CS455 - Introduction To Distributed Systems

[Lab Session 5]

Brandon Gildemaster
Computer Science
Colorado State University

Colorado State University

Topics Covered in Today's Lab

- Quiz 3 Review
- Logistics
- HW2
  - Recommendations
  - Discussion
- Another look at NIO

NOTE: Feel free to bring laptops, code, and questions!

Quiz 3 Review [1/4]

1. In IPv6, intermediate routers will fragment a packet that is larger than the MTU (Maximum Transmission Unit) of the network hop over which the packet needs to be transmitted.
    - **False**
2. To detect packet corruptions, IPv6 includes a 32-bit checksum that is amenable to recovering from dual bit-flips/errors.
    - **False**
3. The IPv6 header does not include a header length field because the size of the IPv6 packet header is fixed at 40 bytes.
    - **True**

Quiz 3 Review [2/4]

4. Port numbers in TCP and UDP are limited to being between 0 through 65535 because:
    - The port field in the IPv4 (and IPv6) header is 16 bits long.
    - **The port field in the TCP and UDP header is 16 bits long.**
    - The operating system (and kernel) libraries use the data type primitive, short, which is 16 bits to identify queues associated with a particular high-level transport protocol such TCP and UDP.
    - All of the above
5. In UDP if the queue associated with a port is full, and a new datagram packet arrives, that packet will be discarded.
    - **True**

Quiz 3 Review [3/4]

6. The size of the TCP sliding window is based on:
    - The Delay x Bandwidth product between the sender and receiver
    - The memory set aside by the sender for the particular connection
    - **The memory set aside by the receiver for the particular connection**
    - All of the above
7. In TCP each byte has a sequence number associated with it.
    - **True**

Quiz 3 Review [4/4]

8. Flow control in TCP is:
    - About how hosts & networks interact with the objective of ensuring that the senders don't cause switches and links to be overloaded.
    - **An end-to-end issue where the sender tries not to overrun the capacity of the receiver**
    - Effective fragmentation of byte streams into variable-sized segments.
    - All of the above
9. Threads within a process must have their own stack.
    - **True**
10. Threads within a process cannot share the process heap.
    - **False**

Logistics

- HW 1 late submission period is closed
- Written component is due by 5:00
- HW 2
  - Please do not run very extreme test cases
    - 1 million messages per second with 10,000 nodes
    - 200 threads in each thread pool
  - Also try to test a few days before the submission

HW 2

Milestones

- Milestone 1: Thread pool
  - Many ways to implement
  - Linked blocking queue
  - Wait / notify
- Milestone 2: Use the java.nio.channels.Selector class to register and deregister incoming SelectionKeys on the server side. You should also be able to create randomized data packets as byte arrays.

Structure

- cs455.scaling.client
- cs455.scaling.server
- cs455.scaling.task
- cs455.scaling.util
    - hashing

Less guidance than HW1, try to think through your design. Where should thread pool, statistics trackers, etc… go?

Timer tasks

```java
import java.util.*;

public class PrintInterval {
    public static void main(String[] args) {

        Timer timer = new Timer();

        PrintTask printTask = new PrintTask();

        timer.scheduleAtFixedRate(printTask, 0L, 5000L);

    }
}

class PrintTask extends TimerTask {
    public void run() {
        System.out.println("Hello World");
    }
}
```

- Expresses intention better than thread.sleep()
  - Code readability
- Timer object runs on one thread, can have one thread execute many different timer tasks
- Other built in functionality

Messages

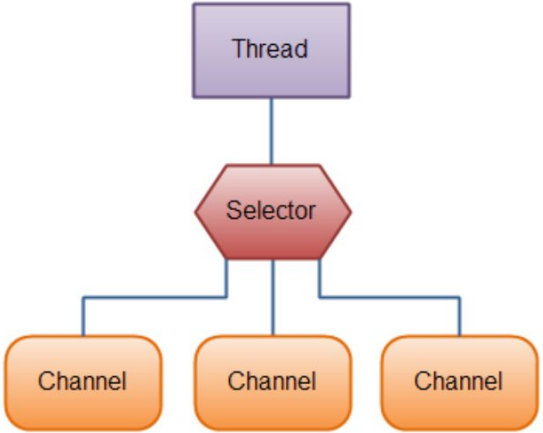
- Client randomly generates an 8kB byte array
- Computes hash
- Sends unhashed byte array to server
- Server computes the hash of the byte array and sends the hash back to client

Padding hashes

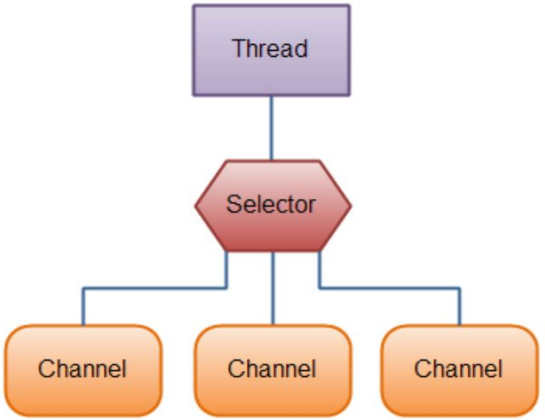
```
public String SHA1FromBytes(byte[] data) {
    MessageDigest digest = MessageDigest.getInstance("SHA1");
    byte[] hash  = digest.digest(data);
    BigInteger hashInt = new BigInteger(1, hash);

    return hashInt.toString(16);
}
```

- Produces a 20 byte hash value
  - Converted to hexadecimal
- 0's dropped from beginning of string
  - Sometimes hashInt.toString(16) will return 40 characters, other times less
- Problem: client needs to know how many bytes to read ahead of time
- Pad with 0's
- Send length of hash along with the hash

Another look at NIO



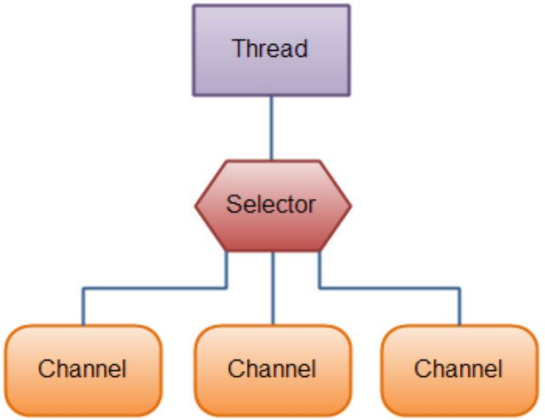
- 3 main constructs
  - Channels - communication channels, associated with an underlying socket connection
  - Selectors - object through which a single thread can monitor and service multiple channels
  - Selection keys - each channel registered with the selector has an associated key
    - Interest set
      - What kind of activity the selector will monitor for
    - Ready set
      - What operations the channel associated with the key is ready for



Selector registration

- When you register a channel with the selector you need to set what the selector will monitor for
  - Connect, Accept, Read, Write
- The server socket channel will be registered with accept for example
- A key is associated with each registered channel
- You may set multiple interests for each key (read and write)
  - Interest set
- Selector monitors all channels
  - Takes action when there is activity on any one channel (ready set)

Interest set



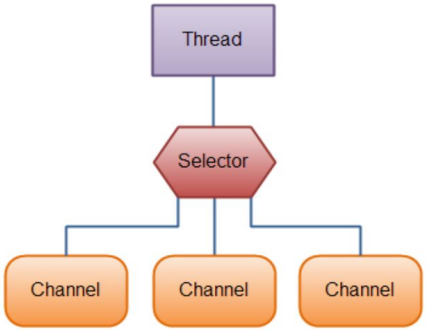
- Writing to and reading from a channel are independent of the interest set
- The interest set is only used by the selector
- If you do not register write interest with the interest set, you can still write to the channel from any random thread

When would you want to register write interest

- When channel.write() returns 0, this means 0 bytes were written and the channel is not currently writable
  - Could be full
- Rather than spinning in the while loop, you can register write interest with the selector, then when the channel does become writable the selector.select() call will pick it up
- Rare situation
- Remember to turn off write interest when done so the selector does not continuously loop

```
ByteBuffer buffer = ByteBuffer.wrap( byteArray );
while ( buffer.hasRemaining() )
{
    socketChannel.write( buffer );
}
```

Read and write interest




```
while (true) {

    // block until one or more channels have activity
    selector.select();

    // get keys that have activity
    Set<SelectionKey> selectedKeys = selector.selectedKeys();

    // loop over keys
    Iterator<SelectionKey> iter = selectedKeys.iterator();

    // get each key and check isAcceptable(), isReadable()
    while (iter.hasNext()) {

        SelectionKey key = iter.next();
        if (key.isAccceptable()) { }
        if (key.isReadable()) { }
        if (key.isWritable()) { }
    }
}
```

- selector.select() blocks, selector.selectNow() does not block
- Clients registered for read interest
- Channels will typically always be writable
- Registering write interest will mean there is always almost always activity on that channel
- Could de-registering read interest after you already know there is something to read so the selector doesn't keep looping
- Not writable: buffer is full

Wrap up

- Advice - focus on implementing the functionality described in the point breakdown first, then go for corner cases or things you may be interested in adding. This helps you allocate your time so you don't spend time working on things that contribute very little to your grade on the assignment
- Questions / discussion?