**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

на тему:

Windows

Выполнил:

студент группы БВТ1902

Атаев Реджеп

Проверил:

Королькова Татьяна Валерьевна

(Ф.И.О., должность преподавателя)

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата 22.12.2021

Москва 2021

Оглавление

[**Цель работы** 3](#_Toc91689735)

[**Задание на курсовую работу** 3](#_Toc91689736)

[Общие указания к выполнению работы 3](#_Toc91689737)

[Дополнительные задания 4](#_Toc91689738)

[**Выполнение** 5](#_Toc91689739)

[**Приложения** 9](#_Toc91689740)

[**Вывод** 10](#_Toc91689741)

# **Цель работы**

Реализовать многозадачное приложение на основе технологии «клиент-сервер». Процесс 1 и 2 выступают как серверы приложений, третий процесс является их клиентом. Реализовать обмен данными между клиентом и сервером при помощи сокетов.

# **Задание на курсовую работу**

Разработать многозадачное приложение, выполняющее получение, сбор и вывод системной информации в соответствии с вариантом задания (таблица 1). Номер варианта задания на курсовую работу совпадает с номером, под которым записана фамилия студента в журнале группы.

|  |  |  |
| --- | --- | --- |
| 4 | * количество мониторов | * полное имя модуля текущего процесса |
| * ширину и высоту основного монитора | * объем свободной физической памяти в единицах, переданных клиентом (в байтах, мегабайтах, гигабайтах по выбору пользователя) |

## 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

1. При обновлении информации на клиенте по таймеру данные передается только в том случае, если что-либо изменилось с момента прошлого обновления.
2. Обновление информации на клиенте происходит по инициативе сервера как реакция на изменения на сервере.
3. Существует отдельный сервер, который ведет лог операций (отдельные для каждого сервера, с сохранением в файл), в котором фиксируются подключения клиентов, переданные запросы, ошибки и т.д. Взаимодействие между серверами осуществляется с помощью каналов.
4. Запуск клиентов в контейнерах.
5. Наличие версии сервера или клиента под ОС Linux (т.е. если клиент и сервер могут работать под разными ОС).

# **Выполнение**

Листинг Сервера 1:

import asyncio  
import websockets  
import win32api  
import json  
  
from tendo.singleton import SingleInstance  
  
HOST, PORT = 'localhost', 1111  
  
  
def display\_monitors*()*:  
 monitors = win32api.EnumDisplayMonitors*()* dynaMon = *[]  
 [*dynaMon.append*(*monitors*[*i*][*2*])* for i in range*(*0, len*(*monitors*))]* return json.dumps*(*dynaMon*)*async def server*(*websocket*)*:  
 async for message in websocket:  
 match json.loads*(*message*)*:  
 case 'stop':  
 await websocket.close*(*code=1000, reason=''*)* case 'once':  
 try:  
 await websocket.send*(*display\_monitors*())* await websocket.close*()* except websockets.ConnectionClosedOK:  
 break  
 case \_:  
 try:  
 dataOld = None  
 for i in range*(*int*(*json.loads*(*message*)))*:  
 dataNew = display\_monitors*()* if dataOld == dataNew:  
 await asyncio.sleep*(*1*)* continue  
 else:  
 dataOld = dataNew  
 await websocket.send*(*dataOld*)* await asyncio.sleep*(*1*)* await websocket.close*()* except websockets.ConnectionClosedOK:  
 break  
  
  
async def main*()*:  
 async with websockets.serve*(*server, HOST, PORT*)*:  
 await asyncio.Future*()*me = SingleInstance*()*asyncio.run*(*main*())*

Листинг Сервера 2:

import asyncio  
import websockets  
import psutil  
import os  
import json  
  
from tendo.singleton import SingleInstance  
  
HOST, PORT = 'localhost', 1112  
DEFAULT\_TIMER = 10  
  
def disc\_free*(*unit*)*:  
 disc = psutil.disk\_usage*(*'/'*)* return disc*[*2*]* / *(*2 \*\* 30*)*def get\_process*()*:  
 p = psutil.Process*(*os.getpid*())*.as\_dict*(*attrs=*[* 'pid', 'ppid', 'name', 'exe', 'cpu\_percent', 'num\_threads', 'username'  
 *])* return p  
  
async def server*(*websocket*)*:  
 async for message in websocket:  
 match json.loads*(*message*)*:  
 case 'stop':  
 await websocket.close*(*code=1000, reason=''*)* case 'once':  
 try:  
 await websocket.send*(*json.dumps*(*get\_process*()))* await websocket.send*(*json.dumps*(*disc\_free*(*json.loads*(*message*))))* await websocket.close*()* except websockets.ConnectionClosedOK:  
 break  
 case \_:  
 try:  
 dataOld1 = None  
 dataOld2 = None  
 for i in range*(*int*(*json.loads*(*message*)))*:  
 dataNew1 = get\_process*()* dataNew2 = disc\_free*(*json.loads*(*message*))* if dataOld1 == dataNew1 and dataOld2 == dataNew2:  
 await asyncio.sleep*(*1*)* continue  
 elif dataOld1 != dataNew1:  
 dataOld1 = dataNew1  
 await websocket.send*(*json.dumps*(*dataOld1*))* elif dataOld2 != dataNew2:  
 dataOld2 = dataNew2  
 await websocket.send*(*json.dumps*(*dataOld2*))* else:  
 await websocket.send*(*json.dumps*(*get\_process*()))* await websocket.send*(*json.dumps*(*disc\_free*(*json.loads*(*message*))))* await asyncio.sleep*(*1*)* await websocket.close*()* except websockets.ConnectionClosedOK:  
 break  
  
  
async def main*()*:  
 async with websockets.serve*(*server, HOST, PORT*)*:  
 await asyncio.Future*()*me = SingleInstance*()*asyncio.run*(*main*())*

Листинг Клиента:

import asyncio  
import websockets  
import json  
  
  
async def cServer1*(*msg='once'*)*:  
 uri = 'ws://localhost:1111'  
 async with websockets.connect*(*uri*)* as client:  
 while client:  
 try:  
 await client.send*(*json.dumps*(*msg*))* data = json.loads*(*await client.recv*())  
 [*print*(*f"Monitor *{*i + 1*}* specs: *{*data*[*i*]}*"*)* for i in range*(*0, len*(*data*))]* except websockets.ConnectionClosedOK:  
 break  
  
  
async def cServer2*(*msg='once'*)*:  
 uri = 'ws://localhost:1112'  
 async with websockets.connect*(*uri*)* as client:  
 while client:  
 try:  
 await client.send*(*json.dumps*(*msg*))* data = json.loads*(*await client.recv*())* print*(*data*)* except websockets.ConnectionClosedOK:  
 break  
  
  
async def main*()*:  
 while True:  
 action = input*(*"\n1 - connect to 1 server \n2 - connect to 2 server\nexit - Exit\n\n"*)*.split*()* if action and action*[*0*]* == '1':  
 if len*(*action*)* != 1:  
 server1\_task = asyncio.create\_task*(*cServer1*(*action*[*1*]))* await server1\_task  
 else:  
 server1\_task = asyncio.create\_task*(*cServer1*())* await server1\_task  
 if action and action*[*0*]* == '2':  
 if len*(*action*)* != 1:  
 server2\_task = asyncio.create\_task*(*cServer2*(*action*[*1*]))* await server2\_task  
 else:  
 server2\_task = asyncio.create\_task*(*cServer2*())* await server2\_task  
 if action and action*[*0*]* == 'exit':  
 break  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 asyncio.run*(*main*())*

## **Приложения**

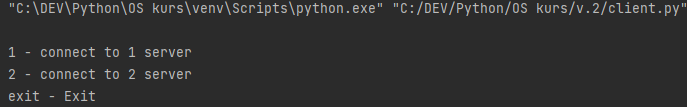


Рисунок 1 – Интерфейс клиента

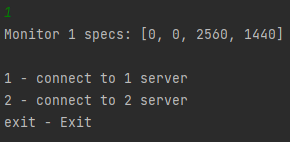


Рисунок 2 – ответ сервера 1 на запрос с клиента

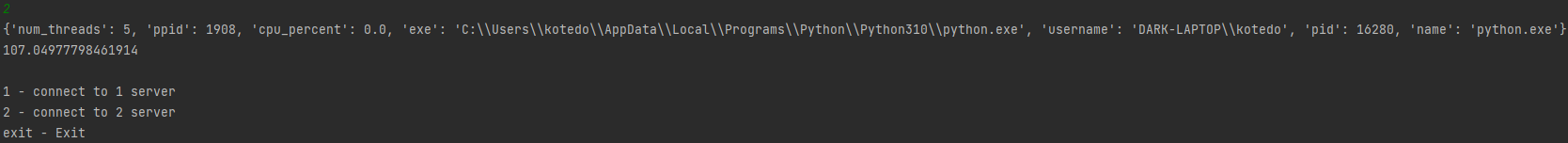


Рисунок 3 – ответ сервера 2 на запрос клиента

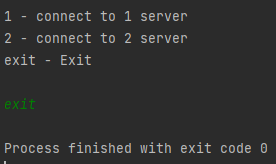


Рисунок 4 – успешное завершение работы клиента по запросу пользователя

# **Вывод**

Приобрел навыки разработки клиент серверных приложений, с передачей данных при помощи сокетов, а также получение системной информации используя ЯП Python3.