Java Byte IPC: Part 1 – Deriving NIO from I/O

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
The Rox Java NIO Tutorial

Contents

1. Introduction
2. Credits
3. General principles
4. The server
5. The client
6. NIO and SSL on 1.4
7. The code
8. About the author

Introduction

This tutorial is intended to collect together my own experiences using the Java NIO libraries and the dozens of hints, tips, suggestions and caveats that litter the Internet. When I wrote Rox all of the useful information existed as just that: hints, tips, suggestions and caveats on a handful of forums. This tutorial actually only covers using NIO for asynchronous networking (non-blocking sockets), and not the NIO libraries in all their glory. When I use the term NIO in this tutorial I'm taking liberties and only talking about the non-blocking IO part of the API.

If you've spent any time looking at the JavaDoc documentation for the NIO libraries then you know they're not the most transparent docs floating around. And if you've spent any time trying to write code based on the NIO libraries and you use more
**Rox Echo Application**

Diagram showing a client-server architecture with three clients communicating with a server.
ASSIGNMENT REQUIREMENTS
Rox Tutorial: Pros and Cons

The Rox Java NIO Tutorial

Contents

1. Introduction
2. Credits
3. General principles
4. The server
5. The client
6. NIO and SSL on 1.4
7. The code
8. About the author

Introduction

This tutorial is intended to collect together my own experiences using the Java NIO. I've found a number of suggestions and caveats that litter the Internet. When I wrote Rox all of the useful tips, suggestions and caveats on a handful of forums. This tutorial actually only covers using NIO for asynchronous networking (non-blocking sockets), and not the NIO libraries in all their glory. When I use the term NIO, I mean the Java NIO APIs, and only talking about the non-blocking IO part of the API.

If you've spent any time looking at the JavaDoc documentation for the NIO libraries, there are a number of transparent docs floating around. And if you've spent any time trying to write code that actually does anything useful, you've found that it's a little difficult to find explanations of how things work. I hope that this tutorial will be more helpful.

Explains NIO and its limitations

Has an implicit application architecture (design pattern)

Some students have trouble understanding what is to be done

Design pattern was not explicit

Abstraction vs. Implicit Design Pattern in Web Material
Points vs. Design Space

- Pipes
- RMI
- Serialized Streams
- Sockets
- Xinu
- NIO
- Sync

Focus on point
Later design space
NIO Discussion from First Principles

- I/O?
- I/O → Inter Process Communication (IPC)?
- Blocking → Non-Blocking?
I/O (API)

Console
- put(char )
- char get()

Implementation and Blocking Semantics?

Device driver?

Application and device events concurrently and at different rates
TTY Output Driver: Producer, Consumer

- Upper half executes in application thread
  - Enqueues printed characters
- Lower half executes in interrupting thread
  - Consumes next char and writes it to device register
  - Device interrupts after output register written/cleared, and after interrupt posting enabled

Diagram:
- put() function
- Producer
- Output Buffer
- Consumer
- char^1
- char^2
- Input?
TTY Input Driver: Producer, Consumer

Producers:
- `put()`
- `tty input interrupt`

Consumers:
- `get()`

Input Buffer:
- `char¹`
- `char²`

Upper half executes in application thread
- Consumes characters, blocks if buffer empty

Lower half executes in interrupting thread
- Reads device register and puts in read buffer
- Device interrupts each time a new character written to register
TTY Output Driver: Blocking

- `put()`: producer
- `put()`: consumer
- `char^1`: `char^2`
- `get()`: Output Buffer
- `tty output interrupt`
- Print blocks if output buffer full
- Output buffer in kernel space
- Lower-half blocking?
**Lower-Half Blocking**

- **Driver Upper Half**
  - put()
  - Request\(^1\)
- **Driver Lower Half**
  - get()
  - Request\(^2\)
  - Bounded Buffer
- **producer**
- **consumer**

**Interrupt routines cannot block**

**Device registers might get full**

If tty interrupt routine blocks, input device register can be overwritten as user enters new character.

**Often interrupt-service disabled while they are executing to ensure atomicity**

While bounded buffer being manipulated, clock interrupt servicing and context switching should be disabled.

**Blocked routine can starve or delay threads**
TTY Output Driver Cannot Block

After outputting the last queued character, the interrupt routine will find output buffer empty.

What triggers its call when a new character is entered, as it cannot wait for non empty buffer?

On finding empty output buffer, interrupt routine disables interrupt posting.

Upper half enables interrupt posting when it writes (to an empty buffer).

Enabling disabled interrupt posting causes an output “interrupt” to be posted and interrupts routine to be called.

After outputting the last queued character, the interrupt routine will find output buffer empty.

What triggers its call when a new character is entered, as it cannot wait for non empty buffer?

On finding empty output buffer, interrupt routine disables interrupt posting.

Upper half enables interrupt posting when it writes (to an empty buffer).

Enabling disabled interrupt posting causes an output “interrupt” to be posted and interrupts routine to be called.

After outputting the last queued character, the interrupt routine will find output buffer empty.

What triggers its call when a new character is entered, as it cannot wait for non empty buffer?

On finding empty output buffer, interrupt routine disables interrupt posting.

Upper half enables interrupt posting when it writes (to an empty buffer).

Enabling disabled interrupt posting causes an output “interrupt” to be posted and interrupts routine to be called.
TTY Input driver Cannot Block

- get()
  - consumer
  - get()
  - char\(^1\)
  - char\(^2\)
  - Input Buffer
  - put()
  - producer
  - tty input interrupt

Tty input routine cannot block if input buffer full
Echoes a bell character to user to not type any more
HALF SYNCHRONIZED BOUNDED BUFFER

Output: Producer synchronized: producer waits but not consumer

Input: Consumer synchronized: consumer waits but not produced

Java ArrayBlockingQueue
(ElementType)

Not synchronizing

Synchronizing

add(<ElementType> e)

<ElementType>get()

int size()

put(<ElementType> e)

<ElementType>take()
I/O: API

Put produces writes characters into a bounded buffer consumed by the console device.

Get consumes characters from a bounded buffer produced by the console device.

Console has two bounded buffers (combining Display and Keyboard devices).
NIO Discussion from First Principles

I/O?

I/O → Inter Process Communication (IPC)?

Blocking → Non-Blocking?