Socket Programming Assignment

Jordan Pyott; 87433186

August 16, 2020

Usage

Running Server

```
python3 server.py port_eng port_mao, port_ger

PARAMETERS [PORT] NO PREFIX
    PORT_English       : Port hosting English service
    PORT_Maori         : Port hosting English service
    PORT_German        : Port hosting English service

OPTIONS [-] PREFIX
    -h --help      : Shows instructions and usage of `server.py`
    -v --verbose   : Shows clients connecting to server
```

Running Client

```
python3 client.py MSG HOST PORT

PARAMETERS [PORT] NO PREFIX
    MESSAGE     : Users request from server `date` or `time`
    HOST        : Hostname of the server
    PORT        : Port of the server

OPTIONS [-] PREFIX
    -h --help      : Shows instructions and usage of `server.py`
    -v --verbose   : Shows clients connecting to server
```

Server Source Code

```python
import socket
import select
import argparse
import sys
import datetime

def start_server(PORT_english, PORT_maori, PORT_german, verbose):
    """
    This function is in essence the main function, it has many sub-functions
    that are used to get certain values to parse into the body of this function,
    at the bottom of this function you will find the server information, note you
    can start this function with the --verbose flag in order to see the options and
    arguments, run `server.py --help` for more information`.
    """

    def check_port(PORT):
        """
        Checks the port and makes sure it complies to requirements (1024 - 64000)
        """
        if PORT < 1024 or PORT > 64000:
            print("The Port number is not within specified range (1024 - 64000)")
            return False
        return True
```

```
def packetCheck(packet):
    """
    checks the integrity of the packet, and makes sure that it is formatted correctly
    """
    info = [packet[i:i+2] for i in range(0, len(packet), 2)]
    MagicNo = int.from_bytes(info[0], 'big')
    PacketType = int.from_bytes(info[1], 'big')
    RequestType = int.from_bytes(info[2], 'big')
    if MagicNo != 0x497E:
        print("Magic wrong")
    if PacketType != 0x0001:
        return False
    if RequestType != 0x0001 and RequestType != 0x0002:
        return False
    return True

def checkRequestType(packet):
    """ Checks to see if the user wants the `date` or `time` """
    info = [packet[i:i+2] for i in range(0, len(packet), 2)]
    RequestType = int.from_bytes(info[2], 'big')
    if RequestType == 0x0001:
        return 'date'
    elif RequestType == 0x0002:
        return 'time'
    else:
        return -1

def getDate(sock):
    """
    Allows user to form the packet the `date` string to be returned to the client
    """
    months = {
        'english': ["January", "February", "March", "April", "May", "June", "July",
        "August", "September", "October", "November", "December"],
        'maori': ["Kohitatea", "Hui-tanguru", "Poutu¯-te-rangi", "Paenga-whawha", "Haratua", "Pipiri",
        "Hongongoi", "Here-turi-koka", "Mahuru", "Whiringa-a-nuku", "Whiringa-a-rangi", "Hakihea"],
        'german': ["Januar", "Februar", "Marz", "April", "Mai", "Juni", "Juli",
        "August", "September", "Oktober", "November", "Dezember"]
    }

    MagicNo = 0x497E.to_bytes(2, 'big')
    PacketType = 0x0002.to_bytes(2, 'big')
    if sock is s_english:
        LanguageCode = 0x0001
        flag = 'english'
    elif sock is s_maori:
        LanguageCode = 0x0002
        flag = 'maori'
    elif sock is s_german:
        LanguageCode = 0x0003
        flag = 'german'
    date = datetime.datetime.today()
    LanguageCode = LanguageCode.to_bytes(2, 'big')
    year = date.year.to_bytes(2, 'big')
    language_months = months[flag]
    chosen_month = language_months[(date.month - 1)]
    month = date.month.to_bytes(1, 'big')
    day = date.day.to_bytes(1, 'big')
    hour = date.hour.to_bytes(1, 'big')
    minute = date.minute.to_bytes(1, 'big')
    if flag == 'english':
        text = "Today's date is {} {}, {}".format(chosen_month, date.day, date.year)
    elif flag == 'maori':
```

```
        text = "Ko te ra o tenei ra ko {} {}, {}".format(chosen_month, date.day, date.year)
    else:
        text = "Heute ist der {} {}, {}".format(chosen_month, date.day, date.year)

    lengthNow = len(text)
    length = lengthNow.to_bytes(1, 'big')

    bytelist = [MagicNo, PacketType, LanguageCode, year, month, day, hour, minute, length]

    out = bytearray()

    for byteset in bytelist:
        out += byteset

    out.extend(text.encode("utf-8"))

    return out

def getTime(sock):
    """
    Server uses this function to form the `time` that will be outputted to the client.
    """
    MagicNo = 0x497E.to_bytes(2, 'big')
    PacketType = 0x0002.to_bytes(2, 'big')
    if sock is s_english:
        LanguageCode = 0x0001
        flag = 'english'
    elif sock is s_maori:
        LanguageCode = 0x0002
        flag = 'maori'
    elif sock is s_german:
        LanguageCode = 0x0003
        flag = 'german'
    date = datetime.datetime.today()
    LanguageCode = LanguageCode.to_bytes(2, 'big')
    year = date.year.to_bytes(2, 'big')
    month = (date.month).to_bytes(1, 'big')
    day = date.day.to_bytes(1, 'big')
    hour = date.hour.to_bytes(1, 'big')
    minute = date.minute.to_bytes(1, 'big')
    if flag == 'english':
        text = f"The current time is {date.hour}:{date.minute}"
    elif flag == 'maori':
        text = f"Ko te wa o tenei wa {date.hour}:{date.minute}"
    else:
        text = f"Die Uhrzeit ist {date.hour}:{date.minute}"

    lengthNow = len(text)
    length = lengthNow.to_bytes(1, 'big')

    bytelist = [MagicNo, PacketType, LanguageCode, year, month, day, hour, minute, length]

    out = bytearray()

    for byteset in bytelist:
        out += byteset

    out.extend(text.encode('utf-8'))

    return out

"""
Here is where we implement the Socket API. This is the main setup for the server to receive and send
the appropriate packets.
```

```
    """
    IP = socket.gethostbyname('localhost')

    s_english = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s_maori = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s_german = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    server_sockets = [[s_english, PORT_english], [s_maori, PORT_maori], [s_german, PORT_german]]

    for sock, port in server_sockets:
        if check_port(port):
            sock.bind((IP, port))
        else:
            print("Server closed: port {} isnt within range (1024 - 64000)".format(port))
            return -1

    sockets = [s_english, s_maori, s_german]

    while True: # while there is a connection
        try:
            read, write, exception = select.select(sockets, [], [])
            for s in read:
                packet, source = s.recvfrom(48)
                if verbose:
                    print(f"Packet recieved from {source}")
                if packetCheck(packet):
                    if checkRequestType(packet) == 'date':
                        msg = getDate(s)
                        s.sendto(msg, source)
                    elif checkRequestType(packet) == 'time':
                        msg = getTime(s)
                        s.sendto(msg, source)
                    else:
                        msg = "Packet discarded"
                        msg = msg.encode("utf-8")
                        s.sendto(bytes(msg))
                else:
                    msg = "Packet discarded"
                    msg = msg.encode("utf-8")
                    s.sendto(bytes(msg))
                    s.close()
        except KeyboardInterrupt:
                print ("")
                print ("Closing Sockets...")
                for s in sockets:
                    s.close()
                print("Server Closed")
                sys.exit()
                break


def Main():
    """ Main function: calls to start the server, provides options for the user """
    parser = argparse.ArgumentParser()
    parser.add_argument("PORT_English", help="The Port number to grab date/time in English.", type=int)
    parser.add_argument("PORT_Maori", help="The Port number to grab date/time in Te Aro Maori.", type=int)
    parser.add_argument("PORT_German", help="The port number to grab date/time in German.", type=int)
    parser.add_argument("-v", "--verbose", action="store_true", \
                help="verbose output: view client requests and where they are comming from")

    args = parser.parse_args()
    if args.verbose:
        start_server(args.PORT_English, args.PORT_Maori, args.PORT_German, verbose=True)
    else:
```

```
        start_server(args.PORT_English, args.PORT_Maori, args.PORT_German, verbose=False)

if __name__ == "__main__":
    Main()
```

Client Source Code

```
import socket
import argparse

def start_client(DATE, HOST, PORT, verbose):
    """
    This function is the main function in order to setup the client. The function
    contains many sub-functions used for finding parameters nessasary to display
    the desired output, the body of this function contains the steps to produce a
    client and calls the nessasary sub-functions to send a request to the server.
    """
    try:
        IP = socket.gethostbyname(HOST)
    except socket.gaierror:
        print(f"Hostname not found: Could not connect to ({HOST}:{PORT})")
        return -1

    def checkInputs(DATE, IP, PORT):
        """ Checks the intergrity of inputs and if they comply to the specifications """
        if not IP:
            print("The Hostname is invalid")
            return False
        if PORT < 1024 or PORT > 64000:
            print("The Port number is not within specified range (1024:64000)")
            return False
        if DATE != 'date' and DATE != 'time':
            print("MSG parameter must be set to `date` or `time`")
            return False
        return True

    def decrypt_message(packet):
        """
        Takes in a packet in the form of a byte array, and decrypts it to pull the relevent data
        that will later be displayed to the client.

        Note: This has a --verbose flag to get the full contents of the packet.
        """
        info = [packet[i:i+1] for i in range(0, len(packet), 1)]
        if len(info) < 13:
            print("Packet does not include minimum headersize")
            return -1

        MagicNo = int.from_bytes(info[0] + info[1], 'big')

        if MagicNo != 0x497E:
            print("MagicNo is incorrect: `{}` recieved, must equal `0x497E`".format(MagicNo))
            return -1
        PacketType = int.from_bytes(info[2] + info[3], 'big')
        if PacketType != 0x0002:
            print("PacketType is incorrect: `{}` received, must equal `0x0002`".format(PacketType))
        LanguageCode = int.from_bytes(info[4] + info[5], 'big')
        if LanguageCode < 0x0001 or LanguageCode > 0x0003:
            print("LanguageCode is incorrect: `{}` received, must be within range (1, 3)".format(LanguageCode))
            return -1
        Year = int.from_bytes(info[6] + info[7], 'big')
        if Year > 2100:
            print("Year is incorrect: `{}` received, must be below 2100".format(Year))
```

```
        return -1
    Month = int.from_bytes(info[8], 'big')
    if Month < 1 or Month > 12:
        print("Month is incorrect: `{}` received, must be between 1 and 12".format(Month))
        return -1
    Day = int.from_bytes(info[9], 'big')
    if Day < 1 or Day > 31:
        print("Day is incorrect: `{}` received, must be between 1 and 31".format(Day))
        return -1
    Hour = int.from_bytes(info[10], 'big')
    if Hour < 0 or Hour > 23:
        print("Hour is incorrect: `{}` received, must be within range (0, 23)".format(Hour))
        return -1
    Minute = int.from_bytes(info[11], 'big')
    if Minute < 0 or Minute > 59:
        print("Minute is incorrect: `{}` received, must be within range (0, 59)".format(Minute))
        return -1
    Length = int.from_bytes(info[12], 'big')
    text = bytearray()
    for i in range(13, len(info)):
        text += info[i]
    text = text.decode('utf-8')

    if len(info) != 13 + Length:
        print("Length of packet does not match packet received")
        return -1

    if verbose:
        print("----------------------------------------")
        print(f"MagicNo: {hex(MagicNo)}")
        print(f"PacketType: {hex(PacketType)}")
        print(f"LanguageCode: {hex(LanguageCode)}")
        print(f"Year: {Year}")
        print(f"Month: {Month}")
        print(f"Day: {Day}")
        print(f"Hour {Hour}")
        print(f"Minute: {Minute}")
        print(f"Length: {Length}")
        print(f"Text: {text}")
        print("----------------------------------------")
        print("")

    return text

def format_request(Date):
    """
    Formats the packet into a byte array to send to the server.
    """
    MagicNo = 0x497E
    PacketType = 0x0001
    if Date == 'date':
        RequestType = 0x0001
    elif Date == 'time':
        RequestType = 0x0002
    else:
        return -1
    bytelist = [MagicNo.to_bytes(2, 'big'), PacketType.to_bytes(2, 'big'), RequestType.to_bytes(2, 'big')]

    arrayBytes = bytearray()

    for x in bytelist:
        arrayBytes += x

    return arrayBytes
```

```
    if checkInputs(DATE, IP, PORT):
        """ Checks if the input date/time is valid """
        request_packet = format_request(DATE)
        if request_packet == -1:
            print("The `date` parameter must be set to either `date` or `time`")
            return -1
        else:
            s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            s.settimeout(1)
            s.sendto(request_packet, (IP, PORT))

            complete_message = bytearray()

        while True:
            try:
                msg, source = s.recvfrom(1024)

                if len(msg) <= 0:
                    break

                complete_message += msg

            except socket.timeout:
                print(f"Client timeout: Could not connect to ({HOST}:{PORT})")
                break

            except socket.error:
                print(f"Client timeout: Could not connect to ({HOST}:{PORT})")
                break

            result = decrypt_message(complete_message)
            if result != -1:
                print(result)
                break
            s.close()
            return result


def Main():
    parser = argparse.ArgumentParser()
    parser.add_argument("MSG", help="The message to receive from server must be `date` or `time`", type=str)
    parser.add_argument("HOST", help="The Hostname to connect to", type=str)
    parser.add_argument("PORT", help="The Port number to connect to", type=int)
    parser.add_argument("-v", "--verbose", action="store_true", \
                                help="verbose output: full output of packet received")

    args = parser.parse_args()

    if args.verbose:
        start_client(args.MSG, args.HOST, args.PORT, verbose=True)
    else:
        start_client(args.MSG, args.HOST, args.PORT, verbose=False)

if __name__ == "__main__":
    Main()
```