server.py

```
##################################
# COSC264 Sockets Assignment 2018 - server.py
# Author: Ambrose Ledbrook
# ID: 79172462
##################################


##################################
# Usage:
# python3 server.py English_port Maori_port German_port
##################################


# Importing used modules
import datetime
import socket as soc
import sys
from select import select


# Defining constants
MAGIC_NUMBER = 0x497E
REQUEST_PACKET = 0x0001
RESPONSE_PACKET = 0x0002
DATE_REQUEST = 0x0001
TIME_REQUEST = 0x0002
ENGLISH_CODE = 0x0001
MAORI_CODE = 0x0002
GERMAN_CODE = 0x0003


# Defining the months of the year in all three languages
english_months = ["January", "February", "March", "April", "May", "June", "July", "August", "September",
                  "October", "November", "December"]
maori_months = ["Kohitātea", "Hui-tanguru", "Poutū-te-rangi", "Paenga-whāwhā", "Haratua", "Pipiri",
                "Hōngongoi", "Here-turi-kōkā", "Mahuru", "Whiringa-ā-nuku", "Whiringa-ā-rangi", "Hakihea"]
german_months = ["Januar", "Februar", "März", "April", "Mai", "Juni", "Juli", "August", "September",
                 "Oktober", "November", "Dezember"]


def get_time():
    """
    Getting the current date and time through a system call
    :return: A list holding the current date and time
    """
    time = datetime.datetime.now()
    year = time.year
    month = time.month
    day = time.day
    hour = time.hour
    minute = time.minute
    return [year, month, day, hour, minute]


def textual_date(year, month, day, lang_code):
    """
    Creating a textual representation of the date in the desired language
    :param year: The current year
    :param month: The current month
    :param day: The current day
    :param lang_code: The code fo the language that the text needs to be in
    :return: The textual representation of the date
    """
    if lang_code == ENGLISH_CODE:
        date_text = "Today's date is {0} {1}, {2}".format(english_months[month-1], day, year)
    elif lang_code == MAORI_CODE:
        date_text = "Ko te ra o tenei ra ko {0} {1}, {2}".format(maori_months[month-1], day, year)
    else:
        date_text = "Heute ist der {0}. {1} {2}".format(day, german_months[month-1], year)
    return date_text


def textual_time(hour, minute, lang_code):
    """
    Creating a textual representation of the time in the desired language
    :param hour: The current hour
    :param minute: The current minute
```

```
    :param lang_code: The code fo the language that the text needs to be in
    :return: The textual representation of the time
    """
    # If-Else structure is used below for when the minute is between 0 and 9,
    # When this is true a leading zero is added to the minute field
    if lang_code == ENGLISH_CODE:
        if minute < 10:
            time_text = "The current time is {0}:0{1}".format(hour, minute)
        else:
            time_text = "The current time is {0}:{1}".format(hour, minute)
    elif lang_code == MAORI_CODE:
        if minute < 10:
            time_text = "Ko te wa o tenei wa {0}:0{1}".format(hour, minute)
        else:
            time_text = "Ko te wa o tenei wa {0}:{1}".format(hour, minute)
    else:
        if minute < 10:
            time_text = "Die Uhrzeit ist {0}:0{1}".format(hour, minute)
        else:
            time_text = "Die Uhrzeit ist {0}:{1}".format(hour, minute)
    return time_text


def process_ports(args):
    """
    Processes the command line arguments to get the three port numbers to
    be used with the three UDP sockets
    :param args: The command line arguments
    :return: A list of the tree port numbers
    """
    text = None
    # Checking the correct number of arguments was passed
    if len(args) != 3:
        text = "Invalid number of ports entered"
    else:
        # Checking that the ports passed are all ints
        try:
            a, b, c = int(args[0]), int(args[1]), int(args[2])
        except ValueError:
            text = "Invalid port type"
        # Checking that the ports are in the correct range
        for port in [a, b, c]:
            if port < 1024 or port > 64000:
                text = "Invalid ports, ports must range from 1024 to 64000"
        # Checking that the ports are all unique
        if a == b or a == c or b == c:
            text = "Invalid ports, ports must be unique"
    if not text:
        # Ports entered are valid server will now wait for requests
        print("-----------------------------")
        print("-----------------------------")
        print("Port number {0} for text in English".format(a))
        print("Port number {0} for text in Maori".format(b))
        print("Port number {0} for text in German".format(c))
        print("-----------------------------")
        return [a, b, c]
    else:
        # Ports entered are invalid, an error message is output
        # along with instructions on how to use the server
        print("*****************************")
        print(text)
        print("*****************************")
        print("Usage: python3 server.py English_port Maori_port German_port")
        print("Program will now exit")
        sys.exit()


def decode_packet(pkt):
    """
    Checking that the received request packet is valid
    :param pkt: The received packet
    :return: 0 if the packet is valid, 1 if the packet is invalid
    """
    text = None
    # Checking if any of the packet fields are invalid
    if len(pkt) != 6: text = "Packet is of invalid length"
    elif ((pkt[0] << 8) + pkt[1]) != MAGIC_NUMBER: text = "Magic number is invalid,"
    elif ((pkt[2] << 8) + pkt[3]) != REQUEST_PACKET: text = "Packet type invalid"
    elif ((pkt[4] << 8) + pkt[5]) not in [TIME_REQUEST, DATE_REQUEST]: text = "Request type invalid"
    if text:
        # Outputting error message as the packet is invalid
        print("*****************************")
```

```
        print("{0}, packet will be discarded".format(text))
        print("*****************************")
        return 1
    else:
        # Returning 0 as the packet is valid and processing can continue
        return 0


def get_lang(sock, sockets):
    """
    Finds out which language the client wants the textual field in
    :param sock: The socket that the request was received on
    :param sockets: A list of all the sockets opened by the server
    :return: The language code corresponding to the language that the client wants
    """
    if sock == sockets[0]:
        return ENGLISH_CODE
    elif sock == sockets[1]:
        return MAORI_CODE
    else:
        return GERMAN_CODE


def handle_packet(pkt, lang_code):
    """
    Finds out the request type of the received packet
    :param pkt: The received packet
    :param lang_code: The language that the client wants the textual
    field in
    :return: The response packet ready to be sent made by using makeResponse()
    """
    # Getting the request type of the packet
    request = (pkt[4] << 8) + pkt[5]
    if request == 0x0001:
        return make_response(True, lang_code)
    elif request == 0x0002:
        return make_response(False, lang_code)
    else:
        # Output error message as request type is invalid
        print("*****************************")
        print("Invalid request type, packet will be discarded")
        print("*****************************")
        return None


def make_response(request_flag, lang_code):
    """
    Makes a response packet using the passed parameters
    :param request_flag: Flag holding if the client wants the date or time
    :param lang_code: Holds the language that the client wants to receive the
    textual field in
    :return: The response packet ready to be sent
    """
    # Getting the textual component of the response
    time = get_time()
    if request_flag:
        text = textual_date(time[0], time[1], time[2], lang_code)
    else:
        text = textual_time(time[3], time[4], lang_code)
    # Encoding the text to be sent to the client
    encoded_text = text.encode('utf-8')
    text_length = len(encoded_text)
    # Checking encoded text is of an allowed length
    if text_length > 255:
        print("Textual field to long, packet will be discarded")
        return None
    # Outputting data of the request packet
    request = "time"
    if request_flag: request = "date"
    lang = "English"
    if lang_code == MAORI_CODE: lang = "Maori"
    elif lang_code == GERMAN_CODE: lang = "German"
    print("-----\nClient requested the {0} in {1}".format(request, lang))
    print("-----")
    # Creating and filling the response packet
    response = bytearray(13 + len(encoded_text))
    response[0:2] = MAGIC_NUMBER.to_bytes(2, "big", signed=False)
    response[2:4] = RESPONSE_PACKET.to_bytes(2, "big", signed=False)
    response[4:6] = lang_code.to_bytes(2, "big", signed=False)
    response[6:8] = time[0].to_bytes(2, "big", signed=False)
    response[8] = time[1]
    response[9] = time[2]
```

```
    response[10] = time[3]
    response[11] = time[4]
    response[12] = text_length
    # Adding the text to the end of the packet
    index = 13
    for i in encoded_text:
        response[index] = i
        index += 1
    # Returning the packet to be sent
    return response


def create_sockets(english_port, maori_port, german_port):
    """
    Opens three UDP sockets and binds them tto the three ports passed
    :param english_port: Port number to be bound to the socket used to get
    a textual representation in English
    :param maori_port: Port number to be bound to the socket used to get
    a textual representation in Maori
    :param german_port: Port number to be bound to the socket used to get
    a textual representation in German
    :return: A list of the three UDP sockets use by the server
    """
    # Opening three UDP sockets
    english_socket = soc.socket(soc.AF_INET, soc.SOCK_DGRAM)
    maori_socket = soc.socket(soc.AF_INET, soc.SOCK_DGRAM)
    german_socket = soc.socket(soc.AF_INET, soc.SOCK_DGRAM)
    try:
        # Binding the three sockets to their ports
        english_socket.bind(('', english_port))
        maori_socket.bind(('', maori_port))
        german_socket.bind(('', german_port))
    except soc.error:
        # Outputting error message as socket binding failed
        print("******************************")
        print("Binding sockets to ports failed, program will terminate")
        print("******************************")
        sys.exit()
    return[english_socket, maori_socket, german_socket]


def wait(sockets):
    """
    Server loops endlessly waiting for requests from the client
    :param sockets: List holding the three UDP sockets used by the server
    """
    while True:
        reads, writes, exceps = select(sockets, [], [], 15.0)
        if len(reads) != 0:
            # A request has been received
            for sock in reads:
                data, address = sock.recvfrom(1024)
                print("------------------------------")
                print("Request packet received from {0}: ".format(address))
                print(data)
                if decode_packet(data) == 0:
                    # The received packet was valid
                    # A response packet is now sent to the client
                    lang_code = get_lang(sock, sockets)
                    response = handle_packet(data, lang_code)
                    if response:
                        sock.sendto(response, address)
                        print("Response packet sent to {0}: ".format(address))
                        print(response)
                        print("------------------------------")
                        print("------------------------------")


def main():
    """
    Runs the server
    """
    args = sys.argv[1:]
    # Getting the three port numbers from the user
    english_port, maori_port, german_port = process_ports(args)
    # Opening sockets
    sockets = create_sockets(english_port, maori_port, german_port)
    # Entering a loop to wait for requests from the client
    wait(sockets)
    # Closing all sockets
    for sock in sockets:
        sock.close()
```

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


##################################
# End of server.py file
##################################
```