Computer Network Laboratory

Basic Applications of Network Programming (II)

Jiawei Chang

Dept. of Computer Science and Information Engineering
National Taichung University of Science and Technology

Outline

1. chat_server_with_select
2. wait_for_remote_service
3. ipc_using_socketpairs
4. unix_domain_socket_server
5. unix_domain_socket_client

CHAT_SERVER_WITH_SELECT

chat_server_with_select

```python
import select
import socket
import sys
import signal
import pickle
import struct
import argparse

SERVER_HOST = 'localhost'
CHAT_SERVER_NAME = 'server'
```

```python
# Some utilities
def send(channel, *args):
    buffer = pickle.dumps(args)
    value = socket.htonl(len(buffer))
    size = struct.pack("L",value)
    channel.send(size)
    channel.send(buffer)

def receive(channel):
    size = struct.calcsize("L")
    size = channel.recv(size)
    try:
        size = socket.ntohl(struct.unpack("L", size)[0])
    except struct.error as e:
        return ''
    buf = ""
    while len(buf) < size:
        buf = channel.recv(size - len(buf))
    return pickle.loads(buf)[0]
```

chat_server_with_select

Server Side

```
33  class ChatServer(object):
34      """ An example chat server using select """
35      def __init__(self, port, backlog=5):
36          self.clients = 0
37          self.clientmap = {}
38          self.outputs = [] # list output sockets
39          self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
40          self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
41          self.server.bind((SERVER_HOST, port))
42          print ('Server listening to port: %s ...' %port)
43          self.server.listen(backlog)
44          # Catch keyboard interrupts
45          signal.signal(signal.SIGINT, self.sighandler)
46
47      def sighandler(self, signum, frame):
48          """ Clean up client outputs"""
49          # Close the server
50          print ('Shutting down server...')
51          # Close existing client sockets
52          for output in self.outputs:
53              output.close()
54          self.server.close()
55
56      def get_client_name(self, client):
57          """ Return the name of the client """
58          info = self.clientmap[client]
59          host, name = info[0][0], info[1]
60          return '@'.join((name, host))
```

chat_server_with_select

Server Side

```
62      def run(self):
63          inputs = [self.server, sys.stdin]
64          self.outputs = []
65          running = True
66          while running:
67              try:
68                  readable, writeable, exceptional = select.select(inputs, self.outputs, [])
69              except select.error as e:
70                  break
71
72              for sock in readable:
73                  if sock == self.server:
74                      # handle the server socket
75                      client, address = self.server.accept()
76                      print ("Chat server: got connection %d from %s" % (client.fileno(), address))
77                      # Read the login name
78                      cname = receive(client).split('NAME: ')[1]
79
80                      # Compute client name and send back
81                      self.clients += 1
82                      send(client, 'CLIENT: ' + str(address[0]))
83                      inputs.append(client)
84                      self.clientmap[client] = (address, cname)
85                      # Send joining information to other clients
86                      msg = "\n(Connected: New client (%d) from %s)" % (self.clients, self.get_client_name(client))
87                      for output in self.outputs:
88                          send(output, msg)
89                      self.outputs.append(client)
90
91                  elif sock == sys.stdin:
92                      # handle standard input
93                      junk = sys.stdin.readline()
94                      running = False
```

```
95                  else:
96                      # handle all other sockets
97                      try:
98                          data = receive(sock)
99                          if data:
100                             # Send as new client's message...
101                             msg = '\n#[' + self.get_client_name(sock) + ']>>' + data
102                             # Send data to all except ourself
103                             for output in self.outputs:
104                                 if output != sock:
105                                     send(output, msg)
106                         else:
107                             print ("Chat server: %d hung up" % sock.fileno())
108                             self.clients -= 1
109                             sock.close()
110                             inputs.remove(sock)
111                             self.outputs.remove(sock)
112
113                             # Sending client leaving information to others
114                             msg = "\n(Now hung up: Client from %s)" % self.get_client_name(sock)
115                             for output in self.outputs:
116                                 send(output, msg)
117                     except socket.error as e:
118                         # Remove
119                         inputs.remove(sock)
120                         self.outputs.remove(sock)
121         self.server.close()
```

chat_server_with_select

Client Side

```
124  class ChatClient(object):
125      """ A command line chat client using select """
126
127      def __init__(self, name, port, host=SERVER_HOST):
128          self.name = name
129          self.connected = False
130          self.host = host
131          self.port = port
132          # Initial prompt
133          self.prompt='[' + '@'.join((name, socket.gethostname().split('.')[0])) + ']> '
134          # Connect to server at port
135          try:
136              self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
137              self.sock.connect((host, self.port))
138              print ("Now connected to chat server@ port %d" % self.port)
139              self.connected = True
140              # Send my name...
141              send(self.sock,'NAME: ' + self.name)
142              data = receive(self.sock)
143              # Contains client address, set it
144              addr = data.split('CLIENT: ')[1]
145              self.prompt = '[' + '@'.join((self.name, addr)) + ']> '
146          except socket.error as e:
147              print ("Failed to connect to chat server @ port %d" % self.port)
148              sys.exit(1)
```

chat_server_with_select

Client Side

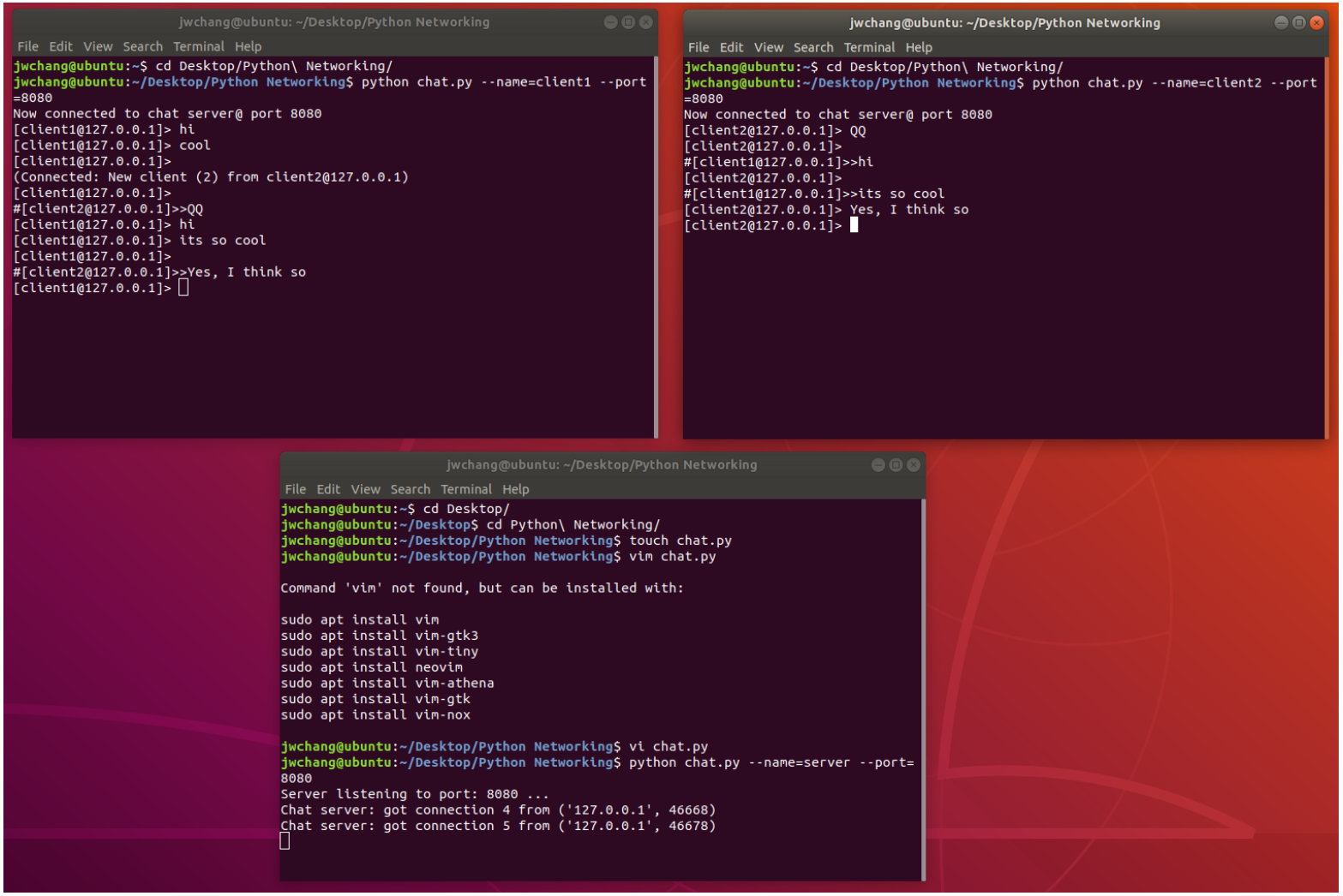
```
150     def run(self):
151         """ Chat client main loop """
152         while self.connected:
153             try:
154                 sys.stdout.write(self.prompt)
155                 sys.stdout.flush()
156                 # Wait for input from stdin and socket
157                 readable, writeable,exceptional = select.select([0, self.sock], [],[])
158                 for sock in readable:
159                     if sock == 0:
160                         data = sys.stdin.readline().strip()
161                         if data: send(self.sock, data)
162                     elif sock == self.sock:
163                         data = receive(self.sock)
164                         if not data:
165                             print ('Client shutting down.')
166                             self.connected = False
167                             break
168                         else:
169                             sys.stdout.write(data + '\n')
170                             sys.stdout.flush()
171
172             except KeyboardInterrupt:
173                 print (" Client interrupted. """)
174                 self.sock.close()
175                 break
```

chat_server_with_select

Main

```
178  if __name__ == "__main__":
179      parser = argparse.ArgumentParser(description='Socket Server Example with Select')
180      parser.add_argument('--name', action="store", dest="name", required=True)
181      parser.add_argument('--port', action="store", dest="port", type=int, required=True)
182      given_args = parser.parse_args()
183      port = given_args.port
184      name = given_args.name
185      if name == CHAT_SERVER_NAME:
186          server = ChatServer(port)
187          server.run()
188      else:
189          client = ChatClient(name=name, port=port)
190          client.run()
```

chat_server_with_select



WAIT_FOR_REMOTE_SERVICE

wait_for_remote_service

1. sudo apt install apache2
2. sudo /etc/init.d/apache2 stop
3. sudo /etc/init.d/apache2 start

```
jwchang@ubuntu:~/Desktop/Python Networking$ sudo /etc/init.d/apache2 stop
[ ok ] Stopping apache2 (via systemctl): apache2.service.
jwchang@ubuntu:~/Desktop/Python Networking$ sudo /etc/init.d/apache2 start
[ ok ] Starting apache2 (via systemctl): apache2.service.
```

wait_for_remote_service

```
1  import argparse
2  import socket
3  import errno
4  from time import time as now
5
6  DEFAULT_TIMEOUT = 120
7  DEFAULT_SERVER_HOST = 'localhost'
8  DEFAULT_SERVER_PORT = 80
9
10 class NetServiceChecker(object):...
45
46 if __name__ == '__main__':
47     parser = argparse.ArgumentParser(description='Wait for Network Service')
48     parser.add_argument('--host', action="store", dest="host",  default=DEFAULT_SERVER_HOST)
49     parser.add_argument('--port', action="store", dest="port", type=int, default=DEFAULT_SERVER_PORT)
50     parser.add_argument('--timeout', action="store", dest="timeout", type=int, default=DEFAULT_TIMEOUT)
51     given_args = parser.parse_args()
52     host, port, timeout = given_args.host, given_args.port, given_args.timeout
53     service_checker = NetServiceChecker(host, port, timeout=timeout)
54     print ("Checking for network service %s:%s ..." %(host, port))
55     if service_checker.check():
56         print ("Service is available again!")
```

wait_for_remote_service

```
10  class NetServiceChecker(object):
11      """ Wait for a network service to come online"""
12      def __init__(self, host, port, timeout=DEFAULT_TIMEOUT):
13          self.host = host
14          self.port = port
15          self.timeout = timeout
16          self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17
18      def end_wait(self):
19          self.sock.close()
20
21      def check(self):
22          """ Check the service """
23          if self.timeout:
24              end_time = now() + self.timeout
25
26          while True:
27              try:
28                  if self.timeout:
29                      next_timeout = end_time - now()
30                      if next_timeout < 0:
31                          return False
32                      else:
33                          print ("setting socket next timeout %ss" %round(next_timeout))
34                          self.sock.settimeout(next_timeout)
35                  self.sock.connect((self.host, self.port))
36              # handle exceptions
37              except socket.timeout as err:
38                  if self.timeout:
39                      return False
40              except socket.error as err:
41                  print ("Exception: %s" %err)
42              else: # if all goes well
43                  self.end_wait()
44                  return True
```

IPC_USING_SOCKETPAIRS

ipc_using_socketpairs

```python
import socket
import os

BUFSIZE = 1024

def test_socketpair():
    """ Test Unix socketpair"""
    parent, child = socket.socketpair()

    pid = os.fork()
    try:
        if pid:
            print ("@Parent, sending message...")
            child.close()

            parent.sendall(bytes("Hello from parent!"))
            # Comment out the above line and uncomment the below line for Python 2.7.
            # parent.sendall("Hello from parent!")

            response = parent.recv(BUFSIZE)
            print ("Response from child:", response)
            parent.close()

        else:
            print ("@Child, waiting for message from parent")
            parent.close()
            message = child.recv(BUFSIZE)
            print ("Message from parent:", message)

            child.sendall(bytes("Hello from child!!"))
            # Comment out the above line and uncomment the below line for Python 2.7.
            # child.sendall("Hello from child!!")

            child.close()
    except Exception as err:
        print ("Error: %s" %err)


if __name__ == '__main__':
    test_socketpair()
```

UNIX_DOMAIN_SOCKET_SERVER

unix_domain_socket_server

```python
import socket
import os
import time

SERVER_PATH = "/tmp/python_unix_socket_server"

def run_unix_domain_socket_server():
    if os.path.exists(SERVER_PATH):
        os.remove( SERVER_PATH )

    print ("starting unix domain socket server.")
    server = socket.socket( socket.AF_UNIX, socket.SOCK_DGRAM )
    server.bind(SERVER_PATH)

    print ("Listening on path: %s" %SERVER_PATH)
    while True:
        datagram = server.recv( 1024 )
        if not datagram:
            break
        else:
            print ("-" * 20)
            print (datagram)
        if "DONE" == datagram:
            break
    print ("-" * 20)
    print ("Server is shutting down now...")
    server.close()
    os.remove(SERVER_PATH)
    print ("Server shutdown and path removed.")

if __name__ == '__main__':
    run_unix_domain_socket_server()
```

UNIX_DOMAIN_SOCKET _CLIENT

unix_domain_socket_client

```python
import socket
import sys

SERVER_PATH = "/tmp/python_unix_socket_server"

def run_unix_domain_socket_client():
    """ Run "a Unix domain socket client """
    sock = socket.socket(socket.AF_UNIX, socket.SOCK_DGRAM)

    # Connect the socket to the path where the server is listening
    server_address = SERVER_PATH
    print ("connecting to %s" % server_address)
    try:
        sock.connect(server_address)
    except socket.error as msg:
        print (msg)
        sys.exit(1)

    try:
        message = "This is the message.  This will be echoed back!"
        print  ("Sending [%s]" %message)

        sock.sendall(bytes(message, 'utf-8'))
        # Comment out the above line and uncomment the below line for Python 2.7.
        # sock.sendall(message)

        amount_received = 0
        amount_expected = len(message)

        while amount_received < amount_expected:
            data = sock.recv(16)
            amount_received += len(data)
            print ("Received [%s]" % data)

    finally:
        print ("Closing client")
        sock.close()

if __name__ == '__main__':
    run_unix_domain_socket_client()
```

延伸閱讀

- Socket Programming in Python (Guide)
  - https://realpython.com/python-sockets/#socket-api-overview
- Python 网络编程
  - http://www.runoob.com/python/python-socket.html

Resource is available by
https://jiaweichang.github.io/biography/

THANKS