# Homework 07 Network Protocol Timeline

**Following table shows the source code of the server/client program and its execution sequence. Complete the execution sequence table for a given program code.**

| Server | Client |
|---|---|
| `01: import socket` | `01: import socket` |
| `02:` | `02:` |
| `03: HOST = '127.0.0.1'   # Standard loopback interface address (localhost)` | `03: HOST = 'localhost'   # The server's hostname or IP address` |
| `04: PORT = 65432         # Port to listen on (non-privileged ports are > 1023)` | `04: PORT = 65432         # The port used by the server` |
| `05:` | `05:` |
| `06: with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:` | `06: with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:` |
| `07:     s.bind((HOST, PORT))` | `07:     s.connect((HOST, PORT))` |
| `08:     s.listen()` | `08:     print("I am going to send Hello world!")` |
| `09:     conn, addr = s.accept()` | `09:     input("Press enter to proceed")` |
| `10:     with conn:` | `10:     s.sendall(b'Hello, world')` |
| `11:         print('Connected by', addr)` | `11:     data = s.recv(1024)` |
| `12:         while True:` | `12:` |
| `13:             data = conn.recv(1024)` | `13: print('Received', repr(data))` |
| `14:             if not data:` | |
| `15:                 break` | |
| `16:             print("Received {} from client:{}".format(data, HOST))` | |
| `17:             input("Press enter to proceed")` | |
| `18:             conn.sendall(data)` | |

| Execution Sequence | |
|---|---|
| 01: import socket | 01: import socket |
| 03: HOST = ‘127.0.0.1’ | 03: HOST = ‘localhost’ |
| 04: PORT = 65432 | 04: PORT = 65432 |
| 06: with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s: | 06: with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s: |
| 07: s.bind((HOST,PORT)) | 07: s.connect((HOST, PORT)) |
| 08: s.listen() | |
| 09: conn,addr = s.accept() | |
| 10: with conn: | 08: print(”I am going to send Hello world!”) |
| 11: print(‘Connected by”, addr) | 09: input(“Press enter to proceed”) |
| 12: while True: | |
| 13: data = conn.recv(1024) | |
| | *User have pressed the enter key* |
| | 10: s.sendadd(b’Hello, world’) |
| 14: if not data: | 11: data = s.recv(1024) |
| 16: print(”Received {} from client {}”.format(data, HOST)) | |
| 17: input(“Press enter to proceed”) | |
| *User have pressed the enter key* | |
| 18: conn.sendall(data | |
| 12: while True: | 13: print(”received”, repr(data)) |
| 13: data = conn.recv(1024) | |
| | *Client terminated, and connection closed* |
| 14: if not data: | |
| 15: break | |

1. Application 1

**Server**

```
import socket

HOST = '127.0.0.1'   # Standard loopback interface address (localhost)
PORT = 65432         # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            if data.decode('ascii') == "what is your name?":
                print("Received what is your name")
                print("Send my name")
                conn.sendall(b'My name is OOO')
            elif data.decode('ascii') == "How old are you?":
                print("Received how old are you")
                print("Send my age")
                conn.sendall(b'I am 22')
            else:
                print("Communication Terminated")
                break
```

**Client**

```
import socket

HOST = '127.0.0.1'   # The server's hostname or IP address
PORT = 65432         # The port used by the server

client_state = ['ask_name', 'asked_name', 'ask_age', 'asked_age', 'termination']

cur_state = 'ask_name'

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    print("Greeing with server")
    input("Press enter to proceed")
    while True:
        if cur_state == 'ask_name':
            s.sendall(b'what is your name?')
            cur_state = 'asked_name'
        elif cur_state == 'asked_name':
            data = s.recv(1024)
            recv_str = data.decode('ascii')
            if recv_str[:10] == 'My name is':
                cur_state = 'ask_age'
            print(recv_str)
        elif cur_state == 'ask_age':
            s.sendall(b'How old are you?')
            cur_state = 'asked_age'
        elif cur_state == 'asked_age':
            data = s.recv(1024)
            recv_str = data.decode('ascii')
            if recv_str[:4] == 'I am':
                cur_state = 'termination'
            print(recv_str)
        else:
            break
```

## 2. Application 2

**Server**

```
import socket
import types
import selectors

sel = selectors.DefaultSelector()

def accept_wrapper(sock):
    conn, addr = sock.accept()   # Should be ready to read
    print('accepted connection from', addr)
    conn.setblocking(False)
    data = types.SimpleNamespace(addr=addr, inb=b'', outb=b'')
    events = selectors.EVENT_READ | selectors.EVENT_WRITE
    sel.register(conn, events, data=data)

def service_connection(key, mask):
    sock = key.fileobj
    data = key.data
    if mask & selectors.EVENT_READ:
        recv_data = sock.recv(1024)   # Should be ready to read
        if recv_data:
            data.outb += recv_data
        else:
            print('closing connection to', data.addr)
            sel.unregister(sock)
            sock.close()
    if mask & selectors.EVENT_WRITE:
        if data.outb:
            print('echoing', repr(data.outb), 'to', data.addr)
            sent = sock.send(data.outb)   # Should be ready to write
            data.outb = data.outb[sent:]

host = 'localhost'
port = 65432

lsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
lsock.bind((host, port))
lsock.listen()
print('listening on', (host, port))
lsock.setblocking(False)
sel.register(lsock, selectors.EVENT_READ, data=None)

try:
        while True:
            events = sel.select(timeout=None)
            for key, mask in events:
                if key.data is None:
                    accept_wrapper(key.fileobj)
                else:
                    service_connection(key, mask)

except KeyboardInterrupt:
    print("caught keyboard interrupt, exiting")
finally:
    sel.close()
```

**Client**

```
import socket
import types
import selectors

sel = selectors.DefaultSelector()
messages = [b'Message 1 from client.', b'Message 2 from client.']

def start_connections(host, port, num_conns):
    server_addr = (host, port)
    for i in range(0, num_conns):
        connid = i + 1
        print('starting connection', connid, 'to', server_addr)
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setblocking(False)
        sock.connect_ex(server_addr)
        events = selectors.EVENT_READ | selectors.EVENT_WRITE
        data = types.SimpleNamespace(connid=connid,
                                     msg_total=sum(len(m) for m in messages),
                                     recv_total=0,
                                     messages=list(messages),
                                     outb=b'')
        sel.register(sock, events, data=data)

def service_connection(key, mask):
    sock = key.fileobj
    data = key.data
    if mask & selectors.EVENT_READ:
        recv_data = sock.recv(1024)   # Should be ready to read
        if recv_data:
            print('received', repr(recv_data), 'from connection', data.connid)
            data.recv_total += len(recv_data)
        if not recv_data or data.recv_total == data.msg_total:
            print('closing connection', data.connid)
            sel.unregister(sock)
            sock.close()
    if mask & selectors.EVENT_WRITE:
        if not data.outb and data.messages:
            data.outb = data.messages.pop(0)
        if data.outb:
            print('sending', repr(data.outb), 'to connection', data.connid)
            sent = sock.send(data.outb)   # Should be ready to write
            data.outb = data.outb[sent:]

host = 'localhost'
port = 65432
start_connections(host, port, 2)

while True:
    events = sel.select(timeout=1)
    if events != None:
        for key, mask in events:
            service_connection(key, mask)

    if not sel.get_map():
            break

sel.close()
```