Segundo Certamen Tiempo 90 [min]
Responder un problema por página

1.- (35 puntos) La transmisión TCP de una radio en Internet no da abasto en su servidor único (maestro) cuando el número de clientes es excesivo. Para aliviar este problema se le piede crear varios servidores replicadores y subdividir los clientes. Un servidor replicador espera por la llegada de clientes. Tan pronto llega el primero, se conecta al servidor maestro y transfiere al cliente todos los datos llegados desde el maestro. Cuando un nuevo cliente contacta al servidor replicador y éste aún atiende al previo, el replicador copia el contenido enviado por el maestro hacia ambos clientes. Cuando se van todos los clientes, el replicador se desconecta del maestro. Suponga que no existen mensajes del cliente al servidor replicador. La sintaxis pedida es:
$ replicador <Servidor Maestro> <puerto maestro> <puerto atención clientes>
Desarrolle un programa en C que use select (no hebras) para programar el servidor replicador.

```
void main( int argc, char * argv[] ) {
   int localSock, masterSock;
   struct sockaddr_in server, from;
   int fromlen;
   fd_set readfds, readfdsCopy;
   int n,i, rc, masterPort, localPort;
   int sockList[64], lastSock=0;
   struct  hostent *hp;
   char buf[512];

   masterPort=htons(atoi(argv[2]));
   localPort=htons(atoi(argv[3]));
  /* prepare welcome socket */       5 pts
   server.sin_family = AF_INET;
   server.sin_addr.s_addr = htonl(INADDR_ANY);
   server.sin_port = localPort;
   localSock = socket (AF_INET,SOCK_STREAM,0);
   bind(localSock, (struct sockaddr *)&server, sizeof(server));
   listen(localSock,4);
   /* prepare conection to master */     5pts
   if  ((hp = gethostbyname(argv[1])) == NULL) {
      server.sin_addr.s_addr = inet_addr(argv[1]);
      if ((hp = gethostbyaddr((void *)&server.sin_addr.s_addr,
                  sizeof(server.sin_addr.s_addr),AF_INET)) == NULL) {
         fprintf(stderr, "Can't find host %s\n", argv[1]);
         exit(-1);
      }
   }
   memcpy(&server.sin_addr, hp->h_addr_list[0], hp->h_length);
   server.sin_port = masterPort;
   fromlen = sizeof(from);
   FD_ZERO(&readfdsCopy);                          Manejo de fd set y select 8pts
   FD_SET(localSock, &readfdsCopy); /* initially attend welcome socket only */
   for(;;){
      memcpy(&readfds, &readfdsCopy, sizeof(fd_set));
      n = select(FD_SETSIZE, &readfds, (fd_set *) 0, (fd_set *) 0, NULL);
      if (n > 0) {
         if (FD_ISSET(localSock, &readfds)) { /* new client */    aceptar cliente 9pts
            sockList[lastSock++] = accept(localSock, (struct sockaddr *)&from, &fromlen);
            FD_SET(sockList[lastSock-1], &readfdsCopy);
            if (lastSock==1) {                  /* first client => connect to master */
               connect(masterSock, (struct sockaddr *)&server, sizeof(server));
```

```
                FD_SET(masterSock, &readfdsCopy);
            }
        }
        if (FD_ISSET(masterSock, &readfds)) { /* is there data from master? */
            rc=read(masterSock, buf, sizeof(buf));    Distribuir datos 8pts.
            for (i=0; i < lastSock; i++)            /*replicate to all clients */
                write(sockList[i], buf, rc);
        }
        for (i=0; i < lastSock; i++) {
            if (FD_ISSET(sockList[i], &readfds)) {       /* client is closing */
                FD_CLR(sockList[i], &readfdsCopy);  /* remove it from slect */
                close (sockList[i]);
                sockList[i] = sockList[--lastSock];
                i--;
                if (lastSock==0){      /* the last client just left => close master*/
                    FD_CLR(masterSock, &readfdsCopy);
                    close (masterSock);
                }
            }
        }
    }
}
```

2.- (35 puntos) Contexto: Una aplicación TCP tiene un conjunto de servidores distribuidos en Internet para así atender a sus clientes con el servidor más cercano a ellos. Para identificar tales servidores, el cliente se conecta al maestro y obtiene de él la lista de IPs de los servidores. Luego el cliente determina en número de saltos a cada servidor, el tiempo viaje de ida y vuelta (RTT: Round-trip time), y se conecta con cada uno de ellos para recibir su porcentaje de carga. Con estos tres valores el cliente decide a quien solicitar el servicio. En esta pregunta se le pide programar un avance del cliente mencionado.

Pregunta: Haga un programa en Java que a partir de un conjunto de servidores definidos por la línea de comandos, imprima por pantalla una línea por servidor:
<IP del servidor> <número de saltos> <Un RTT> <% de carga del servidor>
El cliente es invocado usando:
$ java MyClient <Puerto de los servidores> <IP servidor 1> <IP servidor 2> … <IP servidor N>
Nota: Un segmento de la salida de traceroute es:

```
$ traceroute www.altavoz.com
traceroute to www.altavoz.com (74.117.220.30), 30 hops max, 60 byte packets
 1  wifi1.elo.utfsm.cl (192.168.1.1)  0.291 ms  0.240 ms  0.181 ms
 2  elo-gw.elo.utfsm.cl (200.1.17.1)  2.247 ms  3.033 ms  2.947 ms
 :
```

Para leer un RTT puede usar:

```
Scanner s = new Scanner (<linea>);  // en <linea> ponga una instancia de String.
    s.next(); /* número */  s.next(); /* máquina */ s.next(); /* IP */ int rtt=s.nextInt();
```

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

public class MyClient {
   public static void main(String[] args) {
      int serverPort = Integer.parseInt(args[0]);
      int hopNumber=0, percentage=0;
```

```
        float rtt=0.0F;
        Scanner s;
        for (int i=1; i< args.length; i++) {   5pts.
             /* rtt and hop number */
           hopNumber = 1;
           Runtime runTime= Runtime.getRuntime();    Ejecución de traceroute 9 pts.
           Process process=null;
           try {
              process = runTime.exec("traceroute " + args[i]);
              BufferedReader in = new BufferedReader(
                 new InputStreamReader(process.getInputStream()));
              String lastLine=null, line;
              while ((line=in.readLine())!= null)
                 lastLine = line;
              s = new Scanner (lastLine);
              hopNumber = s.nextInt(); /* read hop number */
              s.next(); /* skip maquina */
              s.next(); /* skip IP */
              rtt = s.nextFloat();                    Lectura de datos en última línea 8 pts.
           } catch (IOException e) {
              System.out.println("traceroute error");
              System.exit(-1);
           }
           if (process!=null)    process.destroy();
           try {
              process.waitFor();
           } catch( InterruptedException e ) {  System.exit(-1);   }
           try {  /* get percentage */                    9 pts
              Socket sock = new Socket(args[i], serverPort);
              s = new Scanner (sock.getInputStream());
              percentage = s.nextInt();
           } catch (IOException e){  System.exit(-1);     }
           System.out.println (args[i]+ " " + hopNumber + " " + rtt + " " + percentage); 4pts
        }
     }
  }
```

3.- (30 puntos) Se desea que instancias de la clase SynchronizedRGB puedan ser manipuladas por vario hilos en Java.

a) ¿Qué cambios debe agregar a esta clase para evitar inconsistencias cuando varias hebras manipulen un mismo objeto SynchronizedRGB?

b) Un uso posible de esta clase se encuentra en el segmento:

```
SynchronizedRGB color = new SynchronizedRGB(0, 0, 0, "Pitch Black");
...
int myColorInt = color.getRGB();
String myColorName = color.getName();
```

¿Puede ocurrir algún problema de consistencia? Si no la puede haber, explique; si puede haber indique cómo se corrige.

c) Se desea asociar un identificador (id) único a cada instancia. Para eso saque los comentarios e indique si es preciso hacer cambios y cuáles para no generar inconsistencias.

a) 10 pts Debemos asegurar accesos exclusivos a cada método de objeto. Para esto agregamos synchronized a cada método.

```
public class SynchronizedRGB {
    private int red;
    private int green;
    private int blue;
    private String name;
//    private int id;
//    private static int nextId=0;
    public SynchronizedRGB(int red, int green, int blue, String name) {
        this.red = red;
        this.green = green;
        this.blue = blue;
        this.name = name;
//        id=nextId;
//        nextId++;
    }
    public synchronized void set(int red, int green, int blue, String name) {
            this.red = red;
            this.green = green;
            this.blue = blue;
            this.name = name;
    }
    public synchronized int getRGB() {
        return ((red << 16) | (green << 8) | blue);
    }
    public synchronized String getName() {
        return name;
    }
}
```

b) 10 pts. Sí, puede haber inconsistencia cuando otra hebra cambia el estado de color entre un llamado y el otro. Obtendremos el RGB del primero y el nombre del segundo.
Para evitar esto debemos acceder a las dos instrucciones en forma exclusiva y controlado con el mismo candado usado para cada uno de los métodos.

```
synchronized(color) {
  int myColorInt = color.getRGB();
  String myColorName = color.getName();
}
```

c) 10 pts. Al sacar los comentarios nada impide que dos hebras creen instancias de dos colores, esto puede generar inconsistencia en el acceso a nextId. Por ejemplo si luego de asignar su valor a id, accede la otra hebra tomando el mismo valor para nextId que la previa. Para esto se debe asegurar acceso exclusivo a la secuencia. Para ello debemos usar el candado asociado a la clase.

id = nextId;
nextId++;
	synchronized(SynchronizedRGB.class) { /* pseudocódigos similares son OK */
	id = nextId;
	nextId++;
	}