Cábula RC

15 de Abril de 2011

Conteúdo

1 Contas

1.1 Notação

- **Tempo de Transmissão:** Tempo que uma máquina demora a colocar um ficheiro ou pacote num canal. Depende do tamanho do ficheiro ou pacote, e da velocidade de transmissão.
- **Velocidade de Transmissão:** Número de unidades de tamanho colocadas num canal numa determinada unidade de tempo (Ex: bits por segundo, Mbs por segundo, Mbs por milissegundo, etc).
- **Tamanho de um ficheiro (ou pacote):** Número de bits, ou bytes, etc, num determinado ficheiro ou pacote.
- **Tempo de Propagação:** Tempo que um bit demora a percorrer o canal de uma ponta à outra. Depende do tamanho do canal e da velocidade de propagação.
- **Velocidade de Propagação:** Unidades de distância percorridas por um bit numa determinada unidade de tempo (Ex: kms por segundo, kms por milissegundo, metros por segundo, etc).
- **Tamanho de um canal:** Distância entre as duas máquinas que partilham esse canal.
- **Volume de um canal:** Número de bits que podem estar dentro do canal ao mesmo tempo, perante uma determinada velocidade de transmissão de um emissor, e o tempo de propagação do canal.

1.2 Fórmulas Úteis

- Tempo de Transmissão, Velocidade de Transmissão e Tamanho do Ficheiro (ou pacote):

$$T_T = \frac{D_F}{V_T}$$

$$D_F = T_T \cdot V_T$$

$$V_T = \frac{D_F}{T_T}$$

- Tempo de Propagação, Velocidade de Propagação e Tamanho de um Canal:

$$T_P = \frac{Tam_C}{V_P}$$

$$Tam_C = T_P \cdot V_P$$

$$V_P = \frac{Tam_C}{T_P}$$

- Quociente aproximado das velocidades médias de transferência de A e B (sendo A e B tempos totais de transferência):

$$Quociente = \frac{\frac{D_F}{T_A}}{\frac{D_F}{T_B}} = \frac{D_F \cdot T_B}{D_F \cdot T_A}$$

- Calcular a que distância está o primeiro bit quando o último bit entra no canal, ou seja, em T_T tempo, que distância é percorrida pelo primeiro bit:

  Sabemos que $T_P = \frac{Tam_C}{V_P}$. Isto indica-nos que, a uma velocidade V_P, o primeiro bit percorre uma distância Tam_C (tamanho do canal), em T_P tempo. Esta equação pode ser transformada para

sabermos qual a distância percorrida em T_P, sabendo que vamos a uma velocidade V_P ($Tam_C = T_P \cdot V_P$). Sendo que nós queremos saber qual a distância percorrida pelo primeiro bit em T_T, basta substituirmos na equação:

$$Dist = T_T \cdot V_P$$

- Velocidade de transmissão a partir do número de pacotes por segundo, e do tamanho de cada pacote:

$$V_T = (\#Pacotes/s) \cdot Dim_P$$

- Volume do canal:

$$Vol_C = V_T \cdot T_P$$

2 UDP Code

```
// Import InetAddress, InetSocketAddress, DatagramSocket
import java.net.* ;

// General stuff
InetAddress address = InetAddress.getByName(<String ip>) ;
InetSocketAddress sockaddr = InetSocketAddress(<InetAddress ip>, <int port>) ;

InetAddress address = sockaddr.getAddress() ;
int port = sockaddr.getPort() ;

//SEND THROUGH SOCKET
DatagramSocket socket = new DatagramSocket() ;

//Message to packet
String msg = "message" ;
byte[] strData = msg.getBytes() ;

DatagramPacket sndpacket = new DatagramPacket(strData, strData.length) ;
packet.setAddress(<InetAddress address>) ;
packet.setPort(<int port>) ;
//or
DatagramPacket sndpacket = new DatagramPacket(strData, strData.length, <
    InetAddress address>, <int port>) ;

//Send Packet
socket.send(<DatagramPacket sndpacket>) ;


//RECEIVE THROUGH SOCKET
DatagramSocket socket = new DatagramSocket(<int ownport>) ;

//Byte buffer to store message
byte[] buffer = new byte[65536] ;

//Create packet to receive message
DatagramPacket rcvpacket = new DatagramPacket(buffer, buffer.length) ;

//Receive from socket into packet
socket.receive(rcvpacket) ;

//Collect packet data
//Address and port
InetAddress fromaddress = rcvpacket.getAddress() ;
int fromport = rcvpacket.getPort() ;

//Message
String msg = new String(rcvpacket.getData(), 0, rcvpacket.getLength()) ;


//CLOSE SOCKET
socket.close() ;
```

3 TCP Code

```
// Import streams
import java.io.*;

//CLIENT
Socket socket = new Socket(<(InetAddress or String) server>, <int port>) ;

//Get output and input streams
OutputStream os = socket.getOutputStream() ;
InputStream is = socket.getInputStream() ;

//Send message
String msg = "message" ;
os.write(msg.getData()) ;

//Receive message
byte[] buf = new byte[65536] ;
os.read(buf) ;
String msg = new String(buf, 0, buf.length) ;

//Get n bytes from input stream
byte[] buf = new byte[n] ;
DataInputStream dis = new DataInputStream(is) ;
dis.readFully(buf) ; // reads a whole byte buffer from the input stream

//Close Socket
socket.close() ;

//SERVER
ServerSocket serverSocket = new ServerSocket(<int port>) ;
Socket clientSocket = serverSocket.accept() ;

//Each client has it's own output and input streams
```

4 Mix Code

4.1 Threads

```
// Import threads
import java.lang.* ;

// GENERAL STRUCTURE
public class <Class name> implements Runnable
{
        <Class name> (...)
        {
        }

        // the thread closes when the function run() ends
        public void run()
        {
        }

        ...
        Thread t = new Thread(this) ;
        t.setDaemon(true) ;
        // the thread starts when the function t.start() is called
        // the thread's execution goes to the function run()
        t.start() ;
        ...
        // The t.join() waits for the thread t to end,
        // to continue the main thread
        t.join() ;
}
// or
public class <Class name> extends Thread
{
        <Class name> (...)
        {
        }

        // the thread closes when the function run() ends
        public void run()
        {
        }

        ...
        <Class name> t = new <Class name>() ;
        t.setDaemon(true) ;
        // the thread starts when the function t.start() is called
        // the thread's execution goes to the function run()
        t.start() ;
        ...
        // The t.join() waits for the thread t to end,
        // to continue the main thread
        t.join() ;
}
```

4.2 Locks

```
// Import Locks
import java.util.concurrent.locks.*;

ReadWriteLock rwlock = new ReentrantReadWriteLock(true) ;

Lock wlock = rwlock.writeLock() ;
Lock rlock = rwlock.readLock() ;

// Locked writing
wlock.lock() ;
...
wlock.unlock() ;

// Locked reading
rlock.lock() ;
...
rlock.unlock() ;

// "The read lock may be held simultaneously by multiple reader
// threads, so long as there are no writers. The write lock is
// exclusive."
//(http://download.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/locks/
    ReadWriteLock.html)
```

4.3 Synchronized

```
// Synchronizes applying the lock to the current object
synchronized int method()
{
        ...
        return 0;
}

// Synchronizes applying the lock to <object>
int method()
{
        syncronized (<object>)
        {
                ...
        }
        return 0;
}
```

4.4 Data Structures

```
// Import Vector, Set
import java.util.* ;

// This is a hash set
Set< ... > set = new HashSet< ... >() ;

Vector< ... > vector = new Vector< ... >() ;

Iterator< <object> > it = <list/set/etc of objects>.iterator() ;

// This is a linked list
List< ... > lst = new LinkedList< ... >() ;

StringBuffer sb = new StringBuffer() ;
sb.append(char c) ;
sb.toString() ;
```