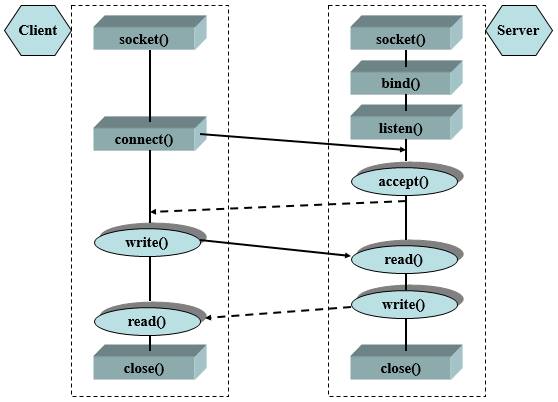
# TÀI LIỆU THIẾT KẾ

# **Người tạo: Nguyễn Đình Thái**

# **Thời gian: 10/7/2017**

# SƠ ĐỒ HOẠT ĐỘNG



# B.CÁC MODULE ĐƯỢC SỬ DỤNG TRONG CHƯƠNG TRÌNH

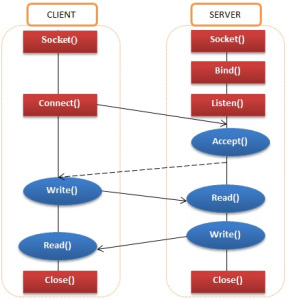
### I. Module Socket

Socket là một cổng logic mà một chương trình sử dụng để kết nối với một chương trình khác chạy trên một máy tính khác trên Internet. Chương trình mạng có thể sử dụng nhiều Socket cùng một lúc, nhờ đó nhiều chương trình có thể sử dụng Internet cùng một lúc. có 2 loại socket thường thấy đó là dùng giao thức TCP hoặc UDP

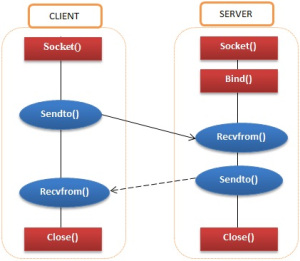
1. Stream Socket: Dựa trên giao thức TCP( Tranmission Control Protocol) việc truyền dữ liệu chỉ thực hiện giữa 2 quá trình **đã thiết lập kết nối**. Giao thức này đảm bảo dữ liệu được truyền đến nơi nhận một cách đáng tin cậy, đúng thứ tự nhờ vào cơ chế quản lý luồng lưu thông trên mạng và cơ chế chống tắc nghẽn.
2. Datagram Socket: Dựa trên giao thức UDP( User Datagram Protocol) việc truyền dữ liệu **không yêu cầu có sự thiết lập kết nối giữa 2 quá trình**. Ngược lại với giao thức TCP thì dữ liệu được truyền theo giao thức UDP không được tin cậy, có thế không đúng trình tự và lặp lại. Tuy nhiên vì nó không yêu cầu thiết lập kết nối không phải có những cơ chế phức tạp nên tốc độ nhanh…ứng dụng cho các ứng dụng truyền dữ liệu nhanh như chat, game…..

Một TCP/IP Socket gồm một địa chỉ IP kết hợp với một port xác định duy nhất một tiến trình (process ) trên mạng.Hay nói cách khác luồng thông tin trên mạng dựa vào IP là để xác định một máy trên mạng còn port xác định 1 tiến trình trên 1 máy.

**Cơ chế gọi hàm trong lập trình socket**

[](https://c6h0st.files.wordpress.com/2015/10/tcp2.jpg)

*Lập trình Socket với TCP*

[](https://c6h0st.files.wordpress.com/2015/10/udp2.jpg)

**Python Socket module**

|  |  |
| --- | --- |
| **Term** | **Description** |
| domain | Họ của các giao thức dùng để truyền dữ liệu: AF\_INET, PF\_INET, PF\_UNIX, PF\_X25, ….  Ở đây chúng ta thường dùng AF\_INET (IPv4) |
| type | Kiểu của socket: SOCK\_STREAM (TCP), SOCK\_DGRAM (UDP) |
| protocol | Dùng để xác định các biến thể của các giao thức, mặc định là 0 |
| hostname | Dùng để xác định hostname, có thể là:   * Chuỗi * Địa chỉ IPv4 nếu dùng AF\_INET * Địa chỉ IPv6 nếu dùng AF\_INET6 |
| port | Xác định port của socket |

Để tạo 1 socket ta làm như sau:

s = socket.socket (socket\_family, socket\_type)

Với socket\_family, socket\_type đã được giới thiệu bên trên (Domain, type)

Sau khi khởi tạo được 1 đối tượng socket, tiếp đến ta có thể khởi tạo chương trình cho Client hoặc Server

**Server Socket Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| s.bind() | Kết nối (hostname, port number) đến socket. |
| s.listen() | Cho Server lắng nghe các Client |
| s.accept() | Thiết lập kết nối giữa Client-Server |

**Client Socket Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| s.connect() | Kết nối đến Server |

**General Socket Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| s.recv() | Nhận dữ liệu trong giao thức TCP |
| s.send() | Truyền dữ liệu trong giao thức TCP |
| s.recvfrom() | Nhận dữ liệu trong giao thức UDP |
| s.sendto() | Truyền dữ liệu trong giao thức UDP |
| s.close() | Đóng kết nối |
| socket.gethostname() | Trả về tên của host |

**Đây là 1 đoạn code TCP Socket đơn giản**

**Server side**

import socket # Import socket module

s = socket.socket() # Tạo đối tượng socket (mặc định là TCP trên nền IPv4)

host = socket.gethostname() # Xác định tên localhost

port = 12345 # Xác định port muốn dùng

s.bind((host, port)) # Kết nối đến socket

s.listen(5) # Server lắng nghe 5 Client cùng 1 lúc

while True:

c, addr = s.accept() # Thiết lập kết nối

print 'Got connection from', addr

c.send('Thank you for connecting') #Gửi dữ liệu đến Client

c.close() # Đóng kết nối

**Client side**

import socket # Import socket module

s = socket.socket() # Tạo đối tượng socket

host = socket.gethostname() # Xác định tên localhost

port = 12345 # Xác định port

s.connect((host, port)) #Kết nối đến Server

print s.recv(1024) #Nhận dữ liệu từ Server với Buffer là 1024 bytes

s.close() # Đóng kết nối

Chạy 2 script trên và đây là kết quả:

Got connection from ('127.0.0.1', 48437) #Server side

Thank you for connecting #Client side

## II.MODULE SELECT

Mục đích:

Thông báo khi 1 kênh socket đầu vào huặc đầu ra đã sẵn sàng

Chức năng select () của Python là một giao diện trực tiếp với việc triển khai hệ điều hành cơ bản. Nó giám sát socket, các tệp mở và các pipe (bất cứ thứ gì với phương thức fileno () trả về một trình mô tả tệp tin hợp lệ) cho đến khi chúng trở nên dễ đọc hoặc ghi được hoặc xảy ra lỗi giao tiếp.

Select () làm cho việc giám sát nhiều kết nối đồng thời dễ dàng hơn, và hiệu quả hơn là viết một vòng lặp bằng Python bằng cách sử dụng timeout socket, bởi vì việc theo dõi xảy ra trong lớp network của hệ điều hành, thay vì thông dịch viên.

Ví dụ server echo từ socket có thể được mở rộng để xem nhiều kết nối tại một thời điểm bằng cách sử dụng select (). Phiên bản mới bắt đầu bằng cách tạo một socket TCP / IP không khóa và cấu hình nó để lắng nghe trên một địa chỉ.

import **select**

import **socket**

import **sys**

import **Queue**

*# Create a TCP/IP socket*

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.setblocking(0)

*# Bind the socket to the port*

server\_address = ('localhost', 10000)

**print** >>sys.stderr, 'starting up on **%s** port **%s**' % server\_address

server.bind(server\_address)

*# Listen for incoming connections*

server.listen(5)

Các đối số để chọn () là ba danh sách có chứa các kênh gia tiếp để giám sát. Đầu tiên là danh sách các đối tượng được kiểm tra để đọc dữ liệu đến, thứ hai chứa các đối tượng sẽ nhận dữ liệu đi ra khi có chỗ trong bộ đệm của chúng, và thứ ba có thể có lỗi (thường là sự kết hợp của Đầu vào và đầu ra kênh đối tượng). Bước tiếp theo của máy chủ là thiết lập các danh sách có chứa các nguồn đầu vào và các đích đầu ra để được truyền cho select ().

Phần chính của vòng lặp, gọi select () để chặn và chờ hoạt động của mạng.

**while** inputs:

*# Wait for at least one of the sockets to be ready for processing*

**print** >>sys.stderr, '**\n**waiting for the next event'

readable, writable, exceptional = select.select(inputs, outputs, inputs)

Select () trả về ba danh sách mới, chứa các tập hợp con của nội dung của các danh sách được chuyển vào. Tất cả các socket trong danh sách có thể đọc được có dữ liệu đến được buffered và có sẵn để được đọc. Tất cả các socket trong danh sách có thể ghi có không gian trống trong bộ đệm của chúng và có thể được ghi vào. Các socket trả về trường hợp đặc biệt đã có một lỗi (định nghĩa thực tế của " exceptional condition" phụ thuộc vào nền tảng hdh).

Các socket "có thể đọc" đại diện cho ba trường hợp có thể. Nếu socket là socket “server” , cổng được sử dụng để nghe các kết nối, thì điều kiện "có thể đọc được" có nghĩa là nó đã sẵn sàng chấp nhận kết nối đến khác. Ngoài việc thêm kết nối mới vào danh sách đầu vào để theo dõi, phần này sẽ đặt socket client không bị chặn.

*# Handle inputs*

**for** s **in** readable:

**if** s **is** server:

*# A "readable" server socket is ready to accept a connection*

connection, client\_address = s.accept()

**print** >>sys.stderr, 'new connection from', client\_address

connection.setblocking(0)

inputs.append(connection)

*# Give the connection a queue for data we want to send*

message\_queues[connection] = Queue.Queue()

Trường hợp tiếp theo là kết nối đã được thiết lập với một máy khách đã gửi dữ liệu. Các dữ liệu được đọc với recv (), sau đó đặt trên hàng đợi để nó có thể được gửi qua socket và trở lại cho client.

else:

data = s.recv(1024)

if data:

# A readable client socket has data

print >>sys.stderr, 'received "%s" from %s' % (data, s.getpeername())

message\_queues[s].put(data)

# Add output channel for response

if s not in outputs:

outputs.append(s)

Một socket có thể đọc được mà không có dữ liệu là từ một client đã ngắt kết nối và luồng đã sẵn sàng để đóng.

else:

# Interpret empty result as closed connection

print >>sys.stderr, 'closing', client\_address, 'after reading no data'

# Stop listening for input on the connection

if s in outputs:

outputs.remove(s)

inputs.remove(s)

s.close()

# Remove message queue

del message\_queues[s]

Có ít trường hợp hơn cho các kết nối có thể ghi. Nếu có dữ liệu trong hàng đợi cho một kết nối, tin nhắn tiếp theo sẽ được gửi đi. Nếu không, kết nối sẽ được gỡ bỏ khỏi danh sách các kết nối đầu ra để thời gian tiếp theo thông qua vòng lặp chọn () không cho biết socket đã sẵn sàng để gửi dữ liệu.

*# Handle outputs*

**for** s **in** writable:

**try**:

next\_msg = message\_queues[s].get\_nowait()

**except** Queue.Empty:

*# No messages waiting so stop checking for writability.*

**print** >>sys.stderr, 'output queue for', s.getpeername(), 'is empty'

outputs.remove(s)

**else**:

**print** >>sys.stderr, 'sending "**%s**" to **%s**' % (next\_msg, s.getpeername())

s.send(next\_msg)

Cuối cùng, nếu có lỗi với một socket, nó sẽ được đóng lại.

*# Handle "exceptional conditions"*

**for** s **in** exceptional:

**print** >>sys.stderr, 'handling exceptional condition for', s.getpeername()

*# Stop listening for input on the connection*

inputs.remove(s)

**if** s **in** outputs:

outputs.remove(s)

s.close()

*# Remove message queue*

**del** message\_queues[s]

Ví dụ chương trình client sử dụng hai socket để chứng minh làm thế nào các máy chủ với select () quản lý nhiều kết nối cùng một lúc. Máy khách bắt đầu bằng cách kết nối mỗi socket TCP / IP với máy chủ.

import **socket**

import **sys**

messages = [ 'This is the message. ',

'It will be sent ',

'in parts.',

]

server\_address = ('localhost', 10000)

*# Create a TCP/IP socket*

socks = [ socket.socket(socket.AF\_INET, socket.SOCK\_STREAM),

socket.socket(socket.AF\_INET, socket.SOCK\_STREAM),

]

*# Connect the socket to the port where the server is listening*

**print** >>sys.stderr, 'connecting to **%s** port **%s**' % server\_address

**for** s **in** socks:

s.connect(server\_address)

Sau đó, nó gửi một mẩu tin một lúc thông qua mỗi socket, và đọc tất cả các phản hồi có sẵn sau khi viết dữ liệu mới.

**for** message **in** messages:

*# Send messages on both sockets*

**for** s **in** socks:

**print** >>sys.stderr, '**%s**: sending "**%s**"' % (s.getsockname(), message)

s.send(message)

*# Read responses on both sockets*

**for** s **in** socks:

data = s.recv(1024)

**print** >>sys.stderr, '**%s**: received "**%s**"' % (s.getsockname(), data)

**if** **not** data:

**print** >>sys.stderr, 'closing socket', s.getsockname()

s.close()

Select () cũng lấy một thông số thứ tư tùy ý, đó là số giây chờ đợi trước khi tắt giám sát nếu không có kênh nào hoạt động. Sử dụng giá trị thời gian chờ cho phép chương trình gọi select () là một phần của vòng lặp xử lý lớn hơn, thực hiện các hành động khác giữa kiểm tra cho đầu vào mạng.

### Timeouts

Select () cũng lấy một thông số thứ tư tùy ý, đó là số giây chờ đợi trước khi tắt giám sát nếu không có kênh nào hoạt động. Sử dụng giá trị thời gian chờ cho phép chương trình gọi select () là một phần của vòng lặp xử lý lớn hơn, thực hiện các hành động khác giữa kiểm tra cho đầu vào mạng.

Khi thời gian chờ hết hạn, hãy chọn () trả về ba danh sách trống. Cập nhật ví dụ máy chủ để sử dụng một thời gian chờ đòi hỏi thêm lập luận bổ sung cho các lựa chọn () cuộc gọi và xử lý các danh sách trống sau khi lựa chọn () trả về.

*# Wait for at least one of the sockets to be ready for processing*

**print** >>sys.stderr, '**\n**waiting for the next event'

timeout = 1

readable, writable, exceptional = select.select(inputs, outputs, inputs, timeout)

**if** **not** (readable **or** writable **or** exceptional):

**print** >>sys.stderr, ' timed out, do some other work here'

**continue**

Phiên bản "slow" này của chương trình client tạm dừng sau khi gửi mỗi tin nhắn, để mô phỏng độ trễ hoặc sự chậm trễ khác trong truyền dẫn.

import **socket**

import **sys**

import **time**

*# Create a TCP/IP socket*

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

*# Connect the socket to the port where the server is listening*

server\_address = ('localhost', 10000)

**print** >>sys.stderr, 'connecting to **%s** port **%s**' % server\_address

sock.connect(server\_address)

time.sleep(1)

messages = [ 'Part one of the message.',

'Part two of the message.',

]

amount\_expected = len(''.join(messages))

**try**:

*# Send data*

**for** message **in** messages:

**print** >>sys.stderr, 'sending "**%s**"' % message

sock.sendall(message)

time.sleep(1.5)

*# Look for the response*

amount\_received = 0

**while** amount\_received < amount\_expected:

data = sock.recv(16)

amount\_received += len(data)

**print** >>sys.stderr, 'received "**%s**"' % data

**finally**:

**print** >>sys.stderr, 'closing socket'

## poll()

Chức năng poll () cung cấp các tính năng tương tự để select (), nhưng việc thực hiện cơ bản hiệu quả hơn. trade-off là poll() không được hỗ trợ trong Windows, do đó, các chương trình bằng cách sử dụng poll () ít di động hơn. Một máy chủ echo được xây dựng trên poll () bắt đầu với cùng mã cấu hình socket được sử dụng trong các ví dụ khác.

import **select**

import **socket**

import **sys**

import **Queue**

*# Create a TCP/IP socket*

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.setblocking(0)

*# Bind the socket to the port*

server\_address = ('localhost', 10000)

**print** >>sys.stderr, 'starting up on **%s** port **%s**' % server\_address

server.bind(server\_address)

*# Listen for incoming connections*

server.listen(5)

*# Keep up with the queues of outgoing messages*

message\_queues = {}

Giá trị thời gian chờ vượt qua cuộc thăm dò () được thể hiện bằng mili giây, thay vì giây, do đó, để tạm dừng cho toàn bộ giây, thời gian chờ phải được đặt thành 1000.

*# Do not block forever (milliseconds)*

TIMEOUT = 1000

Python thực hiện cuộc thăm dò () với một lớp quản lý các kênh dữ liệu đã đăng ký được theo dõi. Kênh được thêm vào bằng cách gọi thanh ghi () với cờ cho biết các sự kiện thú vị cho kênh đó. Bộ cờ đầy đủ là:

| **Event** | **Description** |
| --- | --- |
| **POLLIN** | Input sẵn sàng |
| **POLLPRI** | Sự ưu tiên input sẵn sàng |
| **POLLOUT** | Có thể nhận output |
| **POLLERR** | Lỗi |
| **POLLHUP** | Kênh đã đóng |
| **POLLNVAL** | Kênh không được mở |

echo server sẽ được thiết lập một số socket chỉ để đọc, và những cái khác để đọc huặc ghi. Sự kết hợp thích hợp của cờ được lưu vào các biến địa phương READ\_ONLY và READ\_WRITE.

*# Commonly used flag setes*

READ\_ONLY = select.POLLIN | select.POLLPRI | select.POLLHUP | select.POLLERR

READ\_WRITE = READ\_ONLY | select.POLLOUT

## Tài liệu tham khảo:

*https://docs.python.org/2/library/socket.html?highlight=socket#module-socket*

*https://docs.python.org/2/library/select.html*

*https://pymotw.com/2/select/*