Proxy server

Proxy Server

```
#!/usr/bin/python
# This is a simple port-forward / proxy, written using only the default python
# library. If you want to make a suggestion or fix something you can contact-me
# at voorloop_at_gmail.com
# Distributed over IDC(I Don't Care) license
import socket
import select
import time
import sys


# Changing the buffer_size and delay, you can improve the speed and bandwidth.
# But when buffer get to high or delay go too down, you can broke things
buffer_size = 4096
delay = 0.0001
forward_to = ('localhost', 8081)

class Forward:
    def __init__(self):
        self.forward = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def start(self, host, port):
        try:
            self.forward.connect((host, port))
            return self.forward
        except Exception as e:
            print(">"+ str(e) )
            return False

class TheServer:
    input_list = []
    channel = {}

    def __init__(self, host, port):
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server.bind((host, port))
        self.server.listen(200)

    def main_loop(self):
        self.input_list.append(self.server)
        while 1:
            time.sleep(delay)
```

```
            ss = select.select
            inputready, outputready, exceptready = ss(self.input_list, [], [])
            for self.s in inputready:
                if self.s == self.server:
                    self.on_accept()
                    break

                self.data = self.s.recv(buffer_size)
                if len(self.data) == 0:
                    self.on_close()
                    break
                else:
                    self.on_recv()

    def on_accept(self):
        forward = Forward().start(forward_to[0], forward_to[1])
        clientsock, clientaddr = self.server.accept()
        if forward:
            print(">"+ str(clientaddr) +  "has connected")
            self.input_list.append(clientsock)
            self.input_list.append(forward)
            self.channel[clientsock] = forward
            self.channel[forward] = clientsock
        else:
            print(">"+ "Can't establish connection with remote server.")
            print(">"+ "Closing connection with client side" + str(clientaddr))
            clientsock.close()

    def on_close(self):
        print(">"+ str(self.s.getpeername()) + "has disconnected")
        #remove objects from input_list
        self.input_list.remove(self.s)
        self.input_list.remove(self.channel[self.s])
        out = self.channel[self.s]
        # close the connection with client
        self.channel[out].close()  # equivalent to do self.s.close()
        # close the connection with remote server
        self.channel[self.s].close()
        # delete both objects from channel dict
        del self.channel[out]
        del self.channel[self.s]

    def on_recv(self):
        data = self.data
        # here we can parse and/or modify the data before send forward
        #print(">"+ str(data))
        self.channel[self.s].send(data)

if __name__ == '__main__':
        server = TheServer('', 8080)
```

```
        try:
            server.main_loop()
        except KeyboardInterrupt:
            print(">"+ "Ctrl C - Stopping server")
            sys.exit(1)
```

Revision #1
★Created 2024-03-24 14:39:35 UTC by Hyeon Su Ryu
Updated 2024-03-24 14:40:14 UTC by Hyeon Su Ryu