Computer Network Laboratory
Basic Applications of Network Programming (I)

Jiawei Chang

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

Outline

1. echo_server and echo_client by TCP
2. echo_server and echo_client by UDP
3. forking_mixin_socket_server
4. threading_mixin_socket_server
5. chat_server_with_select

echo_server by TCP

```python
1  import socket
2  import sys
3  import argparse
4
5  host = 'localhost'
6  data_payload = 2048
7  backlog = 5
8
9
10 def echo_server(port):
11     """ A simple echo server """
12     # Create a TCP socket
13     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14     # Enable reuse address/port
15     sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
16     # Bind the socket to the port
17     server_address = (host, port)
18     print ("Starting up echo server  on %s port %s" % server_address)
19     sock.bind(server_address)
20     # Listen to clients, backlog argument specifies the max no. of queued connections
21     sock.listen(backlog)
22     while True:
23         print ("Waiting to receive message from client")
24         client, address = sock.accept()
25         data = client.recv(data_payload)
26         if data:
27             print ("Data: %s" %data)
28             client.send(data)
29             print ("sent %s bytes back to %s" % (data, address))
30         # end connection
31         client.close()
32
33 if __name__ == '__main__':
34     parser = argparse.ArgumentParser(description='Socket Server Example')
35     parser.add_argument('--port', action="store", dest="port", type=int, required=True)
36     given_args = parser.parse_args()
37     port = given_args.port
38     echo_server(port)
```

python 12_echo_server_TCP.py --port=<PORT>

echo_client by TCP

```python
1   import socket
2   import sys
3
4   import argparse
5
6   host = 'localhost'
7
8   def echo_client(port):
9       """ A simple echo client """
10      # Create a TCP/IP socket
11      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12      # Connect the socket to the server
13      server_address = (host, port)
14      print ("Connecting to %s port %s" % server_address)
15      sock.connect(server_address)
16
17      # Send data
18      try:
19          # Send data
20          message = "Test message. This will be echoed"
21          print ("Sending %s" % message)
22          sock.sendall(message.encode('utf-8'))
23          # Look for the response
24          amount_received = 0
25          amount_expected = len(message)
26          while amount_received < amount_expected:
27              data = sock.recv(16)
28              amount_received += len(data)
29              print ("Received: %s" % data)
30      except socket.error as e:
31          print ("Socket error: %s" %str(e))
32      except Exception as e:
33          print ("Other exception: %s" %str(e))
34      finally:
35          print ("Closing connection to the server")
36          sock.close()
37
38  if __name__ == '__main__':
39      parser = argparse.ArgumentParser(description='Socket Server Example')
40      parser.add_argument('--port', action="store", dest="port", type=int, required=True)
41      given_args = parser.parse_args()
42      port = given_args.port
43      echo_client(port)
```

python 12_echo_client_TCP.py --port=<PORT>

4

echo_server by UDP

```python
1   import socket
2   import sys
3   import argparse
4
5   host = 'localhost'
6   data_payload = 2048
7
8   def echo_server(port):
9       """ A simple echo server """
10      # Create a UDP socket
11      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
12
13      # Bind the socket to the port
14      server_address = (host, port)
15      print ("Starting up echo server on %s port %s" % server_address)
16
17      sock.bind(server_address)
18
19      while True:
20          print ("Waiting to receive message from client")
21          data, address = sock.recvfrom(data_payload)
22
23          print ("received %s bytes from %s" % (len(data), address))
24          print ("Data: %s" %data)
25
26          if data:
27              sent = sock.sendto(data, address)
28              print ("sent %s bytes back to %s" % (sent, address))
29
30
31  if __name__ == '__main__':
32      parser = argparse.ArgumentParser(description='Socket Server Example')
33      parser.add_argument('--port', action="store", dest="port", type=int, required=True)
34      given_args = parser.parse_args()
35      port = given_args.port
36      echo_server(port)
```

python 13_echo_server_UDP.py --port=<PORT>

5

echo_client by UDP

```python
1   import socket
2   import sys
3   import argparse
4
5   host = 'localhost'
6   data_payload = 2048
7
8   def echo_client(port):
9       """ A simple echo client """
10      # Create a UDP socket
11      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
12
13      server_address = (host, port)
14      print ("Connecting to %s port %s" % server_address)
15      message = 'This is the message.  It will be repeated.'
16
17      try:
18
19          # Send data
20          message = "Test message. This will be echoed"
21          print ("Sending %s" % message)
22          sent = sock.sendto(message.encode('utf-8'), server_address)
23
24          # Receive response
25          data, server = sock.recvfrom(data_payload)
26          print ("received %s" % data)
27
28      finally:
29          print ("Closing connection to the server")
30          sock.close()
31
32  if __name__ == '__main__':
33      parser = argparse.ArgumentParser(description='Socket Server Example')
34      parser.add_argument('--port', action="store", dest="port", type=int, required=True)
35      given_args = parser.parse_args()
36      port = given_args.port
37      echo_client(port)
```

python 13_echo_client_UDP.py --port=<PORT>

MULTIPLEXING SOCKET I/O FOR BETTER PERFORMANCE

forking_mixin_socket_server

```python
import os
import socket
import threading
import socketserver


SERVER_HOST = 'localhost'
SERVER_PORT = 0 # tells the kernel to pickup a port dynamically
BUF_SIZE = 1024
ECHO_MSG = 'Hello echo server!'


class ForkedClient():
    """ A client to test forking server"""
    def __init__(self, ip, port):
        # Create a socket
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # Connect to the server
        self.sock.connect((ip, port))

    def run(self):
        """ Client playing with the server"""
        # Send the data to server
        current_process_id = os.getpid()
        print ('PID %s Sending echo message to the server : "%s"' % (current_process_id, ECHO_MSG))

        sent_data_length = self.sock.send(bytes(ECHO_MSG, 'utf-8'))

        print ("Sent: %d characters, so far..." %sent_data_length)

        # Display server response
        response = self.sock.recv(BUF_SIZE)
        print ("PID %s received: %s" % (current_process_id, response[5:]))

    def shutdown(self):
        """ Cleanup the client socket """
        self.sock.close()
```

8

forking_mixin_socket_server

```python
class ForkingServerRequestHandler(socketserver.BaseRequestHandler):
    def handle(self):
        # Send the echo back to the client

        #received = str(sock.recv(1024), "utf-8")
        data = str(self.request.recv(BUF_SIZE), 'utf-8')

        current_process_id = os.getpid()
        response = '%s: %s' % (current_process_id, data)
        print ("Server sending response [current_process_id: data] = [%s]" %response)
        self.request.send(bytes(response, 'utf-8'))
        return


class ForkingServer(socketserver.ThreadingMixIn,
                    socketserver.TCPServer,
                    ):
    """Nothing to add here, inherited everything necessary from parents"""
    pass


def main():
    # Launch the server
    server = ForkingServer((SERVER_HOST, SERVER_PORT), ForkingServerRequestHandler)
    ip, port = server.server_address # Retrieve the port number
    server_thread = threading.Thread(target=server.serve_forever)
    server_thread.setDaemon(True) # don't hang on exit
    server_thread.start()
    print ("Server loop running PID: %s" %os.getpid())

    # Launch the client(s)

    client1 =  ForkedClient(ip, port)
    client1.run()

    print("First client running")

    client2 =  ForkedClient(ip, port)
    client2.run()

    print("Second client running")

    # Clean them up
    server.shutdown()
    client1.shutdown()
    client2.shutdown()
    server.socket.close()


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

```
Server loop running PID: 16808
PID 16808 Sending echo message to the server : "Hello echo server!"
Sent: 18 characters, so far...
Server sending response [current_process_id: data] = [16808: Hello echo server!]
PID 16808 received: b': Hello echo server!'
First client running
PID 16808 Sending echo message to the server : "Hello echo server!"
Sent: 18 characters, so far...
Server sending response [current_process_id: data] = [16808: Hello echo server!]
PID 16808 received: b': Hello echo server!'
Second client running
```

threading_mixin_socket_server

```python
import os
import socket
import threading
import socketserver

SERVER_HOST = 'localhost'
SERVER_PORT = 0 # tells the kernel to pickup a port dynamically
BUF_SIZE = 1024


def client(ip, port, message):
    """ A client to test threading mixin server"""
    # Connect to the server
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((ip, port))
    try:
        sock.sendall(bytes(message, 'utf-8'))
        response = sock.recv(BUF_SIZE)
        print ("Client received: %s" %response)
    finally:
        sock.close()


class ThreadedTCPRequestHandler(socketserver.BaseRequestHandler):
    """ An example of threaded TCP request handler """
    def handle(self):
        data = self.request.recv(1024)
        cur_thread = threading.current_thread()
        response = "%s: %s" %(cur_thread.name, data)
        self.request.sendall(bytes(response, 'utf-8'))

class ThreadedTCPServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
    """Nothing to add here, inherited everything necessary from parents"""
    pass
```

threading_mixin_socket_server

```python
if __name__ == "__main__":
    # Run server
    server = ThreadedTCPServer((SERVER_HOST, SERVER_PORT), ThreadedTCPRequestHandler)
    ip, port = server.server_address # retrieve ip address

    # Start a thread with the server -- one  thread per request
    server_thread = threading.Thread(target=server.serve_forever)
    # Exit the server thread when the main thread exits
    server_thread.daemon = True
    server_thread.start()
    print ("Server loop running on thread: %s"  %server_thread.name)

    # Run clients
    client(ip, port, "Hello from client 1")
    client(ip, port, "Hello from client 2")
    client(ip, port, "Hello from client 3")

    # Server cleanup
    server.shutdown()
```

```
Server loop running on thread: Thread-14
Client received: b"Thread-15: b'Hello from client 1'"
Client received: b"Thread-16: b'Hello from client 2'"
Client received: b"Thread-17: b'Hello from client 3'"
```

Thinking Time

Process v.s. Thread
What is relation between them?

chat_server_with_select

```python
1    import select
2    import socket
3    import sys
4    import signal
5    import pickle
6    import struct
7    import argparse
8
9    SERVER_HOST = 'localhost'
10   CHAT_SERVER_NAME = 'server'
```

```python
12   # Some utilities
13   def send(channel, *args):
14       buffer = pickle.dumps(args)
15       value = socket.htonl(len(buffer))
16       size = struct.pack("L",value)
17       channel.send(size)
18       channel.send(buffer)
19
20   def receive(channel):
21       size = struct.calcsize("L")
22       size = channel.recv(size)
23       try:
24           size = socket.ntohl(struct.unpack("L", size)[0])
25       except struct.error as e:
26           return ''
27       buf = ""
28       while len(buf) < size:
29           buf = channel.recv(size - len(buf))
30       return pickle.loads(buf)[0]
```

chat_server_with_select

Server Side

```python
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))
```

14

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()
```
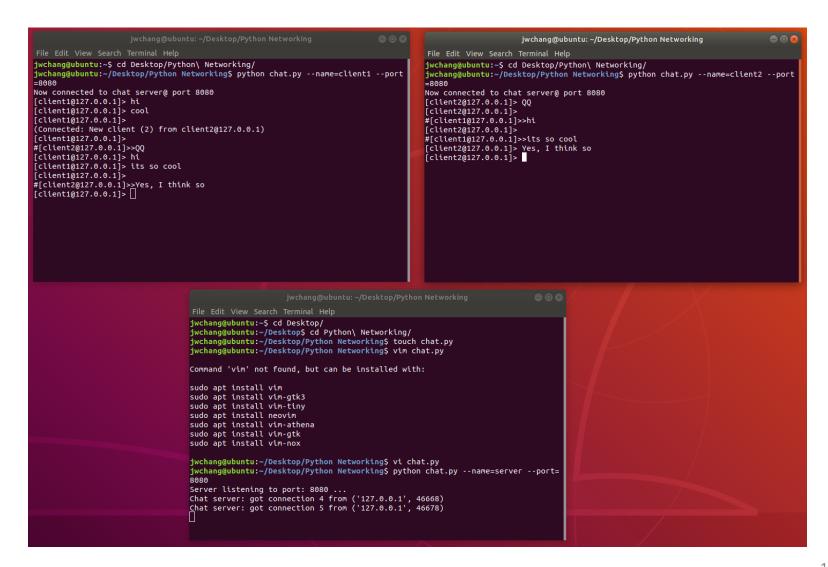
15

chat_server_with_select

Client Side

```python
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

```python
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

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