Java Byte IPC: Part 4-Non-Blocking-NIO-Manager

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)

Multithreading Issues (Last Slide from Previous Part)

Thread that blocks on a select may not be the same as the one that registers or changes interestops

According to (my understanding of Rox tutorial) current implementations of NIO keys are not “thread safe,” meaning writes may not be atomic

So, to be safe, a single same thread should select(), register() and change interestop

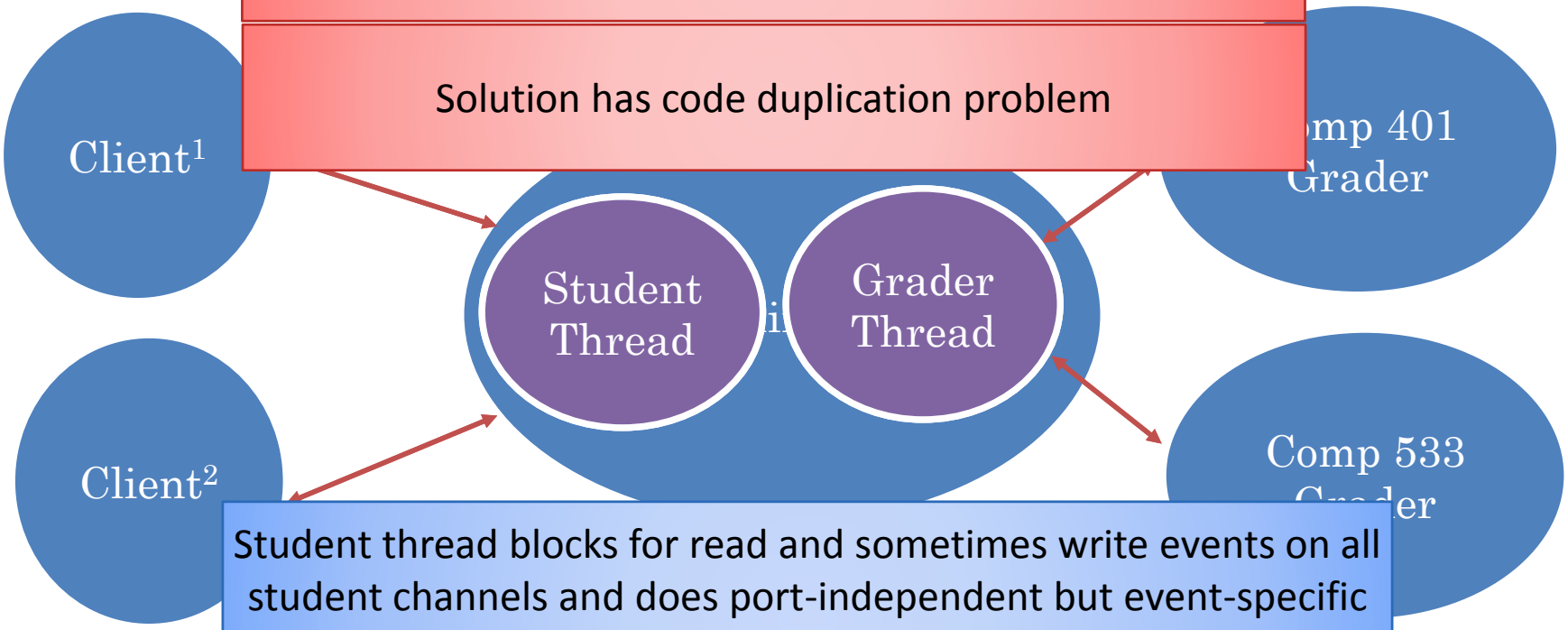
Another thread can call wakeup() for a blocked selector thread

Tutorial recommends a single selector thread in an NIO application

Solution Problems

Solution creates multiple threads

Solution has code duplication problem



Student thread blocks for read and sometimes write events on all student channels and does port-independent but event-specific operation on unblocked ports involving enabling and performing writes to graders after wakeup

Grader thread blocks for read and sometimes write events on all grader channels and does port-independent, but event-specific operation on unblocked ports involving enabling and performing writes to graders after wakeup

Rox Implicit Pattern

A single thread selects(), register() and changes interestop

Application-specific pre-select code (executed by other threads) decides the (register and interestop) actions the selecting thread takes before select

Cli

Thread

Thread

Client²

Comp 533 Grader

Selector thread blocks based on the (registrations and interestops) it executed before select and then does post-block processing based on ready operations

Application post-select code (related to pre-select code) decides the actions the selector takes after select

Issues

How to tell selector thread about the pre- and post-select application-specific code?

Which threads execute the post-select code (pre-select code executed by requesting application-defined thread)

Neither the Rox tutorial (nor NIO) has an explicitly articulated design pattern nor an abstraction

Mechanism, not policy!

Need layering!

Application-specific Processing?

How to tell selector thread about the pre- and post-select application-specific code?

Selector notifies registered observers before and after select?

Can have 0 observers

Notification method is supposed to be informed about actions, not perform actions

Observer's actions are not expected to depend on observable's actions

Future Command?

How to tell selector thread about the pre- and post-select application-specific code?

Application-code needs to tell selector code about some command that needs to be executed in future

Command is operation + args (e.g. put ('a), insert (2, 'a') as opposed to operation (e.g. put, insert)

Command Object

Commmand
(action with target
+ parameters)

Constructor
(targetObject, params)

execute ()

An object that represents a command =operation call , possibly with args

Provides a parameterless “execute” method to invoke the command()

Can have a constructor to take invocation parameters

May provide additional methods such as undo.

A command submitter wishes invocation of the method in the future by a command executor service, which is oblivious of what it does

Alternative to Listener-based Application-specific Processing?

Selector notifies registered observers before and after select?

Can have 0 observers

Notification method is supposed to be informed about actions, not perform actions

Observer's actions are not expected to depend on observable's actions

Application-specific Processing with Command Objects

Selector invokes initiate () methods of pre-select command objects enqueued by other threads and then invokes execute() methods of post-select command objects also queued by other threads

Post-select processing may depend on pre-select processing

Pre-select processing may: (a) register Connect interestop for channel^2 and call connect() on it, (b) set Write interestop for channel^1

Post-select processing may: (a) call finishConnect() on channel^2 , and (b) call write(buffer) on channel^1

Initializable Command Object

Commmand (action with target + parameters)	Constructor (targetObject, params)
	boolean execute ()
	boolean initiate()

Command object enqueued in a bounded buffer by producer thread and dequeued by consumer thread

Initiate() method invoked when dequeued()

Execute() method invoked when executable()

NIO Initializable Command Object

Commmand (action with target + parameters)	Constructor (targetObject, params)
	boolean execute ()
	boolean initiate()

Command object put in a queue bound to a specific channel and/or operation

Connect Queue, Accept Queue, Channel[1] Write Queue, Channel[1] Read Queue, Channel[2] Write Queue

Layering

NIO Initializable Command Objects (NIO/CO) Classes

NIO Manager

NIO Initializable Command Objects (NIO/CO) Interface

Initializable Command Objects (ICO)

NIO

Interrupt Processing

NIO Manager

Client threads use special singleton NIOManager to enqueue command objects and wakeup select thread

Unblocked select thread invokes initiate() methods of enqueued command objects and associates it with a (channel, operation) pair

After select(), thread determines enabled (channel, operation) pairs, find associated command objects, and invoke execute() methods of these command objects

Initiate method may: (a) register Connect interestop for channel[3] and call connect() on it, (b) set Write interestop for channel[1] , (c) set Read intrestop for channel[2].

Execute method may: (a) call finishConnect() on channel[3] and process new connection, (b) call write(buffer) on channel[1], (c) call read(buffer) and process read buffer.

NIO Manager Refinement

At most one single command per channel needed for accept and connect on a channel

At most one connect command needed a single time, expect same accept processing for a channel

Single NIO Manager buffer for reads on a channel, implies single read command

Reduces copying

Writes and reads should not be mixed, implies executing buffered write commands atomically, and a single initiate() command for buffered writes, provided by write buffer

Reduces scattering of writes, implementation bugs.

Default Command Objects

NIO Initializable Command Objects (NIO/CO) Classes

Default command classes and shareable functionality?

NIO Manager

NIO Initializable Command Objects (NIO/CO) Interface

Initializable Command Objects (ICO)

NIO

Interrupt Processing

Default and Shareable Functionality in Pluggable NIO Manager?

Client threads use it to enqueue command objects and wakeup select thread

Unblocked select thread invokes initiate() methods of enqueued command objects and associates it with a (channel, operation) pair

After select(), thread determines enabled (channel, operation) pairs, find associated command objects, and invoke execute() methods of these command objects

Initiate method may: (a) register Connect interestop for channel[3] and call connect() on it, (b) set Write interestop for channel[1] , (c) set Read intrestop for channel[2].

Execute method may: (a) call finishConnect() on channel[3] and process new connection, (b) call write(buffer) on channel[1], (c) call read(buffer) and process read buffer.

Listener-based Application-specific Processing?

Selector notifies registered observers before and after select?

Can have 0 observers

Notification method is supposed to be informed about actions, not perform actions

Observer's actions are not expected to depend on observable's actions

Command objects inform registered observers after execute

Shareable Command Classes

Command Listeners

Predefined Initializable Command-Object Classes

Other Initialized Command-Object Classes

How to switch among command classes?

NIO Manager

Initializable Command-Object Interfaces

NIO

Interrupt Processing

Factory Selector

Factory Singleton	static Factory getSIngleton()
	void setSIngleton(factory)

Factory	Command createCommand(aChannel, aNextInterestOps)

Example Connect Factories

```
public class AConnectCommandFactory implements ConnectCommandFactory {
public ConnectCommand createConnectCommand(
          SelectionManager aSelectionManager, SocketChannel aSocketChannel,
          InetAddress aServerHost, int aPort) {
  return new AConnectCommand (aSelectionManager, aSocketChannel, aServerHost, aPort,
         null);
}
public ConnectCommand createConnectCommand(
          SelectionManager aSelectionManager, SocketChannel aSocketChannel,
          InetAddress aServerHost, int aPort, Integer anInterestOps) {
  return new AConnectCommand(aSelectionManager, aSocketChannel, aServerHost, aPort,
          aNextInterestOps);
}
}
```

```
public class AReadingWritingConnectCommandFactory extends AConnectCommandFactory {
public ConnectCommand createConnectCommand(
          SelectionManager aSelectionManager, SocketChannel aSocketChannel,
          InetAddress aServerHost, int aPort) {
  return new AConnectCommand (aSelectionManager, aSocketChannel, aServerHost, aPort,
          SelectionKey.OP_READ);
}
}
```

Predefined Accept Command

Constructor
(aServerChannel, aNextOP)

```
nextOP ← aNextOP;
serverChannel← aServerChannel
```

boolean initiate()

```
serverChannel.register(selector,  OP_ACCEPT)
return true
```

booleam execute()

```
aChannel = serverChannel.accept()
aChannel.register(selector, nextOP)
notifyAcceptListeners(aChannel)
return true
```