

Отчёт по дисциплине
«Web-программирование»
Лабораторная работа №1

Выполнил:

Носов Артём Иванович ДВИЖ

Санкт-Петербург

2023г

Задание №1

На сервере запускается сокет на порту 9090, который получает и выводит сообщение от клиента. Затем сервер отправляет ответ.

```
1 import socket
2 # Создаем сокет UDP
3 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
4 # Привязываем сокет к текущему хосту (компьютеру) и порту 9090
5 s.bind((socket.gethostname(), 9090))
6 # Запускаем бесконечный цикл для прослушивания и обработки входящих сообщений
7 while True:
8     # Получаем сообщение (msg) и адрес (address) отправителя
9     msg, address = s.recvfrom(2148)
10    # Декодируем и печатаем полученное сообщение
11    print(msg.decode())
12    # Отправим ответное сообщение
13    s.sendto(bytes('Hello, client!', 'utf-8'), address)
```

Run: server x client x

"C:\Users\Artiom\Documents\MEGAsync\Education courses\WEB_ITMO\ITMO_ICT_WebDe
Hello, server!

Рисунок 1. server задание №1

На стороне клиента запускается сокет, затем этот сокет подключается к серверу на порту 9090, после чего отправляет сообщение, получает ответ и выводит его на консоль.

```
1 import socket
2
3 # Создаём User Datagram Protocol сокет
4 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
5
6 # Подключение сокета к локальной машине, порту 9090
7 s.connect((socket.gethostname(), 9090))
8
9 # Отправляем сообщение серверу
10 s.sendto(bytes('Hello, server!', 'utf-8'), (socket.gethostname(), 9090))
11 # Получаем ответ от сервера
12 msg = s.recvfrom(2148)
13 # печатаем декодированное сообщение, полученное от сервера
```

Run: server x client x

"C:\Users\Artiom\Documents\MEGAsync\Education courses\WEB_ITMO\ITMO_ICT_
Hello, client!

Process finished with exit code 0

Рисунок 2. client задание №1

Задание №1

Сервер не выводит никакой информации, только получает, обрабатывает его и отправляет ответ.

```
1 # Поиск площади параллелограмма.
2
3 import socket
4 # Создаем сокет (конечную точку для обмена данными) с использованием протокола TCP
5 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6 # Привязываем сокет к текущему хосту (компьютеру) и порту 7070
7 s.bind((socket.gethostname(), 7070))
8 # Начинаем прослушивать (слушать) входящие соединения, позволяя до 10 клиентских соединений в очереди
9 s.listen(10)
10 # Запускаем бесконечный цикл для ожидания и обработки входящих клиентских соединений
11 while True:
12     # Принимаем входящее клиентское соединение и получаем объект клиентского сокета (clsocket)
13     # и адрес клиента (address)
14     clsocket, address = s.accept()
15     # Получаем данные (строку) от клиента (передаем допустимую длину сообщения в recv)
16     data = clsocket.recv(1000)
17     # Декодируем полученные данные в строку и разделяем их на два значения, a и h
18     a, h = data.decode().split()
19     S = str(float(a) * float(h))
20     # Отправляем результат обратно клиенту в виде байтовой строки
21     clsocket.send(bytes(S, 'utf-8'))
22     # Закрываем клиентский сокет
23     clsocket.close()
```

Run: client (1) × server (1) ×

"C:\Users\Artiom\Documents\MEGAsync\Education courses\WEB_ITMO\ITMO ICT_WebDevelopment_2022-2023\venv

Рисунок 3. server задание №2

Программа запрашивает данные у пользователя, формирует сообщение, подключается к серверу и отправляет запрос. После получения ответа от сервера, программа выводит их пользователю.

```
1 import socket
2
3 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4
5 msg = input("Введите основание и высоту параллелограмма через пробел: ")
6 msg = bytes(msg, 'utf-8')
7
8 s.connect((socket.gethostname(), 7070))
9 s.send(msg)
10 result = float(s.recv(1000).decode())
11 print("Площадь параллелограмма равна ", result)
12
13 s.close()
```

Run: client (1) × server (1) ×

"C:\Users\Artiom\Documents\MEGAsync\Education courses\WEB_ITMO\ITMO ICT_WebDevelopment_2022-2023\venv

Введите основание и высоту параллелограмма через пробел: 4 2

Площадь параллелограмма равна 8.0

Process finished with exit code 0

Рисунок 4. client задание №2

Задание №3

Сокет после установки соединения отправляется html-сообщение клиенту построчно, а затем также передает файл index.html.

```
1 import socket
2 # Создаем сокет (конечную точку для обмена данными) с использованием протокола TCP
3 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4 # Привязываем сокет к текущему хосту (компьютеру) и порту 8080
5 s.bind((socket.gethostname(), 8080))
6 # Начинаем прослушивать (слушать) входящие соединения, позволяя до 10 клиентских соединений в очереди
7 s.listen(10)
8 # Запускаем бесконечный цикл для ожидания и обработки входящих клиентских соединений
9 while True:
10     # Принимаем входящее клиентское соединение и получаем объект клиентского сокета (csocket)
11     # и адрес клиента (addr)
12     csocket, addr = s.accept()
13
14     # Отправляем HTTP-заголовок клиенту, указывая успешный статус и тип содержимого
15     csocket.send(bytes('HTTP/1.1 200 OK\n', 'utf-8'))
16     csocket.send(bytes('Content-Type: text/html; charset=utf-8\n', 'utf-8'))
17     csocket.send(bytes('Content-Length: 150\n', 'utf-8'))
18     csocket.send(bytes('\n', 'utf-8'))
19     csocket.send(bytes('\n', 'utf-8'))
20
21     # Открываем файл 'index.html' в режиме чтения байтов
22     file = open('index.html', 'rb')
23     # Читаем файл построчно и отправляем его содержимое клиенту
24     for line in file:
25         csocket.send(line)
26
27     # Закрываем клиентский сокет
28     csocket.close()
29
30 # Закрываем серверный сокет (по идее, эта строка кода никогда не выполнится, так как цикл while бесконечен)
31 s.close()
```

Рисунок 5. server задание №3

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>My super HTML page</title>
6 </head>
7 <body>
8
9 </body>
10 </html>
```

Рисунок 6. html-страница

Клиент только получает данные, посылаемые сервером, и выводит их в консоль.

```
1 import socket
2
3 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4
5 s.connect((socket.gethostname(), 8080))
6 msg = ''
7 while True:
8     line = s.recv(1000)
9     if not line:
10         break
11     msg += line.decode()
12
13 print(msg)
14
15 s.close()
```

Рисунок 7. client задание №3

```

"C:\Users\Artiom\Documents\MEGAsync\Edu
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 150

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My super HTML page</title>
</head>
<body>

</body>
</html>

Process finished with exit code 0

```

Рисунок 8. Консольный вывод клиентской части

Задание №4

В файле сервера определено несколько функций. Первая отвечает за подключение пользователя, о чем выводит информацию в консоль. Вторая получает сообщение от клиента, если возвращается ошибка, то данное подключение удаляется из списка, а также отправляет всем подключенным клиентам полученное сообщение. Третья отвечает за создание потока для каждого клиента, чтобы они могли иметь одновременное подключение.

```

1 import socket
2 import threading
3
4 # Функция обработки соединений с клиентами
5 def connection(sockets):
6     while True:
7         # Принимаем соединение от клиента и добавляем его сокет в
        список
8         clsocket, addr = s.accept()
9         sockets.append(clsocket)
10        print('Подключился клиент', addr)
11
12 # Функция для получения сообщений от одного клиента и отправки их всем
        остальным клиентам
13 def get(sock, sockets):
14     while True:
15         try:
16             msg = sock.recv(1000)
17             print(f"от {sock} было получено сообщение {msg}")
18         except Exception as e:
19             # Если возникает ошибка при получении сообщения, удаляем
            клиентский сокет и выходим из цикла
20             sockets.remove(sock)
21             print(f"Ох, похоже была ошибка при получении сообщения
            от {sock}")
22             break
23         # Пересылаем сообщение от одного клиента всем остальным
            клиентам
24         for soc in sockets:
25             if soc != sock:
26                 soc.send(msg)
27                 print(f"Переслали сообщение {msg} пользователю {soc}")
28
29 # Функция для создания отдельных потоков для каждого клиента
30 def make_threads(sockets, threads):
31     while True:
32         for soc in sockets:
33             if soc in threads:
34                 continue
35             print(f"Ого, мы создаём поток! для {soc}")
36             # Создаём новый поток для клиента и передаём ему
            соответствующий сокет
37             t = threading.Thread(target=get, args=(soc, sockets))
38             t.start()
39             threads.append(soc)
40

```

Рисунок 9. server часть первая задание №4

В другой части запускается серверный сокет, создаются массивы для подключений и потоков. Затем создаются два потока для одновременной обработки новых подключений и сообщений из чата.

```
41 # Создаем серверный сокет
42 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
43
44 # Привязываем серверный сокет к хосту (этому компьютеру) и порту 2020
45 s.bind((socket.gethostname(), 1010))
46
47 # Начинаем слушать порт, принимая до 10 клиентских соединений
48 s.listen(10)
49
50 # Создаем список для хранения клиентских сокетов и список для хранения потоков
51 sockets = []
52 threads = []
53
54 # Создаем поток для обработки новых соединений
55 t1 = threading.Thread(target=connection, args=(sockets,))
56 t1.start()
57
58 # Создаем поток для создания потоков для каждого клиента
59 t2 = threading.Thread(target=make_threads, args=(sockets, threads,))
60 t2.start()
61
62 # Ожидаем завершения потока t2
63 t2.join()
```

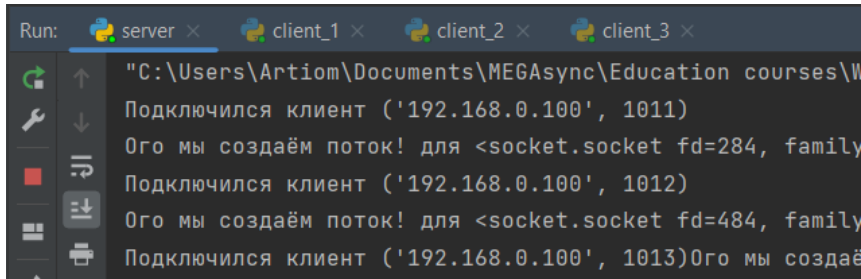
Рисунок 10. server часть два задание №4

У всех пользователей одинаковый скрипт. Отличается только вызов bind.

```
1 import socket
2 import threading
3
4 # Функция для получения сообщений с сервера и их вывода на экран
5 def get_msg(sock):
6     while True:
7         # Получаем сообщение от сервера (максимум 1000 байт)
8         msg = sock.recv(1000)
9         # Декодируем сообщение из байтов в строку и выводим на экран
10        print(msg.decode())
11
12 # Создаем клиентский сокет (конечную точку для обмена данными) с использованием протокола TCP
13 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14 # Привязываем клиентский сокет к текущему хосту (компьютеру) и порту 1011
15 s.bind((socket.gethostname(), 1011))
16 # Устанавливаем соединение с сервером на хосте и порту 1010
17 s.connect((socket.gethostname(), 1010))
18 # Создаем новый поток (Thread) для выполнения функции get_msg, передавая клиентский сокет s в качестве аргумента
19 get = threading.Thread(target=get_msg, args=(s,))
20 # Запускаем поток для приема сообщений от сервера
21 get.start()
22 # Запускаем бесконечный цикл для отправки сообщений серверу
23 while True:
24     # Запрашиваем сообщение для отправки у пользователя
25     msg_send = input()
26     # Отправляем сообщение серверу, предварительно преобразовав его в байты с кодировкой utf-8
27     s.send(bytes(msg_send, 'utf-8'))
28
```

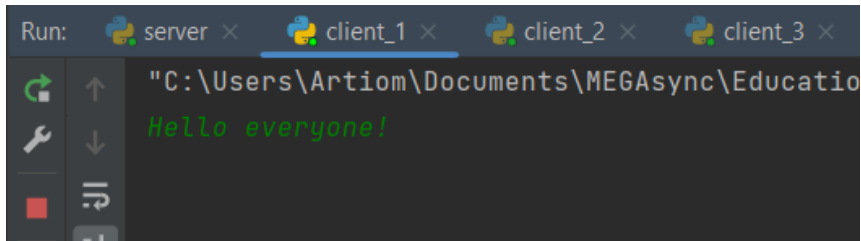
Рисунок 11. client задание №4

Демонстрация работы программы.



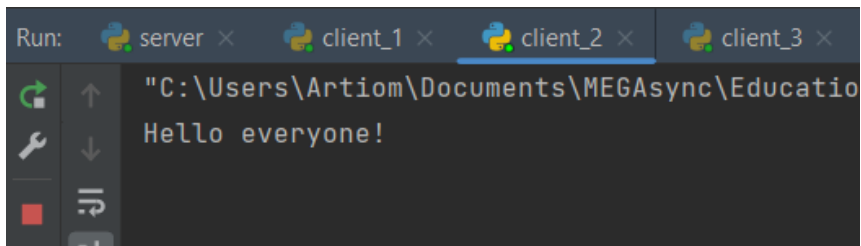
```
Run: server x client_1 x client_2 x client_3 x
"C:\Users\Artiom\Documents\MEGAsync\Education courses\W
Подключился клиент ('192.168.0.100', 1011)
Ого мы создаём поток! для <socket.socket fd=284, family
Подключился клиент ('192.168.0.100', 1012)
Ого мы создаём поток! для <socket.socket fd=484, family
Подключился клиент ('192.168.0.100', 1013)Ого мы создаём
```

Рисунок 12. Запуск программы и подключение пользователей



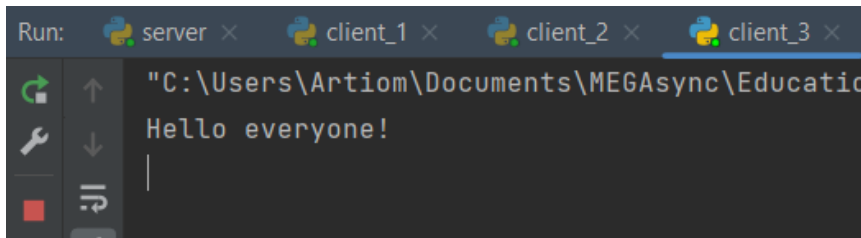
```
Run: server x client_1 x client_2 x client_3 x
"C:\Users\Artiom\Documents\MEGAsync\Education
Hello everyone!
```

Рисунок 13. Первый пользователь отправил сообщение



```
Run: server x client_1 x client_2 x client_3 x
"C:\Users\Artiom\Documents\MEGAsync\Education
Hello everyone!
```

Рисунок 14. Второй пользователь получает сообщение от первого



```
Run: server x client_1 x client_2 x client_3 x
"C:\Users\Artiom\Documents\MEGAsync\Education
Hello everyone!
|
```

Рисунок 15. Третий пользователь получает сообщение от первого

Задание №5

Здесь сервер является http-сервером. В первой функции проходит инициализация сервера и инициализация словаря для дисциплин, куда будут записываться оценки студентов. Далее описана главная функция сервера, где он запускается и обрабатывает клиентские подключения.

```

1  import socket
2  MAX_LINE = 128 * 512
3  MAX_HEADERS = 200
4  class HTTPServer:
5      def __init__(self, host, port, server_name):
6          self._host = host
7          self._port = port
8          self._server_name = server_name
9          # Словарь для хранения оценок по дисциплинам
10         self._discipline = {}
11     def serve_forever(self):
12         # Создание сокета для сервера
13         serv_socket = socket.socket(
14             socket.AF_INET,
15             socket.SOCK_STREAM
16         )
17         try:
18             # Привязка сокета к хосту и порту
19             serv_socket.bind((self._host, self._port))
20             # Начало прослушивания порта
21             serv_socket.listen()
22             while True:
23                 # Принятие нового соединения
24                 conn, _ = serv_socket.accept()
25                 try:
26                     # Обслуживание клиента
27                     self.serve_client(conn)
28                 except Exception as e:
29                     print('Ошибка обслуживания клиента:', e)

```

Рисунок 16. server часть первая задание №5

В обработке клиентского подключения входит парсинг заголовка, составление ответа и его отправка. Парсинг заголовка: данные запроса, разбиваются на метод, адрес и версию протокола, затем адрес разбивается на адрес и переданные параметры, которые в свою очередь оформляются в словарь.


```

34     def serve_client(self, conn):
35         # Парсинг запроса клиента
36         method, url_address, parameters, ver, headers = self.parse_request(conn)
37         # Обработка запроса и формирование ответа
38         resp = self.handle_request(method, url_address, parameters)
39         # Отправка ответа клиенту
40         self.send_response(conn, resp)
41
42     def parse_request(self, conn):
43         rfile = conn.makefile('rb')
44         raw = rfile.readline(MAX_LINE + 1)
45
46         req_line = str(raw, 'iso-8859-1')
47         req_line = req_line.rstrip('\r\n')
48         words = req_line.split()
49         method, url, ver = words
50
51         url_address, parameters = url.split("?")
52
53         pairs = parameters.split('&')
54
55         parameters = {}
56         for pair in pairs:
57             key, value = pair.split('=')
58             parameters[key] = value
59
60         headers = self.parse_headers(rfile)
61
62         return method, url_address, parameters, ver, headers
63

```

Рисунок 17. server часть вторая задание №5

В конце парсинга запроса идёт парсинг заголовков, который описан в функции `parse_headers`. В её цикле построчно читается файл, переданный из предыдущей функции и до появления пустой строки заголовки записываются в соответствующий массив. После идёт функция для создания ответа `handle_request`. В зависимости от переданного метода она вызывает либо функцию записи новых оценок, либо функцию получения информации по дисциплине.

```

64 def parse_headers(self, rfile):
65     headers = []
66     while True:
67         line = rfile.readline(MAX_LINE + 1)
68
69         if line in (b'\r\n', b'\n', b''):
70             break
71
72         headers.append(line)
73
74     return headers
75
76 def handle_request(self, method, url_address, parameters):
77     print('Обработка запроса')
78     if url_address == '/grades' and method == 'POST':
79         return self.post_grades(parameters)
80
81     if url_address == '/grades' and method == 'GET':
82         return self.get_grades(parameters)

```

Рисунок 18. server часть третья задание №5

В функции записи информации проверяется наличие переданной дисциплины, происходит добавление данных в словарь, объявленный при инициализации и возвращаются атрибуты ответа. Если же нужно получить данные, то построчно создается тело ответа с добавлением необходимых данных в соответствии с переданными параметрами и также возвращаются атрибуты ответа.

```

84 def post_grades(self, parameters):
85     # Обработка POST-запроса для добавления оценки
86     if parameters['discipline'] not in self._discipline.keys():
87         self._discipline[parameters['discipline']] = {}
88
89     self._discipline[parameters['discipline']][parameters['name']] = parameters['grade']
90
91     print(self._discipline)
92
93     return [204, 'Created']
94
95 1 usage
96 def get_grades(self, parameters):
97     content_type = 'text/html; charset=utf-8'
98
99     body = '<html><head></head><body>'
100     body += f'<div>Дисциплина ({parameters["discipline"]})</div>'
101     body += '<ul>'
102     for u in self._discipline[parameters["discipline"]].keys():
103         body += f'<li> {u} {self._discipline[parameters["discipline"]][u]}</li>'
104     body += '</ul>'
105     body += '</body></html>'
106
107     body = body.encode('utf-8')
108     headers = [('Content-Type', content_type),
109               ('Content-Length', len(body))]
109     return [200, 'OK', headers, body]

```

Рисунок 19. server часть четвёртая задание №5

В функции отправки ответа `send_response` создается файл, который будет отправлен клиенту, в него в необходимом порядке вставляются атрибуты из предыдущих функций.

```

111     def send_response(self, conn, resp):
112         wfile = conn.makefile('wb')
113         status_line = f'HTTP/1.1 {resp[0]} {resp[1]}\r\n'
114         wfile.write(status_line.encode('iso-8859-1'))
115
116         if len(resp) > 2:
117             for (key, value) in resp[2]:
118                 header_line = f'{key}: {value}\r\n'
119                 wfile.write(header_line.encode('iso-8859-1'))
120
121         wfile.write(b'\r\n')
122
123         if len(resp) > 3:
124             wfile.write(resp[3])
125
126         wfile.flush()
127         wfile.close()
128
129     # Определение хоста, порта и имени сервера
130     host = socket.gethostname()
131     port = 8080
132     name = 'server'
133     # Создание сервера и запуск его бесконечного цикла обслуживания клиентов
134     serv = HTTPServer(host, port, name)
135     try:
136         serv.serve_forever()
137     except KeyboardInterrupt:
138         pass

```

Рисунок 20. server часть пятая задание №5

```

1  import socket
2
3  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4
5  s.connect((socket.gethostname(), 8080))
6
7  msg = 'POST /grades?discipline=theory_of_neural_network&name=Artiom&grade=5 HTTP/1.1\r\nHost: local\r\nAccept: text/html\r\nUser-Agent: Mozilla/5.0\r\n\r\n'
8  s.send(msg.encode('iso-8859-1'))
9  msg_rcv = s.recv(1000)
10 print(msg_rcv.decode())
11 s.close()
12
13 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14
15 s.connect((socket.gethostname(), 8080))
16
17 msg = 'POST /grades?discipline=theory_of_neural_network&name=Anna&grade=4 HTTP/1.1\r\nHost: local\r\nAccept: text/html\r\nUser-Agent: Mozilla/5.0\r\n\r\n'
18 s.send(msg.encode('iso-8859-1'))
19 msg_rcv = s.recv(1000)
20 print(msg_rcv.decode())
21 s.close()
22
23 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
24
25 s.connect((socket.gethostname(), 8080))
26
27 msg = 'GET /grades?discipline=theory_of_neural_network HTTP/1.1\r\nHost: local\r\nAccept: text/html\r\nUser-Agent: Mozilla/5.0\r\n\r\n'
28 s.send(msg.encode('iso-8859-1'))
29 msg_rcv = s.recv(1000)
30 print(msg_rcv.decode())
31 s.close()

```

Рисунок 21. client задание №5

```
"C:\Users\Artiom\Documents\MEGAsync\Education courses\WEB_ITMO\ITMO_ICT_WebDevelopment_2022-2023\venv\Scripts\python.exe" "C:\Users\A
HTTP/1.1 204 Created

HTTP/1.1 204 Created

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 141

<html><head></head><body><div>Дисциплина (theory_of_neural_network)</div><ul><li>* Artiom 5</li><li>* Anna 4</li></ul></body></html>

Process finished with exit code 0
```

Рисунок 22. Вывод в консоль на стороне клиента