JAVA BYTE IPC: PART 3-NON-BLOCKING-NIO

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
NIO Discussion from First Principles (Repeat of Last Slide of Previous Part)

- I/O?
  - ✔
- I/O → Inter Process Communication (IPC)?
  - ✔
- Blocking → Non-Blocking?
**Why Not Block?**

- **connect (host, port)**
  - Block until server accepts connection to server socket
  - Can do other things while waiting for operation

- **Channel accept()**
  - Block until next client tries to contact the server socket
  - Example needing concurrency?

- **write(byte[])**
  - Block until at least one byte in system buffer

- **read(byte[])**
  - Block until at least one byte received

- **Channel accept()**
  - Block until next client tries to contact the server socket
Grading Example with Blocking

Separate reading thread for each student and grader channel

Threads are expensive (compared to method calls) for scalable servers
Grading Server: Single Reading Thread for Multiple Channels

Which client port should a client thread do blocking read from?

Which grader should a grader thread do blocking read from?
Non-Blocking I/O?

Assume polling is ok

Selectable Channel

IS-A

ServerSocket socket()

SocketChannel accept()

bool connect()

HAS-A

ServerSocket

Socket accept()

SocketChannel accept()

bool connect()

ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();

Immediate return, and return values indicate success

HAS-A

Socket

Socket socket()

int read(ByteBuffer)

int write(ByteBuffer)

InputStream getInputStream()

OutputStream getOutputStream()

InputStream getInputStream()

OutputStream getOutputStream()

SocketChannel socketChannel = SocketChannel.open();
**NON-BLOCKING SEMANTICS**

- **connect()**
  - Return true if connection established already

- **accept()**
  - Return channel if connect message pending

- **write()**
  - Return number of bytes written (0.. Length)

- **read()**
  - Read number of bytes received
Grading Server: Single Reading Thread for Multiple Channels

Client\(^1\)  

Student Thread  
Grader Thread

Client\(^2\)

Comp 401 Grader  
Comp 533 Grader

How to create a single read thread for students?  
How to create a single read thread for graders?
**Read Nested Multi-Channel Loops**

1. `readFromStudents()`
   - `s ← nextStudent()`
   - `s.channel.read(s.buffer)`
   - **Full Read?**
     - Service student read()
     - Enqueue write request for grader

2. `readFromGraders()`
   - `g ← nextGrader()`
   - `g.channel.read(g.buffer)`
   - **Full Read?**
     - Service grader read()
     - Enqueue write request for student
**Write Multi-Channel Loop**

```
writeRequests ()

w ← nextRequest()

w.channel.write (w.buffer)

Full Write?

dqueue(w)
```
Accept Loop

\[
\text{acceptRequests}
\]

\[
a \leftarrow \text{nextRequest}()
\]

\[
c = a\text{.serverChannel.accept}()
\]

\[
c \neq \text{null}?
\]

\[
\text{process}(a)
\]

Add c to appropriate read list
CONNECT LOOP

Side effect of connect?
Connect message sent multiple times!

connect() is not idempotent
Multiple invocations of an idempotent operation = single invocation (with same args)

c -> nextRequest()

connected= c.channel.connect()

dqueue(r)

process(r)
CONNECT LOOP WITH FINISH CONNECT

- connect (c)
  - c.connect()
  - enqueueFinishConnectRequest(c)
- dequeueConnectRequests ()
  - c ← nextRequest()
  - connected = c.channel.finishConnect()
  - connected?
- finishConnect() is idempotent
- dequeue(r)
- process(r)

Assume buffer for connect available

Main thread enqueues connect requests
**ServerSocket and Socket Channels**

- **ServerSocketChannel**
  - Selectable Channel
  - ServerSocket socket()
  - SocketChannel accept()
  - boolean (finish) connect()

- **SocketChannel**
  - Socket socket()
  - int read(ByteBuffer)
  - int write(ByteBuffer)

- **ServerSocket**
  - Socket accept()

- **Socket**
  - InputStream getInputStream()
  - OutputStream getOutputStream()

---

```java
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
SocketChannel socketChannel = SocketChannel.open();
```
**Thread Structure**

- One read thread for each port (service kind)
- A separate write, accept, and connect thread
- Main thread creates requests processed by connect and accept threads
- Connect and accept thread create channels processed by read threads
- Read threads create requests for write thread
PROBLEMS

Need multiple event waiting (3 + NumPorts) threads

Better than having single thread per client

Code duplication in reading threads

Threads need to poll!

A blocking I/O operation waits for a particular event on a particular channel but that leads to problem of thread proliferation

A single thread should be able to wait for read, write, finishConnect, and accept operation to be executable on multiple channels, so not waiting for a single event
Grading Server: Single Reading Thread for Multiple Channels (Review)

Client¹

Student Thread

Grader Thread

Comp 401 Grader

Comp 533 Grader

Client²

How to create a single read thread for students?

How to create a single read thread for graders?
**App-Specific Read Nested Multi-Channel Loops (Review)**

1. `readFromStudents()`
   - `s ← nextStudent()`
   - `s.channel.read (s.buffer)`
   - Full Read?
     - Service student read()
     - Enqueue write request for grader

2. `readFromGraders()`
   - `g ← nextGrader()`
   - `s.channel.read (s.buffer)`
   - Full Read?
     - Service grader read()
     - Enqueue write request for student

Manual context switch!
**App-Independent Write Multi-Channel Loop (Review)**

- `writeRequests()`
- `w ← nextRequest()`
- `w.channel.write(w.buffer)`
- **Full Write?**
- `dequeue(w)`
Accept Loop (Review)

```java
acceptRequests

a ← nextRequest()

c = a.serverChannel.accept()

if c != null?
    acceptRequests
    a ← nextRequest()
    c = a.serverChannel.accept()

    c != null?
    process(a)

    Add c to appropriate read list
```
Connect Loop with Finish Connect (Review)

connect (c)

\[ c.connect() \]

enqueueFinishConnectRequest(c)

Assume buffer for connect available

Main thread enqueues connect requests

connect (c)

\[ c.connect() \]

enqueueFinishConnectRequest(c)

\[ c.connect() \]

\[ c.channel.finishConnect() \]

connected?

\[ c \leftarrow \text{nextRequest()} \]

\[ \text{dequeueFinishConnectRequest}(c) \]

finishConnect() is idempotent

\[ \text{dequeue}(r) \]

\[ \text{process}(r) \]
**Thread Structure**

- One read thread for each port (service kind)
- A separate write, accept, and connect thread
- Main thread creates requests processed by connect and accept threads
- Connect and accept thread create channels processed by read threads
- Read threads create requests for write thread
**Problems**

1. Need multiple event waiting (3 + NumPorts) threads
2. Threads perform context switches
3. Code duplication in application-specific reading threads
4. Threads need to poll, need blocking!
5. A blocking I/O operation waits for a particular event on a particular channel but that leads to problem of thread proliferation
6. A single thread should be able to wait for read, write, finishConnect, and accept operation to be executable on multiple channels, so not waiting for a single event
**Review: The Problem of Blocking I/O**

Which client should a client thread do blocking read from?

Which grader should a grader thread do blocking read from?
Grading Server: Single Blocking Reading Thread for Multiple Channels

Student thread blocks for read and write event on all student channels and does port-independent but event-specific operation on unblocked ports.

Grader thread blocks for read and write event on all grader channels and does port-independent, but event-specific operation on unblocked ports.
**MULTI-EVENT BLOCK (SELECTION)**

Block on all pending operations on all channels

A single thread should be able to wait for read, write, finishConnect(), and accept operation to be executable on multiple channels

How to make grader and student threads wait for different channels

The wait should be able to select which channels it blocks on.

How does grader or student thread know which channel and operation to process

On unblocking, wait should know which channel operations are ready
**Selector (Initial Cut)**

- **Resource on which async operation may be executed (replaces Socket, ServerSocket, File)**

- **Selectable Channel**
  - SelectionKey register(Selector s )
  - configureBlocking(boolean)

- **Single selector for multiple channels**
  - Selector selector = Selector.open();

- **Selector**
  - int select()
  - Set<SelectionKey> selectedKeys()

- **Registers interest in (resource) channel referenced by a key id**
  - SelectionKey key = selectableChannel.register( selector );

- **Blocking call waiting until at least one channel to be ready for a new event, and returning # of enabled operations**
  - Keys of actual enabled channels

- **What to do with this set?**
Selection Key

SelectableChannel channel()

boolean isAcceptable()

boolean isConnectable()

boolean isReadable()

boolean isWriteable()

Key to selectable channel, which can be used to execute enabled operation immediately
**NEW PROBLEMS**

- Write operation will usually be ready
  
- select() will unblock whenever write operation is ready
  
- Should wait for write readiness only when we have data to write

**Student thread blocks for read and write events on all student channels and does port-independent but event-specific operation on unblocked ports**

**Grader thread blocks for read and write event on all grader channels and does port-independent, but event-specific operation on unblocked ports**

Client\(^1\)

Client\(^2\)

Comp 401 Grader

Comp 533 Grader
**INITIAL AND DYNAMIC INTERESTOPS**

- **Selectable Channel**
  - `SelectionKey register(selector, iops)`
  - `configureBlocking(boolean)`

- **Selection Key**
  - `SelectableChannel channel()`
  - `int interestOps()`
  - `interestOps(int)`

- **Selector**
  - `int select()`
  - `Set<SelectionKey> selectedKeys()`
  - `Selector wakeup()`

---

- Registers interest in (resource, operation) pair referenced by a key id
  - `SelectionKey key = selectableChannel.register(selector, SelectionKey.OP_READ);`

- Changes interest operations dynamically

- How to make blocked `select()` use new interest ops or registration?

- Unblock `select`, usually after a new registration
**Dynamic Write Interest**

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.
Multithreading Issues

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