# Building blocks of TCP

Internet → IP → Internet Protocol → provides the host-to-host routing and addressing

Internet → TCP → Transmission Control Protocol → provides the abstraction of a reliable network running over an unreliable channel

TCP ;

(1) hides most of the complexity of network communications from our applications
- retransmission of lost data
- in order delivery
- congestion control and avoidance
- data integrity

etc.

(2) All bytes sent will be identical with bytes received and they will arrive in the same order to the client.

(3) optimized for accurate delivery, rather than a timely one.

## Three-way handshake

For the application data to be able to flow between the client and the server, the three-way handshake should be complete.

Each new connection will have a full round-trip of latency before any application can be transferred.

The delay is governed by the latency between the client and the server, not the bandwidth of the connection.

## Congestion avoidance and control

congestion collapse →

# Flow control

a mechanism to prevent the sender from overwhelming the receiver with data it may not be able to process

- receiver may be busy
- under heavy load
- willing to allocate only a fixed amount of buffer space

So, each side of the TCP connection advertises its own receive window (rwnd) which communicates the size of the available buffer space to hold the incoming data.

# CO324: Review of networking fundamentals

Ziyan Maraikar

January 21, 2015

## 1 Layered network models

1. Draw the OSI and Internet protocol stacks beside each other. Map the layers in OSI to those of the Internet layered model.
2. Which of the layers in the Internet protocol stack are implemented in hardware? Where are each of the software layers implemented? (e.g. OS, library, application)
3. Are there any services missing from both the OSI and Internet models, that are vital for modern applications?

## 2 IP networks

4. In what situations are packets lost on a network?
5. How is it possible for packets to be reordered on a network?
6. Can you receive a duplicate of an already received packet? If so, how?
7. What other layer 3 operations are performed by network ***middleboxes*** besides routing?

osi 4

## 3 TCP and UDP

8. Name two network applications that use TCP and two that use UDP.
9. Which of the following properties are required by each applications you stated above.
    1. Reliable packet delivery.
    2. Ordered packet delivery.
10. Which of the properties stated above are provided by UDP and, which are provided by TCP?
11. What is the difference between UDP ***datagrams*** and TCP ***streams***?

22/01/2015

Data Link Layers

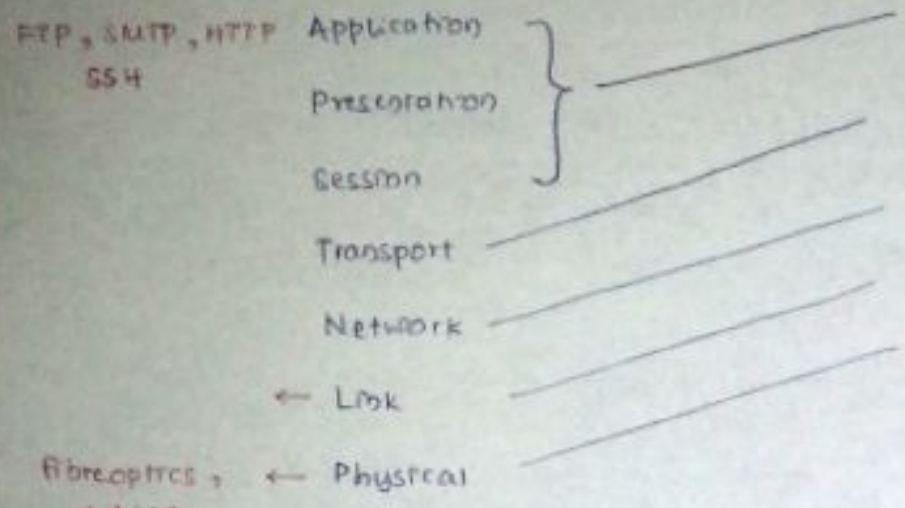
ethernet
SONETS

What is the data link layer of an ADSL connection?

ATM - Asynchronous Transfer Mode

Q1 Layered Network Models

(1) OSI




(2) Link and physical layers are implemented in hardware (network interface card). Part of network layer.

Transport layer → OS kernel  Network layer → library OS kernel

Application layer → applications

(3) Securing services

ARP is specific for ethernet networks. Finds MAC addresses of devices

Q2

SCTP - Stream control and Transmission protocol

How to lookup an IP address of a domain in DNS

root name servers → TLD servers → DNS

ICMP - Internet Control Message Protocol - used in sending routing messages

ethernet networks

Physical Layer → network interface card → generating electrical signals

Link layer → network interface card → framing

wireless networks → WiFi ~~(= wifi)~~

physical layer → radio signals

What is the data link layer of a wifi connection?

802.11 IEEE standard
MAC & LLC sublayer

Why do we need a network layer?

Networks are different from each other. Independent from the data link layer, everything supports IP, so a common platform is made upon IP for connectivity among different networks.

(03) Any services missing from both OSI and TCP/IP stack ?

Security

Why isn't security supported in these architectures?

originated as closed networks with trustworthy end systems and non-mediatries

Security provided by → HTTPS → Hyper Text Transfer Protocol Secure

↑

HTTP + SSH

WEP - Wired Equivalent Protocol

Q2 IP networks

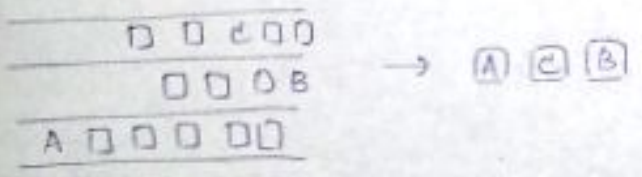
(4) congestion at routers

not enough buffering space allocated.

Routers direct packets from one place to another. The receiving rate of packets can be higher than the sending rate of packets. Then the buffers/queues at the router might not be enough to hold all the incoming packets and packets can be dropped.

(5) The route of the packets can change. Therefore packets might reorder.

Also routers have multiple queues as well.





(6) congestion collapse

If the latency is high, the receiver might request again for the packet. Another packet will be sent again but the previous one will reach the receiver eventually too.

If the message that was sent to acknowledge that the packet was sent to the receiver got lost, another packet will be sent. Ultimately, multiple packets can reach the receiver.

(7) routers, etc → middleboxes

caching

? proxy operation → which layer does this happen at ? → application

03 TCP and UDP

(8) TCP → file transfer , mail services , browsing

UDP → streaming videos , ~~streaming~~ voice over IP , DNS

(9) TCP → requires reliable packet delivery.

streaming videos → ordered packet delivery & , no need to be reliable

ordered delivery is needed for real-time applications

(10) TCP → reliable , ordered → loss-less data

UDP → loss-tolerant data

Web browsing → HTTP → reliable

↳ not real-time applications → no need to be ordered.

What to do if you get the packets out of order?

stutter → drop packets here and there 1 3 → 1 3

results in data loss.

We can arrange the packets orderly ourselves

- If an upp datagram ( IP packet ) is delivered , guaranteed that the packet is not corrupted.

UDP datagrams

discrete packets of data.

each one independent

TCP streams

Stream protocol → data streams

when packets get out of order , TCP has algorithms which order them again.

## Network Programming Concepts

Network applications → 2 roles → clients → mail clients, web browsers, ssh
                                → servers

focused on the software

client → applications used by users to access various services
server → the program providing the service

Any programs that don't fit under the above two categories?

peer-to-peer applications

ex bit torrents

when we are downloading something, we are forced to provide the downloading service to other people downloading the same thing.

## IP addresses and ports

A machine commonly has one single IP address.

When a new machine is connected to a network, DHCP is used to assign an IP address.

A machine runs multiple network programs on a single IP address.

* How to distinguish packets that are meant for different programs?

Port numbers are used to distinguish between programs

Port number → a unique integer identifier for each client/server located a particular IP

privileged ports → 1-1023 require administrative rights ('root') for use, because they are used to host 'well-known' services.

IANA is the authority for this.

Port numbers are assigned at both client and server.

DNS doesn't provide port numbers when a connection is initiated by a client.

Hence, well-known services are assigned standard ports.

## Sockets

A socket represents an endpoint of communication.

A socket is bound to a port on a local machine.

```
// construct socket
DatagramSocket s = new DatagramSocket( )
```

What port is this socket bound to?

Any. Because the port isn't defined any available port can be assigned.

In servers, port numbers don't have to be defined.

Name Resolution

prefer to use hostnames rather than IP addresses to refer to servers.

Need DNS to resolve hostnames to addresses.

Networking API usually provides a function to do this

```
// construct IP address from cmd hostname given in args[0]
InetAddress address = InetAddress.getByName(args[0]);
```

? - DNS provides ease of moving services between IP addresses without inconveniences. Can change IP address without changing hostname.

Mapping IP addresses to physical addresses?

eg. MAC addresses

The IP stack of the OS kernel handles this.

A UDP client

```
// construct IP address
InetAddress address = InetAddress.getByName(args[0]);
// construct socket
DatagramSocket ds = new DatagramSocket();

// construct a packet with destination IP address, port & data
byte buf[] = new byte[256]   <- data portion of UDP packet
DatagramPacket packet = new DatagramPacket(buf, buf.length,
                                address, 12345)
```

address: destination address

12345: destination port number

```
// send the packet
ds.send(packet);
```

Exceptions should be handled.

## sending a datagram

```
// construct IP address from hostname given in args[0]
InetAddress address = InetAddress.getByName(args[0]);
// construct socket
```

## kinds of errors

Handling the error depends on the severity of the error.

- **Permanent failures** → the error condition is not likely to change
  only possibility is to inform user or an administrator (in the case of a server)
  ex. no connectivity, incorrect peer address, name resolution failure
- **Transient errors** → error condition is probably temporary.
  Possibility of recovery by retrying the operation.
  ex. slow or unreliable network connection

## Java Exceptions

Java uses checked exceptions to signify errors. They must be handled in a try-catch block or declared in the method's throws clause.

Permanent failures → display or log the error and abort the operation

Both permanent & transient failures will be signified by exceptions.

Never just discard exceptions in a catch block.

Transient failures → retry the operation a given number of times before giving up.

Bad practise to leave the catch empty

```
try {
    ≡
} catch (Exception e) {
    // nothing here
}
```

catch the exception but do nothing!!
atleast print the exception.
show useful error message to debug the code!

Error handling with throws cause

```
public static void main (String [] args) throws Exception {
}
```

If at any point an exception occured, the method will exit at that point.
using throws clause is again bad practise since it gives no useful error message.

Network Exception Types

| Method / constructor | Exception Type | Error Type |
|---|---|---|
| InetAddress.getByName | UnknownHostException | Permanent |
| DatagramSocket | SocketException | Transient / Permanent |
| DatagramSocket.send | IOException | Transient |

UnknownHostException → host no longer exists

SocketException → creating a socket requires resources. If any are unavailable a SocketException could occur.
can be either permanent or transient

IOException → can be due to a network failure. Most likely able to overcome.

purse

```
try{



}catch(Exception e){
    e.printStackTrace();
}
```

If you no longer need a socket, close it.
a socket is a resource. If you keep up opening sockets but not closing them, OS would run out of resources.

```
public static void main (String[] args) throws Exception {
try{
  ds.send(packet)
  ds.close();
} catch(      ){ . .
```

Running out of sockets could be a real issue

What if ds.send(packet) can cause an IOException. If so, will the socket get closed?

The socket will not be closed.
It will be closed only if there were no exceptions.

```
public state void main(String[] args) {
    DatagramSocket ds = null;
    try {
      InetAddress address =
       ds = new DatagramSocket();
      byte[] buf =
        :
      ds.send(packet);
    } catch( Exception e) {
        - -
    } finally {          // finally block executes whether or not
if(ds!=null) --> ds.close();   an exception occured.
    }
```

Should each exception type be handled in different catch blocks?

```
try {

} catch ( UnknownHostException e ) {

} catch ( SocketException e ) {

} catch ( IOException e ) {

} finally {

}
```

can do different things at each exception.

If all exceptions that might occur can be handled uniformly, no need to have seperate catch blocks.

If error handling logic is different in each case, try use different catch blocks.

UDP server

```
public class UDPServer {
public ... main throws SocketException {
DatagramSocket socket = new DatagramSocket( );
// set up the socket before the server loop
while (true) {
  try {
    byte[] buf = new byte[256];
    DatagramPacket packet = new DatagramPacket( buf, buf.length);
      ↑ packet
    socket.receive(packet);
```

What happens if there's nothing to be received?

- receive operation exists and tells there was no data → non-blocking
- receive operation waits till it gets data → ( returns immediately )

socket waits → blocking
( indefinitely )

default in most stacks for socket operations → blocking operations.

If the network is congested, a send operation can also be blocked. (Rare)

```
// Extract sender address and port
InetAddress address = packet.getAddress(); // client's IP address
int port = packet.getPort(); // client's port
```

loopback address 127.0.0.1

```
        //send back response
        packet = new DatagramPacket(buf, buf.length, address, port)
        socket.send(packet);
      } catch(Exception e) {  e.printStackTrace();
      }
    }
```

Why does the server sits in an infinite loop?

A server is not something that should stop serving clients → Reactive systems

Why does a server need to bind to a well-known port, whereas a client can use any available one?

connection is initiated by the client. Server can extract information easily from the receiving packet.

Why is the SocketException thrown?

## Protocol Properties

UDP provides data integrity only.

An application can implement other properties on top of UDP.

Reliability → Requires message acknowledgment

Ordering → Requires message sequence numbering

Flow control → Ensures that the sender does not overwhelm receiver. Requires a sliding windowing protocol

If all 3 requirements are needed, just use TCP.

What are some applications that require only a few of the properties but not all?

Network gaming → flow control

video streaming → ordering
flow control

## Multicast

cannot be done with TCP.

Multicast lets you send a datagram to a group of receprents, with just a single transmission.

ex. webcast, IPTV

A group of receprents defined by a multicast group that has an address in the class-D range 224.0.00/4. Multicast enabled routers are needed to forward traffic across subnets.

Better supported in IPV6 than IPV4.

## Multicast in Java

```
byte[] mBuf = new byte[256]
try {
  // prepare a join multicast group
MulticastSocket socket60 = new MulticastSocket(8888);
InetAddres address = InetAddress.getByAddress("224.0.0.3")
```

## Fragmentation

An IP packet may be fragmented if its size is larger than the maximum transfer unit MTU of a link.

MTU → Maximum Transfer Unit

Degrades performance.

To avoid this, the maximum recommended size of a UDP payload is 512 bytes.

No fragmentation in TCP.

## Datagrams

TCP → reliable
dedicated point-to-point channel / illusion of one
establish the connection → transmit the data → close the connection
all data sent over the channel received in the same order in which it was sent

Datagrams → send & receive completely independent packets of information
do not have / need a dedicated point-to-point channel.
delivery to destination not guranteed.
Order of arrival not guranteed

Datagram → an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guranteed.

Each datagram packet received by the server indicates a client request for a quotation.
When the server receives a datagram, it replies by sending a datagram packet that contains a one-line "quote of the moment" back to the client.

## TCP clients

TCP is a stream protocol in which packet boundaries are invisible to the application. Data is received and transmitted as a continuous sequence of bytes. It provides,

(1) Reliability
(2) Ordering
(3) Flow control

The java socket class represents a TCP socket.

```
Socket socket = new Socket();
InputStream sin = socket.getInputStream();
OutputStream sout = socket.getOutputStream();
```

Binary data is read & written to the socket via the associated IO streams.

### sending and receiving text

```
InetAddress address = InetAddress.getByName(args[0]);
try {
    Socket socket = new Socket(address, PORT)
    BufferedReader sin = new BufferedReader(new InputStreamReader(
        InputStreamReader → (socket.getInputStream()));
    BufferedWriter sout = new BufferedWriter(new OutputStreamWriter
        (socket.getOutputStream()));
} catch (Exception e) {
    e.printStackTrace();
}
```

Read String

wrap this in InputStreamReader

wrap this in OutputStreamWriter

Reader ← text

↑ InputStreamReader

InputStream ← binary

Note that the application messages must be properly framed

e.g. using a delimiter like \n.

```
sout.write("Hello world\n");
sout.readLine();
```

What does BufferedReader and BufferedWriter classes do?

introduces external mechanisms to collect all the data.
provides reliability. Gives us some additional convenience and efficiency.

## TCP servers

Java uses a seperate ServerSocket class to bind to a port & accept connections from clients.

```
ServerSocket ss = new ServerSocket(PORT);
Socket socket = ss.accept();
```

} 2 step process

accept() returns a new socket connected to the client. (uses a different port)

Client initially connects to the well known port ss. Then the server socket accepts that connection and returns a new socket to the particular client to communicate.

```
try {
    ServerSocket ss = new ServerSocket(PORT);
    Socket socket = ss.accept();

    BufferedReader sin = new BufferedReader(

    BufferedWriter sout =
```

What happens if multiple clients want to connect at once?

Serve them one after another.

use a loop.

```
ServerSocket ss = new ServerSocket(PORT)

while (true) {
 try {
    Socket socket = ss.accept();      // sockets maintain a queue for
                                      accepting client connections
    BufferedReader                    coming in. If more clients are coming
                                      in after the queue is filled up,
                                      those clients are rejected.
    BufferedWriter
                                      This is not very efficient.
   sin.readLine()
   sout.write("  \n");
   socket.close();
 } catch

 }
```

# Framing

## Message as streams

Suppose a client sends two consecutive messages, & the server does a read. Will it get,

1. both messages at once
2. The first message only
3. part of the first message

It depends on the network conditions and TCP/IP stack

Can even get the first message + part of the second message!

* No guarantee about 'how' the data is transfered

TCP can only send & receive from a byte stream, but application protocols are built with discrete messages

We must define a method of framing messages, so that the message boundaries are unambiguous.

Method used depends on kind of data.

- Text
- Binary

## Text Protocols

Text → anything that can be represented by using ASCII

Most applications protocols of the Internet are textual.

- Human readable → so easy to debug
- Historically, most applications were textual ex Telnet, Email

- Disadvantages
  - Vulnerable to security attacks like buffer overflows.
  - Bandwidth and cpu inefficient

ex. SMTP email using a command prompt

How are non-text mail attachments handled?

In the SMTP text-based email example, the data body can have unlimited size. No restrictions applied. When such a long message will be sent, the buffers will overflow. The server would crash.

## Delimiters

We can delimit text protocol messages using a special character.

The usual delimiter used in internet protocols are the line termination characters.

CR \r

LF \n

CRLF

```
Socket socket =
BufferedReader sin =

BufferedWriter

sout.write("hello world\n");
sin.readLine();
```

BufferedReader.readLine splits the newline delimited messages in a stream.

Even though the data was received at the socket as the first message and part of the second message, using a delimiter will result in proper splitting of the message.

hello\n-wor

## Binary Protocols

support transmission of arbitrary data. Not common in internet.

Usually contains a fixed-format header that describes the payload.

| Header | Payload |
|---|---|

TCP/UDP protocols are binary.

- suited to describing structured data
- easier to parse - metadata ~~before~~ received before payload

how to prevent buffer overflows when using binary data?

In the header, specify the length of the message.

When the server receives the message, look at the header, and if the message length is larger than acceptable, reject it.

- efficient use of bandwidth.

ex. Basic encoding rules for ASN.1, an OSI standard used in protocols such as LDAP

| Type | Length | Value | End-of-content |
|---|---|---|---|

basically a delimiter

raw message in gmail?

How to send binary data using text protocols?

ex. send an image attached to an email

Encode the binary data using ASCII

standard way of doing this → Base 64

MIME → Multiprotocol Internet Message Encoder

09/08/2015

# Protocol Patterns

## Protocol specifications

A protocol specification describes what a protocol is supposed to do

1. participants & their roles
2. Types of messages & their format
3. valid message sequences
4. Potential error conditions

A specification can be ;

formal → use specification language

Informal

## Protocol Design

How it can be implemented.

## Client-server protocol patterns

Unidirectional streaming → One party sends the other a continuous stream of data.
May be implemented upon UDP or TCP depending on reliability requirements.

request response → Client alternates between sending a request and receiving a response.
Requires a reliable transport protocol.

## Unidirectional (streaming) protocols

ex. medical monitoring

Client → medical device that monitors signs.

Data from multiple monitors logged to a server

- should we use UDP or TCP ? Text or binary ?
- how to handle multiple devices → universally unique device ID.
- how do we handle client (monitor) or server failure ? suitable method of alerting staff

Binary format can be used since, vital signs like pulse and blood pressure can be represented as binary, it makes no sense to convert them back to ASCII and use a text format. TCP for reliability. Binary for ease of parsing favoured in embedded systems.

UDP can be used if the monitoring frequency is low.

TCP is the option to be used since reliability is essential in recording patient's signs.
The major overhead in TCP compared to UDP is the connection establishment
How often is the client likely to establish a connection to the server?

Once, when the monitor (client) is switched on or reset.

Best option → Binary over TCP.

What should be included in the protocol?

Do we need a header? Yes

What kind of data should be in the header?

- Metadata about readings

  (Metadata → details about the data we are sending over the protocols)

  - Timestamp
  - Types of readings / units of readings
  - Patient ID

- Different message types for readings? or one single message

  There can be correlations between the readings of one patient. It would be useful to have all the data / readings in one single message at once.

- versioning system

  protocol should have a version ID.

  gives manufacturer the flexibility to add / remove / modify features as needed.

  The version ID will identify the data formats.

Serialisation formats → gives versioning, finding metadata

It would be good to use a standard serialisation format if all the above specifications are required.

12/09/2015

- How to keep time in the system

  time-stamp each message

  who should put this time-stamp? Client or server?

  Logically makes sense for the client to put the timestamp. But have to make sure the time in the client is precise & accurate

- How to handle upgrades to client devices

  protocol versioning

a protocol called NTP (Network Time Protocol) controls the time on the server and all of the clients to ensure all the clients have the same time.

The Network Time Server distributes time.

```
public class PatientMonitor

    static final long patientID = "12345678";
           String localhost = "localhost";

public static void main (String[] args) throws IOException, InterruptedException {

    Random rand = new Random();
    InetAddress address = InetAddress.getByName(localhost);

try (Socket socket = new Socket (address, DataLogger.PORT);
    DataOutputStream sout = new DataOutputStream (Socket.getOutputStream()); ) {

     while(true){

         Reading p = new Reading( patientID,
                                  System.currentTimeMillis();
                                  60 + rand.nextInt(40);
                                  30 + 10* rand.nextFloat( ) );

          sout.write(p.data( )); // gives a byte array corresponding to
                                    one Reading
          Thread.Sleep(1000);

        }
    }
  }
}
```

1.

```java
/** Representation of data gathered by patient monitor. */
public class Reading {
  /** The size required to store a reading. */
  static final int SIZE = Long.BYTES * 2
      + Integer.BYTES + Float.BYTES;

  /** ByteBuffer is a handy type for storing binary data. */
  final ByteBuffer buf;

  /** Constructs a Reading from the given values. */
  public Reading(long id, long time, int pulse, float temp) {
    buf = ByteBuffer.allocate(SIZE);
    buf.putLong(id);
    buf.putLong(time);
    buf.putInt(pulse);
    buf.putFloat(temp);
  }

  /** Constructs a Reading from the given stream. */
  public Reading(DataInputStream sin) throws IOException {
    byte[] a = new byte[SIZE];
    sin.readFully(a);
    buf = ByteBuffer.wrap(a);
  }

  /** @return the underlying data array. */
  byte[] data() {
    return buf.array();
  }

  /** Reading data rendered as a string. */
  @Override
  public String toString() {
    return "ID:" + buf.getLong()
        + ", time:" + new Date(buf.getLong())
        + ", pulse:" + buf.getInt()
        + ", temp:" + buf.getFloat() ;
  }

  /** A quick unit test for the class. */
  public static void main(String[] args) throws IOException {
    Reading p1 = new Reading(12345l,
      System.currentTimeMillis(),
      60, 37.5f);

    DataInputStream din = new DataInputStream(
      new ByteArrayInputStream(p1.data()));

    Reading p2 = new Reading(din);

    assert p1.data().equals(p2.data());
  }
}
```

using final ensures all the fields are initialized in the constructors

Serialize the data.

represent the object as a string

First serialize, then deserialize and see if they are equal

2.

```
/**
 * Representation of arithmetic operations.
 * The {@link eval} method computes their value.
 */
enum Operation {
  ADD() {
    int eval(int x, int y) { return x + y; }
  },
  SUB {
    int eval(int x, int y) { return x - y; }
  },
  MUL {
    int eval(int x, int y) { return x * y; }
  },
  DIV {
    int eval(int x, int y) { return x / y; }
  };

  /** Each constant supports an arithmetic operation **/
  abstract int eval(int x, int y);

  /**
   * Reads an arithmetic expression in prefix form.
   * @throws IllegalArgumentException if the expression is malformed.
   **/
  static int eval(Scanner s) {
    Operation op = Operation.valueOf(
    s.next().toUpperCase());
    return op.eval(
    s.nextInt(), s.nextInt());
  }
}
```

16/02/2015

Skeleton

# Concurrency

Programs often need to perform multiple tasks concurrently (or parallel).

- serve multiple clients
- Handle Gui events while doing network I/O

## concurrency mechanisms

Operating systems and languages provide various concurrency mechanisms.

Processes : Traditional unit of execution of in an OS
ex. the JVM runs as an operating system process

Threads : Lightweight units of execution that share a common memory address space. (Processes have many threads in each of them)

Asynchronous I/O : I/O operations are performed in non-blocking fashion

## Multithreading Review

### Threads

Unit of execution in JVM.
one thing happening at a time.
order the JVM executes them is nondeterministic. → JVM+OS decides this.

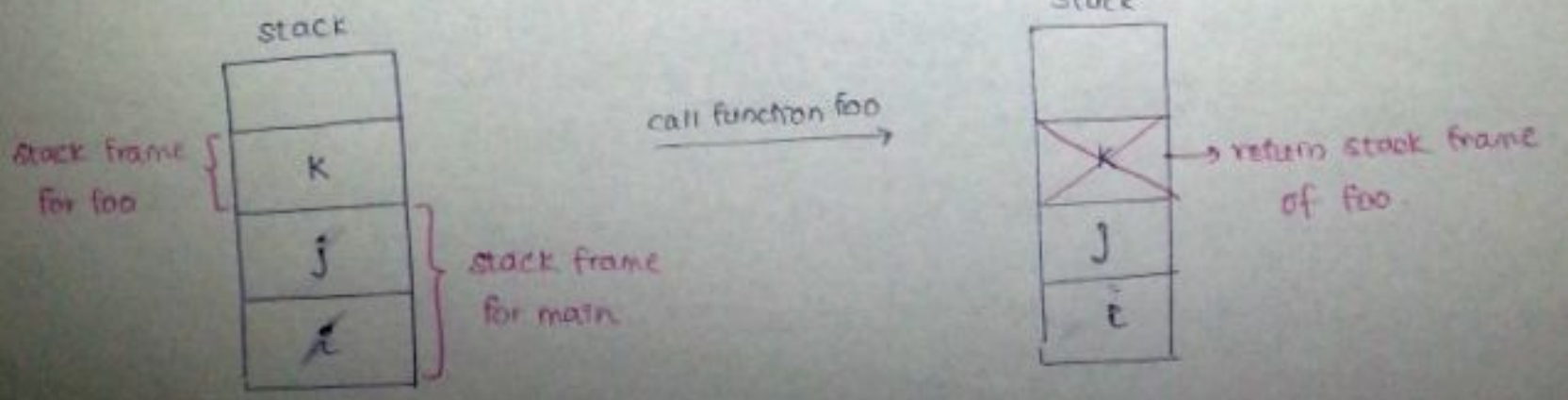
Threads share the JVM's memory. Each thread has,

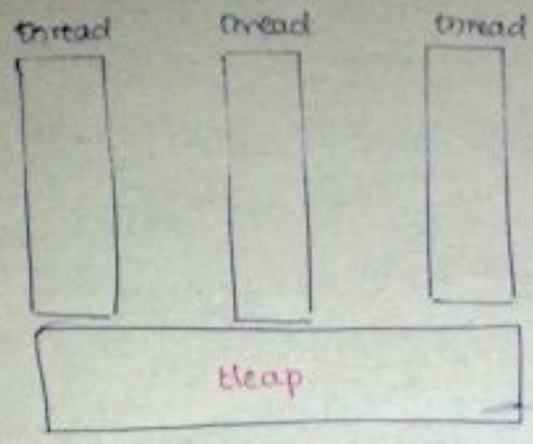
- its own execution stack (stores local variables of primitive types)
- shared access to items on heap (stores items created with the new keyword)

ex.

```
main
    int i, j
    function foo(j)

foo(int k) {
    =
    return
}
```






All operations performed by threads go to Heap.

Only guarantee → at some point, all the threads will come.

Multithreaded Servers

19/02/2015

(*) The synchronized method takes an Object of a monitor (a lock object) which is shared among the threads of clients. synchronized() ensures that requestNum is locked at a particular moment. Once the object is locked by one thread, all other threads have to wait until they get to lock the object.

If the try-catch-finally block of the run() method was put into a huge block of synchronized(), the server will only serve one client since any other threads cannot lock the object. Other threads will be able to lock the object only once one client has finished running. This is called Starvation.

→ next page bottom

# Thread synchronisation

February 19, 2015

```
public class ThreadedServer extends Thread {
  static final int PORT = 4321;
  static int requestNum; tracks the number of requests the server has received (from all the clients)
  final Socket socket;
  static Object lock = new Object();
  public ThreadedServer(Socket s) {
    this.socket = s;
  }

  @Override
  public void run() {   run() overrides the run() in Thread class
     try (Scanner sin = new Scanner(socket.getInputStream() );
     PrintStream sout = new
         PrintStream(socket.getOutputStream()) ) {
      while(true) {
        int i = requestNum;   requestNum copied to local variable
        String req = sin.nextLine();
        sout.println(req + " " + i);
        requestNum = i + 1;
        System.out.println("Total requests served: "+requestNum);
      }
     } catch (IOException e) {
     e.printStackTrace();
    } finally {
     try {
      socket.close();
     } catch (IOException e) {}
    }
  }
```

Since requestNum is a static variable, it's shared among all the clients.

A data race occurs.

Sometimes the requestNum will not be incremented & sometimes even might go backwards.

Instead of the highlighted lines, sout.println(request + " " + requestNum++); Still the problem appear at a lower level, considering the number of instructions taken.

(*)
```
while(true){
  String req = sin.nextLine();
  synchronized (lock){
    sout.println(req + " " + requestNum++)
  }
  System.out.
}
```

* an instance where requestNum is not incremented

| $T_1$ | $T_2$ |
|---|---|
| i = 1 | i = 1 |
| sout.print(i) → | |
| requestNum = 2 | requestNum = 2 |

← for ex. if both threads started at the same time

This should have ended as requestNum = 3

* an instance where requestNum is decremented

| $T_1$ | $T_2$ |
|---|---|
| i = 1 | i = 1 |
| requestNum = 2 | ⋮ |
| ↓ | ∨ |
| requestNum = 3 | requestNum = 2 |

The 2 threads start at once, but one thread serves more clients than the other. So when the slower threads finishes serving one client, value of requestNum might be wrong.

Each thread is executed completely independently.

```
 public static void main(String[] args) throws IOException {
  try( ServerSocket ss = new ServerSocket(PORT)) {
   while (true) {
    Socket socket = ss.accept();
    new ThreadedServer(socket).start();
   }
  }
}
}
```

Each time a client requests, a new thread is created to handle that client alone.

```
public class TCPClient {
 public static void main(String[] args)
 throws IOException {
  InetAddress address = InetAddress.getByName("localhost");
  Random rand = new Random();

  try (Socket socket = new Socket(address,
      ThreadedServer.PORT);
  Scanner sin = new Scanner(socket.getInputStream() );
  PrintStream sout = new PrintStream(socket.getOutputStream()
      );) {

   for (int i=0; i<1000; i++)
   try {
    sout.println(args[0]);
    System.out.println(sin.nextLine());
    Thread.sleep(100+rand.nextInt(400));
   } catch (Exception e) {}
  }
 }
}
```

```
synchronized (lock){
    while(true){

    }
}
```

only one client will be served. Other threads never get a chance of locking the object.

2

Generally a bad idea to have input-output statements inside a synchronized() block. If I/O gets blocked, until it's unblocked, no other thread won't be able to access the lock.

```
while(true){
    synchronized( lock){

    }
}
```

Other threads might have a chance of locking the object once when the locked object exists the synchronized loop and before the next iteration of the while loop starts. No guarantee

| $T_1$ | $T_2$ |
|---|---|
| syn{ c=1 rN=2 } req #2 missing | syn{ c=2 rN=3 } req#3 |
| Total requests 3 | Total requests 3 |

19/09/2015

# Shared Data

In Java, threads may share data via

static variables
shared references

## Data Races

Happens when there are two memory accesses in a program, where both
- target the same location
- are performed concurrently by two threads.

Data races can never be resolved using ~~data races~~ ~~at~~ testing alone.

represents a bank account

```
public class AccountServer extends Thread {
  static final int PORT = 4321;
  final Socket socket;

  public AccountServer(Socket s) {
    this.socket = s;
  }

  @Override
  public void run() {
    try (Scanner sin = new Scanner(
        socket.getInputStream())) {
      while(!interrupted())
      new Transfer(sin).execute();
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      try {
        socket.close();
      } catch (IOException e) {}
  }
}

public static void main(String[] args) throws
    IOException {
  //Create two test accounts for transfers:
  new Account(); new Account();

  try( ServerSocket ss = new
      ServerSocket(PORT)) {
    while (true)
      new AccountServer(
        ss.accept()).start();
    }
  }
}
```

```
public class AccountClient {

  public static void main(String[] args)
      throws IOException {
    InetAddress address =
        InetAddress.getByName("localhost");
    Random rand = new Random();

    try (Socket socket = new Socket(address,
        AccountServer.PORT);
      PrintStream sout = new
          PrintStream(socket.getOutputStream()
          )) {

      int fromID = Integer.parseInt(args[0]),
        toID = Integer.parseInt(args[1]);

      for (int i=0; i<1000; i++)
      try {
        int amount = rand.nextInt(100) - 50;
        sout.println(new Transfer(fromID, toID,
            amount));
        Thread.sleep(rand.nextInt(100));
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }
}
```

# transfer random amounts

can also transfer negative amounts.

```java
/** Message format for account transfer */
public class Transfer {

 final int fromID, toID, amount;

 /** Constructor for serialising transfer message */
 public Transfer(int fromID, int toID, int
     amount) {
  this.fromID = fromID;
  this.toID = toID;
  this.amount = amount;
 }

 /** Deserialise a transfer message */
 public Transfer(Scanner sin)
    throws ProtocolException {
  if (!sin.hasNextInt()) throw new
      ProtocolException("missing account ID");
  fromID = sin.nextInt();

  if (!sin.hasNextInt()) throw new
      ProtocolException("missing account ID");
  toID = sin.nextInt();

  if (!sin.hasNextInt()) throw new
      ProtocolException("missing amount");
  amount = sin.nextInt();

 }

 /** Execute the specified transfer */
 public void execute() {
  Account ac1 = Account.find(fromID);
  Account ac2 = Account.find(toID);
  if (ac1==null || ac2==null)
   throw new IllegalArgumentException(
    "invalid account ID");

  ac1.transfer(ac2, amount);
  System.err.println("Trasferred "+amount+
   " from a/c "+ac1+" to a/c "+ac2);
 }

 @Override
 public String toString() {
  return String.format("%d %d %d\n", fromID,
      toID, amount);
 }
}
```

```java
/** Represents a bank account */
public class Account {

 private static int count = 0;
 /** Hash table of all accounts that have been
     created */
 private static Map<Integer,Account> accounts
  = new HashMap<Integer,Account>();

 /** Find account by ID */
 public static Account find(int id) {
  return accounts.get(id);
 }

 final int id;
 private int balance;

 /** Construct account with unique ID and add it to
     global hash table */
 public Account() {
  synchronized(accounts) {
   id = count++;
   accounts.put(id, this);
  }
 }

 /** Negative amount indicates withdrawal */
 public void deposit(int amount) {
  balance += amount;
 }

 /** Transfer amount from this account to another. */
 public void transfer (Account to, int
     amount) {
  deposit(-amount);
  to.deposit(amount);
 }

 public int balance() {
  return balance;
 }

 @Override
 public String toString() {
  return "ID:"+id+", balance:"+balance;
 }
}
```

(i) Describe a data race present in Accounts class.

i.e. What can happen if two or more threads access the same account?

There has to be data shared by multiple threads.

The variable 'balance' is shared between threads.

balance = 10

| $T_1$ | $T_2$ |
|---|---|
| deposit(1) | |
| | deposit(2) |
| | balance ← 12 |
| balance ← 11 | |

Both threads read balance as 10 in the beginning.

The method balance() would not create a data race even though it reads the shared variable 'balance'. This is because a data race would occur only if multiple var threads tried to modify the same shared variable.

The account Object itself can be used to synchronize and lock the deposit() method.

```
public void deposit (int amount) {
    synchronized (this) {  <-- using accounts Object as the lock object
        balance += amount;
    }
}
```

This can also be coded in a shortcut method as following;

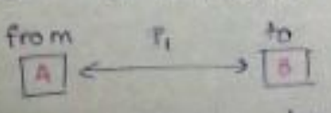
```
public synchronized void deposit (int amount) {
    balance += amount;
}
```

The entire method body is in the synchronized loop.

Any other method that modifies 'balance' is also prone to data races.

Even if synchronized, things can still go wrong.

Consider the transfer() method.




balance -= amount  balance += amount

Both the accounts and the balance is shared by multiple threads.

ex: While $T_1$ is transferring from A to B, another thread could be trying to deposit into A or B.

ex. while $T_1$ is transferring from A to B, another thread would be trying to transfer amounts between another account C and either A or B.

synchronization required

```
static Object lock = new Object();
```

Since two shared objects are modified, two synchronizations have to be done.

We have already synchronized the deposit() method. Why do we need to synchronize the transfer() method again?

Assume the deposit() method has not been synchronized yet. Consider balance is directly modified as following;

```
void transfer(Account from, Account to, int amount) {
    balance -= amount
    to.balance += amount
}
```

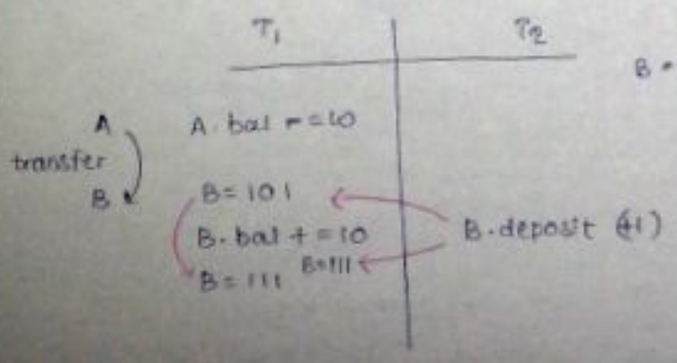
why can't we use a lock object & lock the whole method?

```
// synchronization
void transfer(from, to, amount){
  synchronized(this){
      balance -= amount;
  }
      synchronized(to){
          to.balance += amount;
      }
  }
}
```

Now there are no races conditions. Two accounts are individually synchronized.

If deposit() method is synchronized, this doesn't have to be done. Both methods are same in synchronization.

Can something still go wrong when the transfer() method is synchronized?



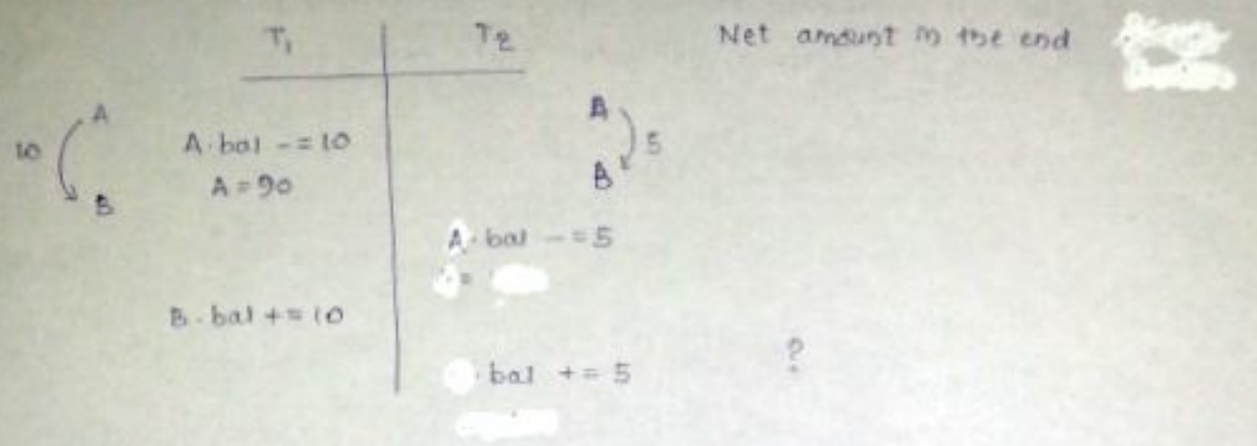

even if B.deposit() happened before or after B.bal +=10, the final amount of B would be 111.

deposit() also synchronized.

Still no data races occur.

What if another thread wanted to transfer from A to A ? while $T_1$ transfers from A to B.

A = 100, B = 200




Skip 1(b).

2(b) can the synchronization cause starvation?

Not likely

Because the locking is done only for an instance of modifying the balance.

## Deadlock

Assume the transfer method is as following ;

```
void transfer (Bank to, amount) {
    synchronized (this) {
        balance -= amount;
        synchronized (to) {
        to.balance += amount;
        }
    }
}
```

nested synchronized blocks.

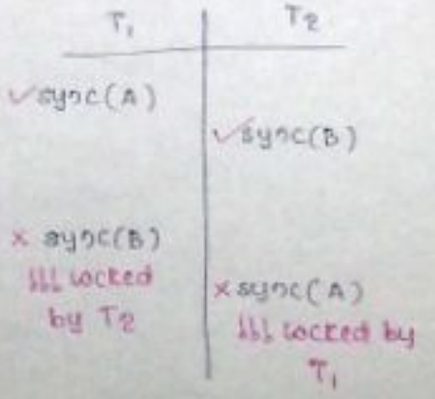
This can be done in Java.

consider the following scenario.

$T_1$ : A → B

$T_2$ : B → A

A transfer requires both accounts be locked.




$T_1$ cannot lock B since $T_2$ has already locked B. Therefore, $T_1$ is blocked now $T_2$ cannot lock A since $T_1$ already locked A and also because $T_1$ is now blocked. $T_1$ and $T_2$ can never get unblocked. This situation is called a Deadlock.

A deadlock happens if the modification the threads are trying to achieve is circular and when those modifications are done on exclusive resources.

Data races are fairly easy problems to solve. But deadlocks are much harder to solve. Because the circular weights can be huge!