Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра системного программирования

Курсовая работа по дисциплине

«Операционные системы»

Выполнила: студентка гр. БСТ2104  
Первухина А.А.

Вариант №13

Проверила:

Алексанян Д.А.

Москва, 2023

Оглавление

[**1.** **Цель работы** 3](#_Toc153904495)

[**2. Задание на лабораторную работу** 3](#_Toc153904496)

[**Дополнительные задания на курсовую работу** 4](#_Toc153904497)

[**Введение** 5](#_Toc153904498)

[**3. Выполнение лабораторной работы** 6](#_Toc153904499)

[**3.1. Разработка и описание алгоритма работы приложения** 6](#_Toc153904500)

[**3.2. Описание структуры программы** 11](#_Toc153904501)

[**Реализация Logserver:** 17](#_Toc153904502)

[**3.3. Результаты применения программы** 18](#_Toc153904503)

[**Заключение** 21](#_Toc153904504)

[**Список использованных источников** 22](#_Toc153904505)

[**Приложение** 23](#_Toc153904506)

# **Цель работы**

**Целью** данной курсовой работы является закрепление и углубление теоретических знаний в области современных операционных систем, приобретение практических навыков разработки клиент-серверных приложений, использующих стандартные механизмы меж процессного взаимодействия.

# **2. Задание на лабораторную работу**

Разработать многозадачное приложение, выполняющее получение, сбор и вывод системной информации в соответствии с **вариантом 13**.

Таблица 1. Индивидуальные варианты заданий

|  |  |  |
| --- | --- | --- |
| Вариант | Сервер 1 | Сервер 2 |
|  | [получает от клиента/выполняет и] возвращает в качестве результата | |
| 13, 31 | * координаты окна серверного процесса | * количество модулей серверного процесса |
| * разрешение основного монитора | * количество потоков серверного процесса |

**Общие указания к выполнению работы**

Многозадачное приложение реализуется на основе технологии «клиент-сервер» средствами интегрированной среды разработки Microsoft Visual Studio/Visual C++, по согласованию с преподавателем возможно использование других компиляторов и языков программирования. Процессы 1 и 2 выступают как серверы приложений, третий процесс является их клиентом. Обмен данными между клиентами и серверами осуществляется при помощи сокетов.

И клиентский, и серверные процессы оформляются как отдельные приложения, запускаются и завершают работу независимо друг от друга. Серверы должны быть многопоточными, то есть допускать одновременное подключение нескольких клиентов. Серверные процессы по запросу получают системную информацию в соответствии с вариантом задания и передают ее клиенту с указанием текущего времени. На стороне клиента необходимо реализовать интерфейс, позволяющий пользователю подключиться по выбору к первому или второму серверу, а также возможность раздельного получения и отображения данных по каждому из пунктов индивидуального задания.

Необходимо предусмотреть проверку на повторный запуск серверов, т.е. исключить ситуацию, когда запускается более чем один экземпляр каждого сервера. Клиентов может быть несколько. В течение всего времени подключения клиента к серверу информация в клиентской части должна обновляться либо автоматически, либо по команде от пользователя.

# **Дополнительные задания на курсовую работу**

1. Существует отдельный сервер, который ведет лог операций (отдельный для каждого сервера задачи, с сохранением лога в файл), в котором фиксируются все события сервера: подключения клиентов, переданные запросы, ошибки и т.д. Взаимодействие между серверами задач и сервером логирования осуществляется с помощью каналов.
2. Реализация графического интерфейса на стороне клиента.

# **Введение**

В условиях постоянного развития информационных технологий и повышения требований к эффективности систем, разработка многозадачных приложений, способных эффективно обмениваться системной информацией, становится ключевой задачей для современных программистов. В данном контексте актуальность исследования механизмов межпроцессного взаимодействия, в частности, с использованием технологии "клиент-сервер" на основе сокетов, является неоспоримой.

Цель данной работы заключается в разработке многозадачного приложения, осуществляющего сбор и передачу системной информации с использованием технологии "клиент-сервер". Клиентское приложение предоставляет пользователю возможность подключения к двум серверам, каждый из которых выполняет свою уникальную функцию, собирая и возвращая информацию в соответствии с индивидуальным заданием. Серверы должны быть многопоточными, обеспечивая одновременное обслуживание нескольких клиентов, а обмен данными между процессами осуществляется с использованием сокетов.

Проект направлен на создание удобного и функционального интерфейса для пользователя, обеспечивая возможность выбора сервера, а также отображение раздельной системной информации в реальном времени. При этом важным аспектом является обеспечение уникальности серверных процессов и поддержание актуальности информации в течение всего времени подключения клиента.

# **3. Выполнение лабораторной работы**

В данной курсовой работе я использовала программу PyCharm Community Edition на языке программирования Python. Проект создавался в консольном приложение (Майкрософт).

## **3.1. Разработка и описание алгоритма работы приложения**

Пользователь входит и запускает приложение клиента, одновременно с этим приложением работают два сервера. Пользователь может в клиенте получить данные с сервера 1 или с сервера 2 на выбор. После выбора сервера написать «1», пользователь получит данные от 1 сервера, при этом будет проверяться обновления на сервере по таймеру (если полученные данные не равны предыдущим, отправляем их на приложение клиента). После написания «2», пользователь получит данные от 2 сервера, при этом он будет по таймеру получать данные, если они следующее отличается от предыдущего.

**1. Запуск серверов:**

* **Сервер 1 (Порт 8081):** Сервер ожидает подключений от клиентов. При получении подключения, создается новый поток для обработки клиента. Он обрабатывает запросы, связанные с получением информацию о ширине и высоте курсора, количестве клавиш мыши.
* **Сервер 2 (Порт 8082):** Аналогично серверу 1, но обрабатывает запросы клиентов, связанные с информацией об количестве всех потоков в системе, и модулях.
* **Log server (порт 8888):** для связи с серверами используется 8083 и 8084 порты

**2. Клиентское приложение:**

* **Графический интерфейс:** Клиент имеет графический интерфейс с двумя кнопками: "Connect to server 1" и "Connect to server 2".  
  Есть кнопка Clear data, которая очищает информацию о выводе. Также есть текстовое представление для вывода результатов.
* **Обработка задач:**
  + Connect to server 1*:* При нажатии кнопки "Video Adapter Task", клиент проверяет соединение с сервером 1 (порт 8081). Если соединение установлено, клиент отправляет запрос на сервер 1 и получает информацию о координатах окна серверного процесса, разрешении основного монитора. В случае отсутствия соединения, клиент выводит сообщение об ошибке.
  + Connect to server 2*:* При нажатии кнопки "Memory Usage Task", клиент проверяет соединение с сервером 2 (порт 8082). Если соединение установлено, клиент отправляет запрос на сервер 2 и получает информацию о количестве модулей серверного процесса, количестве потоков серверного процесса. В случае отсутствия соединения, клиент выводит сообщение об ошибке.
* **Обновление информации:**
  + Клиент периодически обновляет информацию о использовании памяти (раз в 5 секунд) с использованием таймера.
* **Обработка Отсутствия Соединения:**
  + При попытке отправить запрос на выполнение задачи без установленного соединения, клиент выводит сообщение об ошибке.
  + При разрыве соединения в процессе выполнения запроса, клиент также выводит сообщение об ошибке.
  + Созданы два сокета, для проверки сервера (работает или нет), чтобы два раза нельзя было вызывать.

**3. Алгоритм обновления информации на клиенте:**

* Connect to server 1**:**
  + Клиент проверяет соединение с сервером 1.
  + Если соединение установлено, клиент отправляет запрос на сервер 1.
  + Ожидает ответ от сервера 1.
  + Получив ответ, обновляет текстовое представление и получает информацию о координатах окна серверного процесса, разрешении основного монитора.
  + В случае отсутствия соединения, клиент выводит сообщение об ошибке.
* Connect to server 2**:**
  + Клиент проверяет соединение с сервером 2.
  + Если соединение установлено, клиент отправляет запрос на сервер 2.
  + Ожидает ответ от сервера 2.
  + Получив ответ, обновляет текстовое представление с информацией о количестве модулей серверного процесса, количестве потоков серверного процесса.
  + В случае отсутствия соединения, клиент выводит сообщение об ошибке.

**4. Алгоритм проверки на повторный запуск сервера:**

* **Сервер 1 (Порт 8081):**
  + Ожидает подключения от клиентов.
  + Обрабатывает каждое подключение в отдельном потоке.
  + Не прекращает слушать порт для новых подключений.
  + Используется отдельный сокет, для предотвращения повторного создания лог-сервера (8083).
* **Сервер 2 (Порт 8082):**
  + Точно также как и сервер 1, ожидает подключения и обрабатывает каждое в отдельном потоке.
  + Не прекращает слушать порт для новых подключений.

**5. Перечень запросов от клиента к серверу:**

* **От клиента к серверу 1 (Порт 8081):**
  + Запрос на выполнение задачи "Connect to server 1".
  + Нет запроса на обновление информации, так как эта задача выполняется по требованию.
* **От клиента к серверу 2 (Порт 8082):**
  + Запрос на выполнение задачи "Connect to server 2".
  + Запрос на обновление информации о использовании памяти с периодичностью.

**6. Входные и выходные данные:**

* **Входные данные (клиент):**
  + Выбор задачи для выполнения (нажатие кнопок "Connect to server 1" или "Connect to server 2").
  + Периодический запрос на обновление информации о памяти.
* **Выходные данные (клиент):**
  + Информация координатах окна серверного процесса, разрешении основного монитора (полученная от сервера 1).
  + Информация о координатах окна серверного процесса, разрешении основного монитора. (полученная от сервера 2).
  + Сообщения об ошибках при отсутствии соединения или разрыве соединения.
* **Входные данные (сервер 1):**
  + Запрос на выполнение задачи "VConnect to server 1" от клиента.
* **Выходные данные (сервер 1):**
  + Информация о координатах окна серверного процесса, разрешении основного монитора.
* **Входные данные (сервер 2):**
  + Запрос на выполнение задачи "Connect to server 2" от клиента.
* **Выходные данные (сервер 2):**
  + Информация о координатах окна серверного процесса, разрешении основного монитора.
* **Сообщения об ошибках:**
  + При отсутствии соединения или разрыве соединения могут появляться сообщения об ошибках как на стороне клиента, так и на стороне сервера.

## **3.2. Описание структуры программы**

Программа состоит из трех основных частей: сервер 1, сервер 2 и клиент. Рассмотрим структуру каждой из них.

**Сервер 1 (Порт 8081):**

Модули и Основные Функции:

1. **server1.py:**
   * start\_server(): Функция инициализации и запуска сервера. Ожидает подключения от клиентов, создает новый поток для каждого клиента и передает управление функции handle\_client().
   * handle\_client(client\_socket): Функция, запускаемая в отдельном потоке для обработки клиента. Выполняет задачу "Connect to server1", отправляет информацию о координатах окна серверного процесса, разрешении основного монитора
   * Функции для запроса информации у get\_server\_window\_info(device\_id).  
     Делает запрос на информацию о координатах окна серверного процесса

Структура:

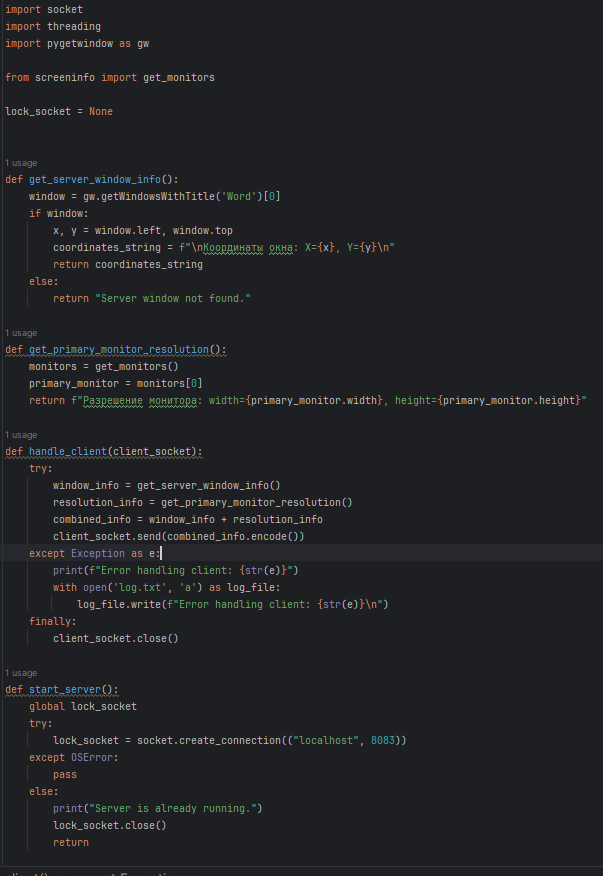


Рисунок 1 – структура 1 сервера

**Сервер 2 (Порт 8082):**

Модули и Основные Функции:

1. **server2.py:**
   * start\_server(): Функция инициализации и запуска сервера. Ожидает подключения от клиентов, создает новый поток для каждого клиента и передает управление функции handle\_client(client\_socket).
   * handle\_client(client\_socket): Функция, запускаемая в отдельном потоке для обработки клиента. Выполняет задачу "Connect to server 2", отправляет информацию о количестве модулей серверного процесса, количестве потоков серверного процесса. Также устанавливает таймер для периодического обновления информации.
   * Функции для работы с графическим интерфейсом get\_thread\_info().  
     Делает запрос на информацию о количестве модулей серверного процесса, количестве потоков серверного процесса.

Структура:

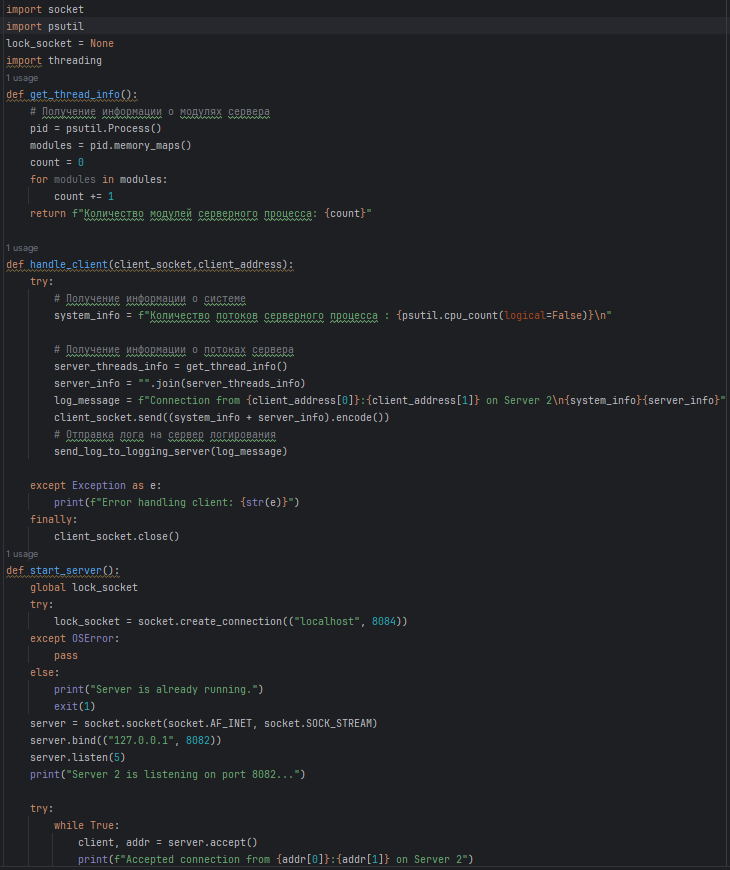


Рисунок 2 – структура 2 сервера

**Клиент:**

Модули и Основные Функции:

1. **client.py:**
   * create\_widgets(self) : Основная функция клиента. Инициализирует графический интерфейс GTK, создает окно с кнопками и текстовым представлением.
   * clear\_output(self): Очищает текстовое представление.
   * create\_widgets(self): Добавляет текс представление.
   * get\_data(self): Отправляет задачу на сервер в зависимости от типа задачи (Connect to server 1 или Connect to server 2).
   * connect\_to\_server(self,server\_port): Отправляет запрос на сервер с указанием типа задачи.

Структура:

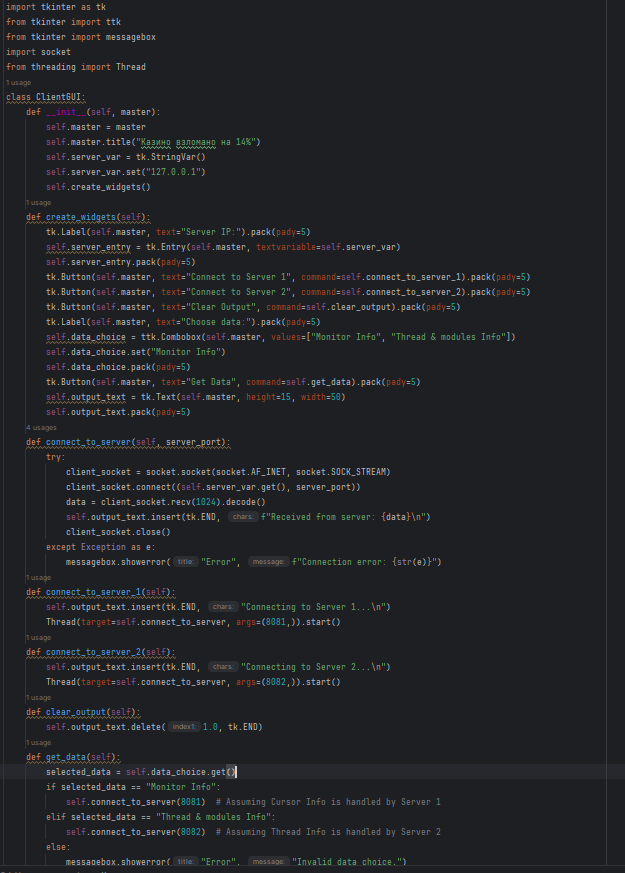


Рисунок 3 – структура клиентского приложения

## **Реализация Logserver:**

**Модули и Основные Функции:**

1. **logserver.py**
   * start\_logging\_server() : Основная функция логсервера  
     включает сокет для прослушивания и принимает запросы серверов
   * handle\_log(log\_client, addr): Производит запись в логфайл.

**Структура:**

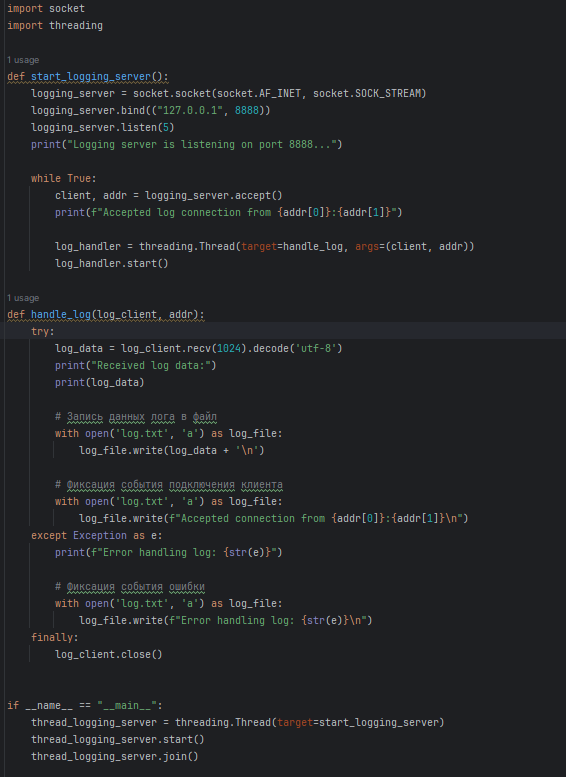


Рисунок 4– структура лог-сервера

## **3.3. Результаты применения программы**

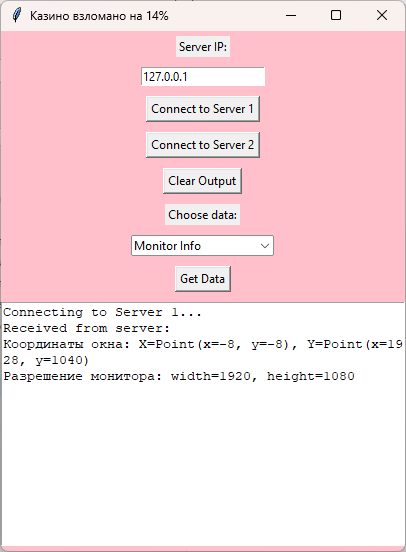


Рисунок 5 – запуск 1 сервера

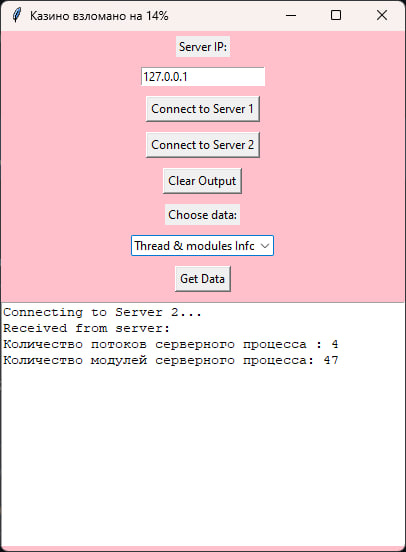


Рисунок 6 – запуск 2 сервера

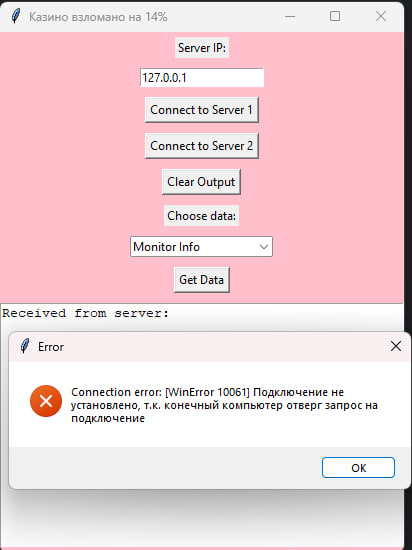


Рисунок 6 – запуск без подключения

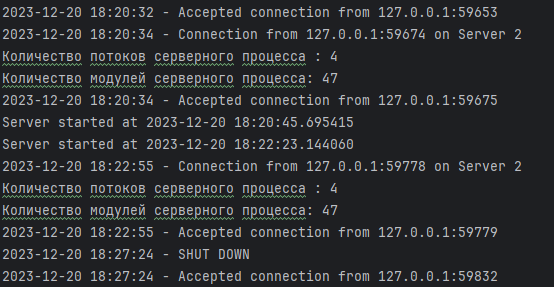
****

Рисунок 7 – содержимое лог-файла

# **Заключение**

В рамках пройденного материала за курс «Операционные системы» я научилась основам проектирования и работы операционных систем. В рамках курсовой работы я реализовала приложение на основе технологии «клиент-сервер» с двумя серверами, каждый из которых отправляет системные данные клиенту, а также реализовала графический интерфейс пользователя и отдельный сервер, который ведет лог операций, в котором фиксируются все события сервера: подключения клиентов, переданные запросы, ошибки и т.д.

# **Список использованных источников**

1. Э.Таненбаум. Современные операционные системы. 3-изд. – СПб.: Питер, 2011
2. Э.Таненбаум, Х.Бос. Современные операционные системы. 4-изд. – СПб.: Питер, 2015
3. В.Г.Олифер, Н.А.Олифер. Сетевые операционные системы. – СПб.: Питер, 2009
4. С.В. Назаров, А.И. Широков. «Современные операционные системы». — М., Бином, 2013
5. Марк Руссинович, Дэвид Соломон, Алекс Ионеску, Павел Йосифович. Внутреннее устройство Windows.7-е издание . - СПб.: Питер, 2018
6. Вдовикина Н.В., Машечкин И.В., Терехин А.Н., Томилин А.Н. «Операционные системы: взаимодействие процессов: учебно-методическое пособие». — Издательский отдел факультета ВМиК МГУ, 2008

# **Приложение**

Исходный код первого сервера Файл «server1.py»:

import socket  
import threading  
import pygetwindow as gw  
  
from screeninfo import get\_monitors  
  
lock\_socket = None  
  
def get\_server\_window\_info():  
 window = gw.getWindowsWithTitle('Word')[0]  
 if window:  
 x = window.topleft  
 y = window.bottomright  
 coordinates\_string = f"\nКоординаты окна: X={x}, Y={y}\n"  
 return coordinates\_string  
 else:  
 return "Server window not found."  
  
def get\_primary\_monitor\_resolution():  
 monitors = get\_monitors()  
 primary\_monitor = monitors[0]  
 return f"Разрешение монитора: width={primary\_monitor.width}, height={primary\_monitor.height}"  
  
def handle\_client(client\_socket):  
 try:  
 window\_info = get\_server\_window\_info()  
 resolution\_info = get\_primary\_monitor\_resolution()  
 combined\_info = window\_info + resolution\_info  
 client\_socket.send(combined\_info.encode())  
 except Exception as e:  
 print(f"Error handling client: {str(e)}")  
 with open('log.txt', 'a') as log\_file:  
 log\_file.write(f"Error handling client: {str(e)}\n")  
 finally:  
 client\_socket.close()  
  
def start\_server():  
 global lock\_socket  
 try:  
 lock\_socket = socket.create\_connection(("localhost", 8083))  
 except OSError:  
 pass  
 else:  
 print("Server is already running.")  
 lock\_socket.close()  
 return  
  
 server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 try:  
 server.bind(("127.0.0.1", 8081))  
 except:  
 log\_message1 = f"Server 1 SHUT DOWN!"  
 send\_log\_to\_logging\_server(log\_message1)  
 exit(1)  
 else:  
 server.listen(5)  
 print("Server 1 is listening on port 8081...")  
  
  
 try:  
 while True:  
 client, addr = server.accept()  
 print(f"Accepted connection from {addr[0]}:{addr[1]}")  
  
 # Фиксация события подключения клиента в файл лога  
 with open('log.txt', 'a') as log\_file:  
 log\_file.write(f"Accepted connection from {addr[0]}:{addr[1]}\n")  
  
 client\_handler = threading.Thread(target=handle\_client, args=(client,))  
 client\_handler.start()  
 except KeyboardInterrupt:  
 print("Server 2 is shutting down...")  
 finally:  
 server.close()  
  
def send\_log\_to\_logging\_server(log\_message):  
 with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as log\_client\_socket:  
 log\_client\_socket.connect(('127.0.0.1', 8888))  
 log\_client\_socket.send(log\_message.encode('utf-8'))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 thread\_server = threading.Thread(target=start\_server)  
 thread\_server.start()  
 thread\_server.join()

Исходный код второго сервера Файл «server2.py»:

import socket  
import psutil  
lock\_socket = None  
import threading  
def get\_thread\_info():  
 # Получение информации о модулях сервера  
 pid = psutil.Process()  
 modules = pid.memory\_maps()  
 count = 0  
 for modules in modules:  
 count += 1  
 return f"Количество модулей серверного процесса: {count}"  
  
def handle\_client(client\_socket,client\_address):  
 try:  
 # Получение информации о системе  
 system\_info = f"Количество потоков серверного процесса : {psutil.cpu\_count(logical=False)}\n"  
  
 # Получение информации о потоках сервера  
 server\_threads\_info = get\_thread\_info()  
 server\_info = "".join(server\_threads\_info)  
 log\_message = f"Connection from {client\_address[0]}:{client\_address[1]} on Server 2\n{system\_info}{server\_info}"  
 client\_socket.send((system\_info + server\_info).encode())  
 # Отправка лога на сервер логирования  
 send\_log\_to\_logging\_server(log\_message)  
  
 except Exception as e:  
 print(f"Error handling client: {str(e)}")  
 finally:  
 client\_socket.close()  
def start\_server():  
 global lock\_socket  
 try:  
 lock\_socket = socket.create\_connection(("localhost", 8084))  
 except OSError:  
 pass  
 else:  
 print("Server is already running.")  
 exit(1)  
 server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 try:  
 server.bind(("127.0.0.1", 8082))  
 except:  
 log\_message1 = f"Server 2 SHUT DOWN!"  
 send\_log\_to\_logging\_server(log\_message1)  
 exit(1)  
 else:  
 server.listen(5)  
 print("Server 2 is listening on port 8082...")  
  
 try:  
 while True:  
 client, addr = server.accept()  
 print(f"Accepted connection from {addr[0]}:{addr[1]} on Server 2")  
 client\_handler = threading.Thread(target=handle\_client, args=(client,addr))  
 client\_handler.start()  
 except KeyboardInterrupt:  
 print("Server 2 is shutting down...")  
 finally:  
 server.close()  
  
def send\_log\_to\_logging\_server(log\_message):  
 with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as log\_client\_socket:  
 log\_client\_socket.connect(('127.0.0.1', 8888))  
 log\_client\_socket.send(log\_message.encode('utf-8'))  
if \_\_name\_\_ == "\_\_main\_\_":  
 thread\_server2 = threading.Thread(target=start\_server)  
 thread\_server2.start()  
 thread\_server2.join()

Исходный код клиента Файл «client.py»:

import tkinter as tk  
from tkinter import ttk  
from tkinter import messagebox  
import socket  
from threading import Thread  
class ClientGUI:  
 def \_\_init\_\_(self, master):  
 self.master = master  
 self.master.title("Казино взломано на 14%")  
 self.server\_var = tk.StringVar()  
 self.server\_var.set("127.0.0.1")  
 self.create\_widgets()  
 def create\_widgets(self):  
 tk.Label(self.master, text="Server IP:").pack(pady=5)  
 self.server\_entry = tk.Entry(self.master, textvariable=self.server\_var)  
 self.server\_entry.pack(pady=5)  
 tk.Button(self.master, text="Connect to Server 1", command=self.connect\_to\_server\_1).pack(pady=5)  
 tk.Button(self.master, text="Connect to Server 2", command=self.connect\_to\_server\_2).pack(pady=5)  
 tk.Button(self.master, text="Clear Output", command=self.clear\_output).pack(pady=5)  
 tk.Label(self.master, text="Choose data:").pack(pady=5)  
 self.data\_choice = ttk.Combobox(self.master, values=["Monitor Info", "Thread & modules Info"])  
 self.data\_choice.set("Monitor Info")  
 self.data\_choice.pack(pady=5)  
 tk.Button(self.master, text="Get Data", command=self.get\_data).pack(pady=5)  
 self.output\_text = tk.Text(self.master, height=15, width=50)  
 self.output\_text.pack(pady=5)  
 def connect\_to\_server(self, server\_port):  
 try:  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client\_socket.connect((self.server\_var.get(), server\_port))  
 data = client\_socket.recv(1024).decode()  
 self.output\_text.insert(tk.END, f"Received from server: {data}\n")  
 client\_socket.close()  
 except Exception as e:  
 messagebox.showerror("Error", f"Connection error: {str(e)}")  
 def connect\_to\_server\_1(self):  
 self.output\_text.insert(tk.END, "Connecting to Server 1...\n")  
 Thread(target=self.connect\_to\_server, args=(8081,)).start()  
 def connect\_to\_server\_2(self):  
 self.output\_text.insert(tk.END, "Connecting to Server 2...\n")  
 Thread(target=self.connect\_to\_server, args=(8082,)).start()  
 def clear\_output(self):  
 self.output\_text.delete(1.0, tk.END)  
 def get\_data(self):  
 selected\_data = self.data\_choice.get()  
 if selected\_data == "Monitor Info":  
 self.connect\_to\_server(8081) # Assuming Cursor Info is handled by Server 1  
 elif selected\_data == "Thread & modules Info":  
 self.connect\_to\_server(8082) # Assuming Thread Info is handled by Server 2  
 else:  
 messagebox.showerror("Error", "Invalid data choice.")  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 root.configure(bg='pink')  
 client\_gui = ClientGUI(root)  
 root.mainloop()

Исходный код лог сервера Файл «logserver.py»:

import socket  
import threading  
from datetime import datetime  
  
  
def start\_logging\_server():  
 log\_server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 log\_server.bind(("127.0.0.1", 8888))  
 log\_server.listen(5)  
 print("Logging server is listening on port 8888...")  
  
 # Запись информации о запуске сервера  
 with open('log.txt', 'a') as log\_file:  
 log\_file.write(f"Server started at {datetime.now()}\n")  
  
 while True:  
 client, addr = log\_server.accept()  
 print(f"Accepted log connection from {addr[0]}:{addr[1]}")  
  
 log\_handler = threading.Thread(target=handle\_log, args=(client, addr))  
 log\_handler.start()  
  
  
def handle\_log(log\_client, addr):  
 try:  
 log\_data = log\_client.recv(1024).decode('utf-8')  
 current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
 log\_message = f"{current\_time} - {log\_data}"  
  
 print("Received log data:")  
 print(log\_message)  
  
 # Запись данных лога в файл  
 with open('log.txt', 'a') as log\_file:  
 log\_file.write(log\_message + '\n')  
  
 # Фиксация события подключения клиента  
 with open('log.txt', 'a') as log\_file:  
 log\_file.write(f"{current\_time} - Accepted connection from {addr[0]}:{addr[1]}\n")  
 except Exception as e:  
 error\_message = f"{current\_time} - Error handling log: {str(e)}"  
 print(error\_message)  
  
 # Фиксация события ошибки  
 with open('log.txt', 'a') as log\_file:  
 log\_file.write(error\_message + '\n')  
 log\_file.write(f"{current\_time} - disconnect error")  
 finally:  
 log\_client.close()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 thread\_logging\_server = threading.Thread(target=start\_logging\_server)  
 thread\_logging\_server.start()  
 thread\_logging\_server.join()