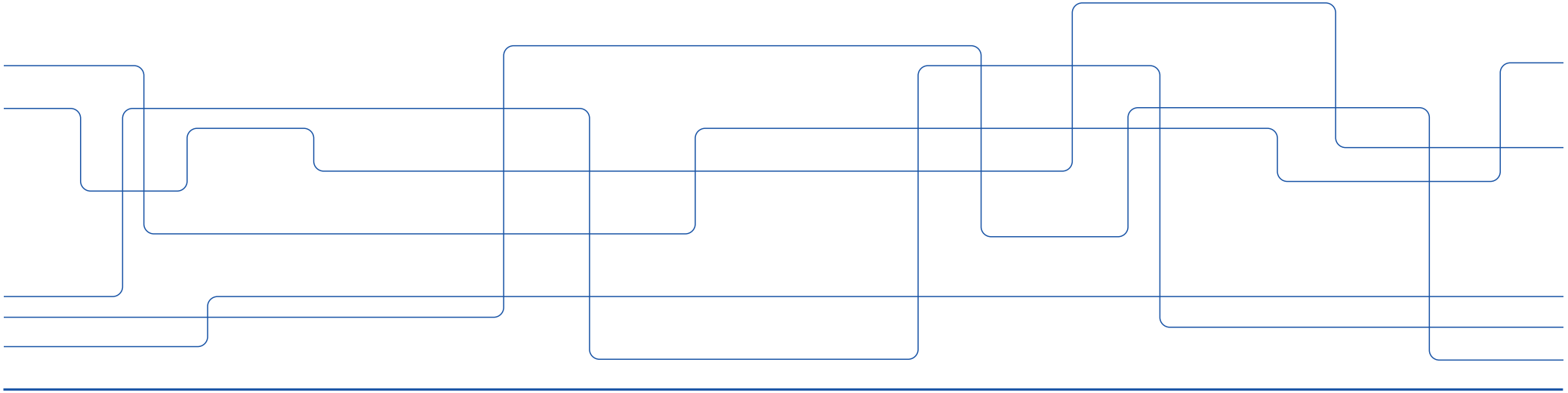
KTH ROYAL INSTITUTE
OF TECHNOLOGY

Socket Programming and Data Encoding

Peter Sjödin

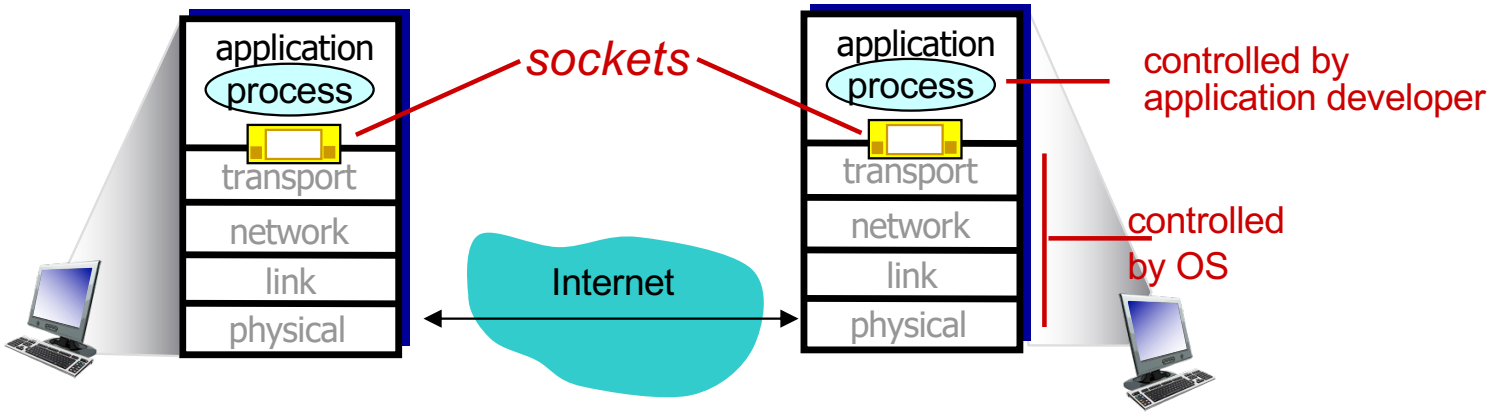


Synopsis

- The Sockets Applications Programming Interface (API)
- Socket programming in Java
- Java I/O
- Data encoding and decoding

Sockets

- process sends/receives messages to/from its socket
  - How application protocols access transport protocol services



Socket Programming

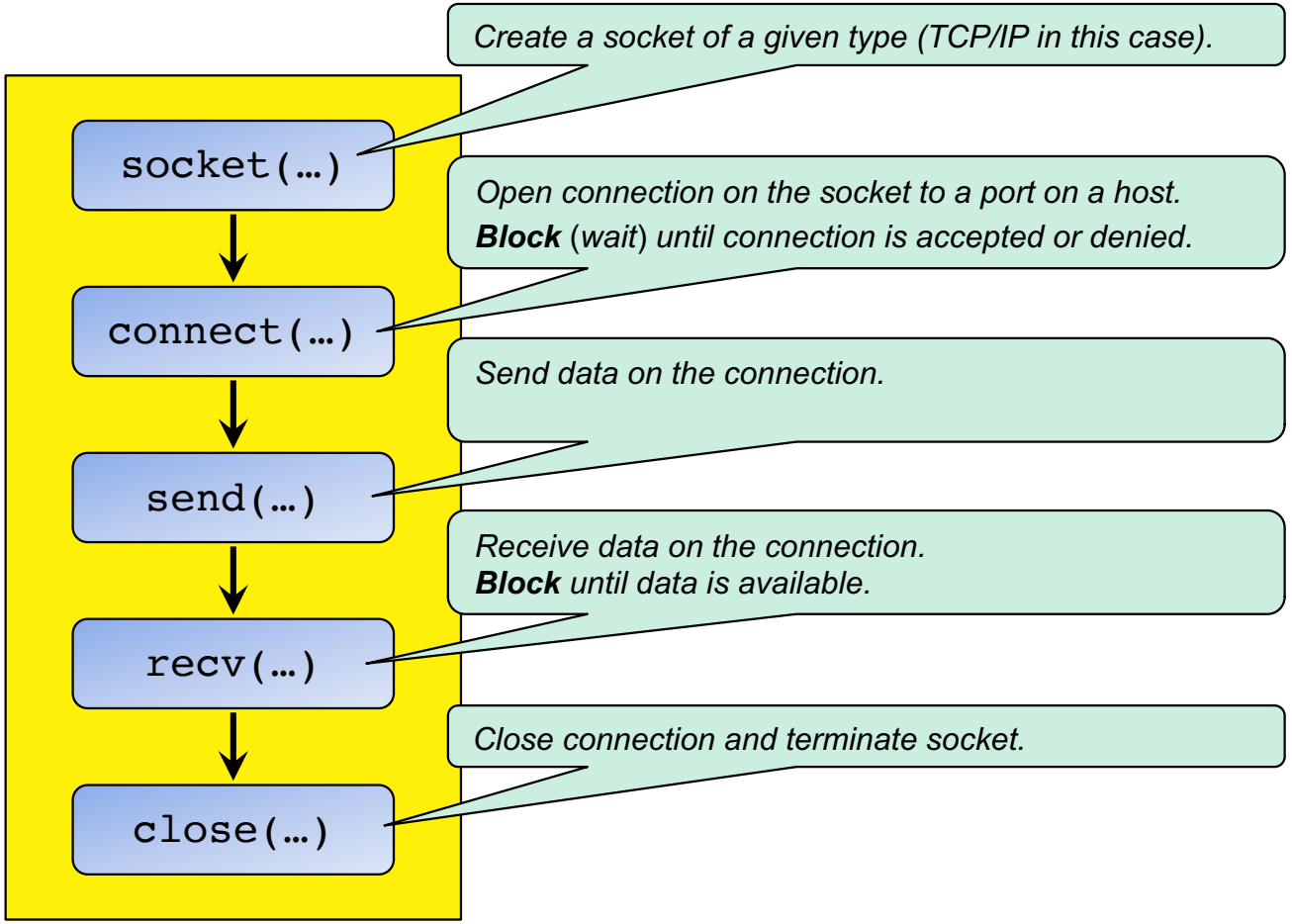
- Two socket types for two transport services:
- TCP: reliable, byte stream-oriented
  - > *TCP responsible for dividing the stream into packets*
- UDP: unreliable, byte block-oriented (datagram)
  - > *Application responsible for dividing data into packets*

Socket Programming with TCP

- First, client must open a TCP connection to the server
- TCP connection
  - Ordered delivery of data as a *stream of bytes*
    - > Bytes delivered in the same order they are sent
  - Reliable
    - > If the data does not make it to the other side, the application will learn
  - Bi-directional
    - > Both sides (client and server) can send and receive
- When communication is completed, the connection is closed
  - Both client and server may take initiative to close connection (FIN flag)
  - The two directions are closed independently
    - > *A connection can be half open*

TCP Client

Client creates a socket and performs a number of **socket system calls** on it – functions in the operating system



Java Streams

- In Java, streams (InputStream and OutputStream) are the basic classes for byte I/O

Write/send all bytes in buffer.

```
OutputStream()

void write(byte [] buffer)
void write(byte [] buffer, int offset, int length)
void write(int)
```

Write/send length bytes from buffer, starting at offset.

Write/send a single byte (0 – 255). Larger integers will be truncated.

Java Streams

- Reads data into an existing (pre-allocated) array
- Returns the number of bytes being read
- Returns -1 at end of data
  - End of file, connection closed, etc.

Read/receive bytes into buffer

```
InputStream()

int read(byte [] buffer)
int read(byte [] buffer, int offset, int length)
```

Read/receive at most length bytes into buffer, starting at offset.

Socket Programming Example Application

1. Client reads data from system input and sends data to server
2. Server receives the data and computes a response
3. Server sends response to client
4. Client receives response and and writes it to system output

Example TCP Client in Java

```
import java.net.*;
class TCPClient {
    private static int BUFFERSIZE=1024;

    public static void main(String argv[]) throws Exception
    {
        // Pre-allocate byte buffers for reading/receiving
        byte[] fromUserBuffer = new byte[BUFFERSIZE];
        byte[] fromServerBuffer = new byte[BUFFERSIZE];

        Socket clientSocket = new Socket("hostname", 6789);

        int fromUserLength = System.in.read(fromUserBuffer); // User input
        clientSocket.getOutputStream().write(fromUserBuffer, 0, fromUserLength);

        int fromServerLength = clientSocket.getInputStream().read(fromServerBuffer);
        System.out.print("FROM SERVER: "); // Use print method since it is a string
        System.out.write(fromServerBuffer, 0, fromServerLength);
        clientSocket.close();
    }
}
```

Defines Socket class

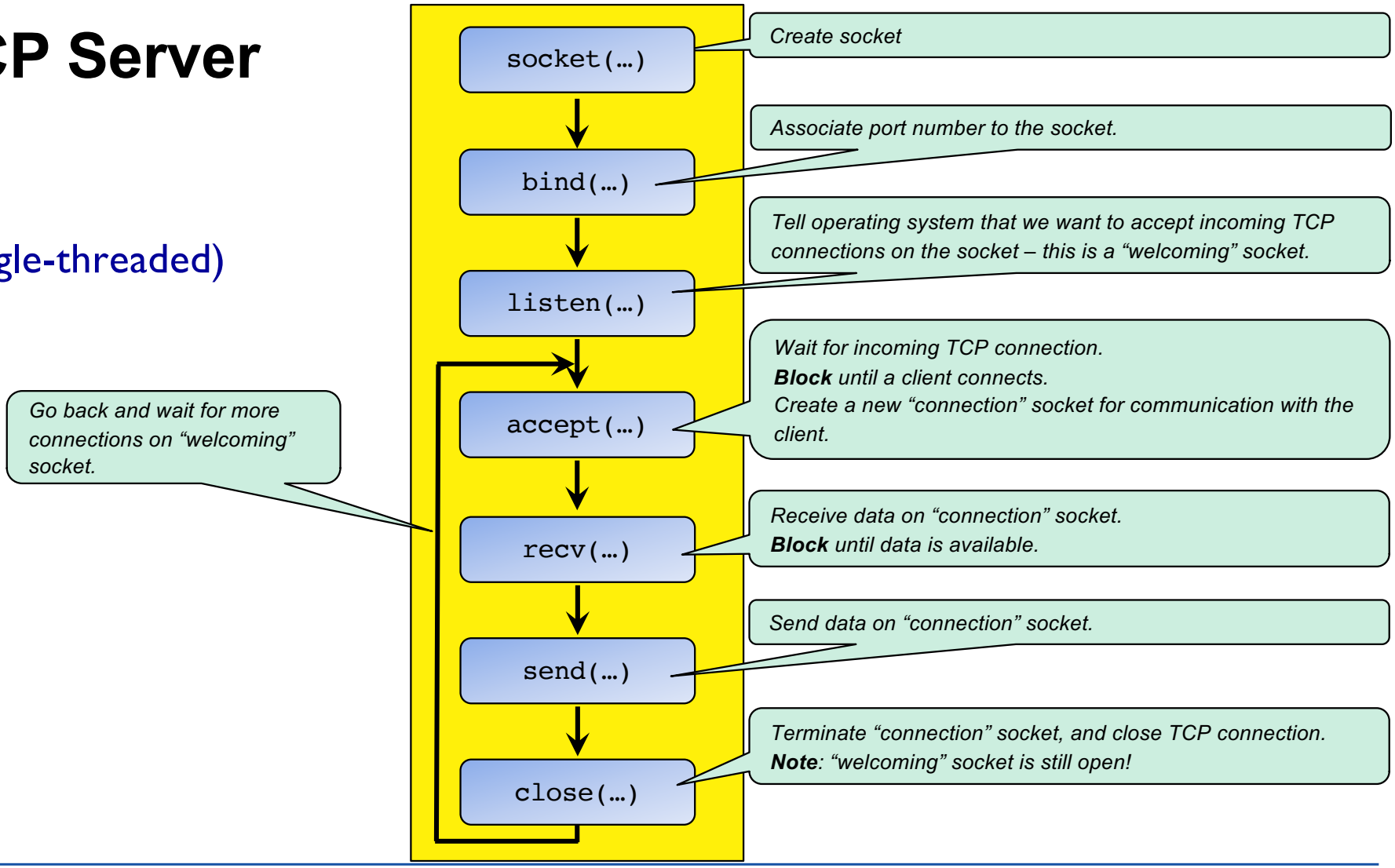
Calls socket() and then connect() system calls to open connection to server “hostname” at port 6789

Send bytes on socket

Receive bytes on socket

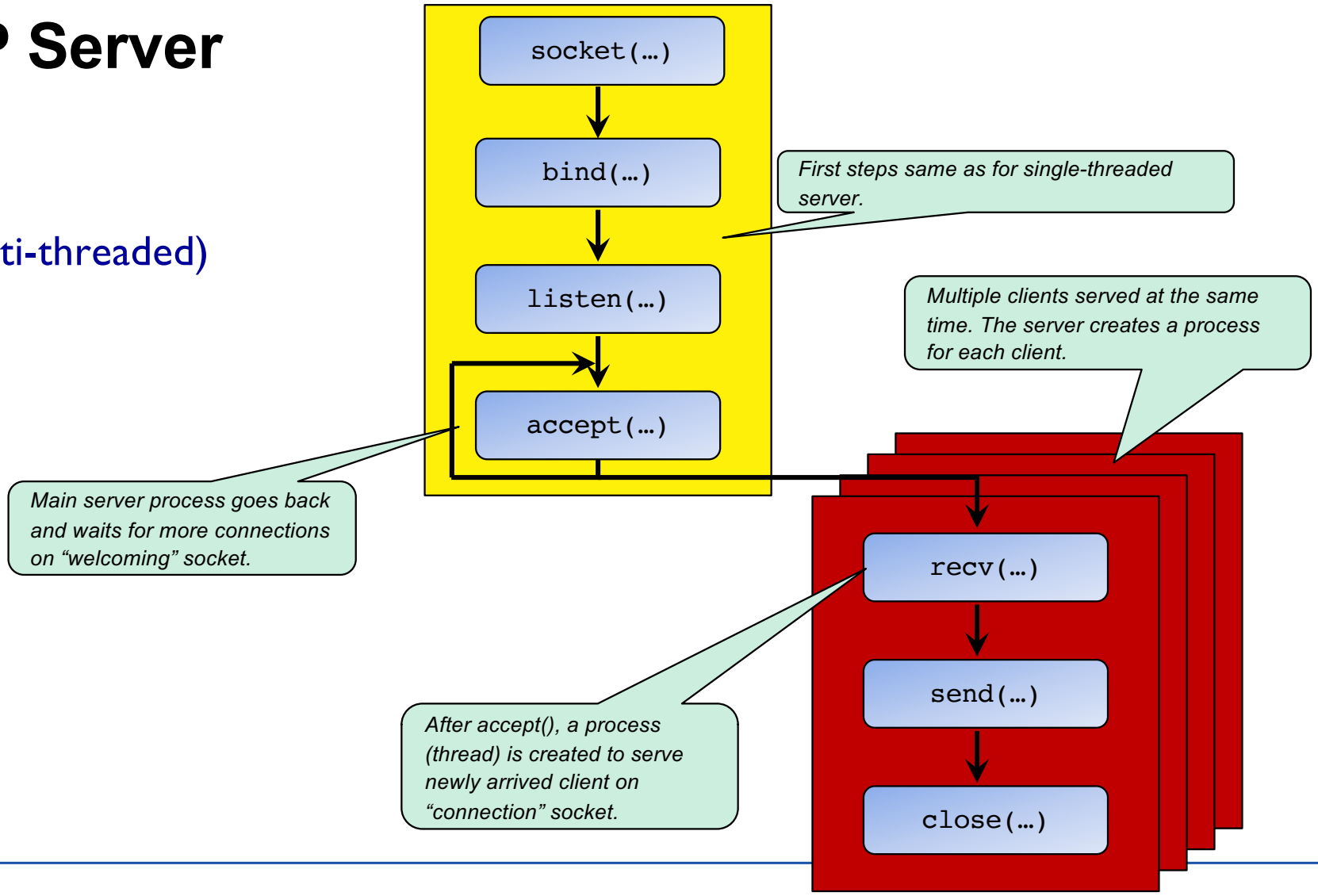
TCP Server

Sequential (single-threaded) version



TCP Server

Concurrent (multi-threaded) version



TCP Server

```
import java.net.*;
class TCPServer {
  static int BUFFERSIZE=1024;

    public static void main(String argv[]) throws Exception
    {
        ServerSocket welcomeSocket = new ServerSocket(6789);

        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            byte[] fromClientBuffer = new byte[BUFFERSIZE];
            int fromClientLength = connectionSocket.getInputStream().read(fromClientBuffer);

            // Compute response for client
            byte[] toClientBuffer = ...

            connectionSocket.getOutputStream().write(toClientBuffer);
            connectionSocket.close();
        }
    }
}
```

Calls socket(), bind() and listen() system calls to create a “welcoming” socket with port number 6789.

Wait for a client to connect. Create a new socket for communication with the client.

End of loop statement. Go back and wait for another client.

Sequential or concurrent server?

Encoding/Decoding

- Programming languages have data types
  - > *Integers, floating point numbers, strings, booleans, etc.*
- To transfer a string between two programs (processes) over the network, for example, we must decide how to represent the string as a sequence of bytes
  - In other words, how convert between bytes and data types?
  - That is what encoding/decoding is about
- Another example: Save a floating point number to file

Text Encoding/Decoding

- Translate between string symbols (such as ”A”, ”?”, ”d”, ”ü”, ”戏”) and bytes
- Define how each symbol is represented as one or more bytes
  - Character encoding standard
  - ASCII, UTF-8, ISO 8859-1, …

ASCII

- American Standard Code for Information Interchange
- 7 bits per symbol
  - > *In practice, one byte per symbol*
  - > *8th bit always zero*
  - – ”A” is ASCII 65 (41 hexadecimal)
  - – ”z” is ASCII 122 (7A hexadecimal)
  - – ASCII 0 to 31 represent control characters
    - > *7 (07 hex) is bell (”beep”)*
    - > *10 (0A hex) is line feed*
    - > *13 (0D hex) is carriage return*

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

UTF-8

- Unicode Transformation Format – 8-bit
- Variable length encoding
- Up to four bytes per symbol
- The first 128 are the same as for ASCII
  - Backwards compatibility – ASCII text is also valid UTF-8
- Dominating format on the Web

There are Many Text/String Coding Schemes

- Families or series of encodings
  - EBCDIC (IBM)
- ISO 8859
- MS-Windows
- Mac OS Roman
- Unicode
  - > *UTF-8*
  - > *UTF-16*
  - …

"knäckebröd av råg" encoded in UTF-8 and decoded as ISO-8859-1 becomes "knÃ¤ckebrÃ¶d av rÃ¥g"

From Wikipedia

Encoding/Decoding in Java

- Methods and constructors to convert between strings and bytes
- Many take encoding scheme (”Charset”) as parameter
  - > *Optional parameter*
  - > *If unspecified, there is a default encoding*
    - Typically UTF-8

```
String string = "Fruit flies like a banana";

// encode a string into a byte array
byte [] encodedBytes = string.getBytes(StandardCharsets.UTF_8);

// decode a byte array into a string
String decodedString = new String(encodedBytes, StandardCharsets.UTF_8);
```

Other Data Types

- In general, any object needs a encoding/decoding scheme to export/import it in a program
  - Transfer over network, save in a file, …
- Floating numbers
  - > *IEEE 754-2008*
  - > *Single-precision (approx 7 decimal digits): 32 bits*
  - > *Double precision (approx 16 decimal digits): 64 bits*
  - Integer
    - > *8, 16, 32, 64 bits* (*or more*)
    - > *Byte order – big-endian or little-endian?*
  - …

Complete Server with Data Processing

```
import java.net.*;
import java.nio.charset.StandardCharsets;

class TCPServer {
  static int BUFFERSIZE=1024;

    public static void main(String argv[]) throws Exception
    {
        ServerSocket welcomeSocket = new ServerSocket(6789);

        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            byte[] fromClientBuffer = new byte[BUFFERSIZE];
            int fromClientLength = connectionSocket.getInputStream().read(fromClientBuffer);

            String clientSentence = new String(fromClientBuffer, 0,
                                               fromClientLength, StandardCharsets.UTF_8);
            // Capitalize string
            String capitalizedSentence = clientSentence.toUpperCase();
            byte[] toClientBuffer = capitalizedSentence.getBytes(StandardCharsets.UTF_8);

            connectionSocket.getOutputStream().write(toClientBuffer);
            connectionSocket.close();
        }
    }
}
```

Class with Charset definitions for coding schemes

Decode byte sequence into string.

Encode string into byte sequence

Java I/O Plumbing

- In the examples so far, we use explicit encoding and decoding
  - String() constructor, getBytes() method, etc
- In Java I/O, another method is to wrap readers/writers onto streams
- InputStreamReader converts a byte stream to a character stream by encoding it with a given character set
- OutputStreamWriter does the opposite

```
inStreamReader = new InputStreamReader(connectionSocket.getInputStream(),
                                       StandardCharsets.UTF_8);

outStreamWriter = new OutputStreamWriter(connectionSocket.getOutputStream(),
                                         StandardCharsets.UTF_8);
```

Java I/O Plumbing II

- With InputStream/OutputStream, every read/write operation leads to send/receive socket operations
- BufferedReader/BufferedWriter buffers data instead, and calls read/write when needed
  - For instance, when buffer gets full

```
inReader = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream(),
                                                    StandardCharsets.UTF_8));

outWriter = new BufferedWriter(new OutputStreamWriter(connectionSocket.getOutputStream(),
                                                      StandardCharsets.UTF_8));
```

TCP Server According to Kurose-Ross, 5th ed.

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
    {
      String clientSentence;
      String capitalizedSentence;

      ServerSocket welcomeSocket = new ServerSocket(6789);

      while(true) {

         Socket connectionSocket = welcomeSocket.accept();

         BufferedReader inFromClient =
            new BufferedReader(new
            InputStreamReader(connectionSocket.getInputStream()));
```

Create input stream attached to socket →

TCP Server According to Kurose-Ross 5th ed., cont

create output stream attached to socket →

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());

clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';

outToClient.writeBytes(capitalizedSentence);
            }
        }
    }
```

Summary

- Socket Programming API
  - Create socket for communication
  - Perform operations on sockets (system calls)
    - *connect(), bind(), listen(), read(), write(), …*
- Byte transfer services
  - Sequence of bytes – stream (TCP)
  - Block of bytes – datagram (UDP)
- Conversion between data types in programming language and bytes
  - Encoding and decoding
- A variety of encoding schemes for conversion between characters (strings) and bytes
  - ASCII, UTF-8, ISO 8859-1, …