```
import socket
import sys
import select
import datetime

MAGIC_NUMBER = 0x497E

def check_ports(argv):
    """Checks to ensure that there are the correct number of ports and that the
       ports are in the correct range"""
    if len(argv) > 3:
        print("Too many ports!")
        sys.exit()
    elif len(argv) < 3:
        print("Too few ports!")
        sys.exit()

    ports = [int(argv[0]), int(argv[1]), int(argv[2])]

    for port in ports:
        if port < 1024 or port > 64000:
            print("Port numbers must be between 1024 and 6400 (inclusive)")
            sys.exit()

    return ports


def open_sockets(ports):
    """Opens and binds three UDP / datagram sockets to the given port numbers"""
    try:
        #Create three sockets
        eng_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #AF_INET = IPv4
        maori_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #SOCK_DGRAM =
UDP
        ger_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    except Exception as e:
        print("Error creating sockets: {}".format(e))
        sys.exit()

    try:
        #Bind the sockets
        eng_sock.bind(("0.0.0.0", ports[0]))
        maori_sock.bind(("0.0.0.0", ports[1]))
        ger_sock.bind(("0.0.0.0", ports[2]))
    except Exception as e:
        print("Error binding sockets: {}".format(e))
        sys.exit()

    return [eng_sock, maori_sock, ger_sock]


def packet_valid(packet):
    """Checks whether a given packet is valid"""
    #Checks the length of the packet
    if len(packet) != 6:
        print("Incorrect packet size ({})".format(len(packet)))
        return False

    #Checks the magic number
    magicNum = packet[0] << 8 | packet[1]
    if magicNum != MAGIC_NUMBER:
        print("Magic number not correct ({})".format(magicNum))
        return False

    #Checks the PacketType field
    packetType = packet[2] << 8 | packet[3]
    if packetType != 1: #0x0001
        print("PacketType field not valid ({})".format(packetType))
        return False

    #Checks the RequestType field
    requestType = packet[4] << 8 | packet[5]
    if requestType != 1 and requestType != 2: #0x0001, 0x0002
```

```
        print("RequestType field not valid ({})".format(requestType))
        return False

    return True


def create_string(requestType, lang, time):
    """Creates and returns a string"""
    date_options = ["Today's date is", "Ko te ra o tenei ra ko", "Heute ist der"]
    time_options = ["The current time is", "Ko te wa o tenei wa", "Die Uhrzeit ist"]
    months = [["January", "February", "March", "April", "May", "June", "July",
               "August", "September", "October", "November", "December"],
              ["Kohitatea", "Hui-tanguru", "Poutu-te-rangi", "Paenga-whawha",
               "Haratua", "Pipiri", "Hongongoi", "Here-turi-koka", "Mahuru",
               "Whiringa-a-nuku", "Whiringa-a-rangi", "Hakihea"],
              ["Januar", "Februar", "Marz", "April", "Mai", "Juni", "Juli",
               "August", "September", "Oktober", "November", "Dezember"]]
    str_to_add = ""

    if requestType == 1: #Date
        date_str = date_options[lang]
        if lang == 2:
            date_str += " {0}. {1} {2}".format(time.day, months[lang][(time.month -
1)], time.year)
        else:
            date_str += " {0} {1}, {2}".format(months[lang][(time.month - 1)], time.
day, time.year)
        str_to_add = date_str
    elif requestType == 2: #Time
        time_str = time_options[lang]
        time_str += time.strftime(" %H:%M")
        str_to_add = time_str

    return str_to_add


def prepare_packet(recv_packet, lang):
    """Prepares a DT-Response packet to send back to the client"""
    #Gets the string
    time = datetime.datetime.now() #Create here and parse in to create_string so it'
s exactly the same
    requestType = int((bin(recv_packet[4])[2:] + bin(recv_packet[5])[2:]), 2)
    str_to_add = create_string(requestType, lang, time)
    str_bytes = str_to_add.encode('utf-8')
    string_len = len(str_bytes)
    if string_len > 255:
        print("String to send too long")
        return False, None

    #Begin creating the packet
    #Create an empty packet
    packet = bytearray(13 + string_len) #13 Bytes of header

    #Add the magic number to the packet
    magicNum_16 = format(MAGIC_NUMBER, "016b")
    packet[0] = int(magicNum_16[:8], 2)
    packet[1] = int(magicNum_16[8:], 2)

    #Add the PacketType information to the packet
    packet[2] = 0 #Since it will always be lead by 8 0s
    packet[3] = 2 #Since this is will be 6 0s followed by a 1, followed by a 0

    #Add the LanguageCode information to the packet
    packet[4] = 0 #Since it will always be lead by 8 0s
    packet[5] = (lang + 1) #As lang is currently an index

    #Add the Year information to the packet
    year = format(time.year, "016b")
    packet[6] = int(year[:8], 2)
    packet[7] = int(year[8:], 2)

    #Add the Month information to the packet
    packet[8] = time.month
```

```
    #Add the day information to the packet
    packet[9] = time.day

    #Add the hour information to the packet
    packet[10] = time.hour

    #Add the minute information to the packet
    packet[11] = time.minute

    #Add the length of the text information to the packet
    packet[12] = string_len + 13

    #Add the text representation of the date/time to the packet
    packet[13:] = str_bytes

    return True, packet


def run_server(sockets, ports):
    """Runs the infinite loop that uses the select() call to wait for a request
       packet on any of the sockets. When one is recieved the server retrieves
       the packet, checks the packet for validity, processes the packet and
       finally prepares and sends a response packet"""
    while True:
        readable, writeable, exceptional = select.select(sockets, [], [], 1)
        for sock in readable:
            packet, address = sock.recvfrom(4096)
            _, port_recieved_on = sock.getsockname()
            lang = ports.index(port_recieved_on)
            if packet_valid(packet):
                #Process packet
                success, resp_packet = prepare_packet(packet, lang)
                if success:
                    #Send packet
                    sock.sendto(resp_packet, address)


def main(argv):
    ports = check_ports(argv)
    sockets = open_sockets(ports)
    run_server(sockets, ports)

if __name__ == '__main__':
    main(sys.argv[1:]) #Cuts out the server.py argument
```