало игры!",bg="#FFF",font=("Times New Roman", 18, "bold"),)  
 self.game\_data\_label = Label(self.root,justify="left",font=self.mini\_button\_font,text=self.game\_data\_text(),bg="#FFF",)  
 self.opponent\_label = Label(self.root,justify="right",font=self.mini\_button\_font,text=f"Оппонент: Нет",bg="#FFF",)  
  
 self.game\_start\_label.place(x=150, y=5)  
 self.game\_data\_label.place(x=5, y=5)  
 self.opponent\_label.place(x=145, y=55)  
  
 self.txt = tk.Text(self.root, font=self.mini\_button\_font, width=47, height=8, bg="#c3d7df")  
  
 self.txt.configure(state="disabled")  
 self.txt.place(x=430, y=0)  
 self.entry = tk.Entry(self.root, font=self.mini\_button\_font, width=36, bg="white")  
 self.entry.place(x=430, y=177)  
 send = Button(self.root,text="Отправить",font=self.mini\_button\_font,command=self.send\_button,width=9,height=2,)  
 send.place(x=764, y=160)  
  
  
class SocketClient:  
 def \_\_init\_\_(self, name: str):  
 self.client = None  
 self.name = name  
 self.game = None  
  
 def result\_handler(self, message: str):  
 if message == "draw":  
 self.game.draw += 1  
 self.game.game\_start\_label.configure(text="Ничья")  
 if message == "win":  
 self.game.win += 1  
 self.game.game\_start\_label.configure(text="Победа")  
 if message == "lose":  
 self.game.lose += 1  
 self.game.game\_start\_label.configure(text="Проигрыш")  
 self.game.game\_data\_label.configure(text=self.game.game\_data\_text())  
 for btn in self.game.game\_buttons:  
 btn["state"] = tk.NORMAL  
  
 def socket\_start(self, host: str, port: int):  
 self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.client.connect((host, port))  
 while True:  
 data = self.client.recv(1024)  
 if not data:  
 continue  
 data = json.loads(data.decode())  
 command = data["command"]  
 nickname = data["nickname"]  
 message = data["message"]  
  
 # *todo: handler* self.game.opponent\_label.configure(text=f"Оппонент: {nickname}")  
 if command == "result":  
 self.result\_handler(message)  
 if command == "chat":  
 self.game.txt.configure(state="normal")  
 self.game.txt.insert(END, f"{nickname} : {message}\n")  
 self.game.txt.see("end")  
 self.game.txt.configure(state="disabled")  
  
 def send(self, command: str, message: str):  
 data = json.dumps(  
 {"command": command, "nickname": self.name, "message": message}  
 )  
 self.client.sendall(data.encode())  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main\_root = Tk()  
 main\_root.geometry("860x220+200+200")  
 main\_root.title("Камень, ножницы, бумага")  
 main\_root.resizable(False, False)  
 main\_root["bg"] = "#FFF"  
 nick = "keksik"  
 print(nick)  
 client = SocketClient(name=nick)  
 app = Main(main\_root, client)  
 app.pack()  
  
 game\_thread = threading.Thread(target=main\_root.mainloop)  
 socket\_thread = threading.Thread(target=client.socket\_start, args=("192.168.200.243", 9090))  
 socket\_thread.start()  
 game\_thread.run()

# Вывод

Мы научились связывать, при помощи программных средств, две вычислительных машины в локальной сети на примере игры «Камень, ножницы, бумага». Это поможет нам в дальнейшем использовать полученные, в ходе данной работы, навыки для дальнейшей работы.

# Список литературы

1. Курс «Компьютерные сети» [Электронный ресурс] URL: [http://math.gsu.by/wp-content/uploads/courses/networks/построение\_больших\_сетей](http://math.gsu.by/wp-content/uploads/courses/networks/r5.2.html#:~:text=IP%2D%D0%B0%D0%B4%D1%80%D0%B5%D1%81%D0%B0%20%D0%BF%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D1%8F%D1%8E%D1%82%20%D1%81%D0%BE%D0%B1%D0%BE%D0%B9%20%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D0%BE%D0%B9,%D0%B2%D1%80%D0%B5%D0%BC%D1%8F%20%D0%BA%D0%BE%D0%BD%D1%84%D0%B8%D0%B3%D1%83%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%20%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BE%D0%B2%20%D0%B8%20%D0%BC%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80%D0%BE%D0%B2.)
2. Статья «Протоколы передачи данных» [Электронный ресурс] URL: <https://tproger.ru/explain/protokoly-peredachi-dannyh-chto-jeto-kakie-byvajut-i-v-chjom-razlichija/>
3. Документация IBM «Маршрутизация TCP/IP» [Электронный ресурс] URL: <https://www.ibm.com/docs/ru/aix/7.1?topic=protocol-tcpip-routing>
4. Таблица «безопасных» цветов от студии Артемия Лебедева [Электронный ресурс] URL: <https://www.artlebedev.ru/colors/>
5. Конвертер цветов из одной цветовой модели в ряд других [Электронный ресурс] URL: <https://colorscheme.ru/color-converter.html>