CS455 – Lab Session 05

NAMAN SHAH

02/17/2017

Agenda

- Quiz-3 : review
- HW2-PC : Communication using Java NIO
- HW2-PC : Restrictions and Suggestions

Quiz-3 review

1. Consider a class A that has two synchronized methods s1() and s2(); this class also has two unsynchronized methods u1() and u2(). Class A was used to create two object instances, a1 and a2, in a particular process P. Within the process P, there are N threads that are represented as T1, T2, …, TN.

(a) Threads T1, T2, …, TN can all be active in instance a1 at the same time and have different program counters. [**True**/False]

(b) Threads T1 and T2 can be active inside method a1.s1() at the same time. [True/**False**]

Quiz-3 review

(c) Thread T1 can be active in a1.s1() and Thread T2 can be active in a1.s2() at the same time. [True/**False**]

(d) Thread T1 can be active in a1.u1() and Thread T2 can be active in a1.u2() at the same time. [**True**/False]

(e) Thread T1 can be active in a1.s1() and Thread T2 can be active in a2.s2() at the same time. [**True**/False]

Quiz-3 review

2. The scope of a lock impacts the degree of concurrency for threads within a process. [**True**/False]

3. Java uses the same mutex lock to ensure thread safety for both static synchronized methods and non-static synchronized methods. [True/**False**]

4. The `volatile` keyword for a variable is used for performance reasons because it allows Threads to cache that variable in a register. [True/**False**]

Quiz-3 review

6. Which of the following statement is incorrect about wait-and-notify in Java?

a) It has an inherent race condition that is solved by the JVM.

b) It can be used as a communication mechanism between threads.

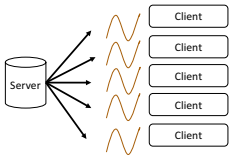
c) wait() and notify() methods are inherited from the Object class.

d) It can be used instead of synchronization to achieve thread safety.

HW2-PC

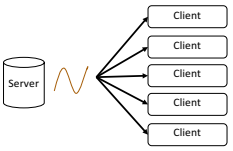
Java NIO

- Java Regular IO Server
  - Context switching between threads may be very high



Java NIO

- **Java Non-blocking IO**
  - One thread can handle multiple connections



Java NIO Buffers and Channels

- Data is read/written from a buffer
  - Types:
    - ByteBuffer, CharBuffer etc..
    - We will use ByteBuffer
- Buffer reads data from/writes data to a Channel
  - Types:
    - FileChannel, SocketChannel, and ServerSocketChannel
    - We will use ServerSocketChannel and SocketChannel

Java NIO important classes

- Selector
  - A multiplexer for SelectableChannel objects
- SelectionKey
  - A selectable channel's registration with a selector is represented by a SelectionKey.
  - 4 main interest sets:
    - Connect
      - Wants to establish connection to another server
    - Accept
      - Wants to accept waiting connection
    - Read
      - Wants to read data from channel
    - Write
      - Wants to write data to channel

Java NIO SelectionKeys

- Only one operation is valid for ServerSocket channels.
  - SelectionKey.OP_ACCEPT (type: int, value: 16)
- SocketChannels support connecting, reading, and writing.
  - SelectionKey.OP_CONNECT (type: int, value: 8)
  - SelectionKey.OP_READ (type: int, value: 1)
  - SelectionKey.OP_WRITE (type: int, value: 4)
- Registration example
  - `channel.register(selector, SelectionKey.OP_READ);`
- Question
  - How to assign more than one interestOps to channel

NIO Server

```
private void startServer() throws IOException {
  //housekeeping not shown
  // processing
  serverSocketChannel.socket().bind(new InetAddres(...));
  while (true) {
    // wait for events
    this.selector.select();
    // wake up to work on selected keys
    Iterator keys = this.selector.selectedKeys().iterator();
    while (keys.hasNext()) {
      //more housekeeping
      if (key.isAcceptable ()) {
         this.accept(key);
      }
      /*other cases such as isReadable() and isWriteable() not shown*/
  } }}
```

Key is Acceptable

```
private void accept(SelectionKey key) throws IOException {
  ServerSocketChannel servSocket = (ServerSocketChannel) key. channel();
  SocketChannel channel = servSocket.accept();

  System.out.println("Accepting incoming connection ");
  channel.configureBlocking(false);
  channel.register(selector, SelectionKey.OP_READ);
}
```

Key is Readable

```
private void read(SelectionKey key) throws IOException {
SocketChannel channel = (SocketChannel) key.channel();
ByteBuffer buffer = ByteBuffer.allocate(buffSize);
int read = 0;
try {
   while (buffer.hasRemaining() && read != -1) {
         read = channel.read(buffer);
      }
} catch (IOException e) {
/* Abnormal termination */
server.disconnect(key); return;
} //continued
```

Key is Readable cont…

```
if (read == -1) {
/* Connection was terminated by the client. */
server.disconnect(key);
return;
}
key.interestOps(SelectionKey.OP_WRITE);
}
```

Key is Writable

```
private void write(SelectionKey key) throws IOException {
  SocketChannel channel = (SocketChannel) key.channel();
    //You have your data stored in 'data', (type: byte[])
    ByteBuffer buffer = ByteBuffer.wrap(data);
    channel.write(buffer);
    key.interestOps(SelectionKey.OP_READ);
  }
}
```

NIO Client

```
private void startClient() throws IOException {
  //housekeeping not shown
  // processing
  channel.configureBlocking(false);
  channel.register(selector, SelectionKey.OP_CONNECT);
  channel.connect(new InetAddres(...));
  while(true){
    //other operations
    if(key.isConnectable()){
        this.connect(key);
    }
  }
}
```

Key is Connectable

```
private void connect(SelectionKey key) throws IOException {
 SocketChannel channel = (SocketChannel) key.channel();
   channel.finishConnect();
   key.interestOps(SelectionKey.OP_WRITE);
 }
```

Java NIO insights

- `Selectors` are thread-safe, but their `SelectionKey` set is not.
- `register(Selector selector, int interestOps, Object attachment)`

HW2-PC : Restrictions and Suggestions

HW2-PC : Suggestions

- Please implement and run your solution in environment where it is going to be graded
- Code above just explains how to code with `java.nio`
  - Implementation might be different for HW2-PC

HW2-PC : Restrictions

- You cannot use the Executor interface or any of the thread pool classes that are part of the `java.util.concurrent` package
- You cannot use third-party implementations of the thread pool. This is something you must implement all by yourself.
- Threads should be pre-allocated when the server component starts and live the duration of program execution. Do NOT create new Threads for each new message.
- The stop, suspend, and resume Thread methods should not be used. These methods are deprecated and can cause concurrency bugs.
- The data that you will be sending will be byte[]. None of your classes can implement the `java.io.Serializable` interface
- No GUIs should be built under any circumstances. These are auxiliary paths and the deduction is in place to ensure that none of you attempt to do this

QUESTIONS ?