```python
"""
Server.py

"""

import select
import socket
import sys
import datetime
from packet import DT_Response, check_request, get_request

# get command line arguments
try:
    ports = [int(sys.argv[1]), int(sys.argv[2]), int(sys.argv[3])]
    for port in ports:
        if port < 1024 or port > 64000:
            raise ValueError
except IndexError:
    print("Please enter three ports")
    sys.exit(1)
except ValueError:
    print("Please enter ports between 1024 and 64000")
    sys.exit(1)
except:
    print("Usage: python3 server.py port1 port2 port3")
    sys.exit(1)

# te reo months list
te_reo_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 list
german_months = [
    "Januar",
    "Februar",
    "M¨arz",
    "April",
    "Mai",
    "Juni",
    "Juli",
    "August",
    "September",
    "Oktober",
    "November",
    "Dezember"
]


def date(language):
    """
    return the text literal for the current date in the given language
    """
    date = datetime.date.today()
    day = date.strftime("%d")
    year = date.strftime("%Y")
    if language == "te_reo":
        month = te_reo_months[date.month - 1]
        return "Ko te ra o tenei ra ko %s %s, %s" %(month, day, year)
    elif language == "german":
        month = german_months[date.month - 1]
        return "Heute ist der %s. %s %s" %(day, month, year)
    else:
        month = date.strftime("%B")
        return "Today's date is %s %s, %s" %(month, day, year)

def time(language):
    """
    return the text literal for the current time in the given language
```

```
    """
    time = datetime.datetime.now().strftime("%H:%M")
    if language == "te_reo":
        return "Ko te wa o tenei wa %s" % time
    elif language == "german":
        return "Die Uhrzeit ist %s" % time
    else:
        return "The current time is %s" % time


class Server(socket.socket):
    """
    Server class
    """
    def __init__(self, port, language) -> None:
        self.port = port
        self.language = language
        super().__init__(socket.AF_INET, socket.SOCK_STREAM)
        self.bind(("", port))
        self.listen(5)

    def connection(self) -> None:
        """
        handle the connection
        """
        client, address = self.accept()
        request = self.recieve(client)
        if request: # if the request is valid, send the response, else, ignore the request
            self.send(client, get_request(request))
        client.close()

    def recieve(self, client) -> None:
        """
        recieve the request from the client and check if it is valid
        """
        request = client.recv(1024).decode()
        request = bytearray(request, "utf-8")
        if check_request(request):
            return request
        return None

    def send(self, client, request) -> None:
        """
        send the response to the client
        """
        date_time = list(map(int,datetime.datetime.now().strftime("%Y %m %d %H %M %S").split()))
        data = time(self.language) if request == "time" else date(self.language)
        data = data.encode("utf-8")
        packet = DT_Response(self.language, date_time, data)
        packet.pack()
        client.send(packet.buffer)


#globals
server = Server(ports[0], "english")
server2 = Server(ports[1], "te_reo")
server3 = Server(ports[2],  "german")
inputs = [server, server2, server3]


def main():
    """
    main function
    """
    while True: # loop forever
        readable, writable, exceptional = select.select(inputs, [], []) # select the sockets that are ready to be read
        for s in readable: # read the sockets
            s.connection() # handle the connection

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

```python
"""
client.py
"""

import select
import socket
import sys
from packet import DT_Request, check_response, get_whole_response

# get command line arguments
try:
    request = sys.argv[1]
    ip = sys.argv[2]
    port = int(sys.argv[3])
except IndexError:
    print("Please enter three arguments")
    sys.exit(1)

if port < 1024 or port > 64000:
    print("Please enter ports between 1024 and 64000")
    sys.exit(1)

if request not in ["date", "time"]:
    print("Argument 1 must be either 'date' or 'time'")
    sys.exit(1)

if socket.getaddrinfo(ip, port)[0][0] != socket.AF_INET:
    print("Argument 2 must be a valid IP address")
    sys.exit(1)


def print_nicely(List):
    """
    print the list nicely
    """
    for (name, value) in List:
        print(name + ":" + " " * (16 - len(name)), value)
    print()

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((ip, port))
packet = DT_Request(request)
packet.pack()
client.send(packet.buffer)
writeable, readable, exceptional = select.select([client], [], [], 1)

if writeable:
    response = client.recv(1024)
    if check_response(response):
        print_nicely(get_whole_response(response))
    elif response == b"":
        print("Server timed out")
    else:
        print("Invalid response")
else:
    print("Server timed out")
```

```python
"""
packet.py
"""

class Packet:
    """
    Base class for all packets.
    """
    magic_no = 0x497E

    def __init__(self, length):
        self.buffer = bytearray(length)
        self.length = length


class DT_Request(Packet):
    """
    packet for requesting date or time
    """
    type = 0x0001

    def __init__(self, request):
        super().__init__(6)
        self.request = 0x0001 if request == "date" else 0x0002

    def pack(self):
        """
        pack the packet
        """
        self.buffer[0] = self.magic_no >> 8
        self.buffer[1] = self.magic_no & 0xFF
        self.buffer[2] = self.type >> 8
        self.buffer[3] = self.type & 0xFF
        self.buffer[4] = self.request >> 8
        self.buffer[5] = self.request & 0xFF


def check_request(buffer):
    """
    check if the packet is a valid request packet
    """
    if buffer[0] << 8 | buffer[1] != 0x497E:                    # check magic number
        return False
    if buffer[2] << 8 | buffer[3] != 0x0001:                    # check type
        return False
    if buffer[4] << 8 | buffer[5] not in [0x0001, 0x0002]:      # check language
        return False
    return True

def get_request(buffer):
    """
    get the request type from a request packet
    """
    return buffer[4] << 8 | buffer[5]

class DT_Response(Packet):
    """
    packet for sending date or time
    """
    type = 0x0002

    def __init__(self, language, datetime, data):
        super().__init__(13 + len(data))
        self.language = 0x0001 if language == "english" else 0x0002 if language == "te_reo" else 0x0003
        self.datetime = datetime
        self.data = data

    def pack(self):
        """
        pack the packet
        """
        self.buffer[0] = self.magic_no >> 8
        self.buffer[1] = self.magic_no & 0xFF
        self.buffer[2] = self.type >> 8
        self.buffer[3] = self.type & 0xFF
        self.buffer[4] = self.language >> 8
        self.buffer[5] = self.language & 0xFF
        self.buffer[6] = self.datetime[0] >> 8
        self.buffer[7] = self.datetime[0] & 0xFF
```

```
        self.buffer[8] = self.datetime[1]
        self.buffer[9] = self.datetime[2]
        self.buffer[10] = self.datetime[3]
        self.buffer[11] = self.datetime[4]
        self.buffer[12] = self.length - 13
        for i in range(13, self.length):
            self.buffer[i] = self.data[i - 13]


def check_response(buffer):
    """
    check if the packet is a valid response packet
    """
    if len(buffer) < 13:                                        # check if buffer is at least 13 bytes long
        return False
    if buffer[0] << 8 | buffer[1] != 0x497E:                    # check magic number
        return False
    if buffer[2] << 8 | buffer[3] != 0x0002:                    # check type
        return False
    if buffer[4] << 8 | buffer[5] not in [0x0001, 0x0002, 0x0003]:      # check language
        return False
    if (buffer[6] << 8 | buffer[7]) > 2100:                     # check year is below 2100
        return False
    if not(0x1 <= (buffer[8]) <= 0x12):                         # check month is between 1 and 12
        return False
    if not(0x1 <= buffer[9] <= 0x1f):                           # check day is between 1 and 31
        return False
    if not(0x0 <= buffer[10] <= 0x18):                          # check hour is between 0 and 24
        return False
    if not(0x0 <= buffer[11] <= 0x3c):                          # check minute is between 0 and 60
        return False
    if len(buffer) != 13 + buffer[12]:                          # check length is correct
        return False
    return True

def get_whole_response(buffer):
    """
    return the whole response from a response packet in a list with the field names as the first element
    """
    temp = []
    temp.append(("Magic Number", hex(buffer[0] << 8 | buffer[1])))      # magic number
    temp.append(("type", hex(buffer[2] << 8 | buffer[3])))              # type
    temp.append(("language", buffer[4] << 8 | buffer[5]))               # language
    temp.append(("year", buffer[6] << 8 | buffer[7]))                   # year
    temp.append(("month", buffer[8]))                                   # month
    temp.append(("day", buffer[9]))                                     # day
    temp.append(("hour", buffer[10]))                                   # hour
    temp.append(("minute", buffer[11]))                                 # minute
    temp.append(("length", buffer[12]))                                 # length
    temp.append(("data", str(buffer[13:], 'utf-8')))                    # data
    return temp
```