Computer Network Laboratory

Basic Network Programming (II)

Jiawei Chang

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

Outline

1. modify_buff_size
2. socket_modes
3. reuse_socket_address
4. print_machine_time
5. echo_server and echo_client by TCP
6. echo_server and echo_client by UDP
7. forking_mixin_socket_server
8. threading_mixin_socket_server

modify_buff_size

```
import socket

SEND_BUF_SIZE = 4096
RECV_BUF_SIZE = 4096

def modify_buff_size():
    sock = socket.socket( socket.AF_INET, socket.SOCK_STREAM )

    # Get the size of the socket's send buffer
    bufsize = sock.getsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF)
    print ("Buffer size [Before]:%d" %bufsize)

    sock.setsockopt(socket.SOL_TCP, socket.TCP_NODELAY, 1)
    sock.setsockopt(
            socket.SOL_SOCKET,
            socket.SO_SNDBUF,
            SEND_BUF_SIZE)
    sock.setsockopt(
            socket.SOL_SOCKET,
            socket.SO_RCVBUF,
            RECV_BUF_SIZE)
    bufsize = sock.getsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF)
    print ("Buffer size [After]:%d" %bufsize)

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

```
Buffer size [Before]:65536
Buffer size [After]:4096
```

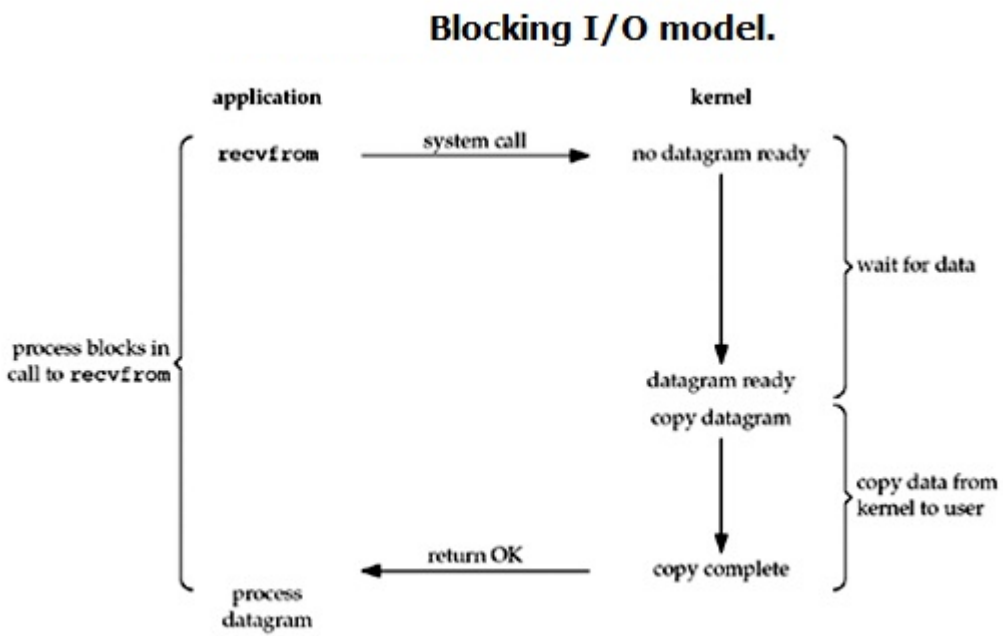
socket_modes

```python
import socket

def test_socket_modes():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setblocking(0) #0 = non-blocking mode, 1 = blocking mode and default value
    s.settimeout(0.5)
    s.bind(("127.0.0.1", 0))

    socket_address = s.getsockname()
    print ("Trivial Server launched on socket: %s" %str(socket_address))
    while(1):
        s.listen(1)

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

```
Trivial Server launched on socket: ('127.0.0.1', 64604)
```

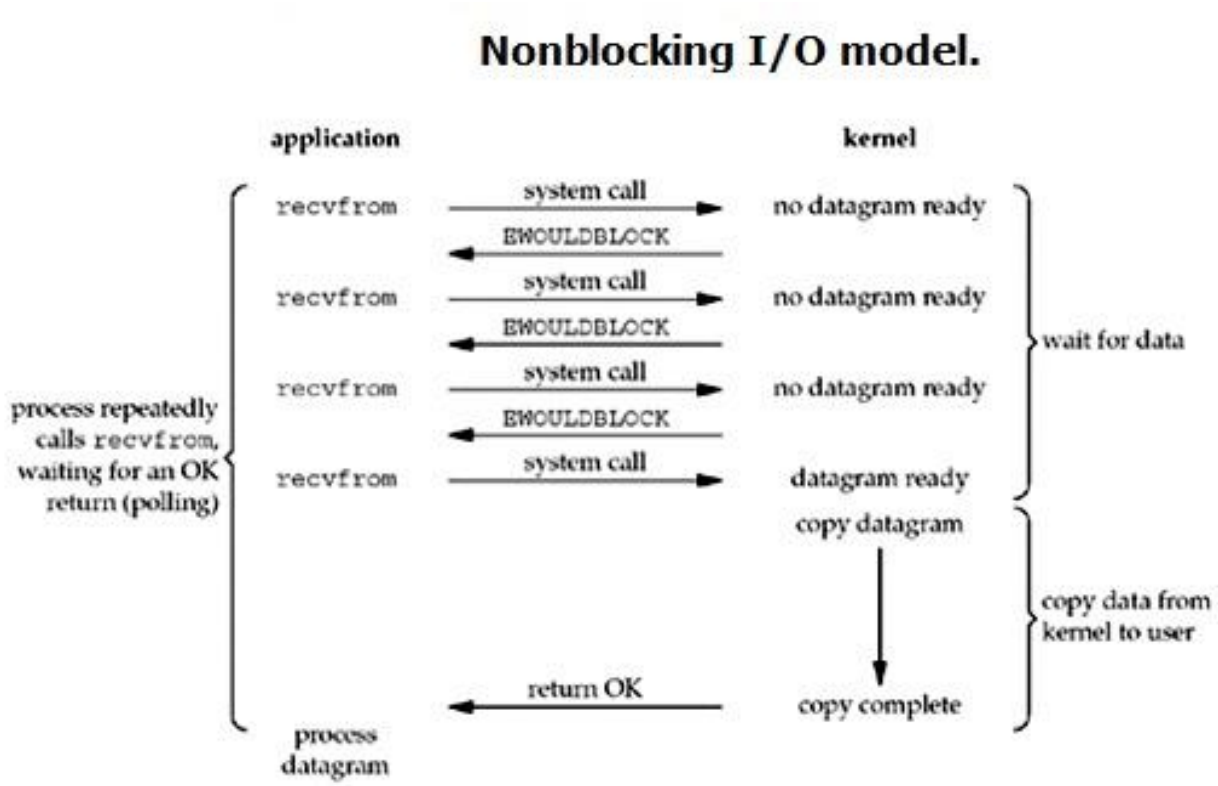
Blocking and non-Blocking

- 在預設的狀況，Server 通常是開啟 block 機制(同步)，將導致程式在執行時會被阻塞住，導致暫停執行。
- 非同步 I/O 的想法其實很單純，假如程式在執行過程中因為 I/O 暫停，但如果不會被阻塞住就能暫時把控制權切換給其它程式，這樣就不會浪費執行時間。



Blocking and non-Blocking

1. 當使用者程序發出read操作時，如果kernel中的資料還沒有準備好，那麼它並不會block使用者程序，而是立刻返回一個error。
2. 從使用者程序角度講 ，它發起一個read操作後，並不需要等待，而是馬上就得到了一個結果。
3. 使用者程序判斷結果是一個error時，它就知道資料還沒有準備好，於是它可以再次傳送read操作。
4. 一旦kernel中的資料準備好了，並且又再次收到了使用者程序的system call，那麼它馬上就將資料拷貝到了使用者記憶體，然後返回。
5. 所以，使用者程序其實是需要不斷的主動詢問kernel資料好了沒有。

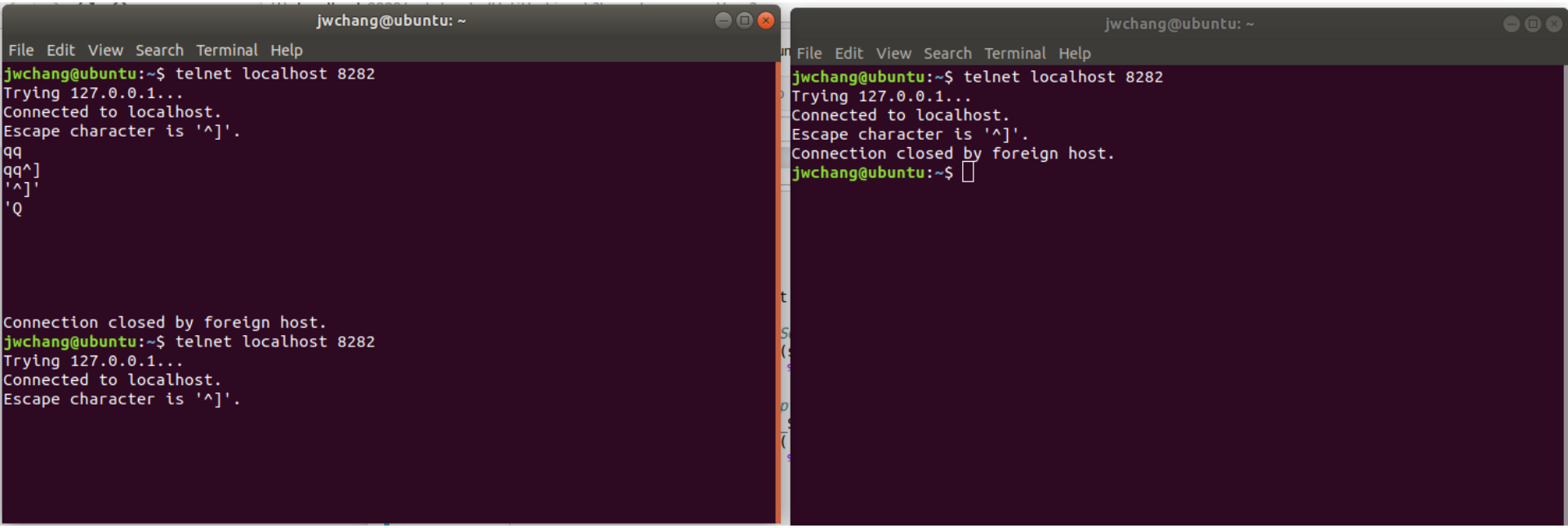


reuse_ socket_ address

```python
import socket
import sys


def reuse_socket_addr():
    sock = socket.socket( socket.AF_INET, socket.SOCK_STREAM )

    # Get the old state of the SO_REUSEADDR option
    old_state = sock.getsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR )
    print ("Old sock state: %s" %old_state)

    # Enable the SO_REUSEADDR option
    sock.setsockopt( socket.SOL_SOCKET, socket.SO_REUSEADDR, 1 )
    new_state = sock.getsockopt( socket.SOL_SOCKET, socket.SO_REUSEADDR )
    print ("New sock state: %s" %new_state)

    local_port = 8282

    srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    srv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    srv.bind( ('', local_port) )
    srv.listen(1)
    print ("Listening on port: %s " %local_port)
    while True:
        try:
            connection, addr = srv.accept()
            print ('Connected by %s:%s' % (addr[0], addr[1]))
        except KeyboardInterrupt:
            break
        except socket.error as msg:
            print ('%s' % (msg,))


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

```
Old sock state: 0
New sock state: 1
Listening on port: 8282
Connected by 127.0.0.1:36272
Connected by 127.0.0.1:36406
Connected by 127.0.0.1:36412
```

reuse_socket_address



print_machine_time

```
import ntplib
from time import ctime


def print_time():
    ntp_client = ntplib.NTPClient()
    response = ntp_client.request('pool.ntp.org')
    print (ctime(response.tx_time))


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

```
Sun Mar  3 22:43:32 2019
```

echo_server by TCP

```python
import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048
backlog = 5


def echo_server(port):
    """ A simple echo server """
    # Create a TCP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Enable reuse address/port
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # Bind the socket to the port
    server_address = (host, port)
    print ("Starting up echo server  on %s port %s" % server_address)
    sock.bind(server_address)
    # Listen to clients, backlog argument specifies the max no. of queued connections
    sock.listen(backlog)
    while True:
        print ("Waiting to receive message from client")
        client, address = sock.accept()
        data = client.recv(data_payload)
        if data:
            print ("Data: %s" %data)
            client.send(data)
            print ("sent %s bytes back to %s" % (data, address))
        # end connection
        client.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)
```

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

echo_client by TCP

```python
import socket
import sys

import argparse

host = 'localhost'

def echo_client(port):
    """ A simple echo client """
    # Create a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Connect the socket to the server
    server_address = (host, port)
    print ("Connecting to %s port %s" % server_address)
    sock.connect(server_address)

    # Send data
    try:
        # Send data
        message = "Test message. This will be echoed"
        print ("Sending %s" % message)
        sock.sendall(message.encode('utf-8'))
        # Look for the response
        amount_received = 0
        amount_expected = len(message)
        while amount_received < amount_expected:
            data = sock.recv(16)
            amount_received += len(data)
            print ("Received: %s" % data)
    except socket.error as e:
        print ("Socket error: %s" %str(e))
    except Exception as e:
        print ("Other exception: %s" %str(e))
    finally:
        print ("Closing connection to the server")
        sock.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(port)
```

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

echo_server by UDP

```python
import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048

def echo_server(port):
    """ A simple echo server """
    # Create a UDP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # Bind the socket to the port
    server_address = (host, port)
    print ("Starting up echo server on %s port %s" % server_address)

    sock.bind(server_address)

    while True:
        print ("Waiting to receive message from client")
        data, address = sock.recvfrom(data_payload)

        print ("received %s bytes from %s" % (len(data), address))
        print ("Data: %s" %data)

        if data:
            sent = sock.sendto(data, address)
            print ("sent %s bytes back to %s" % (sent, address))


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)
```

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

echo_client by UDP

```python
import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048

def echo_client(port):
    """ A simple echo client """
    # Create a UDP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    server_address = (host, port)
    print ("Connecting to %s port %s" % server_address)
    message = 'This is the message.  It will be repeated.'

    try:

        # Send data
        message = "Test message. This will be echoed"
        print ("Sending %s" % message)
        sent = sock.sendto(message.encode('utf-8'), server_address)

        # Receive response
        data, server = sock.recvfrom(data_payload)
        print ("received %s" % data)

    finally:
        print ("Closing connection to the server")
        sock.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(port)
```

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

forking_mixin_socket_server

```
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()
```

forking_mixin_socket_server

```
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

```
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

```
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?

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