Segundo Certamen
Tiempo 11:30 hrs - 13:00 hrs.

1.- Abajo está el código de un servidor de eco simple. Se pide modificar simpleEcho.c para ofrecer dos servicios: uno en “puerto_eco” y otro en “puerto_monitor”. En el primer puerto el servidor se comporta igual como el código adjunto. Cuando un cliente se conecta al puerto_monitor, el servidor inmediatamente responde con el número de bytes total enviados como eco desde su inicio y luego cierra ese socket. El servidor atiende sólo un cliente monitor a la vez. El servidor termina cuando finaliza la atención del primer cliente eco. El segundo argumento del programa corresponde al puerto donde corre el servicio monitor (el primer argumento es el puerto del servicio eco).

Obs: i) Codificar la solución en lenguaje C. El diseño del programa puede usar cualquier mecanismo visto en el curso.
ii) Es OK suponer que todos los llamados al sistema son exitosos.

```
/* simpleEcho.c
 * Copyright 2016 Agustin Gonzalez
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <arpa/inet.h>

int main(int argc, char **argv) {
    char buf[10];
    int s, n, ns, len;
    struct sockaddr_in name;

    s = socket(AF_INET, SOCK_STREAM, 0);
    name.sin_family = AF_INET;
    name.sin_port = htons(atoi(argv[1]));
    name.sin_addr.s_addr = htonl(INADDR_ANY);
    len = sizeof(struct sockaddr_in);
    bind(s, (struct sockaddr *) &name, len);
    listen(s, 1);
    ns = accept(s, (struct sockaddr *) &name, &len);
    while ((n = recv(ns, buf, sizeof(buf), 0)) > 0)
        send(ns, buf, n, 0);
    close(ns);
    close(s);
    exit (0);
}
```

A continuación se presentan dos soluciones posibles, una usando threads y otra usando select.
Solución usando POSIX threads

```
/* Threaded_EchoMonitored.c
 * Copyright 2016 Agustin Gonzalez
 */
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <string.h>
#include <pthread.h>

int nbytes=0;
pthread_mutex_t   count_lock = PTHREAD_MUTEX_INITIALIZER;

static void * monitor(void *arg) {
    char buf[10];
    int sm, n, cm, len;
    struct sockaddr_in name;

    sm = socket(AF_INET, SOCK_STREAM, 0);
    name.sin_family = AF_INET;
    name.sin_port = htons(atoi((char *)arg));
    name.sin_addr.s_addr = htonl(INADDR_ANY);
    len = sizeof(struct sockaddr_in);
    bind(sm, (struct sockaddr *) &name, len);
    listen(sm, 1);
    while (1) {
      cm = accept(sm, (struct sockaddr *) &name, &len);
      pthread_mutex_lock( &count_lock);
      sprintf(buf, "%i", nbytes);
      pthread_mutex_unlock( &count_lock);
      send(cm, buf, strlen(buf), 0);
      close(cm);
    }
}

int main(int argc, char **argv){
    char buf[10];
    int se, n, ce, len;
    struct sockaddr_in name;
    pthread_t tid;

    pthread_create(&tid,NULL, monitor,argv[2]);

    se = socket(AF_INET, SOCK_STREAM, 0);
    name.sin_family = AF_INET;
    name.sin_port = htons(atoi(argv[1]));
    name.sin_addr.s_addr = htonl(INADDR_ANY);
    len = sizeof(struct sockaddr_in);
    bind(se, (struct sockaddr *) &name, len);
    listen(se, 1);
    ce = accept(se, (struct sockaddr *) &name, &len);
    while ((n = recv(ce, buf, sizeof(buf), 0)) > 0){
        send(ce, buf, n, 0);
        pthread_mutex_lock( &count_lock);
        nbytes+=n;
        pthread_mutex_unlock( &count_lock);
    }
    close(ce);
    close(se);
    exit (0);
```

```
}
```

Solución usando Select

```
/* Select_EchoMonitored.c
 * Copyright 2016 Agustin Gonzalez
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <string.h>

int main(int argc, char **argv) {
    char buf[10];
    int es, ms, n, ec, mc, len;
    struct sockaddr_in name;
    int nbytes = 0;
    fd_set readfds, readfdsCopy;

    es = socket(AF_INET, SOCK_STREAM, 0);
    name.sin_family = AF_INET;
    name.sin_port = htons(atoi(argv[1]));
    name.sin_addr.s_addr = htonl(INADDR_ANY);
    len = sizeof(struct sockaddr_in);
    bind(es, (struct sockaddr *) &name, len);
    listen(es, 1);

    ms = socket(AF_INET, SOCK_STREAM, 0);
    name.sin_family = AF_INET;
    name.sin_port = htons(atoi(argv[2]));
    name.sin_addr.s_addr = htonl(INADDR_ANY);
    len = sizeof(struct sockaddr_in);
    bind(ms, (struct sockaddr *) &name, len);
    listen(ms, 1);

    FD_ZERO(&readfdsCopy);
    FD_SET(es,&readfdsCopy);
    FD_SET(ms,&readfdsCopy);
    while (1) {
      memcpy(&readfds, &readfdsCopy, sizeof(fd_set));
      n = select(FD_SETSIZE, &readfds, (fd_set *) 0, (fd_set *) 0, NULL);
      if (n>0) {
        if (FD_ISSET(es, &readfds)) {
          ec = accept(es, (struct sockaddr *) &name, &len);
          FD_SET(ec, &readfdsCopy);
        }
        if (FD_ISSET(ec, &readfds)) {
          if ((n = recv(ec, buf, sizeof(buf), 0)) > 0) {
            send(ec, buf, n, 0);
            nbytes += n;
          } else break;
        }
        if (FD_ISSET(ms, &readfds)) {
          mc = accept(ms, (struct sockaddr *) &name, &len);
          sprintf(buf, "%i", nbytes);
          send(mc, buf, strlen(buf), 0);
          close(mc);
```

```
        }
      }
    }
    close(ms);
    close(ec);
    close(es);
    exit (0);
}
```

2.- Abajo está la implementación en lenguaje Java de un servidor eco concurrente. Es de interés conocer el número de clientes atendidos desde el inicio del servidor (se considera atendido tan pronto su conexión es aceptada). Se pide modificar ThreadedEchoServer.java para ofrecer dos servicios: uno en puerto_eco y otro en puerto_monitor. En el primer puerto el servidor se comporta igual como el código adjunto. Cuando un cliente se conecta al puerto_monitor, el servidor inmediatamente responde con el número de clientes eco atendidos desde el inicio del servidor y luego cierra la conexión. Otro cliente monitor se puede conectar una vez atendido el cliente monitor anterior. El programa termina cuando termina de atender al primer cliente eco. El servicio monitor se ofrece en puerto_monitor el cual es ingresado como segundo argumento (el puerto del servicio eco es el primero).

Obs: Es OK considerar que todos los llamados al sistema son exitosos.

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

/** ThreadedEchoServer.java
   This program implements a multithreaded server that listens to
   port 8189 and echoes back all client input.
*/
public class ThreadedEchoServer {
   public static void main(String[] args ){
      try{
         long i = 1;
         ServerSocket s = new ServerSocket(8189);
         for (;;){
            Socket incoming = s.accept( );
            System.out.println("Spawning " + i);
            Thread t = new ThreadedEchoHandler(incoming);
            t.start();
            i++;
         }
      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

```
/**
   This class handles one client input.
*/
class ThreadedEchoHandler extends Thread {
   public ThreadedEchoHandler(Socket i) {
      incoming = i;
   }

   public void run() {
      try {
         BufferedReader in = new BufferedReader
            (new InputStreamReader(incoming.getInputStream()));
         PrintWriter out = new PrintWriter
            (incoming.getOutputStream(), true /* autoFlush */);
         out.println( "Hello! Enter BYE to exit." );
         boolean done = false;
         while (!done) {
            String str = in.readLine();
            if (str == null) done = true;
            else  {
                out.println("Echo = "+str);
                if (str.trim().equals("BYE"))
                  done = true;
            }
         }
         incoming.close();
      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }
   private Socket incoming;
}
```

Obs.: Si su solución no hace terminar el programa con el término del primer cliente, no importa. Esto lo olvidé borrar de la pregunta 1 y no es importante para evaluar su dominio de hebras y acceso exclusivo.

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

/** ThreadedEchoServer.java
   This program implements a multithreaded server that listens to
   port given by two command line arguments.
   The server echoes back all client input on port given by first argument
   and monitored the number of echo clients on port given by seconf argument.
*/
class ClientCounter {
    public ClientCounter(){
        counter=0;
    }
    public synchronized void increment(){
        counter++;
    }
```

```
    public synchronized long getValue(){
        return counter;
    }
    private long counter;
}
public class ThreadedEchoServerMonitored {
   public static void main(String[] args ){
      try{
         ClientCounter i = new ClientCounter();
         new Monitor(i, Integer.parseInt(args[1])).start();
         ServerSocket s = new ServerSocket(Integer.parseInt(args[0]));
         for (;;){
            Socket incoming = s.accept( );
            i.increment();
            System.out.println("Spawning " + i.getValue());
            Thread t = new ThreadedEchoHandler(incoming, i.getValue());
            t.start();
         }
      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }
}

class Monitor extends Thread {
    public Monitor( ClientCounter i, int port) {
        this.i=i;
        this.port=port;

    }
    public void run (){
       try {
         ServerSocket ms = new ServerSocket (port);
         while(true) {
            Socket mc = ms.accept();
            PrintWriter out = new PrintWriter(mc.getOutputStream(), true);
            out.println(" "+i.getValue());
            mc.close();
         }
       } catch (Exception e) {
           e.printStackTrace();
       }
    }
    private ClientCounter i;
    private int port;
}

/**
   This class handles one client input.
*/
class ThreadedEchoHandler extends Thread {
   public ThreadedEchoHandler(Socket i, long n) {
      incoming = i;
      numOrder = n;
   }

   public void run() {
```

```
      try {
         BufferedReader in = new BufferedReader
            (new InputStreamReader(incoming.getInputStream()));
         PrintWriter out = new PrintWriter
            (incoming.getOutputStream(), true /* autoFlush */);
         out.println( "Hello! Enter BYE to exit." );
         boolean done = false;
         while (!done) {
            String str = in.readLine();
            if (str == null) done = true;
            else  {
                out.println("Echo = "+str);
                if (str.trim().equals("BYE"))
                  done = true;
            }
         }
         incoming.close();
         if(numOrder==1)
           System.exit(0);
      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }
   private Socket incoming;
   private long numOrder;
}
```