Санкт-Петербургский государственный инженерно-экономический университет.

Кафедра вычислительных систем и программирования

Лабораторная работа «Клиент-серверное приложение»

Составитель: Федоров Д.Ю.

http://pycode.ru/edu/

Оглавление

1. Предварительные сведения

Задача заключается в передаче зашифрованного сообщения по прослушиваемой сети.

Две стороны заранее договариваются о ключе шифрования методом Цезаря. На стороне клиента сообщение должно шифроваться и передаваться по сети серверу, на стороне сервера должно происходить дешифрование сообщения и вывод его на экран.

Интерфейс работы с программой на стороне клиента:

```
>>>
Enter your message:
rrr
Enter the key number (1-26)
3
Send ciphertext:  uuu
>>>
```

Пользователь вводит сообщение, которое хочет передать по сети, указывает ключ шифрования. Программа выводит на экран зашифрованное сообщение и отправляет его серверу.

Интерфейс работы с программой на стороне сервера:

```
>>>
Connected client
Enter the key number (1-26)
3
Received ciphertext:  uuu
Plaintext:  rrr
```

```
>>>
```

На экране пользователя со стороны сервера в первую очередь выводится сообщение о том, что установлена связь с клиентом. Далее указывается ключ шифрования и выводятся зашифрованный и открытый тексты.

2. Теоретические сведения

2.1. Работа с сокетами

Применяемая в IP-сетях архитектура клиент-сервер использует IP-пакеты для коммуникации между клиентом и сервером. Клиент отправляет запрос серверу, на который тот отвечает. В случае с TCP/IP между клиентом и сервером устанавливается соединение (обычно с двусторонней передачей данных), а в случае с UDP/IP - клиент и сервер обмениваются пакетами (дейтаграммами) с негарантированной доставкой.

Каждый сетевой интерфейс IP-сети имеет уникальный в этой сети адрес (IP-адрес). Упрощенно можно считать, что каждый компьютер в сети Интернет имеет собственный IP-адрес. При этом в рамках одного сетевого интерфейса может быть несколько сетевых портов. Для установления сетевого соединения приложение клиента должно выбрать свободный порт и установить соединение с серверным приложением, которое слушает (*listen*) порт с определенным номером на удаленном сетевом интерфейсе. Пара IP-адрес и порт характеризуют *сокет* (гнездо) - начальную (конечную) точку сетевой коммуникации. Для создания соединения TCP/IP необходимо два сокета: один на локальной машине, а другой - на удаленной. Таким образом, каждое сетевое соединение имеет IP-адрес и порт на локальной машине, а также IP-адрес и порт на удаленной машине.

Модуль *socket* в Python обеспечивает доступ к интерфейсу *BSD сокетов* и обеспечивает возможность работать с сокетами из Python. Сокеты используют транспортный уровень согласно семиуровневой модели OSI (Open Systems Interconnection, взаимодействие открытых систем).

2.2. Уровни модели OSI

Физический: поток битов, передаваемых по физической линии. Определяет параметры физической линии.

Канальный (Ethernet, PPP, ATM и т.п.): кодирует и декодирует данные в виде потока битов, справляясь с ошибками, возникающими на физическом уровне в пределах физически единой сети.

Сетевой (IP): маршрутизирует информационные пакеты от узла к узлу.

Транспортный (TCP, UDP и т.п.): обеспечивает прозрачную передачу данных между двумя точками соединения.

Сеансовый: управляет сеансом соединения между участниками сети. Начинает, координирует и завершает соединения.

Представления: обеспечивает независимость данных от формы их представления путем преобразования форматов. На этом уровне может выполняться прозрачное (с точки зрения вышележащего уровня) шифрование и дешифрование данных.

Приложений (HTTP, FTP, SMTP, NNTP, POP3, IMAP и т.д.): поддерживает конкретные сетевые приложения. Протокол зависит от типа сервиса.

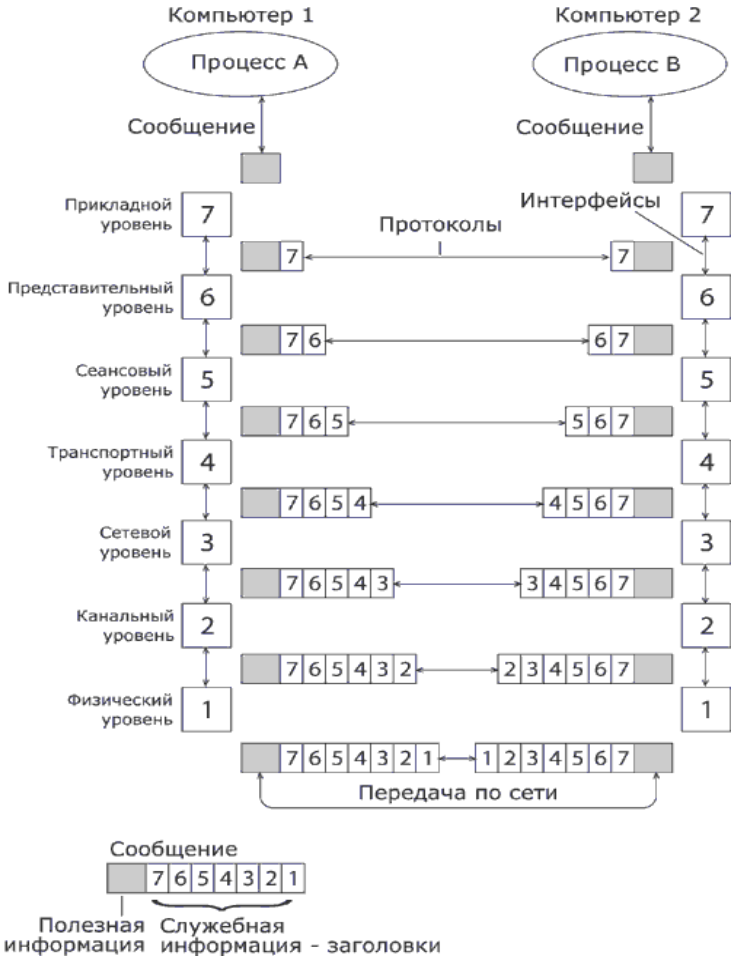

Рисунок. Модель взаимодействия открытых систем ISO/OSI.

Каждый сокет относится к одному из *коммуникационных доменов*. Модуль socket поддерживает домены *UNIX* и *Internet*. Каждый домен подразумевает свое семейство протоколов и адресацию. Данное изложение будет затрагивать только домен *Internet*, а именно протоколы *TCP/IP* и *UDP/IP*, поэтому для указания коммуникационного домена при создании сокета будет указываться константа *socket.AF_INET* (и *socket.SOCK_STREAM* для надежной потокоориентированной службы).

Далее представлены примеры двух программ, использующих протокол TCP/IP для передачи текстовой строки: сервер принимает строку и пересылает ее обратно клиенту.

2.3. Исходный текст программы-сервера

```
# Echo server program

import socket
```

```
HOST = '127.0.0.1'
PORT = 50007
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print 'Connected client'
while 1:
    data = conn.recv(1024)
    if not data:
        break
    else:
        print 'Received[2]: ', data
    conn.send(data)
    print 'Send[3]: ', data
conn.close()
```

2.4. Исходный текст программы-клиента

```
# Echo client program
import socket
HOST = '127.0.0.1'
PORT = 50007
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
data = 'Hello world'
s.send(data)
print 'Send[1]: ', data
data = s.recv(1024)
```

```
s.close()

print 'Received[4]: ', data
```

Прежде всего, нужно запустить сервер, затем, запустив еще одну копию IDLE, запустить в ней клиент. Сервер открывает сокет на локальной машине на порту *50007*, и адресе *127.0.0.1* (метод *bind* связывает локальный сетевой адрес транспортного уровня с сокетом). После этого сервер слушает (*listen*) порт. Когда на порту появляются данные, принимается (*accept*) входящее соединение, создается сокет, соответствующий новому соединению клиента и сервера. Сокет, для которого был вызван *accept*, остается в состоянии *listen* и готов к принятию следующих соединений. Метод *accept* возвращает пару – socket-объект и адрес удаленного компьютера, устанавливающего соединение (пара – IP-адрес, порт на удаленной машине). После этого можно применять методы *recv* и *send* для общения с клиентом. В *recv* задается число байтов в очередной порции, от клиента может прийти и меньшее количество данных.

Код программы-клиента достаточно очевиден. Метод *connec* устанавливает соединение с удаленным хостом (в приведенном примере он расположен на той же машине). Данные передаются методом *send* и принимаются методом *recv* – аналогично тому, что происходит на сервере.

Для реализации поставленной задачи можно воспользоваться модулем шифрования по методу Цезаря: импортировать модуль и воспользоваться необходимыми методами.

3. Литература для дополнительного чтения

1. Документация Python версия 2.7: http://docs.python.org/library/socket.html
2. Сетевые приложения на Python:

   http://www.intuit.ru/department/pl/python/9/1.html
3. В. Олифер, Н. Олифер. Компьютерные сети. Принципы, технологии, протоколы.
4. Алгоритмы работы системных вызовов TCP: от уровня ядра к уровню приложений:

http://www.ibm.com/developerworks/ru/library/au-tcpsystemcalls/index.html

5. Основы сетей передачи данных:

   http://www.intuit.ru/department/network/networkbasics/