JAVA BYTE IPC: PART 5-NON-BLOCKING-NIO MANAGER-COMMAND OBJECTS

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
NIO MANAGER: API TO DEFAULT COMMAND OBJECTS

```java
public interface NIOManager {
    void enableListenableAccepts(ServerSocketChannel aChannel,
                                  SocketChannelAcceptListener... aListener);
    void connect(SocketChannel aSocketChannel, InetAddress aServerHost,
                 int aPort, SocketChannelConnectListener... listeners);
    void write(SocketChannel aSocketChannel, ByteBuffer aByteBuffer,
               SocketChannelWriteListener... listeners);
    void addReadListener(SocketChannel aSocketChannel,
                          SocketChannelReadListener aListener);
    void addCloseListener(SocketChannel aSocketChannel,
                          SocketChannelCloseListener aListener);
}
```
**NIO Manager**

Client threads use special singleton NIOManager to enqueue command objects and wake up select thread.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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, finds associated command objects, and invokes execute() methods of these command objects.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 interestop 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

<table>
<thead>
<tr>
<th>At most one single command per channel needed for accept and connect on a channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>At most one connect command needed a single time, expect same accept processing for a channel</td>
</tr>
<tr>
<td>Single NIO Manager buffer for reads on a channel, implies single read command</td>
</tr>
<tr>
<td>Reduces copying</td>
</tr>
<tr>
<td>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</td>
</tr>
<tr>
<td>Reduces scattering of writes, implementation bugs.</td>
</tr>
</tbody>
</table>
**Predefined Accept Command**

**Constructor**
(aServerChannel, aNextOP)

serverChannel ← aServerChannel

nextOP ← aNextOP

boolean initiate()

serverChannel.register(selector, OP_ACCEPT)

boolean execute()

aChannel = serverChannel.accept()

notifyAcceptListeners(aChannel)

aChannel.register(selector, nextOP)

---

Observer pattern details - Listener registration and notification ignored
EXAMPLE PRE-CONNECT ACCEPT TRACE

```java
I*** {main} (SelectorFactorySet)
EvtSrc(AcceptCommandFactorySelector) inputport.nio.manager.factories.classes.AReadingAcceptCommandFactory@65ae6ba4

I*** {main} (SocketChannelBound) EvtSrc(AMeaningOfLifeNIOServer) 0.0.0.0/0.0.0.0:9000
sun.nio.ch.ServerSocketChannelImpl[/0:0:0:0:0:0:0:0:9000]

I*** {main} (ListenableAcceptsEnabled)
EvtSrc(ANIOManager) [examples.nio.manager.server.AMeaningOfLifeNIOServer@4f063c0a]<-
sun.nio.ch.ServerSocketChannelImpl[/0:0:0:0:0:0:0:0:9000]

I*** {Selecting Thread} (SelectCalled) EvtSrc(AScatterGatherSelectionManager) sun.nio.ch.WindowsSelectorImpl@2dab1979

I*** {main} (SelectorRequestNextInterestOp) EvtSrc(AnAcceptCommand) Ops:1
Channel:sun.nio.ch.ServerSocketChannelImpl[/0:0:0:0:0:0:0:0:9000]

I*** {main} (SelectorRequestEnqueued)
EvtSrc(ASelectionConnectionManager) Command:inputport.nio.manager.commands.classes.AnAcceptCommand@131276c2

I*** {main} (SelectorWokenUp) EvtSrc(ASelectionConnectionManager) sun.nio.ch.WindowsSelectorImpl@2dab1979

I*** {Selecting Thread} (SelectUnblocked) EvtSrc(AScatterGatherSelectionManager) sun.nio.ch.WindowsSelectorImpl@2dab1979

I*** {Selecting Thread} (SelectorRequestDequeued)
EvtSrc(ASelectionConnectionManager) Command:inputport.nio.manager.commands.classes.AnAcceptCommand@131276c2

I*** {Selecting Thread} (SocketChannelBlockingConfigured) EvtSrc(AnAcceptCommand) false
sun.nio.ch.ServerSocketChannelImpl[/0:0:0:0:0:0:0:0:9000]

I*** {Selecting Thread} (SocketChannelRegistered)
EvtSrc(AnAcceptCommand) Ops:16, Selector:sun.nio.ch.WindowsSelectorImpl@2dab1979,
sun.nio.ch.ServerSocketChannelImpl[/0:0:0:0:0:0:0:0:9000]

I*** {Selecting Thread} (SelectCalled) EvtSrc(AScatterGatherSelectionManager) sun.nio.ch.WindowsSelectorImpl@2dab1979
```
**Predefined Connect Command**

**Constructor**
(aChannel, aNextOPs)

nextOPs ← aNextOPs;
channel ← aChannel

**boolean initiate()**

key = channel.register(selector, OP_CONNECT)
return true

**boolean execute()**

channel.finishConnect()
notifyConnectListeners()
key.interestOps(aNextOPs)
return true
**Example Connect Trace**

```java
I***{main}(SelectorFactorySet)
EvtSrc(ConnectCommandFactorySelector: inputport.nio.manager.factories.classes.AConnectCommandFactory@135fbaa4
I***{main}(SocketChannelConnectRequested)
EvtSrc(AnNIOManager) (localhost/127.0.0.1:9000,[examples.nio.manager.client.AMeaningOfLifeNIOClient@5d099f62]) <-
java.nio.channels.SocketChannel[unconnected]
I***{Selecting Thread}(SelectCalled) EvtSrc(AScatterGatherSelectionManager) sun.nio.ch.WindowsSelectorImpl@50586a8e
I***{main}(SelectorRequestEnqueued)
EvtSrc(ASelectionConnectionManager) Command:inputport.nio.manager.commands.classes.AConnectCommand@37f8bb67
I***{main}(SelectorWokenUp) EvtSrc(ASelectionConnectionManager) sun.nio.ch.WindowsSelectorImpl@50586a8e
I***{Selecting Thread}(SelectUnblocked)
EvtSrc(AScatterGatherSelectionManager) sun.nio.ch.WindowsSelectorImpl@50586a8e
I***{Selecting Thread}(SelectorRequestDequeued)
EvtSrc(ASelectionConnectionManager) Command:inputport.nio.manager.commands.classes.AConnectCommand@37f8bb67
I***{Selecting Thread}(SocketChannelBlockingConfigured) EvtSrc(AConnectCommand) false
java.nio.channels.SocketChannel[unconnected]
I***{Selecting Thread}(SocketChannelConnectInitiated)
EvtSrc(AConnectCommand) java.nio.channels.SocketChannel[connection-pending remote=localhost/127.0.0.1:9000]
I***{Selecting Thread}(SocketChannelRegistered)
EvtSrc(AConnectCommand) Ops:8, Selector:.sun.nio.ch.WindowsSelectorImpl@50586a8e,
java.nio.channels.SocketChannel[connection-pending remote=localhost/127.0.0.1:9000]
I***{Selecting Thread}(SelectCalled) EvtSrc(AScatterGatherSelectionManager) sun.nio.ch.WindowsSelectorImpl@50586a8e
I***{Selecting Thread}(SelectUnblocked)
EvtSrc(AScatterGatherSelectionManager) sun.nio.ch.WindowsSelectorImpl@50586a8e
I***{Selecting Thread}(SocketChannelConnected)
EvtSrc(AConnectCommand) [examples.nio.manager.client.AMeaningOfLifeNIOClient@5d099f62]<-
java.nio.channels.SocketChannel[connected local=/127.0.0.1:49840 remote=localhost/127.0.0.1:9000]
Ready to send messages to server
I***{Selecting Thread}(SocketChannelInterestOp) EvtSrc(AConnectCommand) Ops:0 Selection
Key: sun.nio.ch.SelectionKeyImpl@5aa001fe
I***{Selecting Thread}(SelectCalled) EvtSrc(AScatterGatherSelectionManager) sun.nio.ch.WindowsSelectorImpl@50586a8e
```
EXAMPLE POST-CONNECT ACCEPT TRACE

I***{Selecting Thread}(SelectUnblocked)
EvtSrc(AScatterGatherSelectionManager)  sun.nio.ch.WindowsSelectorImpl@2dab1979
I***{Selecting Thread}(SocketChannelAccepted)
EvtSrc(AnAcceptCommand)  [examples.nio.manager.server.AMeaningOfLifeNIOServer@4f063c0a]<-java.nio.channels.SocketChannel[connected local=/127.0.0.1:9000
remote=/127.0.0.1:49840]:sun.nio.ch.ServerSocketChannelImpl[/0:0:0:0:0:0:0:9000]
I***{Selecting Thread}(ReadListenerAdded)
EvtSrc(AnNIOManager)  examples.nio.manager.server.AMeaningOfLifeServerReceiver@17004870<-java.nio.channels.SocketChannel[connected local=/127.0.0.1:9000 remote=/127.0.0.1:49840]
I***{Selecting Thread}(SocketChannelRegistered)
EvtSrc(AnAcceptCommand)  Ops:1,  Selector:sun.nio.ch.WindowsSelectorImpl@2dab1979,
java.nio.channels.SocketChannel[connected local=/127.0.0.1:9000 remote=/127.0.0.1:49840]
I***{Selecting Thread}(SelectCalled)
EvtSrc(AScatterGatherSelectionManager)  sun.nio.ch.WindowsSelectorImpl@2dab1979]
Write Command?

Constructor (aChannel, aBuffer)

buffer ← aNextBuffer;
channel ← aChannel

boolean initiate()

key = channel.register(selector, OP_WRITE)

boolean execute()

Int aSize = channel.write(buffer)

If (aSize == 0) return false
else return true;
INCREASE WRITE GATHER

How to make sure writes and other operations do not mix (to increase gather) and are retried (to overcome scatter)?

Keep a bounded buffer of pending writes on a channel and keep executing them, without recalling select, while there is buffer space.
WRITE COMMAND (NO INITIATE)

Constructor
(aChannel, aBuffer)

buffer ← aNextBuffer;
channel ← aChannel

boolean execute()

Int aSize = channel.write(buffer)

If (aSize == 0) return false
else return true;
**Write Bounded Buffer and Write Command**

- **Initiate called before each select if write buffer is non empty**
- **Buffer execute called after each select if channel writeable and write buffer is non-empty**
- **Children execute() called and children dequeued until buffer empty or all bytes of child not written. Child remains in queue if partially written (execute returns false).**

**WriteBoundedBuffer Command Object**

- **initiate()**
- **put(WriteCommand)**
- **WriteCommand remove()**
- **execute ()**
- **invoke**

**HAS-A**

- **Write Command Object**
- **boolean execute ()**
- **Write Command Object**
- **boolean execute ()**
WRITECOMMAND BOUNDED BUFFER

boolean initiate()

savedInterestOps = channel.interestOPs();

key = channel.register(selector, OP_WRITE)

boolean execute()

for (WriteCommand aChild: children) {
    if (!aChild.execute()) return false;
    else remove(aChild);
}

if (isEmpty()) {
    channel.register(selector, savedInterestOps);
    notifyBufferEmptyListeners();
}

if (savedInterestOps | OP_WRITE?)
**Example Write Trace (Input 42)**

I***{main}(NotifiedPropertyChangeEvent)
EvtSrc(AMeaningOfLifeModel) java.beans.PropertyChangeEvent[propertyName=Meaning; oldValue=; newValue=42; propagationId=null; source=examples.nio.manager.mvc.AMeaningOfLifeModel@17f052a3;]
 EXAMPLE_WRITE_TRACE
examples.nio.manager.client.AMeaningOfLifeClientSender@2dda6444,
examples.nio.manager.mvc.AMeaningOfLifeView@49c2faae]
I***{main}(NotifyCalled)
EvtSrc(AMeaningOfLifeModel) java.beans.PropertyChangeEvent[propertyName=Meaning; oldValue=; newValue=42; propagationId=null; source=examples.nio.manager.mvc.AMeaningOfLifeModel@17f052a3;]

I***{main}(RequestToWrite)
EvtSrc(AMeaningOfLifeView@49c2faae)
java.nio.channels.SocketChannelWriteRequested
EvtSrc(AnNIOManager) java.nio.HeapByteBuffer[pos=0 lim=6 cap=6], []
java.nio.channels.SocketChannel[connected local=/127.0.0.1:49840 remote=localhost/127.0.0.1:9000]
I***{main}(WriteRequestEnqueued)
EvtSrc(AWriteBoundedBuffer) Command:inputport.nio.manager.commands.classes.AHeaderWriteCommand@5010be6
I***{main}(WriteRequestEnqueued)
EvtSrc(AWriteBoundedBuffer) Command:inputport.nio.manager.commands.classes.AWriteCommand@2e0fa5d3
I***{main}(SelectorWokenUp)
EvtSrc(AScatterGatherSelectionWriteManager) sun.nio.ch.WindowsSelectorImpl@5058a8e
Meaning of life:42
Meaning of life? (or enter "experiment" for programmed input)
I***{Selecting Thread}(SelectUnblocked)
EvtSrc(AScatterGatherSelectionManager) sun.nio.ch.WindowsSelectorImpl@5058a8e
I***{Selecting Thread}(SocketChannelInterestOp) EvtSrc(AWriteBoundedBuffer) Ops:4 Selection
Key:sun.nio.ch.SelectionKeyImpl@5aa001fe
I***{Selecting Thread}(SelectCalled) EvtSrc(AScatterGatherSelectionManager) sun.nio.ch.WindowsSelectorImpl@5058a8e
I***{Selecting Thread}(SelectUnblocked)
EvtSrc(AScatterGatherSelectionManager) sun.nio.ch.WindowsSelectorImpl@5058a8e
I***{Selecting Thread}(SocketChannelWritten) EvtSrc(AHeaderWriteCommand) java.nio.HeapByteBuffer[pos=0 lim=4 cap=4]
java.nio.channels.SocketChannel[connected local=/127.0.0.1:49840 remote=localhost/127.0.0.1:9000]
I***{Selecting Thread}(WriteRequestDequeued)
EvtSrc(AWriteBoundedBuffer) Command:inputport.nio.manager.commands.classes.AHeaderWriteCommand@5010be6
I***{Selecting Thread}(SocketChannelWritten) EvtSrc(AWriteCommand) java.nio.HeapByteBuffer[pos=0 lim=6 cap=6]
java.nio.channels.SocketChannel[connected local=/127.0.0.1:49840 remote=localhost/127.0.0.1:9000]
I***{Selecting Thread}(WriteRequestDequeued)
EvtSrc(AWriteBoundedBuffer) Command:inputport.nio.manager.commands.classes.AWriteCommand@2e0fa5d3
I***{Selecting Thread}(SocketChannelInterestOp) EvtSrc(AWriteBoundedBuffer) Ops:0 Selection
Key:sun.nio.ch.SelectionKeyImpl@5aa001fe
I***{Selecting Thread}(WriteBufferIsEmpty) EvtSrc(AWriteBoundedBuffer) []<--java.nio.channels.SocketChannel[connected
local=/127.0.0.1:49840 remote=localhost/127.0.0.1:9000]
I***{Selecting Thread}(SelectCalled) EvtSrc(AScatterGatherSelectionManager) sun.nio.ch.WindowsSelectorImpl@5058a8e
Is NIO write buffer a copy of application sent buffer?

- No need for app to do buffer management
- Copy inefficient
- Inconsistent with NIO philosophy
- Object layer will do buffer management
- Application buffer not copied
- Header information?

Pros and cons means we should have our own command objects sometimes.
Matching Input and Output Buffers (Missing Voice)

- **NIO Driver Upper Half**
- **Header Buffer**
- **App Buffer**
- **NIO Driver Lower Half**

- **Producer**: Send header byte buffer before application byte buffer
- **Inefficient**: Allocate header buffer on each send?
- **Inefficient**: Have fixed header pool, which can overflow

- **Get()**: Bounded Buffer
- **Put()**: Producer
- **Consumer**
**Input Buffer Copying?**

- Is listener buffer a copy of NIO read buffer?
- Listener can manipulate its contents
- Copy inefficient
- Inconsistent with NIO philosophy
- A single input buffer with complex scatter/gather
How many Input Buffers

Create new input buffer for each sent output buffer - header/data pair?

- Inefficient
- Makes scatter/gather easier

A single input buffer with complex scatter/gather

Read listener delivered view containing next sent output buffer
Multiple Views of Byte Array

Multiple ByteBuffer can share a common byte array

wrap(bytes[])
**Predefined Singleton Read Command**

Constructor (aChannel)

channel ← aChannel

boolean execute()

channel.read(buffer)
If (extractSentData())
    notifyReadListeners(message)
return true
EXAMPLE READ TRACE

I***{Selecting Thread}(SelectUnblocked)  
EvtSrc(AScatterGatherSelectionManager)  sun.nio.ch.WindowsSelectorImpl@2dab1979
I***{Selecting Thread}(SocketChannelRead)  EvtSrc(AScatterGatherReadCommand)  (10)java.nio.ByteBuffer[pos=10  
lim=4194304 cap=4194304]  java.nio.channels.SocketChannel[connected local=/127.0.0.1:9000 remote=/127.0.0.1:49840]
I***{Selecting Thread}(SocketChannelHeaderRead)  EvtSrc(AScatterGatherReadCommand)  (6)java.nio.ByteBuffer[pos=4  
lim=10 cap=4194304]  java.nio.channels.SocketChannel[connected local=/127.0.0.1:9000 remote=/127.0.0.1:49840]
I***{Selecting Thread}(SocketChannelFullMessageRead)  
EvtSrc(AScatterGatherReadCommand)  [examples.nio.manager.server.AMeaningOfLifeServerReceiver@17004870]<-  
(6)java.nio.ByteBuffer[pos=4 lim=10 cap=4194304]  java.nio.channels.SocketChannel[connected local=/127.0.0.1:9000  
remote=/127.0.0.1:49840]
Meaning of Life from Bob:42
I***{Selecting Thread}(SelectCalled)  EvtSrc(AScatterGatherSelectionManager)  sun.nio.ch.WindowsSelectorImpl@2dab1979
**public interface NIOManager {**

void enableListenableAccepts(ServerSocketChannel aChannel, SocketChannelAcceptListener... aListener);

void connect(SocketChannel aSocketChannel, InetAddress aServerHost, int aPort, SocketChannelConnectListener... listeners);

void write(SocketChannel aSocketChannel, ByteBuffer aByteBuffer, SocketChannelWriteListener... listeners);

void addReadListener(SocketChannel aSocketChannel, SocketChannelReadListener aListener);

void addCloseListener(SocketChannel aSocketChannel, SocketChannelCloseListener aListener);**
NIO Manager vs General Device Driver Architecture

Analogy with NIO Manager?

Driver Upper Half

Request\(^1\)  \rightarrow  \text{consumer}

Driver Lower Half

Request\(^2\)

Lower half executes in special driver thread

Processes request

Upper half executes in application thread

Gives request to lower half
**Analogous NIO Manager Architecture**

**NIO Manager Upper Half**
- **Command** \(^1\)
- **Command** \(^2\)

**NIO Manager Lower Half**

**Synchronization?**
- Upper half executes in application thread
  - Gives command to lower half
  - Identifies channel and buffer and post operation interestops
    - Blocks on full queue and wakes up selector

**Lower half executes in selection thread**
- Processes request
- Registers, selects, issues write, calls listener
- Unblocks upper half and blocks on select() if queue empty
Synchronous NIO Receive Upper and Lower Halves

Listener method executed by selecting thread
Should listener method be allowed to block?
**Select Thread Blocking Rule**

- The only blocking call selection thread should make is `select()` and it should execute efficiently.
- Unblocking done by some application thread(s).
- Those threads can delay or starve communication of other threads the selection thread serves.
- Can starve a single thread if selector waits for non-empty buffer.
- Application listeners should not block or take too long as they delay or starve same or other threads.
- Listener can be executed in a separate thread.
SELECT THREAD PERFORMANCE RULE

Listeners should execute quickly so other requests do not suffer for the same reason interrupts should execute quickly.

NIO Manager
Upper Half

Request¹

NIO Driver
Lower Half

Request²
ACCOMMODATING BLOCKING OR SLOW LISTENERS?

- Use thread pool or new thread for blocking or slow calls
- NIOManager does not use new thread to avoid thread overhead when not needed