МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Проект

по дисциплине

«Параллельные вычисления»

Вариант 1

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Мартынов Д.С.\_\_\_

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сухоруков В.А.\_\_\_

(подпись) (фамилия, и.,о.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Мосташов В.С.\_\_\_

(подпись) (фамилия, и.,о.)

\_\_\_\_\_\_19-В-2\_\_\_\_\_\_\_\_

(шифр группы)

Работа защищена «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2021

# Цель работы

Построить модель и разработать программную модель для следующей задачи:

1. Моделирование работы электронной торговой площадки.

Электронная торговая площадка «получает» от «внешних процессов» заявки двух видов: заявки на покупку и заявки на продажу того или иного товара. Заявки имеют унифицированный формат, включающий в себя информацию об идентификаторе участника торгов (заказчика услуги со стороны торговой площадки), описание заказываемого товара, заявленная цена.

Для реализации работы «участников торгов» необходимо написать отдельные программы. Список товаров и их цен должен храниться в отдельном текстовом файле в csv-формате.

Для организации процесса взаимодействии потоков воспользоваться: интерфейсом сокетов.

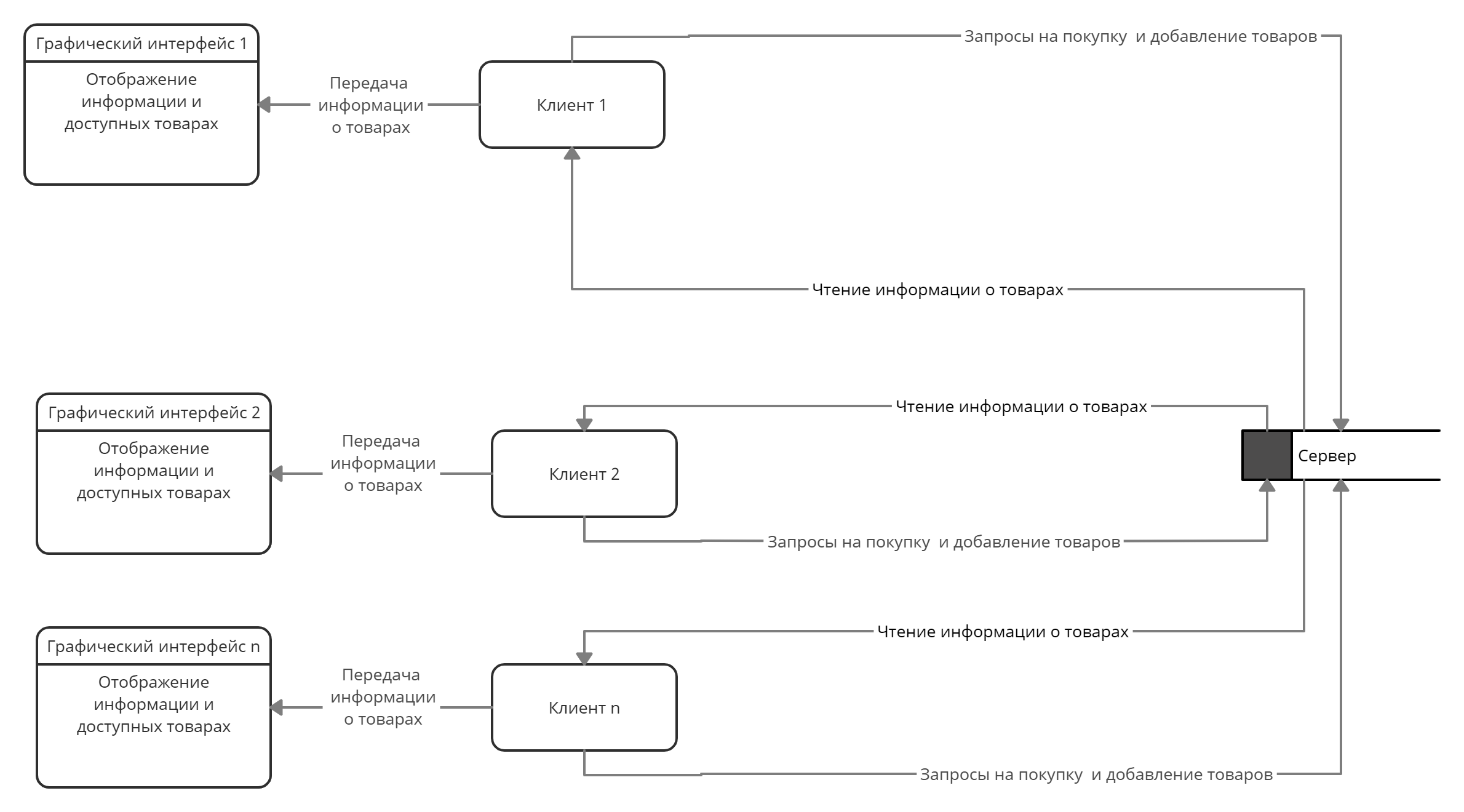
# Анализ задачи

На торговой площадке могут присутствовать несколько участников в различные моменты времени. Необходимо реализовать механизм подключению клиентов к серверу торговой площадки. При подключении у участника торгов доступны 2 функции, которые можно выполнить из графического интерфейса: разместить товар, купить товар.

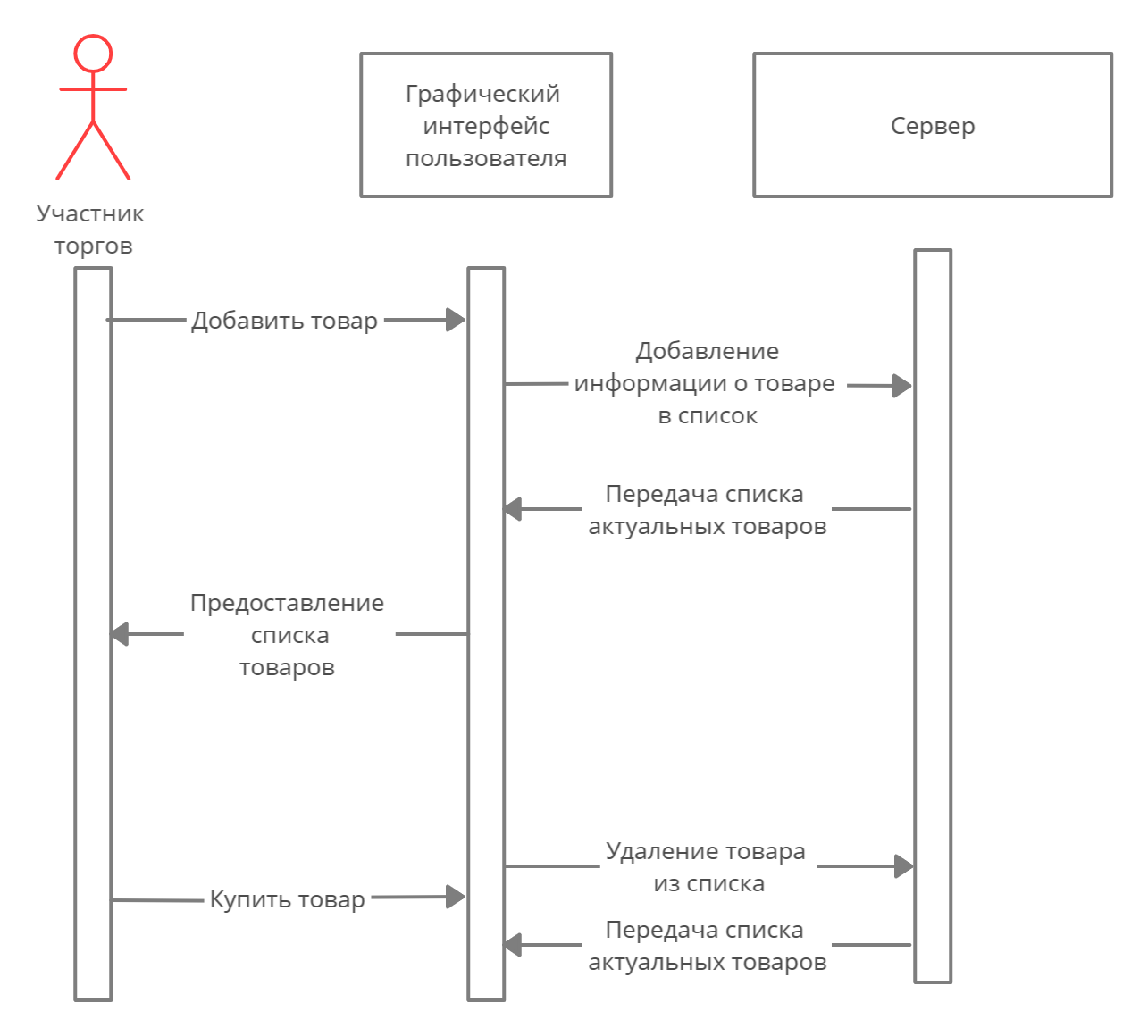
При размещении товара, информация о нём записывается в csv файл. Программа, запущенная у клиентов, должна получать сведения из общего файла и отображать информацию о товарах в графическом интерфейсе пользователя. Для проверки актуальности сведений необходимо реализовать функцию обновления информации, считанной из файла. Если товар, который собирается купить один из клиентов уже был приобретён другим клиентом, у которого не обновлена информация, отображаемая в графическом интерфейсе, то должно быть выведено сообщение о невозможности совершения сделки.

Данная модель работает в асинхронном режиме. Для реализации асинхронности используются генераторы. Сервер при подключении клиента заносит сокет клиента в список задач, которые он постоянно отслеживает. При поступлении запроса от клиента, сервер заносит клиента в список задач для выполнения. После чего сервер обрабатывает запрос клиента и опять заносит его в список задач для отслеживания.

# Диаграмма потоков данных



# Диаграмма взаимодействия



# Программная реализация

Программная модель была реализована средствами языка программирования Python. Для организации процесса взаимодействии потоков был использован интерфейс сокетов.

Для создания GUI использовалась библиотека Tkinter.

При помощи интерфейса мы можем добавить и купить товар, размещённый на торговой площадке. Каждый из клиентов, подключенных к площадке работает в отдельном потоке. Для каждого клиента используется его собственный GUI.

Информация о товарах пишется в отдельный csv файл откуда она считывается клиентами и выводится в соответствующей области графического интерфейса.

Код программной реализации разделен на 2 файла:

**Server.py** – содержит реализацию работы сервера.

**Client.py**– содержит реализацию работы клиента и его графический интерфейс.

# Листинг программы

## Server.py

import socket

import csv

import pickle

from select import select

tasks = [] #список с задачами

to\_read = {} #словарь для чтения

to\_write = {} #словарь для записи

#отправка данных из таблицы клиенту

def send\_request(client\_socket):

#открытие файла для чтения

with open('products.csv', "r", newline="") as file:

reader = csv.reader(file)

request = []

for row in reader:

request.append(row)

data = pickle.dumps(request)

#отправка данных клиенту

client\_socket.send(data)

#создание сервера

def server():

#создание сокета

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind(('localhost', 7000))

server.listen()

open('products.csv', "a", newline="")

while True:

yield ('read', server)

client\_socket, addr = server.accept()

print('Connection from', addr)

yield ('write', client\_socket)

send\_request(client\_socket)

#занесение генератора в список задач

tasks.append(client(client\_socket))

#обработка запросов клиента

def client(client\_socket):

while True:

yield ('read', client\_socket)

data = client\_socket.recv(4096)

if not data:

break

else:

request = pickle.loads(data)

#запрос на добавление данных

if request[0][0] == 1:

Id = 0

#открытие файла для чтения

with open('products.csv', "r", newline="") as file:

reader = csv.reader(file)

for row in reader:

if Id < int(row[0]):

Id = int(row[0])

request[1] = [[str(Id+1)] + request[1][0]]

#открытие файла для записи

with open('products.csv', "a", newline="") as file:

writer = csv.writer(file)

writer.writerows(request[1])

client\_socket.send('0'.encode())

#запрос на получение данных из файла

if request[0][0] == 2:

send\_request(client\_socket)

#запрос на покупку товара

if request[0][0] == 3:

data = []

temp = "f"

#открытие файла для чтения

with open('products.csv', "r", newline="") as file:

reader = csv.reader(file)

#нахождение записи с нужным id

for row in reader:

if request[1] != row[0]:

data.append(row)

if request[1] == row[0]:

temp = request[1]

#открытие файла для записи

with open('products.csv', "w", newline="") as file:

writer = csv.writer(file)

writer.writerows(data)

client\_socket.send(temp.encode())

client\_socket.close()

#цикл событий

def event\_loop():

while any([tasks, to\_read, to\_write]):

while not tasks:

#отслеживание объектов готовых к обработке

ready\_to\_read, ready\_to\_write, \_ = select(to\_read, to\_write, [])

#занесение генераторов в список задач

for sock in ready\_to\_read:

tasks.append(to\_read.pop(sock))

for sock in ready\_to\_write:

tasks.append(to\_write.pop(sock))

try:

task = tasks.pop(0) #получение генератора

reason, sock = next(task) #распаковка кортежа

#заполнение словарей

if reason == 'read':

to\_read[sock] = task

if reason == 'write':

to\_write[sock] = task

except StopIteration:

pass

#занесение генератора в список задач

tasks.append(server())

event\_loop()

## Client.py

import socket

import pickle

from tkinter import \*

import tkinter.ttk as ttk

from datetime import datetime

#создание сокета

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client.connect(('localhost', 7000))

#получение таблицы

data = client.recv(4096)

temp = pickle.loads(data)

#отправление запроса на добавление товара

def send\_request():

description = txt2.get()

price = txt3.get()

time = datetime.now()

#создание запроса

request = [[1], [[name, description, price, time]]]

data = pickle.dumps(request)

client.send(data)

client.recv(4096)

#обновление таблицы

update\_table()

#обновление таблицы

def update\_table():

#отправка запросв

request = [[2]]

data = pickle.dumps(request)

client.send(data)

#получение данных с сервера

data = client.recv(4096)

temp = pickle.loads(data)

#очистка таблицы

table.delete(\*table.get\_children())

#занесение полученных данных в таблицу

for row in temp:

table.insert(parent='', index=0, values=row)

#запрос на покупку

def send\_order():

if not table.selection():

return

#выбор выделенного товара

select = table.selection()[0]

value = table.item(select, option="values")

#занесение id товара в запрос

request = [[3], value[0]]

data = pickle.dumps(request)

client.send(data)

#получение ответа от сервера

answer = client.recv(4096)

answer = answer.decode()

lbl4 = Label(window, text="")

#если товар недоступен - вывести сообщение об этом

if answer == "f":

lbl4.configure(text="Товар с Id = " + value[0] + " недоступен.")

else:

lbl4.configure(text="Товар с Id = " + value[0] + " оформлен.")

lbl4.grid(column=1, row=11)

#обновление таблицы

update\_table()

#вход пользователя

def user\_login():

global name

name = txt1.get()

lbl1.destroy()

txt1.destroy()

btn0.destroy()

window.quit()

window = Tk()

window.title("Client")

window.geometry('600x320')

name = ""

lbl1 = Label(window, text="Введите имя пользователя: ")

lbl1.place(x=200, y=100)

txt1 = Entry(window, width=24)

txt1.place(x=200, y=120)

btn0 = Button(window, text="Войти", command=user\_login)

btn0.place(x=265, y=150)

#отображение окна с вводом имени пользователя

window.mainloop()

lbl2 = Label(window, text="Введите название товара: ")

lbl2.grid(column=0, row=1)

txt2 = Entry(window, width=24)

txt2.grid(column=0, row=2)

lbl3 = Label(window, text="Введите цену: ")

lbl3.grid(column=0, row=3)

txt3 = Entry(window, width=24)

txt3.grid(column=0, row=4)

btn = Button(window, text="Добавить", command=send\_request)

btn.grid(column=0, row=5)

btn2 = Button(window, text="Обновить", command=update\_table)

btn2.grid(column=1, row=0)

#создание таблицы

table = ttk.Treeview(window)

table['columns'] = ('Id','Name', 'Description', 'Price')

table.column('#0', width=0, stretch=NO)

table.column('Id', anchor=CENTER, width=40)

table.column('Name', anchor=CENTER, width=100)

table.column('Description', anchor=CENTER, width=180)

table.column('Price', anchor=CENTER, width=80)

table.heading('#0', text='', anchor=CENTER)

table.heading('Id', text='Id', anchor=CENTER)

table.heading('Name', text='Продавец', anchor=CENTER)

table.heading('Description', text='Название товара', anchor=CENTER)

table.heading('Price', text='Цена', anchor=CENTER)

#заполнение таблицы данными

for row in temp:

table.insert(parent='', index=0, values=row)

table.grid(column=1, row=1, rowspan=9)

btn3 = Button(window, text='Купить', command=send\_order)

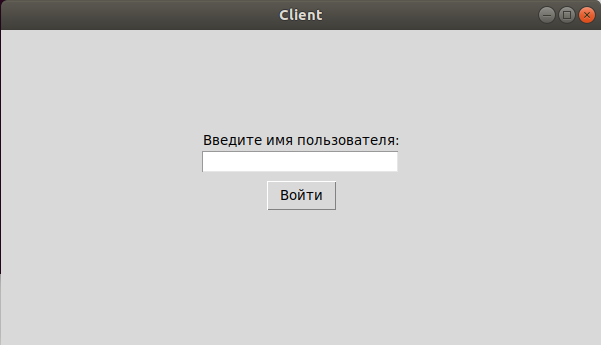
btn3.grid(column=1, row=10)

window.mainloop()

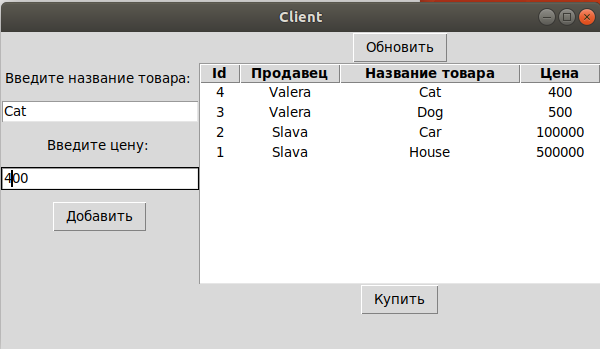
# Вывод

В ходе данной работы были закреплены навыки разработки многопоточных приложений на языке Python средствами библиотеки socketи разработки GUIпри помощи Tkinter.

# Приложения



Приложение 1. Вход пользователя в систему.



Приложение 2. Графический интерфейс пользователя на торговой площадке.