Homework Assignment 5:
A Sliding Window Data Transfer Protocol

Reliable data transfer means:
• Data is delivered in the same order without loss or duplication.
• Data sent is eventually delivered.
• Provided that:
  • Source and sink are always connected and correctly initialized.
NetworkSocket

```java
NetworkSender
```

SourceTester

SinkTester

Blocking vs. nonblocking

- **Blocking TCP/IP**
  - Method `sendData(byte data[])` blocks until all bytes of data are sent and acked.
  - User cannot call another `sendData` unless the current one is executed.

- **Nonblocking TCP/IP**
  - Method `sendData(byte data[])` adds data to the buffer, and returns to the caller.
  - Data is sent later.
Testing and Debugging

- Program involves multiple computers.
- Ability to discover errors becomes harder.
- A research method to test distributed systems
  - Specify interactions between different entities of a distributed system in a single service file.
  - Specifications of interactions describe the relations between different interactions.
- Use Remote Method Invocation (RMI) to implement this test method
- Lock before issuing an RMI call!

```
class SourceTester {
    TesterInterface tester; // lookup...
    ...
    public void sendData(byte[] data) {
        try {
            tester.lock();
            tester.sendData(data);
            tester.printLog(Thread.currentThread().getName() + "SW_Source:acceptData()" +);
            checkAssertions(true);
            // method body
            // ...........
            tester.unlock();
        } catch (RemoteException re) {
            re.printStackTrace();
        }
    }
}
```
Tester Execution

- Change "cicada.cs.yale.edu" to an appropriate machine, e.g., "csy02.cs.wmich.edu".
- Execute Tester on machine A.
- Execute Sink on machine B.
- Execute Source on machine C.
- Machines A, B and C may be the same machine during development.

Assertions

- Assertion definition.
- Have method checkAssertion at the start of every method.

```java
class DT { ... }

public void sendData(...) throws RemoteException {
    synchronized (tester.lock)
    { ...
        checkAssertions(true);
        ...
    }
}

public void deliverData(...) throws RemoteException {
    synchronized (tester.lock)
    { ...
        checkAssertions(true);
        ...
    }
}

//end class DT

void checkAssertions(boolean debugInfo)
{
    tester.printLog(":");
    if ( ( srcBufUsed >= 0 ) && ( srcBufUsed <= srcBufSize ) )
        tester.printLog(":bufCondition(true)" );
    else
        tester.printLog(":bufCondition(false)" );
    tester.printlnLog("");
    try{
        tester.printLog(":");
        if ( ... )
            tester.printLog(":methodDefinition(" );
        else
            tester.printLog(":methodDefinition(" );
        ...
    }catch (RemoteException re) {}
} }
```
**Synchronization**

- A simple solution to obtain a mutual-exclusive lock for a code block
  - Define a **static** Object
  - Use `synchronized` statement
- For example:

  ```java
  static Object lock = new Object();
  ...
  synchronized (lock) {
    // do things exclusively
    ..........
  }
  ```

**Common Synchronization Errors**

- `lock()` not followed by `unlock()`.

  ```java
  void xyz()
  {
    ..........
    lock();
    if (condition)
      break;
    ..........
    unlock();
  }
  ```

  **Fix:**

  ```java
  void xyz()
  {
    ..........
    lock();
    if (condition) {
      unlock();
      break;
    }
    ..........
    unlock();
  }
  ```

- `lock()` not followed by `unlock()`.

  ```java
  void xyz()
  {
    ..........
    lock();
    while (condition)
    wait();
    ..........
    unlock();
  }
  ```

  **Fix:**

  ```java
  void xyz()
  {
    ..........
    lock();
    while (condition)
    {
      unlock();
      wait();
      lock();
    }
    ..........
    unlock();
  }
  ```
Common Synchronization Errors

- lock() not followed by unlock().

```c
void abc()
{    lock();
    xyz();
    lock();
    unlock();
}
```

**Fix:**
```c
void xyz()
{    lock();
    //lock();
    unlock();
}
```

How to start?

1. Implement "stop and wait" protocol.
2. Increase the send window.
3. Increase LRD probabilities.
4. Change the sending window from fixed window to adaptable window.
5. Make random scenarios.

Design Specifications

- Demonstrate that you have thought about the problem and your solution is plausible
  - Use diagrams and/or pseudo-code
  - Concise and simple

- Demonstrate how you will test your code
  - Check for edge/boundary cases
  - Accurately determine correctness of your code

- Due date: March 17, 2005
Chapter 3: Transport Layer

Our goals:

- understand principles behind transport layer services:
  - multiplexing/demultiplexing
  - reliable data transfer
  - flow control
  - congestion control

- learn about transport layer protocols in the Internet:
  - UDP: connectionless transport
  - TCP: connection-oriented transport
  - TCP congestion control

Source: Guizani, Kurose and Ross textbooks and Internet. Material from textbooks is copyrighted by appropriate authors. For example, All material copyright 1996-2006, J.F Kurose and K.W. Ross. All Rights Reserved.

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
- 3.6 Principles of reliable data transfer
- 3.7 Flow control
- 3.8 Connection management
- 3.9 TCP congestion control
- 3.10 TCP in brief
- 3.11 TCP fairness
- 3.12 Delay Modeling

Delay modeling

Q: How long does it take to receive an object from a Web server after sending a request?

Ignoring congestion, delay is influenced by:

- TCP connection establishment
- data transmission delay
- slow start

Notation, assumptions:

- Assume one link between client and server of rate R
- S: MSS (bits)
- O: object size (bits)
- no retransmissions (no loss, no corruption)
- Window size:
  - First assume: fixed congestion window, W segments
  - Then dynamic window, modeling slow start

What is the delay if no congestion control? \( \text{delay} = 2\text{RTT} + \frac{O}{R} \)
Fixed congestion window (1)

First case:
WS/R > RTT + S/R:
ACK for first segment in window returns before window’s worth of data sent

delay = 2RTT + O/R

Fixed congestion window (2)

Second case:
WS/R < RTT + S/R: wait for ACK after sending window’s worth of data sent

delay = 2RTT + O/R
- (K-1)[S/R + RTT - WS/R]

K = # of windows of data needed to cover the object
K = Q / (WS/R)
Server is stalled for K-1 periods of time!

TCP Delay Modeling: Slow Start (1)

Now suppose window grows according to slow start

Will show that the delay for one object is:

\[ \text{Latency} = 2RTT + \left\lfloor \frac{Q}{R} \right\rfloor RTT + \frac{3}{R} \left( 2^P - 1 \right) \frac{S}{R} \]

where \( P \) is the number of times TCP idles at server:

\( P = \min(Q, K-1) \)

- where \( Q \) is the number of times the server idles
- if the object were of infinite size.
- and \( K \) is the number of windows that cover the object.
TCP Delay Modeling: Slow Start (2)

Delay components:
- 2 RTTs for connection estab and request
- O/R to transmit object
- time server idles due to slow start

Server idles:
\[ P = \min(K-1,Q) \text{ times} \]

Example:
- O/S = 15 segments
- K = 4 windows
- Q = 2
- P = \min(K-1,Q) = 2

Server idles \( P = 2 \) times

TCP Delay Modeling (3)

\[ \frac{S}{R} \cdot RTT = \text{time from when server starts to send segment until server receives acknowledgement} \]

\[ 2^{k-1} \cdot \frac{S}{R} \] is time to transmit the \( k \)-th window

\[ \frac{S}{R} \cdot RTT - 2^{k-1} \cdot \frac{S}{R} \] is idle time after the \( k \)-th window

Delay:
\[ \frac{O}{R} + 2RTT + \sum \text{idleTime} \]
\[ = \frac{O}{R} + 2RTT + \sum \text{RTT} - 2^{k-1} \cdot \frac{S}{R} \]
\[ = \frac{O}{R} + 2RTT + PRTT + \frac{S}{R} \cdot (2^{K-1}) \]

TCP Delay Modeling (4)

Recall \( K \) = number of windows that cover object

How do we calculate \( K \)?

\[ K = \min \{ k : 2^k S + 2^{k-1} S + \ldots + 2 \cdot S \geq O \} \]
\[ = \min \{ k : 2^k \geq \frac{O}{S} + 1 \} \]
\[ = \left\lceil \log \left( \frac{O}{S} + 1 \right) \right\rceil \]

Calculation of \( Q \), number of idles for infinite-size object, is similar.
HTTP Modeling

- Assume Web page consists of:
  - 1 base HTML page (of size O bits)
  - M images (each of size O bits)

- Non-persistent HTTP:
  - M+1 TCP connections in series
  - Response time = \((M+1)O/R + (M+1)2RTT + \text{sum of idle times}\)

- Persistent HTTP:
  - 2 RTT to request and receive base HTML file
  - 1 RTT to request and receive M images
  - Response time = \((M+1)O/R + 3RTT + \text{sum of idle times}\)

- Non-persistent HTTP with X parallel connections
  - Suppose M/X integer.
  - 1 TCP connection for base file
  - M/X sets of parallel connections for images.
  - Response time = \((M+1)O/R + (M/X + 1)2RTT + \text{sum of idle times}\)

HTTP Response time (in seconds)

RTT = 100 msec, O = 5 Kbytes, M=10 and X=5

For low bandwidth, connection & response time dominated by transmission time.
Persistent connections only give minor improvement over parallel connections.

HTTP Response time (in seconds)

RTT = 1 sec, O = 5 Kbytes, M=10 and X=5

For larger RTT, response time dominated by TCP establishment & slow start delays. Persistent connections now give important improvement, particularly in high delay/bandwidth networks.