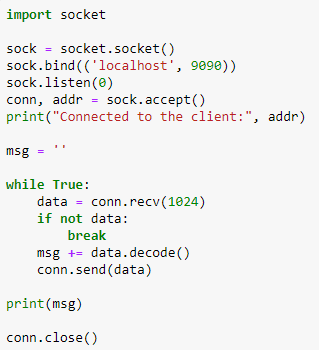
Устимова Снежана, ПИ19-3

Лабораторная работа №3

Простейшие TCP-клиент и эхо-сервер

**Задания для выполнения**

1. Создать простой TCP-сервер, который принимает от клиента строку (порциями по 1 КБ) и возвращает ее. (Эхо-сервер).



1. Сервер должен выводить в консоль служебные сообщения (с пояснениями) при наступлении любых событий:

* Запуск сервера;



* Начало прослушивания порта;



* Подключение клиента;



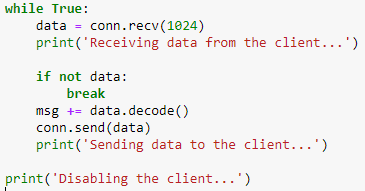
* Прием данных от клиента;



* Отправка данных клиенту;



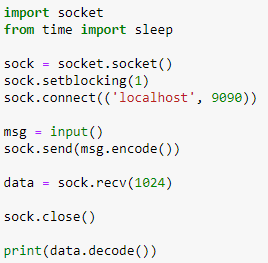
* Отключение клиента;



* Остановка сервера.

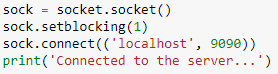


1. Напишите простой TCP-клиент, который устанавливает соединение с сервером, считывает строку со стандартного ввода и посылает его серверу.



1. Клиент должен выводить в консоль служебные сообщения (с пояснениями) при наступлении любых событий:

* Соединение с сервером;



* Разрыв соединения с сервером;



* Отправка данных серверу;

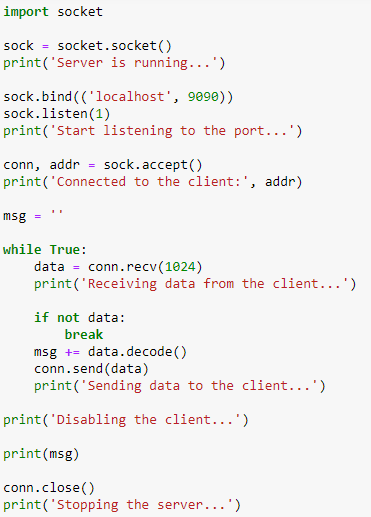


* Прием данных от сервера.

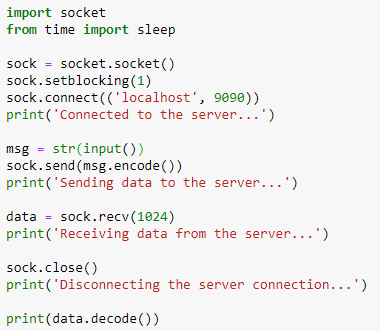


Теперь код выглядит так:

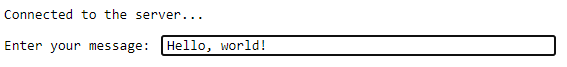
* Сервер

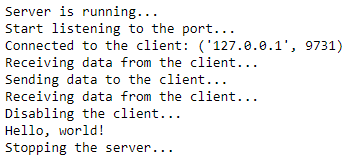


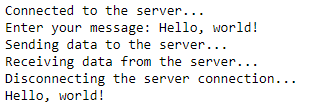
* Клиент



Проверим их работу:







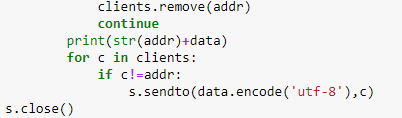
Все прекрасно работает.

**Задания для самостоятельного выполнения**

*Перед тем, как приступить к выполнению, скажу, что на другой дисциплине мы уже прошли данную тему, поэтому далее буду использовать другой, более удобный для реализации код программы.*

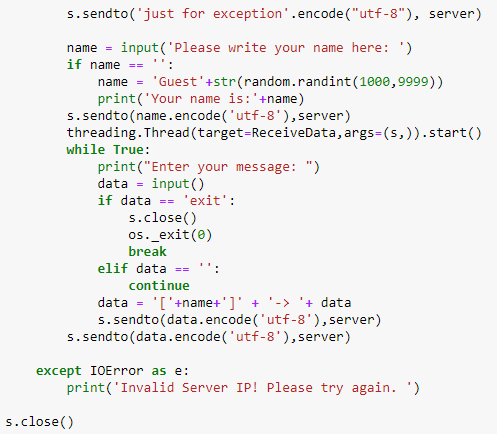
*Изначальный вид сервера:*





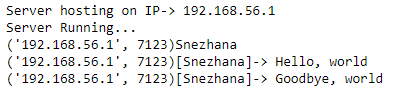
*Код клиента:*



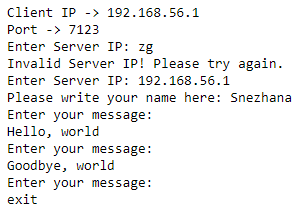


Посмотрим на работу:

(сервер)



(клиент)



*Теперь приступим к пунктам.*

1. Проверьте возможность подключения к серверу с локальной, виртуальной и удаленной машины.

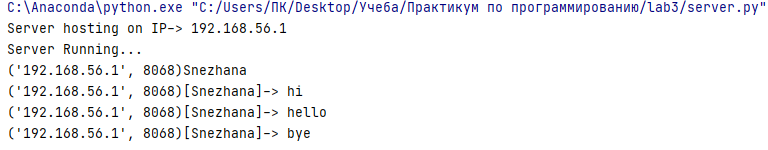
*По непонятной мне причине моя виртуальная машина не видит сервер локальной, несмотря на то, что я прописала в настройках сети сетевой мост и виртуальный адаптер хоста.*

1. Модифицируйте код клиента таким образом, чтобы он читал строки в цикле до тех пор, пока клиент не введет “exit”. Можно считать, что это команда разрыва соединения со стороны клиента.

*Уже реализовано. Проверим на запуске:*

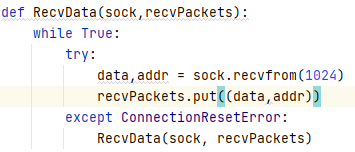
**

*Тем временем на сервере:*

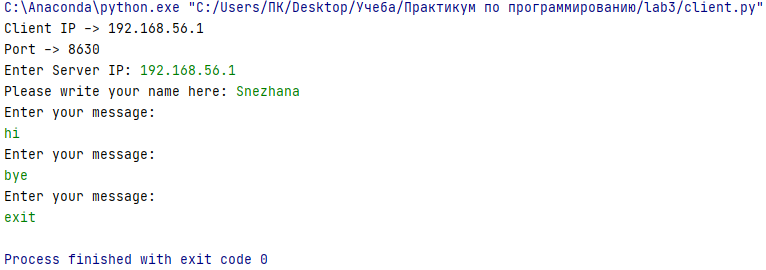
**

1. Модифицируйте код сервера таким образом, чтобы при разрыве соединения клиентом он продолжал слушать данный порт и, таким образом, был доступен для повторного подключения.

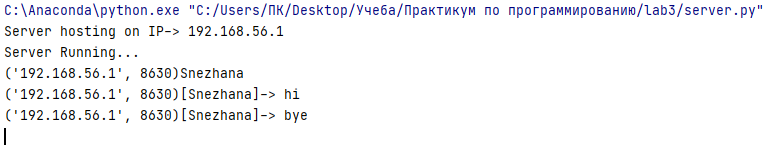
*В коде клиента перехватываем исключение, возникающее в случае принудительного завершения соединения, и вставляем рекурсивную функцию.*

**

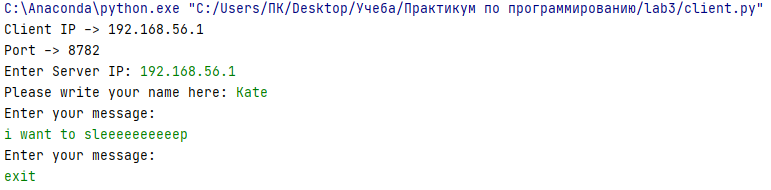
*Проверим работу:*



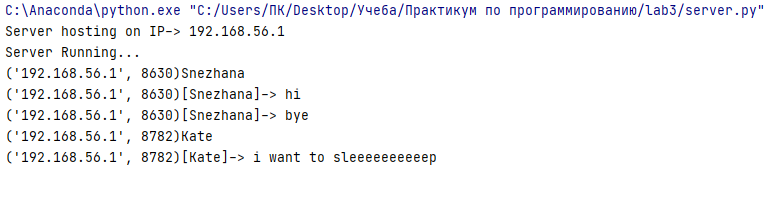
*Мы отправили несколько сообщений и завершили работу, набрав exit. Посмотрим, что происходит на сервере:*

**

*Как видим, он все еще работает. Запустим код клиента еще раз, что убедиться в этом:*

**

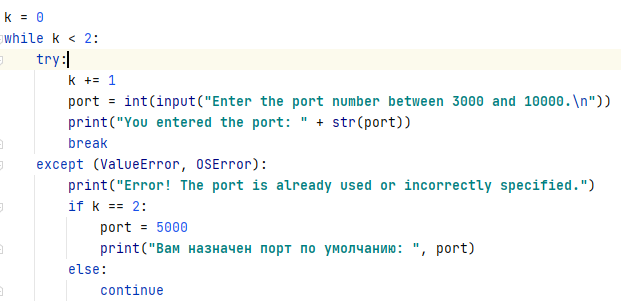
*А что на сервере?*

**

*Он благополучно продолжает принимать данные.*

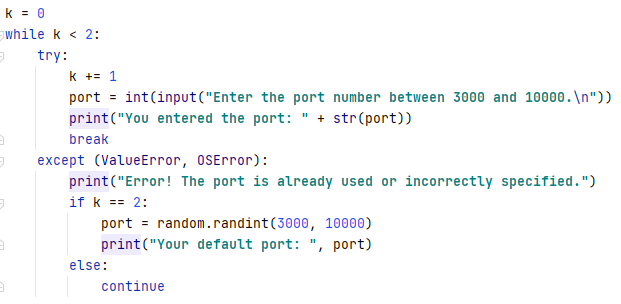
1. Модифицируйте код клиента и сервера таким образом, чтобы номер порта и имя хоста (для клиента) они спрашивали у пользователя. Реализовать безопасный ввод данных и значения по умолчанию.

*В код сервера добавили следующие строки:*

**

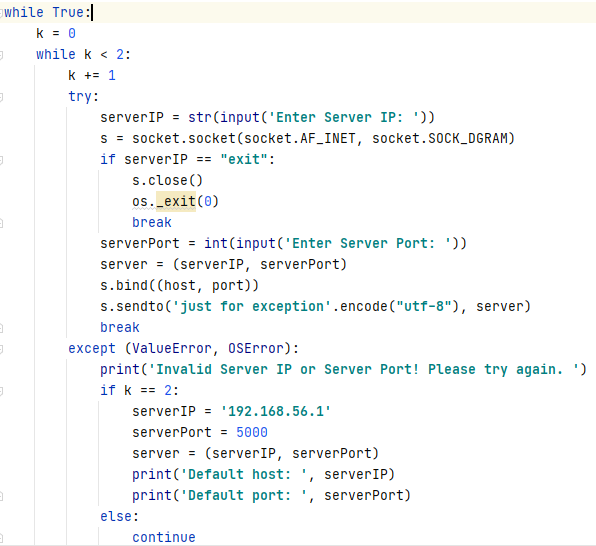
*Таким образом, мы даем пользователю 2 попытки на самостоятельное введение номера порта. Иначе по умолчанию выбираем 5000.*

*Код клиента для собственного порта:*

**

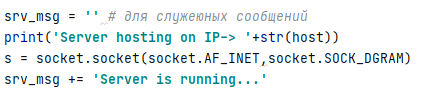
*Аналогично, но порт выбирается рандомным методом от 3000 до 10000.*

*Для данных сервера:*

**

1. Модифицировать код сервера таким образом, чтобы все служебные сообщения выводились не в консоль, а в специальный лог-файл.

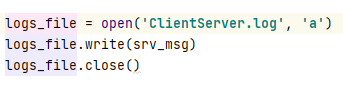
*Создам отдельное хранилище данных для строк и прибавляем к нему каждое служебное сообщение.*

**

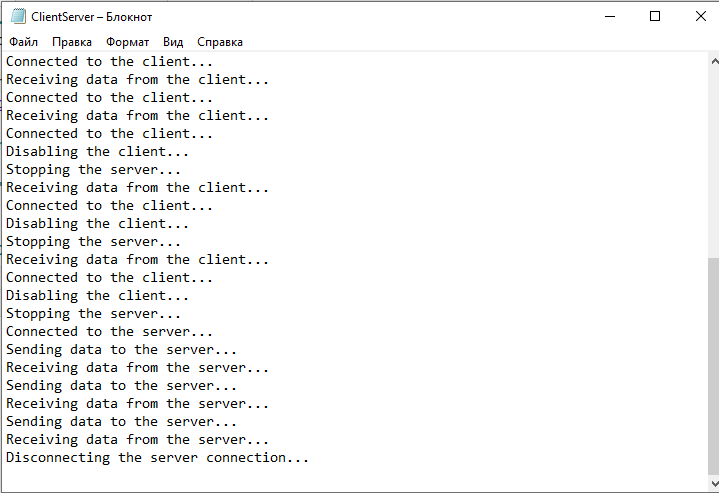
*Далее аналогично с каждым.*

*И так же для клиента.*

*И добавим последние строки для записи в log-файл.*

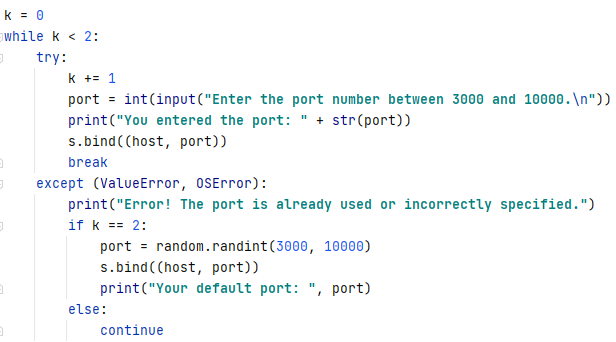
**

*Проверим работу:*

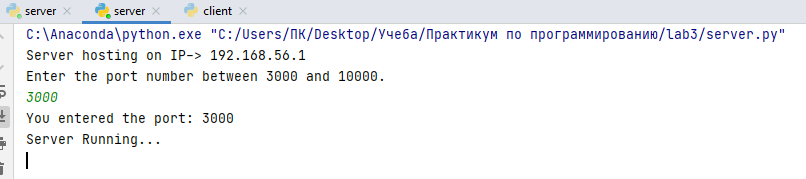
**

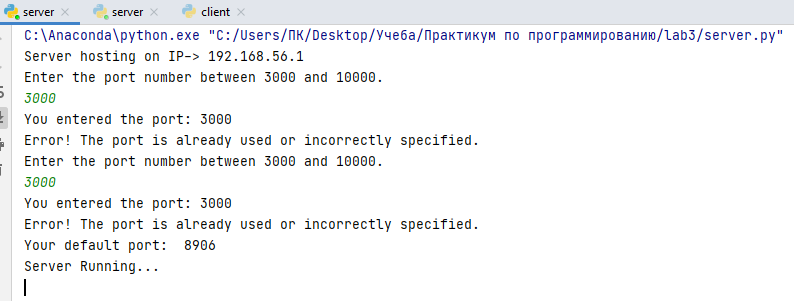
1. Модифицируйте код сервера таким образом, чтобы он автоматически изменял номер порта, если он уже занят. Сервер должен выводить в консоль номер порта, который он слушает.

*Уже реализовано:*

**

*Попробуем запустить второй сервер, выбрав тот же порт, что и первый.*

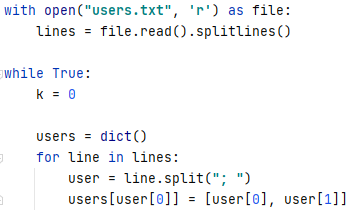
**

**

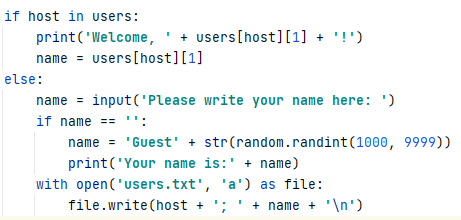
*Так как порт уже занят, программа дала нам вторую попытку, чтобы выбрать другой порт. Но так как пользователь был настойчивым, ему автоматически был назначен другой порт методом рандома.*

1. Реализовать сервер идентификации. Сервер должен принимать соединения от клиента и проверять, известен ли ему уже этот клиент (по IP-адресу). Если известен, то поприветствовать его по имени. Если неизвестен, то запросить у пользователя имя и записать его в файл. Файл хранить в произвольном формате.

*Создадим текстовый файл и словарь для работы:*

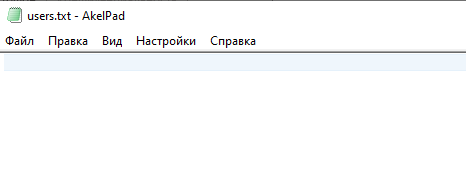
**

*И изменим следующие строки перед вопросом об имени клиента:*

**

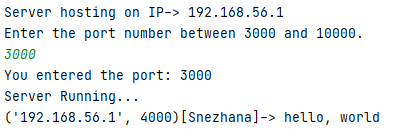
*То есть, если адрес клиента уже записан в книге, то вы здороваемся с ним по ключу, иначе просим пользователи ввести свое имя и записываем его в файл вместе с текущим IP-адресом с разделителем ‘;’.*

*Посмотрим на работу. Пустой файл users.txt:*

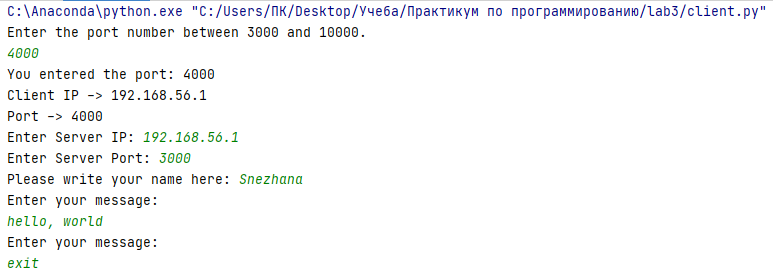
**

*Теперь, пока наши данные еще не записаны, подключимся к серверу и введем пару команд.*

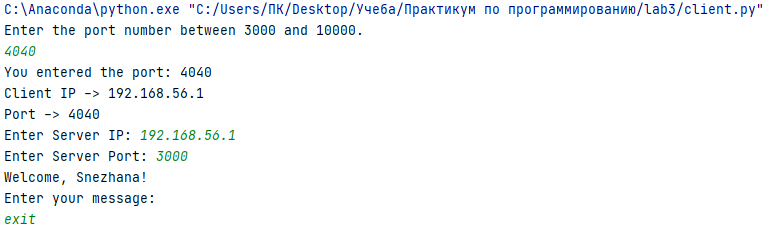
*На сервере:*

**

*У клиента:*

**

*Теперь, завершив этот процесс, попробуем снова подключиться к работающему серверу. По идее нас должны поприветствовать, так как мы не меняли наш текущий ip:*

**

*Все работает!*

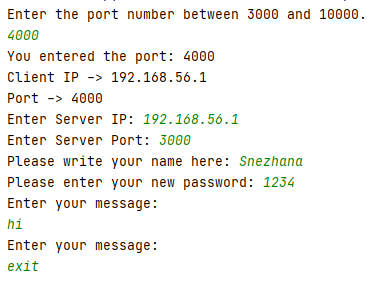
1. Реализовать сервер аутентификации. Похоже на предыдущее задание, но вместе с именем пользователя сервер отслеживает и проверяет пароли. Дополнительные баллы за безопасное хранение паролей. Дополнительные баллы за поддержание сессии на основе токена наподобие cookies.

*Немного изменим словарь и дополним его новыми данными для хранения в файле. Для БЕЗОПАСНОГО хранения паролей я решила использовать хэширование.*

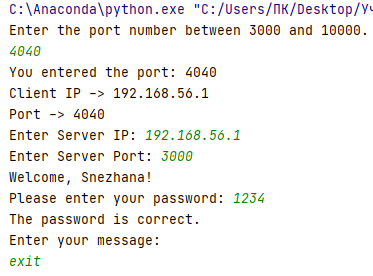
**

*Посмотрим на вывод, предварительно удалив из файла предыдущую историю.*

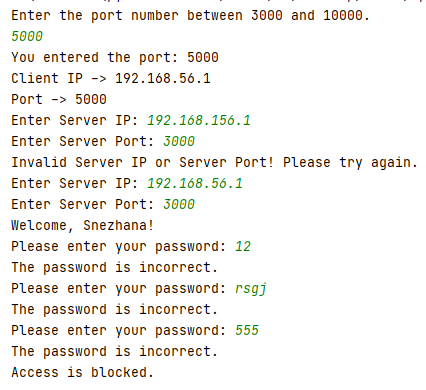
*Первое вхождение:*

**

*Повторное (без ошибки):*

**

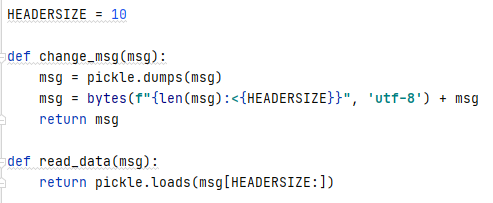
*Мы зашли. Теперь попробуем поиграться и ввести неправильный пароль. Как можно видеть из кода программы, мы предоставляем клиенту 3 попытки.*

**

*После 3неправильных попыток нам заблокировали доступ в систему. А еще я случайно ввела не тот IP-адрес, но ничего. Зато лишний раз проверили работу исключений.*

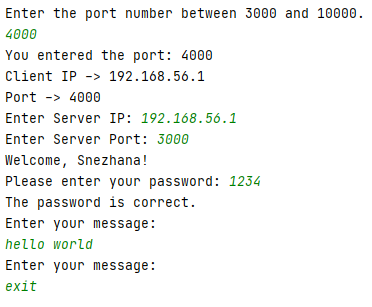
1. Напишите вспомогательные функции, которые реализуют отправку и принятие текстовых сообщений в сокет. Функция отправки должна дополнять сообщение заголовком фиксированной длины, в котором содержится информация о длине сообщения. Функция принятия должна читать сообщение с учетом заголовка. В дополнении реализуйте преобразование строки в байтовый массив и обратно в этих же функциях. Дополнительно оценивается, если эти функции будут реализованы как унаследованное расширение класса socket библиотеки socket.

*Добавим следующие функции и в файл client, и в server. По две функции в оба файла (особенно это пригодится для работы следующего пункта данной методички).*

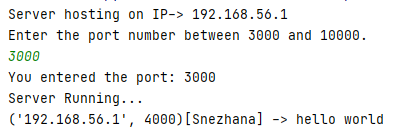


*Далее эта функция будет запускать на местах sendto вместо encode и decode.*

*Проверим работу. Пусть клиент отправит стандартное сообщение:*

**

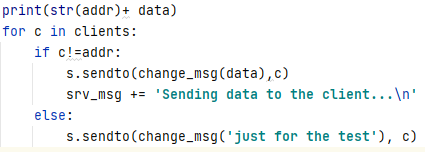
*Теперь посмотрим, правильно ли сервер прочитал сообщение:*

**

*Все корректно!*

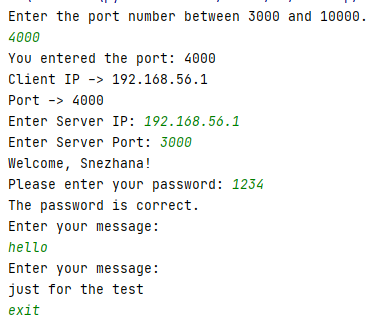
1. Дополните код клиента и сервера таким образом, чтобы они могли посылать друг другу множественные сообщения один в ответ на другое.

*Я не хочу постоянно вводить сообщения для сервера, поэтому просто покажу, где это сделать и введу в том месте фиксированное сообщение «just for the test».*

**

*То есть теперь сервер будет отправлять сообщение не только «всем остальным клиентам», но и самому написавшему. Если мы заменим, к примеру, часть ‘just for the test’ в «s.sendto(change\_msg('just for the test'), c)» на input(), то это и будет расцениваться как ответное сообщение сервера.*

*Посмотрим на работу:*

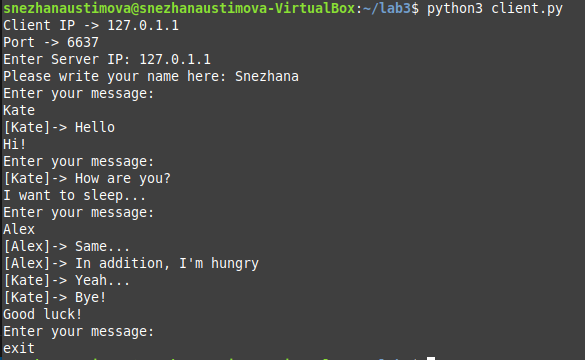
**

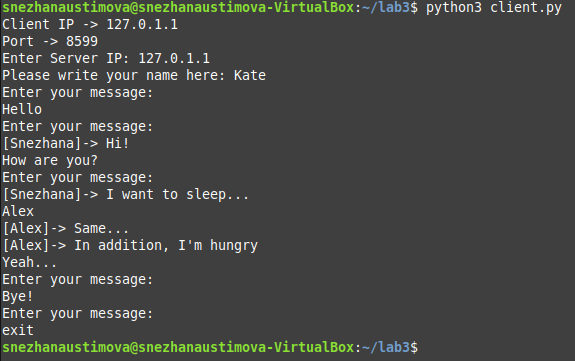
*Как можно заметить, сервер нам ответил.*

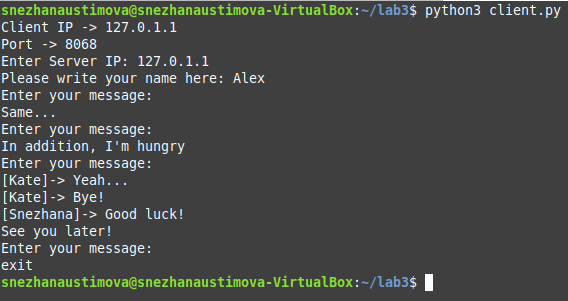
1. Напишите многопользовательский чат. Подсказка: используйте сокеты, основанные на протоколе UDP.

*Так как я с самого начала реализовывала это задание при помощи потоков, то оно сразу ориентировано на работу с несколькими клиентами. Однако после мы модифицировали код программы, что новый клиент у нас считывается только при новом ip-адресе, но я, к сожалению, не могу показать вам работу данного пункта, поскольку не имею иной машины. Тем не менее, если взглянуть на самый изначальный вариант кода, то можно понять, что он будет работать. К примеру, покажу, как это работало, запустив код на ОС linux без занесения ip-адреса в файл.*

*На скриншотах представлена «переписка» между 3 клиентами. Первые два из них (Snezhana и Kate) переписывались еще до того, как пришел Alex. В общем, наглядно:*

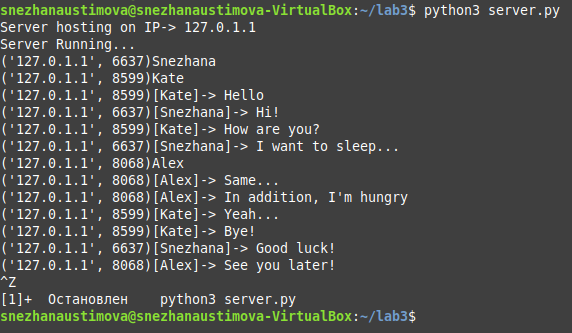
**

**

**

*Каждый клиент добровольно вышел из чата, введя команду «exit».*

*Теперь посмотрим, что происходило на сервере:*

**

*Как мы видим, после выхода всех «собеседников» сервер все еще продолжает свою работу и к нему могут подключиться новые клиенты. Я же принудительно завершила работу.*