**Создание простого многопоточного сервера**

**Цель работы**

Познакомиться с приемами работы с многопоточностью на примере создания сокетного TCP-сервера, способного работать с несколькими клиентами одновременно

**Задания для выполнения**

1. Создать простой эхо-сервер и клиент для него.
2. Модифицировать код сервера таким образом, чтобы при подключении нового клиента создавался новый поток и вся работа с клиентом выполнялась в нем.
3. Проверить возможность подключения нескольких клиентов к этому серверу одновременно.

**Методические указания**

Потоки управления (threads) образуются и работают в рамках одного процесса. В однопоточном приложении (программе, которая не использует дополнительных потоков) имеется только один поток управления. Говоря упрощенно, при запуске программы этот поток последовательно исполняет встречаемые в программе операторы, направляясь по одной из альтернативных ветвей оператора выбора, проходит через тело цикла нужное число раз, выбирается к месту обработки исключения при возбуждении исключения. В любой момент времени интерпретатор Python знает, какую команду исполнить следующей. После исполнения команды становится известно, какой команде передать управление. Эта ниточка непрерывна в ходе выполнения программы и обрывается только по ее завершении.

Теперь можно представить себе, что в некоторой точке программы ниточка раздваивается, и каждый поток идет своим путем. Каждый из образовавшихся потоков может в дальнейшем еще несколько раз раздваиваться. (При этом один из потоков всегда остается главным, и его завершение означает завершение всей программы.) В каждый момент времени интерпретатор знает, какую команду какой поток должен выполнить, и уделяет кванты времени каждому потоку. Такое, казалось бы, незначительное усложнение механизма выполнения программы на самом деле требует качественных изменений в программе - ведь деятельность потоков должна быть согласована. Нельзя допускать, чтобы потоки одновременно изменяли один и тот же объект, результат такого изменения, скорее всего, нарушит целостность объекта.

В следующем примере создается два дополнительных потока, которые выводят на стандартный вывод каждый свое:

import threading

def proc(n):

print "Процесс", n

p1 = threading.Thread(target=proc, name="t1", args=["1"])

p2 = threading.Thread(target=proc, name="t2", args=["2"])

p1.start()

p2.start()

Сначала получается два объекта класса Thread, которые затем и запускаются с различными аргументами. В данном случае в потоках работает одна и та же функция proc(), которой передается один аргумент, заданный в именованном параметре args конструктора класса Thread. Нетрудно догадаться, что метод start() служит для запуска нового потока. Таким образом, в приведенном примере работают три потока: основной и два дополнительных (с именами "t1" и "t2" ).

То же самое можно проделать через наследование от класса threading.Thread с определением собственного конструктора и метода run():

import threading

class T(threading.Thread):

def \_\_init\_\_(self, n):

threading.Thread.\_\_init\_\_(self, name="t" + n)

self.n = n

def run(self):

print "Процесс", self.n

p1 = T("1")

p2 = T("2")

p1.start()

p2.start()

**Контрольные вопросы**

1. Почему однопоточное приложение не может решить задачу одновременного подключения?
2. Чем поток отличается от процесса?
3. Как создать новый поток?
4. Как выделить участок кода так, чтобы он выполнялся в другом потоке?
5. В чем проблема потокобезопасности?
6. Какие методы обеспечения потокобезопасности существуют?

**Дополнительные задания**

1. Модифицировать простой эхо-сервер таким образом, чтобы при подключении клиента создавался новый поток, в котором происходило взаимодействие с ним.

Может подключаться одновременно несколько клиентов, и всем сервер присылает ответные сообщения:

Изображение выглядит как текст

Автоматически созданное описание

1. Реализовать простой чат сервер на базе сервера аутентификации. Сервер должен обеспечивать подключение многих пользователей одновременно, отслеживание имен пользователей, поддерживать историю сообщений и пересылку сообщений от каждого пользователя всем остальным.

Изображение выглядит как текст

Автоматически созданное описание

1. Реализовать сервер с управляющим потоком. При создании сервера прослушивание портов происходит в отдельном потоке, а главный поток программы в это время способен принимать команды от пользователя. Необходимо реализовать следующие команды:
   1. Отключение сервера (завершение программы);
   2. Пауза (остановка прослушивание порта);
   3. Показ логов;
   4. Очистка логов;
   5. Очистка файла идентификации.

Логи (в виде csv таблицы и текстового файла).

Изображение выглядит как стол

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Список юзеров:

Изображение выглядит как стол

Автоматически созданное описание

Теперь чистим логи:

Изображение выглядит как текст

Автоматически созданное описание

И список пользователей:

Изображение выглядит как стол

Автоматически созданное описание