SOLUCIÓN - 1º parcial de Redes - ETSINF - 25 de enero de 2010

Apellidos, Nombre:_____

Grupo de matrícula:_____

1. a) Utilizando un cliente ftp en modo texto como el de prácticas se quiere: abrir una sesión con un servidor FTP, descargar tres ficheros almacenados en él y, a continuación, cerrar la sesión. ¿Cuántas conexiones TCP serán necesarias? ¿De qué tipo será cada una? Indica para qué se emplea cada tipo de conexión. ***(0,5 puntos)***

   b) Si los tres ficheros se encuentran almacenados en el mismo directorio del servidor y disponibles a través de un servidor HTTP, ¿cuántas conexiones TCP se requerirían para descargarlos? Justifica la respuesta, teniendo en cuenta las distintas posibilidades que hay (a nivel de HTTP). ***(0,5 puntos** – 0,25 por versión de HTTP**)***

a) Se necesitan cuatro conexiones: una de control para el diálogo con el servidor y tres de datos, una por cada fichero que hay que descargarse del servidor.

b) Depende de que las conexiones sean persistentes o no. Con conexiones persistentes (comportamiento por defecto en HTTP/1.1) una única conexión sería suficiente. Si la conexión es no persistente (comportamiento por defecto en HTTP/1.0) y el servidor cierra la conexión después de enviar el objeto solicitado se necesitarían tres. Una para descargarse cada uno de los ficheros.

2. Un servidor SMTP ejecuta la sentencia "`out.print("354 Enter mail, end with '.' on a line by itself" + "\r\n");`" habiendo ejecutado previamente la sentencia "`PrintWriter out = new PrintWriter(s.getOutputStream());`". Suponiendo que s es un *socket* conectado con el cliente, y que el cliente está esperando la respuesta del servidor para enviarle un correo electrónico, ¿qué efecto tendría ejecutar la primera sentencia? ***(0,25 puntos)*** En caso de que exista algún problema en el diálogo, indica que habría que modificar (o añadir) para que funcione correctamente.***(0,5 puntos** – una respuesta de las dos posibles es suficiente**)***

Al ejecutarse la sentencia "out.print ..." el mensaje del servidor se almacenaría en el *buffer* TCP a la espera de ser enviado. Los datos que hay en el *buffer* se envían en dos circunstancias: cuando el *buffer* se llena o cuando la aplicación solicita su envío. Puesto que el *buffer* se habría vaciado al enviar la respuesta anterior del servidor, estos datos serían los únicos actualmente almacenados y el *buffer* no estaría lleno. Para provocar su envío la aplicación debe solicitarlo expresamente para lo que tiene dos posibilidades:

a) Añadir a continuación la sentencia "**`out.flush();`**".

b) Definir el objeto PrintWriter mediante la sentencia "`PrintWriter out = new PrintWriter(s.getOutputStream(), true);`" y emplear después el método println de la clase PrintWriter: "`out.println("354 Enter mail, end with '.' on a line by itself");`"

(Como el `println` ya añade un final de línea hemos eliminado la parte "`\r\n`" de la sentencia "`out.println...`").

3. a) ¿Por qué cuando nos conectamos a un servidor Web mediante telnet tras teclear "GET / HTTP/1.0", debemos pulsar dos veces la tecla INTRO para obtener una respuesta del servidor? ***(0,75 puntos)***

   b) Indica la primera línea de la respuesta del servidor Web (línea de estado) suponiendo que el recuso solicitado se encuentra disponible en el servidor. ***(0,25 puntos –*** *una respuesta de las dos posibles es suficiente****)***

a) El formato de los mensajes HTTP es:

Línea Inicial

Cabeceras

Línea en blanco

Cuerpo

La petición de página web que vamos a enviar al servidor Web deberá seguir este formato. Por ello, dado que se trata de HTTP 1.0 no se exige la existencia de ninguna cabecera, y dado que la orden GET no requiere el envío de ninguna información al servidor, nuestro mensaje no tendrá cuerpo. Por todo ello, nuestra petición sólo estará compuesta por la linea inicial: "GET / HTTP/1.0", y por la línea en blanco.

La primera vez que pulsamos la tecla INTRO, enviamos al servidor la orden "GET / HTTP/1.0", y al pulsar nuevamente la tecla INTRO enviamos una línea en blanco.

b) Existen dos primeras líneas de respuesta posibles: "HTTP/1.0 200 OK" o "HTTP/1.1 200 OK"

4. ***(1 punto)*** Explica de forma breve cómo funciona la opción SACK de TCP.

Cuando un receptor TCP emplea la técnica de retransmisión selectiva, es decir, almacena los segmentos recibidos fuera de orden, no puede reconocer estos segmentos mediante el mecanismo de ACK convencional como consecuencia de los ACK acumulativos, y por tanto estos segmentos serán retransmitidos cuando venza su TimeOut. Mediante la opción SACK el receptor puede indicar al emisor la recepción correcta de estos segmentos, indicando expresamente qué bloques de datos se encuentran ya en la ventana de recepción.

Ejemplo opcional: el emisor envía una serie de cinco segmentos cada uno con 1.000 bytes de datos y números de secuencia 1.000, 2.000, 3.000, 4.000 y 5.000. De los cinco, el receptor recibe correctamente sólo el primero, el tercero y el quinto. Tras recibir el primer segmento con el número de secuencia 1.000, y suponiendo que no hay segmentos anteriores pendientes de ser recibidos podría generar un segmento con ACK=2.000. Si a continuación recibe el tercero, podría generar un segmento con ACK = 2.000 (sigue esperando el segundo segmento) y SACK = 3.000- 4.000. Tras recibir el quinto, podría enviar ACK = 2.000, SACK = 5.000-6.000, 3.000-4.000.

5. ***(1,5 puntos)*** Refleja en la siguiente tabla la evolución del **umbral** y de las **ventanas de congestión y transmisión** con respecto al tiempo (en RTT's), teniendo en cuenta los eventos que se indican (suceden durante el RTT, y se detectan al final del mismo). Todos los valores se expresan en segmentos. El valor de WIN indicado en cada RTT es válido desde el **principio** del mismo. El emisor siempre tiene nuevos datos que transmitir. No se emplean ACKs retrasados.

(**) = recepción de 3 ACK's duplicados (*) = TimeOut

					**					*						**					
RTT	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
WIN	64	62	59	52	47	45	53	43	32	16	16	32	32	16	13	10	8	9	6	5	3
Umbral	64	64	64	64	64	16	16	16	16	16	8	8	8	8	8	8	5	5	5	5	5
Vcong	2	4	8	16	32	16	17	18	19	20	1	2	4	8	9	10	5	6	7	8	9
Vtrans	2	4	8	16	32	16	17	18	19	16	1	2	4	8	9	10	5	6	6	5	3

Indica qué algoritmos para el control de la congestión actúan en cada tramo y explica cómo se calculan los nuevos valores de Umbral, Vcong y Vtrans tras la recepción de 3 ACKs duplicados en el RTT 5 y tras el vencimiento de un temporizador en el RTT 10.

RTTs 1-5 y 11-14: arranque lento (*slow-start*).

RTTs 6-10 y 15-21: incremento aditivo.

El nuevo valor del Umbral en los tres casos se calcula como Umbral = max(2,Vtrans/2), tomando el valor de Vtrans en los RTTs 5 y 10. Es decir:

a) Nuevo valor de Umbral en el RTT 6, Umbral_RTT6 = 32/2 = 16.

b) Nuevo valor de Umbral en el RTT 11, Umbral_RTT11 = 16/2 = 8.

Tras recibir 3 ACKs duplicados Vcong = Umbral. Por lo tanto, Vcong_RTT6 = 16.

Tras vencer un temporizador Vcong = 1. Por lo tanto, Vcong_RTT11 = 1.

Por último, Vtrans = min(WIN, Vcong).

Vtrans_RTT6 = min(45,16) = 16

Vtrans_RTT11 = min(16,1) = 1

6. Justifica brevemente la utilidad del GET condicional ***(0,5 puntos)***, indicando claramente sus ventajas frente al GET estándar ***(0,25 puntos)***.

El GET condicional se emplea cuando el cliente ya dispone en su caché de una copia del recurso solicitado. Antes de mostrarlo al usuario, es necesario confirmar que dicho recurso no ha sido modificado recientemente. Para ello se envía una solicitud GET condicional indicando la fecha de modificación del recurso disponible en caché local, mediante la cabecera IF_MODIFIED_SINCE. En el mejor de los casos – el objeto está actualizado – el cliente sólo tendrá que esperar una respuesta del tipo 304 NOT MODIFIED, ahorrándose la transmisión del recurso. En el peor de los casos – si el objeto está obsoleto – el servidor enviará el recurso al cliente de forma análoga a un GET convencional.

7. ***(2 puntos)*** Implementa en Java un servidor de nombres iterativo sobre TCP (llámalo ServidorNombres) que escuche en el puerto 5001. Al conectarse con un cliente, el servidor recibirá una línea que contenga un nombre de dominio y devolverá un *string* con la dirección IP correspondiente en formato decimal separado por puntos o un mensaje de error en caso de que el nombre no tenga dirección IP asociada. Para realizar la traducción se servirá de la clase InetAddress.

Algunos métodos de la clase InetAddress:

`static InetAddress` **`getByAddress`**`(byte[] addr)throws UnknownHostException`

Devuelve un objeto `InetAddress` a partir de su dirección IP formato de bytes.

`static InetAddress` **`getByAddress`**`(String host, byte[] addr)`

Crea un `InetAddress` basado en el nombre de host proporcionado y su dirección IP.

`static InetAddress` **`getByName`**`(String host) throws UnknownHostException`

Determina la dirección IP de un host, dado el nombre del host.

`String` **`getHostAddress`**`()`

Devuelve la dirección IP en formato string.

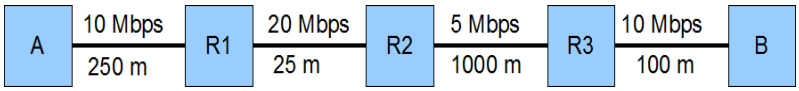
`String` **`getHostName`**`()`

Devuelve el nombre de host para esta dirección IP, o si la operación no está permitida por el control de seguridad, la representación en formato texto de la dirección IP.

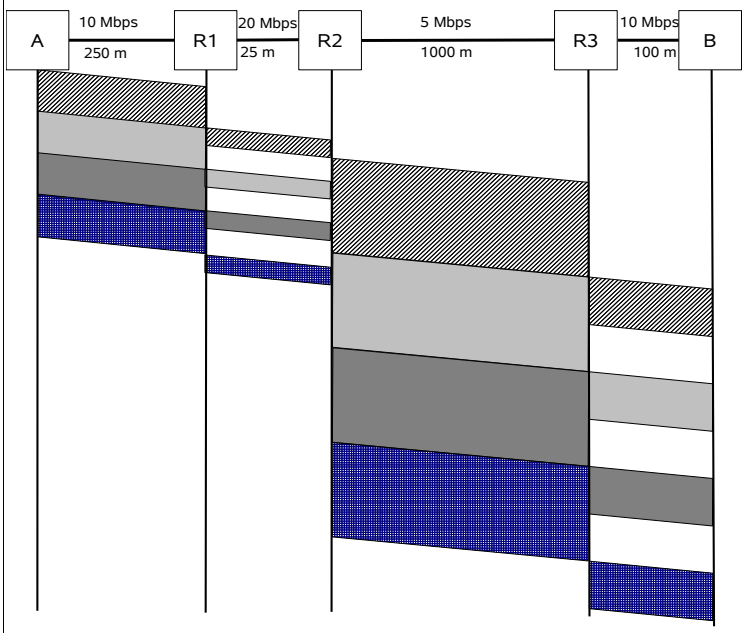
```
import java.net.*; import java.io.*; import java.util.*;

class ServidorNombres {
  public static void main(String args[]) throws Exception {
    ServerSocket ss=new ServerSocket(5001);
    while(true) {
      Socket s=ss.accept();
      PrintWriter salida=new PrintWriter(s.getOutputStream(),true);
      Scanner entrada = new Scanner(s.getInputStream());
      try{
       salida.println(InetAddress.getByName(entrada.nextLine()).getHostAddress());
      } catch(UnknownHostException e) {
        salida.println("ERROR: nombre desconocido");
      }
      s.close();
    }
  }
}
```

Desglose puntuación: 1 punto por la estructura del servidor (ServerSocket, accept, while(true), PW, Scanner, close).
1 punto por el servicio ofrecido (getByName, getHostAddress, tratamiento excepciones).

8. El computador A desea transmitir un mensaje de 500 bytes al computador B, atravesando la red de conmutación de paquetes del dibujo. El tamaño máximo de paquete es de 1000 bits, considerándose despreciable el tamaño de las cabeceras. Suponiendo también despreciable el tiempo de proceso en los *routers*, una velocidad de propagación de $2{,}5 \times 10^8$ m/s y que no existe otro tráfico en ese momento en la red:
   a) Representa en un esquema gráfico el tránsito de los paquetes desde A hasta B a través de los tres *routers*. ***(0,75 puntos)***
   b) Calcula el tiempo transcurrido desde el comienzo de la transmisión en A hasta que el mensaje se encuentra disponible en B. ***(0,5 puntos + 0,75 puntos)***



En primer lugar, para la transmisión de 500 Bytes = 4000 bits serán necesarios cuatro paquetes de 1000 bits. Gráficamente:



Como puede observarse, los cuatro paquetes salen de forma consecutiva de A. En R1 la línea de salida (R1-R2) queda disponible antes de que se reciba el siguiente paquete, por lo que se intercala un espacio de inactividad de línea entre paquetes. Cuando los paquetes llegan a R2, como consecuencia de la baja velocidad de transmisión, cada paquete debe esperar a que finalice la transmisión de los anteriores, por lo que el tiempo en colas **no es despreciable** en este caso.

Puesto que la línea R3-B tiene una mayor velocidad de transmisión que la R2- R3, volverá a producirse el mismo efecto que en la línea R1-R2, por lo que la llegada a B **no será consecutiva**. La tasa de llegadas a B (tiempo entre dos paquetes consecutivos) será el tiempo de transmisión de la línea más lenta, en este caso R2-R3.

Teniendo en cuenta que los tiempos de propagación también son diferentes en todas las líneas, la recepción completa del primer paquete en B (que no espera en colas) costaría:

$\mathbf{T_1 = Ttrans_{A\text{-}R1} + Tprop_{A\text{-}R1} + Ttrans_{R1\text{-}R2} + Tprop_{R1\text{-}R2} + Ttrans_{R2\text{-}R3} + Tprop_{R2\text{-}R3} + Ttrans_{R3\text{-}B} + Tprop_{R3\text{-}B}}$

Donde:

$Ttrans_{A\text{-}R1} = Ttrans_{R3\text{-}B} = 1000 \text{ bits} / 10^7 \text{ bits/s} = \mathbf{0{,}1 \text{ ms}}$

$Ttrans_{R1\text{-}R2} = 1000 \text{ bits} / 2 \times 10^7 \text{ bits/s} = \mathbf{0{,}05 \text{ ms}}$

$Ttrans_{R2\text{-}R3} = 1000 \text{ bits} / 5 \times 10^6 \text{ bits/s} = \mathbf{0{,}2 \text{ ms}}$

$Tprop_{A\text{-}R1} = 250 \text{ m} / 2{,}5 \times 10^8 \text{ m/s} = 1 \times 10^{-6} = \mathbf{1 \text{ µs}}$

$Tprop_{R1\text{-}R2} = 25 \text{ m} / 2{,}5 \times 10^8 \text{ m/s} = 1 \times 10^{-7} = \mathbf{0{,}1 \text{ µs}}$

$Tprop_{R2\text{-}R3} = 1000 \text{ m} / 2{,}5 \times 10^8 \text{ m/s} = 4 \times 10^{-6} = \mathbf{4 \text{ µs}}$

$Tprop_{R3\text{-}B} = 100 \text{ m} / 2{,}5 \times 10^8 \text{ m/s} = 4 \times 10^{-7} = \mathbf{0{,}4 \text{ µs}}$

Luego $\mathbf{T_1}$= 0,1 ms + 0,05 ms + 0,2 ms + 0,1 ms + 1 µs + 0,1 µs +4 µs + 0,4 µs = 0,4555 ms = **455,5 µs**

A partir de este instante, el segundo paquete llega a B un tiempo equivalente a $Ttrans_{R2\text{-}R3}$ (0,2 ms = 200 µs) y lo mismo pasa con el tercero y cuarto. Así, el tiempo total será

Ttotal = $T_1 + 3 \times 200$ µs **= 1055,5 µs**