Project 0

Danyang Zhuo
CSE 461 15sp Section #2
Proj0

- You have to implement:
  - Multi-Thread UDP Server
  - Multi-Thread UDP Client
  - Async IO UDP Server
  - Async IO UDP Client

- Your server must support multiple clients
Submit Format

• Use any programming language you like
  – Tips: TAs are familiar with C/C++, Java, Python

• Must write scripts to make testing generic:
  – Server:
    • ./server <portnum>
  – Client:
    • ./client <hostname> <portnum>
UDP (User Datagram Protocol)

- Communication Protocol on the top of Internet Protocol
  - Use IP address and port as a identifier
- Server:
  - Bind to a port
- Client:
  - Connect through Server’s IP address and port
- Send/Receive
Server

//Bind to a port (5533):
DatagramSocket socket = new DatagramSocket(5533);

// Receive Message
byte[] buf = new byte[256];
DatagramPacket p = new DatagramPacket(buf, buf.length);
socket.receive(p);
Client

// Connect to server
DatagramSocket socket = new DatagramSocket();
socket.connect(InetAddress.getByName("127.0.0.1"), 5533);

// Send Message (byte[] buffer)
DatagramPacket p = new DatagramPacket(buffer, buffer.length);
socket.send(p);
Demo

- UDP Server
- UDP Client
Interactive Client/Server

• Use one thread to listen to user command
• Use another thread to send/receive message
Demo

• Interactive UDP Server
Multi-Thread Server

- Packet Session2
- Session 1 Thread
  - Packet Session2
- Session 2 Thread
  - Packet Session2
- Session 3 Thread
  - Packet Session4
- Session 4 Thread
  - Packet Session4

UDP Listening Thread
Multi-thread Server

• Easy to reason
  – Each thread handles a session
  – All the logic for a given session is in one thread

• High overhead
  – Each thread to handle the current incoming packet can be blocked by other threads
  – High memory utilization
Multi-thread Server

Packet Session2

UDP Listening Thread

Session 1 Thread

Packet Session2

Session 2 Thread

Session 3 Thread

Session 4 Thread

Active Thread
Multi-thread Server

- Packet Session 2
- UDP Listening Thread
- Session 1 Thread
- Packet Session 2
- Session 2 Thread
- Session 3 Thread
- Session 4 Thread
- Active Thread
Multi-thread Server

- Packet Session2
- UDP Listening Thread
- Session 1 Thread
- Packet Session2
- Session 2 Thread
- Session 3 Thread
- Session 4 Thread

Active Thread

May be blocked by other threads for a long time
ConcurrentLinkedQueue

• UDP listening thread is writing to a data structure that session thread is reading from
• Race condition may happen
• Java has a race free implementation of a queue
Non-blocking IO

UDP Listening Thread

Packet Session2

Packet Session4

Update data structure
Why is this design block-free?

• Server is always processing incoming packet rather than waiting for the correct thread to wake up

• High performance

• Slightly harder to reason
Demo

• Non-blocking Server

• You non-blocking server and client still should have 2 threads because 1 thread is handling user input
Tips

• Use a shared Object (Java class) implementation for p0p packet format for both server and client

• ByteBuffer is easier to use than byte[]
• TimerTask for timeout implementation

• Start Early!!!
• Ask Questions!!
Reference

• Java:

• Node.js

• Python
DatagramSocket socket = new DatagramSocket(5533);
    while (true) {
        byte[] buf = new byte[256];
        DatagramPacket p = new DatagramPacket(buf, buf.length);
        socket.receive(p);
        String string = new String(buf);
        System.out.println(string);
        if (string.contentEquals("BYE")) {
            break;
        }
    }
    socket.close();
Sample Code – UDP Client

String msg = args[0];
DatagramSocket socket = new DatagramSocket();
socket.connect(InetAddress.getByName("127.0.0.1"), 5533);
ByteBuffer buffer = ByteBuffer.allocate(256);
buffer.asCharBuffer().put(msg);
DatagramPacket p = new DatagramPacket(buffer.array(), buffer.array().length);
socket.send(p);
socket.close();
static class InputWatcher extends Thread {
    @Override
    public void run() {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String msg;
        try {
            while((msg = reader.readLine()) != null && msg.length() != 0) {
                System.out.println("You just typed: "+msg);
            }
        } catch (IOException e) {
            // A Java programmer would know what to do here...
        }
        System.exit(0);
    }
}
Selector selector = Selector.open();
DatagramChannel channel = DatagramChannel.open();
channel.configureBlocking(false);
channel.socket().bind(new InetSocketAddress(5533));
channel.register(selector, SelectionKey.OP_READ);

while (true) {
    selector.select();
    Iterator<SelectionKey> keyIter = selector.selectedKeys().iterator();
    while (keyIter.hasNext()) {
        SelectionKey key = keyIter.next();
        if (key.isReadable()) {
            handle(channel);
        }
        keyIter.remove();
    }
}