Eric Craaybeek and Benjamin Tozier  
Network Design Project; Phase 2 Documentation

**NetDesignServer.py**

|  |
| --- |
|  |
|  | #Eric Craaybeek, Ben Tozier #Python Net Design Project Phase 1 |
|  |  |
|  | #code adapted from : |
|  | #"Computer Networking: A Top-Down Approach" by Keith Ross and James Kurose |
|  |  |
|  | from socket import \* |
|  |  |
|  | serverPort = 12000 |
|  | serverSocket = socket(AF\_INET, SOCK\_DGRAM) |
|  | serverSocket.bind(('',serverPort)) |
|  |  |
|  | dstFile = 'dstPic.png' |
|  |  |
|  | print ('The server is ready to receive') |
|  | while 1: |
|  | # Wait here until recieve message from socket |
|  | message, clientAddress = serverSocket.recvfrom(2048) |
|  | # Write local file |
|  | fileWrite = open(dstFile, 'ab') |
|  | fileWrite.write(message) |
|  | fileWrite.seek(2048) |
|  | # If EOF, close the file |
|  | if message == b"": |
|  | fileWrite.close() |

**NetDesignClient.py**

|  |
| --- |
|  |
|  | #Eric Craaybeek, Ben Tozier  #Python Net Design Project Phase 1 |
|  |  |
|  | #code adapted from : |
|  | #"Computer Networking: A Top-Down Approach" by Keith Ross and James Kurose |
|  |  |
|  | from socket import \* |
|  | from tkinter import \* |
|  |  |
|  |  |
|  | global root |
|  |  |
|  | # Size of packets to send |
|  | PacketSize = 1024 |
|  | # Message to signify end of file |
|  | TerminateCharacter = b'' |
|  |  |
|  | ServerName = 'localhost' |
|  | # Port for socket to attach to |
|  | ServerPort = 12000 |
|  |  |
|  | class App(Frame): |
|  | # Tkinter initializing |
|  | def \_\_init\_\_(self, master=None): |
|  | super().\_\_init\_\_(master) |
|  | self.pack() |
|  |  |
|  |  |
|  | self.instructions = Text(height=2, width=15) |
|  | self.instructions.pack() |
|  | self.instructions.insert(END, "Enter name of\n local file.") |
|  |  |
|  |  |
|  | # GUI will have place for string entry entryPath |
|  | # entryPath will be at the bottom of the GUI |
|  | self.entryPath = Entry() |
|  | self.entryPath.pack() |
|  | self.contents = StringVar() |
|  |  |
|  | # Default contents of variable will be null |
|  | self.contents.set('') |
|  | # tell the entry widget to watch this variable |
|  | self.entryPath["textvariable"] = self.contents |
|  |  |
|  | # Begin send\_file member function on press of enter key |
|  | self.entryPath.bind('<Key-Return>', |
|  | self.send\_file) |
|  |  |
|  | ######## Function: |
|  | ######## send\_file |
|  | #### Purpose: |
|  | #### Take input string file name from GUI. Open and send file to server |
|  | ## Paramters: |
|  | ## None |
|  |  |
|  | def send\_file(self, event): |
|  |  |
|  | # Setup socket |
|  | clientSocket = socket(AF\_INET, SOCK\_DGRAM) |
|  |  |
|  | # Get variable obtained via GUI |
|  | srcFile = self.contents.get() |
|  |  |
|  | # Procedure to automatically close window if invalid file is given |
|  | try: |
|  | fileRead = open(srcFile, "rb") |
|  | except FileNotFoundError: |
|  | print(srcFile, "not found") |
|  | self.Quit() |
|  | raise |
|  | except: |
|  | self.Quit() |
|  | raise |
|  |  |
|  | # initial read of file |
|  | message = fileRead.read(PacketSize) |
|  | # loop to read and send packets to the server |
|  | while message != b"": |
|  | packet = message |
|  | clientSocket.sendto(packet, (ServerName, ServerPort)) |
|  | message = fileRead.read(PacketSize) |
|  |  |
|  | # Send a final message to the server to signify end |
|  | clientSocket.sendto(TerminateCharacter, (ServerName, ServerPort)) |
|  |  |
|  | # End by closing the file and the socket |
|  | fileRead.close() |
|  | clientSocket.close() |
|  |  |
|  | ######## Function: |
|  | ######## Quit |
|  | #### Purpose: |
|  | #### Quit and close tkinter GUI window on error |
|  | ## Paramters: |
|  | ## None |
|  | def Quit(self): |
|  | root.destroy() |
|  |  |
|  |  |
|  |  |
|  | root = Tk() |
|  | app = App(master=root) |
|  | # Run the tkinter GUI app |
|  | app.mainloop() |

The function of this program is to send the data of a file from the client to the UDP server over the port 12000. The server will use these packets to replicate the file at a potentially remote location.

This process begins by running the NetDesignServer.py module. A Socket is created for the server, of address family “AF\_INET” (IPv4) and socket type “SOCK\_DGRAM”, or a UDP socket. This socket is then bound using its own address as the hostname and 12000 as the port number. This means that any message addressed to that port will then pass through to the socket of this server. After this, the server enters an endless loop that receives messages and the address of the client that sent the message. The messages can be up to a length of 2048 bytes. The client program sends packets that are smaller than this. Upon the arrival of the first packet a file is opened as ‘ab’ (append;bytes). This allows packets to append to the file easily while avoiding any usage of strings. Each consequent packet is appended to this file until an empty packet is received indicating an EOF on the side of the client. At this point the server closes the file and begins waiting again for another connection.

In the Client program, a similar setup occurs for the socket. The servers address is specified at the beginning of the program. The address will be ‘localhost’ if both modules are being run on the same machine. If each module is being run on separate machines this address will be the IP address of the server machine.

The GUI is then initialized using the tkinter toolkit. This GUI takes user input for a filename. This filename will be the source file to be transferred.

A function called send\_file() uses the path given from the GUI. The file is opened as ‘rb’ (read; bytes) to remove any encoding or decoding of strings. The client opens and reads the first 1024 bytes of the file, then sends these bytes as a packet to the server at port 12000. It repeatedly sends these packets until it an EOF is read from the source file (b’’) at which point the loop exits. The files and socket are then closed.

**Classes and Data Types  
socket**

A socket in this regard, is a class that takes in an addressing protocol (in this case IPv4) and a messaging protocol (in this case UDP.) The resulting structure then can be used as an endpoint for the server to access the port (12000) being used for the transactions.

The class has the member functions bind, recvfrom, sendto, and close used in this project. Bind associates a port with the socket, allowing the socket to access the traffic through that port. Recvfrom allows a socket to receive data of a size in bytes through its assigned port. Sendto is the inverse of the Recvfrom function, writing to the port instead of reading. Close simply closes the socket.

**Tkinter**

Tkinter is the standard and most widely used toolkit for python.