```
'''
Lab 5- Sockets

Trang Van
CIS 41B

Description: Uses high level and low level concepts of multiprocessing and
socket programming
             to measure data transfer speeds between processes

'''

import multiprocessing as mp
import socket
import time
import platform

HOST = 'localhost'    # Own machine
PORT = 4540           # An unassigned port

'''
Function for child process - Sets the event, retrieves a number from the
                             queue. It then increments and puts this new
                             number into the queue and gets the next
number.
'''
def highChild(q,e):
    e.set()
    data = q.get()
    while e.is_set():
        while data != 0:
            data += 1
            q.put(data)
            q.get()
'''
Uses Queue and Event to transfer data- Checks if value is 2 and then
proceeds
                                     to loop and increment data, putting it
into
                                     the queue and retrieving the
incremented
                                     number from the child
'''
def highLevel():
    x = 0
    x += 1
    q = mp.Queue()
    e = mp.Event()
    p = mp.Process(target = highChild , args = (q,e)) # child process
    p.start()

    q.put(x)
    e.wait(2)
```

```
    while e.is_set():
        expected = q.get() # 2

        if expected == 2:
            x = 1
            start = time.process_time()

            for i in range(10000):
                x += 1
                q.put(x)
                transfer = q.get()
        else:
            raise ValueError
        e.clear()

    elapsed = time.process_time() - start

    #print('\nElapsed time:', elapsed)
    return transfer/2/elapsed             #number of one-way data transfer
is half of what was being passed back & forth

'''
Function for child process - Acts as client for server. recieves the data
                             from server and increments it to send back to
server
'''
#Client
def lowChild():
    with socket.socket() as client:
        client.connect((HOST, PORT))
        #print("Client connect to:", HOST, "port:", PORT)

        x = client.recv(1024).decode('utf-8')
        data = int(x)
        while data != 0:
            try:
                data += 1
                client.send(str(data).encode('utf-8'))
                next_data = client.recv(1024).decode('utf-8')
            except ConnectionResetError:
                pass
'''
Creates a socket to transfer data - Acts as server. Recieves incremented
data from
                                    client and sends data to client
'''
def lowLevel():
    p = mp.Process(target = lowChild, name = 'Socket Process')
    p.start()

    with socket.socket() as server:
        server.bind((HOST, PORT))   #Address for server
        #print("Server hostname:", HOST, ",port:", PORT)
```

```
        server.listen()

        while True:
            (conn, addr) = server.accept()
            #print("From client:", addr)
            x = 0
            x += 1
            conn.send(str(x).encode('utf-8'))
            fromClient = conn.recv(1024).decode('utf-8')

            if int(fromClient) == 2:
                x = 1
                start = time.process_time()
                for i in range(10000):
                    x += 1
                    conn.send(str(x).encode('utf-8'))

                    fromClient = conn.recv(1024).decode('utf-8')
                    transfers = float(fromClient)

                elapsed = time.process_time()-start
            else:
                raise ValueError
            #print('Elapsed time:', elapsed)

            conn.send(str(0).encode('utf-8'))
            break
        return transfers/2/elapsed

'''
Main- prints platform and cpu count; runs highLevel and lowLevel 3 times
'''
if __name__ == '__main__':
    print('OS:',platform.system())
    print('Num of cores:', mp.cpu_count())

    hi_run1 = int(round(highLevel()))
    print('High Level 1:',hi_run1)
    lo_run1 = int(round(lowLevel()))
    print('Low Level 1:',lo_run1)

    hi_run2= int(round(highLevel()))
    print('High Level 2:',hi_run2)
    lo_run2 = int(round(lowLevel()))
    print('Low Level 2:',lo_run2)

    hi_run3= int((highLevel()))
    print('High Level 3:',hi_run3)
    lo_run3 = int((lowLevel()))
    print('Low Level 3:',lo_run3)

'''
Conclusion:
```

Using sockets allow the data to transfer easily. Whereas multiprocessing required the use of Queues to send data back and forth, socket programming allows communication between two hosts and there's less of a concern about race conditions. In multiprocessing, we had to factor in the race condition
and has mechanisms like Event to control it.

My results seems like multiprocessing has a quicker transfer speed than the
sockets. I think this is because it takes more time to communicate between hosts. Multiprocessing and using queues make the transfer faster since it
runs independently.

Sample Run:

```
OS: Windows
Num of cores: 4
High Level 1: 3479
Low Level 1: 11431
High Level 2: 3442
Low Level 2: 8650
High Level 3: 2857
Low Level 3: 8650

'''
```