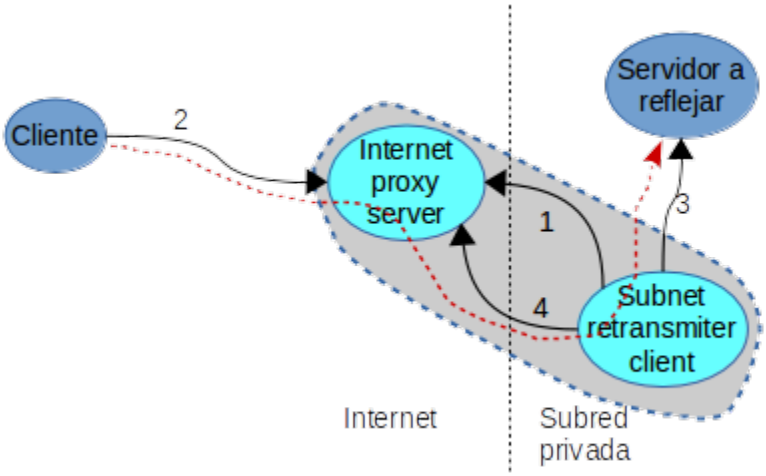
Segundo Certamen
Tiempo 90 min. Responder un problema por página

1.- En la tarea 3 usted desarrolló programas para Internet proxy server (ips) y Subnet retransmiter client de la figura adjunta. El primero se ejecutó usando:
ips <puerto_servicio_reflejado> <puerto_tunel>



Un aspecto a mejorar de ips es la forma para terminar su ejecución. Buscar su identificador de proceso (pid) de la lista de procesos y luego ejecutar comando kill genera un corte abrupto de los servicios siendo atendidos vía el túnel. Para lograr un término cortés, es decir, esperar la partida de todos los clientes en curso y luego terminar el proceso ips, se incorpora un nuevo puerto de escucha. Cuando a éste se conecta un cliente **ejecutado en la misma máquina** (127.0.0.1), ips terminará cuando todos los clientes pendientes hayan partido.
Modifique el código disponible en http://profesores.elo.utfsm.cl/~agv/elo330/2s17/C2/p1
para aceptar un nuevo puerto como parámetro en ips y poner término del servicio de manera cortés. Ips no requiere recibir datos en este puerto nuevo, la conexión a él es entendida como petición del cierre cortés del servicio proxy.
En su respuesta use nombre de archivo y la numeración de las líneas del código dado para indicar dónde usted incluye y/o modifica el código.

Elementos a evaluar:

** Debe definir variable global para controlar término de thread principal*	*5*
** El acceso a la variable global debe ser controlado por mutex:*	
- definición global del mutex	*5*
** Definición de hebra servidora para atender conexiones para shutdown:*	
- Creación de socket servidor	*7*
- accept	*5*
- Cambio de variable de control para cierre cortés.	*7*
- Mecanismo para sacar a thread principal su accept bloqueante.	*7*
** En hebra principal (main): crear hebra shutdown*	*7*
** En hebra principal: lógica de término con variable de control.*	*7*

```
#include <stdio.h>  /* printf */
#include <stdlib.h>   /* exit */
#include <unistd.h>   /* write  */
#include <sys/socket.h>  /* accept, inet_ntoa, recv  */
#include <netinet/in.h>  /* inet_ntoa  */
#include <arpa/inet.h>   /* inet_ntoa  */
#include <pthread.h>     /* pthread_create */

8 #include "tcp_util.h"
```

```
/* new */
int done=0;
int serverPort;
pthread_mutex_t  doneLock = PTHREAD_MUTEX_INITIALIZER;

void * shutdown_server(void * arg) {
   struct sockaddr_in name;
   int shutdown_s = createTCPserverSocket(atoi((char *) arg));
   int cs, len;
   while (1) {
     cs = accept(shutdown_s, (struct sockaddr *) &name, &len);
     if (strcmp(inet_ntoa(name.sin_addr), "127.0.0.1")) break;
     close(cs);
   }
   pthread_mutex_lock(&doneLock);
     done = 1;
   pthread_mutex_unlock(&doneLock);
   cs=createTCPclientSocket("localhost", serverPort);  /* to unblock accept */
}

int isDone () {
   int state;
   pthread_mutex_lock(&doneLock);
     state = done;
   pthread_mutex_unlock(&doneLock);
   return state;
}
/** end new */

9  int main(int argc, char * argv[])  {
     int intServerSocket, tunnelServerSocket, relayClientSocket, len;
     int * nc;
     struct sockaddr_in name;
     pthread_t threadID;

     if (argc!=4) {
        printf("Usage: %s  <puerto_servicio_reflejado> <puerto_tunel> <shutdown_port>\n", argv[0]);
        exit(-1);
17    }
     serverPort = atoi(argv[1]);   /* new line  */
18    /* Create the socket. */
     intServerSocket = createTCPserverSocket(atoi(argv[1]));
     tunnelServerSocket = createTCPserverSocket(atoi(argv[2]));
     relayClientSocket = accept(tunnelServerSocket,(struct sockaddr *) &name, &len);
22    printf("Connection from : %s\n",inet_ntoa(name.sin_addr));
/*  New  */
     pthread_create(&threadID, NULL, shutdown_server, argv[3]);
/* end New  */

23    while(1) {
24       int fd;

       if (!isDone()) {        /*new line*/

25           nc = calloc (2, sizeof(int));
       /* Accept a connection.  */
27           nc[0] = accept(intServerSocket, (struct sockaddr *) &name, &len);

       }  else break;      /* new line */
       if (isDone())  break;   /* new line */

28     printf("New connection from a client looking for the server %s\n",inet_ntoa(name.sin_addr));
       write(relayClientSocket, &nc[0], sizeof(int));
       nc[1] = accept(tunnelServerSocket, (struct sockaddr *) &name, &len);
       printf("Connection from subnet retransmit client: %s\n",inet_ntoa(name.sin_addr));
       read(nc[1],&fd, sizeof(int));
       if(fd==nc[0]) {
           printf("Realy to make the logic connection betwen fd=%i and fd=%i.\n",nc[0], nc[1]);
           pthread_create(&threadID, NULL, TCPRelay, nc);
       } else {
```

```
            printf("Protocol error.... exiting...\n");
            close(nc[0]);
            close(nc[1]);
            free (nc);
            break;
        }
    }
    close(relayClientSocket);
    close(intServerSocket);
    close(tunnelServerSocket);
    exit(0);
}
```

2.- El servidor de eco ubicado en http://profesores.elo.utfsm.cl/~agv/elo330/2s17/C2/p2/ atiende a un único cliente, cuando éste cierra la conexión o envía "BYE", el servidor eco termina. Se pide incluir el puerto de servicio 12345 a través del cual un cliente puede hacer llegar comandos "pause" y "resume" (ambos strings). Cuando llega el comando "pause", el servidor de eco envía a lo más un eco adicional y suspende su trabajo hasta la llegada del comando "resume". Este ciclo se puede repetir varias veces con comandos enviados por el mismo cliente.
Haga los cambios necesarios al servidor de eco en Java proporcionado para permitir pausar y reanudar sus ecos.
En su respuesta use la numeración de las líneas del código dado para indicar dónde usted incluye y/o modifica el código.

Elementos a evaluar:
** Creación e inicio de hebra para atender puerto 12345 10*
** Código en hebra principal (main)*
- para consultar por estado de pausa y 6
- espera sincronizada para resumir. 6
** Implementación de hebra:*
- Constructor 6
- Método run para recepcionar y decodificar comandos 10
- Métodos para acceso sincronizado a variable de control pausa/resume del servicio. 12

```
import java.io.*;
import java.net.*;
/** This program implements a simple server that listens to
   port 8189 and echoes back all client input.
   It also listens to port 12345 and turns ON/OFF the echo service on arrival of
   "resume"/"pause" commands on this port.
*/
public class EchoServer  {
   public static void main(String[] args ) {
      try {
         // establish server socket
19         ServerSocket s = new ServerSocket(8189);
         PauseControl pc = new PauseControl();   /*   new line  */
         pc.start();    /* new line */

21         // wait for client connection
         Socket incoming = s.accept( );
         BufferedReader in = new BufferedReader
            (new InputStreamReader(incoming.getInputStream()));
         PrintWriter out = new PrintWriter
            (incoming.getOutputStream(), true /* autoFlush */);

         out.println( "Hello! Enter BYE to exit." );

         // echo client input
         boolean done = false;
32         while (!done) {
```

```
            if (pc.paused())     /* new code */
                synchronized(pc) {
                   pc.wait();
                }        /* new code */
              String line = in.readLine();
            if (line == null) done = true;
            else {
               out.println("Echo: " + line);

               if (line.trim().equals("BYE"))
                  done = true;
            }
         }
         incoming.close();
      }
      catch (Exception e){
         e.printStackTrace();
      }
   }
}

/*   new lines */
class PauseControl extends Thread {
   private boolean paused = false;
   private ServerSocket ss;
   public PauseControl () {
     try {
       ss = new ServerSocket(12345);
     } catch (Exception e) {
       e.printStackTrace();
     }
   }

   public void run() {
     try {
       Socket cs = ss.accept();
       BufferedReader in = new BufferedReader (
          new InputStreamReader(cs.getInputStream()));
       boolean done = false;
       while (!done) {
         String line = in.readLine();
         if (line == null) done = true;
         else {
           if (line.trim().equals("pause"))
              myPause();
           if (line.trim().equals("resume"))
              myResume();
         }
       }
       cs.close();
       ss.close();
     } catch (Exception e) {
        e.printStackTrace();
     }
   }

   private synchronized void myPause() {
     paused=true;
   }

   private synchronized void myResume() {
     paused=false;
     notify();
   }

   public synchronized boolean paused() {
     return paused;
   }
}
/* end new */
```