Using Sockets

1. Communication between Programs
   - client/server interaction ~ telephone exchange

2. Remote Server and Client
   - getting IP addresses of computers
   - code for remote client and server

3. Two Clients and one Talk Host
   - swapping information between two clients
   - code for client and server

4. Monte Carlo for $\pi$
   - a pleasingly parallel computation
   - the multiprocessing module

MCS 275 Lecture 28
Programming Tools and File Management
Jan Verschelde, 15 March 2017
Using Sockets

1 Communication between Programs
   - client/server interaction \( \sim \) telephone exchange

2 Remote Server and Client
   - getting IP addresses of computers
   - code for remote client and server

3 Two Clients and one Talk Host
   - swapping information between two clients
   - code for client and server

4 Monte Carlo for \( \pi \)
   - a pleasingly parallel computation
   - the multiprocessing module
Analogy with Telephone Exchange
client/server communication with sockets

Analogy between a telephone exchange and sockets:

1. dial company on 1-312-666-9000
   connect to IP address 127.0.0.1

2. call answered by reception
   connection established to remote host

3. ask for computer center
   route using specified port (8732)

4. call answered by computer center
   server handles request from client

5. hang up the phone
   close sockets
# Methods on Socket Objects

Most commonly used methods:

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept()</td>
<td>accepts connection and returns new socket for passing data</td>
</tr>
<tr>
<td>bind()</td>
<td>binds a socket to an address</td>
</tr>
<tr>
<td>close()</td>
<td>closes the socket</td>
</tr>
<tr>
<td>connect(a)</td>
<td>connects to address a</td>
</tr>
<tr>
<td>listen(c)</td>
<td>listen for c connections</td>
</tr>
<tr>
<td>shutdown(flag)</td>
<td>shut down for reading, writing, or both</td>
</tr>
<tr>
<td>recv(b)</td>
<td>receives data in buffer of size b</td>
</tr>
<tr>
<td>send(d)</td>
<td>sends data in d</td>
</tr>
</tbody>
</table>
Using Sockets

1. Communication between Programs
   - client/server interaction ~ telephone exchange

2. Remote Server and Client
   - getting IP addresses of computers
   - code for remote client and server

3. Two Clients and one Talk Host
   - swapping information between two clients
   - code for client and server

4. Monte Carlo for $\pi$
   - a pleasingly parallel computation
   - the multiprocessing module
getting IP addresses of computers with `ifconfig`

On a Mac: open System Preferences -> Network.

On Unix: `ifconfig` configures network interface parameters. Without parameters, `ifconfig` displays current configuration.

To select what we need, we type at the prompt $ 

```
$ ifconfig | grep "inet " | grep -v 127.0.0.1 \ 
   | cut -d\  -f2
```

The firewall on the server computer must allow incoming connections for Python.

On Mac, see System Preferences -> Security.
nslookup and firewall configurations

$ nslookup people.uic.edu
Server: 128.248.171.50
Address: 128.248.171.50#53

Name: people.uic.edu
Address: 128.248.156.140
$

To allow tcp communications via a port number on Linux, one must modify the Firewall Configuration and add a port number.
Using Sockets

1. Communication between Programs
   - client/server interaction ~ telephone exchange

2. Remote Server and Client
   - getting IP addresses of computers
   - code for remote client and server

3. Two Clients and one Talk Host
   - swapping information between two clients
   - code for client and server

4. Monte Carlo for $\pi$
   - a pleasingly parallel computation
   - the multiprocessing module
code for the client in `remote_client.py`

```python
from socket import socket as Socket
from socket import AF_INET, SOCK_STREAM
SERVER = '131.193.178.183'  # IP address
PORTNUMBER = 41267        # port number
BUFFER = 80               # buffer size

SERVER_ADDRESS = (SERVER, PORTNUMBER)
CLIENT = Socket(AF_INET, SOCK_STREAM)
try:
    CLIENT.connect(SERVER_ADDRESS)
    print('client is connected')
    DATA = input('Give message : ')
    CLIENT.send(DATA.encode())
except OSError:
    print('connection failed')
CLIENT.close()
```
from socket import socket as Socket
from socket import AF_INET, SOCK_STREAM, SHUT_RDWR

HOSTNAME = ''  # blank for any address
PORTNUMBER = 41267  # number for the port
BUFFER = 80  # size of the buffer

SERVER_ADDRESS = (HOSTNAME, PORTNUMBER)
SERVER = Socket(AF_INET, SOCK_STREAM)
SERVER.bind(SERVER_ADDRESS)
SERVER.listen(1)
print('server waits for connection')
CLIENT, CLIENT_ADDRESS = SERVER.accept()

if CLIENT_ADDRESS[0] == '131.193.41.130':
    print('server accepted connection from ',\
          CLIENT_ADDRESS)
    print('server waits for data')
    DATA = CLIENT.recv(BUFFER).decode()
    print('server received ', DATA)
else:
    print('server does not accept data from ',\
          CLIENT_ADDRESS)
    CLIENT.shutdown(SHUT_RDWR)

SERVER.close()
Using Sockets

1 Communication between Programs
   • client/server interaction ~ telephone exchange

2 Remote Server and Client
   • getting IP addresses of computers
   • code for remote client and server

3 Two Clients and one Talk Host
   • swapping information between two clients
   • code for client and server

4 Monte Carlo for \( \pi \)
   • a pleasingly parallel computation
   • the multiprocessing module
Protocol to swap Client Data

two clients and one server

Often the server acts as moderator between clients.

Suppose two clients want to exchange information.

Protocol:

1. server listens to two incoming clients
2. both clients connect to the server
   and the server accepts two connections
3. both clients send their data to the server
4. server sends the data of the first client to the second and the data
   of the second client to the first.

→ same program for the clients
two clients swapping data

Terminal window at the server:

$ python talk_host.py
talk host waits for first client
server accepted connection from .. 
     .. ('127.0.0.1', 49158)
talk host waits for second client
server accepted connection from .. 
     .. ('127.0.0.1', 49159)
talk host waits for first client
talk host received "this is alpha"
talk host waits for second client
talk host received "this is beta"
talk host sends "this is beta" to first client
talk host sends "this is alpha" to second client
$
running the clients

Terminal window for the first client:

$ python talk_client.py
client is connected
Give message : this is alpha
client waits for reply
client received "this is beta"
$

Terminal window for the second client:

$ python talk_client.py
client is connected
Give message : this is beta
client waits for reply
client received "this is alpha"
$
Using Sockets

1. Communication between Programs
   - client/server interaction \sim telephone exchange

2. Remote Server and Client
   - getting IP addresses of computers
   - code for remote client and server

3. Two Clients and one Talk Host
   - swapping information between two clients
   - code for client and server

4. Monte Carlo for $\pi$
   - a pleasingly parallel computation
   - the multiprocessing module
code for the client in talk_client.py

```python
from socket import socket as Socket
from socket import AF_INET, SOCK_STREAM
HOSTNAME = 'localhost'  # on same host
PORTNUMBER = 11267  # same port number
BUFFER = 80  # size of the buffer

SERVER_ADDRESS = (HOSTNAME, PORTNUMBER)
CLIENT = Socket(AF_INET, SOCK_STREAM)
CLIENT.connect(SERVER_ADDRESS)

print('client is connected')
ALPHA = input('Give message : ')
CLIENT.send(ALPHA.encode())

print('client waits for reply')
BETA = CLIENT.recv(BUFFER).decode()
print('client received "' + BETA + '"')
CLIENT.close()
```
code for starting the server in `talk_host.py`

from socket import socket as Socket
from socket import AF_INET, SOCK_STREAM

HOSTNAME = '' # blank for any address
PORTNUMBER = 11267 # number for the port
BUFFER = 80 # size of the buffer

SERVER_ADDRESS = (HOSTNAME, PORTNUMBER)
SERVER = Socket(AF_INET, SOCK_STREAM)
SERVER.bind(SERVER_ADDRESS)
SERVER.listen(2)
code for accepting connections

print('talk host waits for first client')
FIRST, FIRST_ADDRESS = SERVER.accept()
print('server accepted connection from ', \
    FIRST_ADDRESS)

print('talk host waits for second client')
SECOND, SECOND_ADDRESS = SERVER.accept()
print('server accepted connection from ', \
    SECOND_ADDRESS)
code for exchanging data

```python
print('talk host waits for first client')
ALPHA = FIRST.recv(BUFFER).decode()
print('talk host received "' + ALPHA + '"')
print('talk host waits for second client')
BETA = SECOND.recv(BUFFER).decode()
print('talk host received "' + BETA + '"')

print('talk host sends "' + BETA + '" to first client')
FIRST.send(BETA.encode())
print('talk host sends "' + ALPHA + '" to second client')
SECOND.send(ALPHA.encode())

SERVER.close()
```
Using Sockets

1. Communication between Programs
   - client/server interaction ~ telephone exchange

2. Remote Server and Client
   - getting IP addresses of computers
   - code for remote client and server

3. Two Clients and one Talk Host
   - swapping information between two clients
   - code for client and server

4. Monte Carlo for $\pi$
   - a pleasingly parallel computation
   - the multiprocessing module
Estimating $\pi$
Monte Carlo methods: throwing darts

The area of the unit disk is $\pi$:

Generate random uniformly distributed points with coordinates $(x, y) \in [-1, +1] \times [-1, +1]$. We count a success when $x^2 + y^2 \leq 1$.

1. generate $n$ points $P$ in $[0, 1] \times [0, 1]$
2. $m := \{ (x, y) \in P : x^2 + y^2 \leq 1 \}$
3. the estimate is then $4 \times m/n$

→ dual core processor: use two clients.

pleasingly parallel: optimal speedup (twice as fast)
server and client terminals

$ python mc4pi2.py
server waits for connections...
server waits for results...
approximation for pi = 3.141487

$ python mc4pi_client.py
client is connected
client received 1
client computes 0.785253200000

$ python mc4pi_client.py
client is connected
client received 2
client computes 0.785490300000
from socket import socket as Socket
from socket import AF_INET, SOCK_STREAM

HOSTNAME = ''  # blank for any address
PORTNUMBER = 11267  # number for the port
BUFFERSIZE = 80  # size of the buffer

SERVER_ADDRESS = (HOSTNAME, PORTNUMBER)
SERVER = Socket(AF_INET, SOCK_STREAM)
SERVER.bind(SERVER_ADDRESS)
one server distributes seeds in `mc4pi2.py`

SERVER.listen(2)
print('server waits for connections...')
FIRST, FIRST_ADDRESS = SERVER.accept()
SECOND, SECOND_ADDRESS = SERVER.accept()
FIRST.send('1'.encode())
SECOND.send('2'.encode())

print('server waits for results...')
NBR1 = FIRST.recv(BUFFERSIZE).decode()
NBR2 = SECOND.recv(BUFFERSIZE).decode()
RESULT = 2*(float(NBR1)+float(NBR2))
print('approximation for pi =', RESULT)
SERVER.close()
import random
from socket import socket as Socket
from socket import AF_INET, SOCK_STREAM

HOSTNAME = 'localhost'  # on same host
PORTNUMBER = 11267  # same port number
BUFFERSIZE = 80  # size of the buffer

SERVER_ADDRESS = (HOSTNAME, PORTNUMBER)
CLIENT = Socket(AF_INET, SOCK_STREAM)
CLIENT.connect(SERVER_ADDRESS)

print('client is connected')
SEED = CLIENT.recv(BUFFERSIZE).decode()
print('client received %s' % SEED)
random.seed(int(SEED))

NBR = 10**7
CNT = 0
for i in range(NBR):
    XPT = random.uniform(0, 1)
    YPT = random.uniform(0, 1)
    if XPT**2 + YPT**2 <= 1:
        CNT = CNT + 1
RESULT = float(CNT)/NBR
print('client computes %.12f' % RESULT)

CLIENT.send(str(RESULT).encode())

CLIENT.close()
Using Sockets

1. Communication between Programs
   - client/server interaction \sim \text{telephone exchange}

2. Remote Server and Client
   - getting IP addresses of computers
   - code for remote client and server

3. Two Clients and one Talk Host
   - swapping information between two clients
   - code for client and server

4. Monte Carlo for $\pi$
   - a pleasingly parallel computation
   - the multiprocessing module
the multiprocessing module

The multiprocessing module provides two classes.

1. **Objects of the class** `Process` **represent processes**:  
   - We instantiate with a function the process will execute.  
   - The method `start` starts the child process.  
   - The method `join` waits till the child terminates.  

   In UNIX, we say: the main process *forfs* a child process.

2. **Objects of the class** `Queue` **are used to pass data from the child processes to the main process that forked the processes.**

The two classes suffice for a simple manager-worker model:

- The manager distributes the work with functions.
- The workers are the child processes which run the functions.
processes report back via a queue

from multiprocessing import Process, Queue
from math import pi

def monte_carlo4pi(nbr, nsd, result):
    """
    Estimates pi with nbr samples, using nsd as seed.
    Adds the result to the queue q.
    """
    from random import uniform as u
    from random import seed
    seed(nsd)
    cnt = 0
    for _ in range(nbr):
        (x, y) = (u(-1, 1), u(-1, 1))
        if x**2 + y**2 <= 1:
            cnt = cnt + 1
    result.put(cnt)
def main():
    """
    Prompts the user for the number of samples and the number of processes.
    """

    nbr = int(input('Give the number of samples : '))
    npr = int(input('Give the number of processes : '))
    queues = [Queue() for _ in range(npr)]
    procs = []
    for k in range(1, npr+1):
        procs.append(Process(target=monte_carlo4pi, \
                         args=(nbr, k, queues[k-1])))
    for process in procs:
        process.start()
    for process in procs:
        process.join()
    app = 4*sum([q.get()/nbr for q in queues])/npr
    print(app, 'error : %.3e' % abs(app - pi))
Summary + Assignments

We covered more of chapter 12 in *Making Use of Python*; and introduced *interactive* parallel computation.

Exercises:

1. Explore the possibilities of a client/server interaction between people.uic.edu and computers in the labs, or between your laptop and home computer.

2. Modify the code for `talk_host.py`: the user is prompted to enter the number of clients. Change the code for `talk_client.py`: the user enters the number of the client. Extend the data swapping protocol: data from the $i$th client goes to the $(i + 1)$st client, data from the last client is sent to the first one.

3. The natural logarithm of 2 is $\ln(2) = \int_1^2 \frac{1}{x} \, dx$. Adjust the code for estimating $\pi$, to estimate $\ln(2)$ with an interactive parallel client/server computation.