Segundo Certamen: Tiempo: 19:00 hrs - 20:30 hrs.
Responder un problema por página

1.- (40 puntos) Esta pregunta explora una implementación simplificada del servidor "acumulador" de la Tarea 3. Se pide crear un servidor que acumule valores enteros (en rango 0..5) enviados por los clientes TCP que a él se conecten. Inmediatamente después de conectarse, cada cliente envía un valor entero (0..5) y luego se desconecta del servidor. Se pide que el servidor sea implementado usando el llamado a select; es decir, no puede usar hilos. Además se desea que el servidor muestre por pantalla el resultado acumulado de cada opción cada 3 segundos.

```
#include <stdio.h>
#include <sys/time.h> /*gettimeofday()*/
#include <string.h>  /*memcpy()*/
#include <sys/select.h>  /*select()*/
#include <sys/socket.h>
#include <netdb.h>

#define MAX_HOSTNAME 80
#define MAX_CLIENT 64

int main( int argc, char * argv[] ) {
  int s, rc, n , i, length;
  struct sockaddr_in server;
  struct sockaddr_in from;
  fd_set readfds, readfdsCopy;
  int sockList[MAX_CLIENT], lastSock=0;
  char buf[32];
  int option[6]={0,0,0,0,0,0};
  struct timeval now, nextDisplay, remainingTime;

  server.sin_family = AF_INET;
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  server.sin_port = htons(atoi(argv[1]));
  s = socket (AF_INET, SOCK_STREAM, 0);
  length = sizeof(server);
  bind(s, (struct sockaddr *)&server, length);
  listen(s, 5);
  FD_ZERO(&readfdsCopy);
  FD_SET(s,&readfdsCopy);
  gettimeofday(&nextDisplay, NULL);
  nextDisplay.tv_sec+=3;
  remainingTime.tv_usec=0;
  for(;;){
    gettimeofday(&now, NULL);
    remainingTime.tv_sec = nextDisplay.tv_sec - now.tv_sec;
    if (remainingTime.tv_sec < 0) remainingTime.tv_sec=0;
    memcpy(&readfds, &readfdsCopy, sizeof(fd_set));
    n = select(FD_SETSIZE, &readfds, (fd_set *) 0, (fd_set *) 0, &remainingTime);
    if (n > 0) {
      if (FD_ISSET(s, &readfds)) {
        printf("Accepting a new connection...\n");
        sockList[lastSock++] = accept(s, (struct sockaddr *)&from, &length);
        FD_SET(sockList[lastSock-1], &readfdsCopy);
```

```
    }
    for (i=0; i < lastSock; i++)
     if (FD_ISSET(sockList[i], &readfds)) {
       rc=read(sockList[i], buf, sizeof(buf));
       buf[rc]='\0';
       option[atoi(buf)]++;
       FD_CLR(sockList[i], &readfdsCopy);
       close (sockList[i]);
       sockList[i] = sockList[--lastSock];
       i--;
     }
   } else {  /* n=0 */
    printf("Resultado acumulado opción, 0,1,2,3,4,5 es: %i, %i, %i, %i, %i, %i\n",
          option[0], option[1], option[2], option[3], option[4], option[5]);
    nextDisplay.tv_sec+=3;
   }
  }
}
```

2.- (60 puntos) Se desea controlar la tasa (vulgarmente referido como “ancho de banda”) entrante a una aplicación servidora. A este servidor lo llamaremos ServidorControlado. ServidorControlado normalmente corre en puerto 1234, se ejecuta al estilo de:
$ ServidorControlado 1234

Para lograr limitar la tasa entrante de todas las conexiones, usted cambia el punto de ejecución del ServidorControlado a 4321 y en su lugar pone su programa ControladorTasa. La sintaxis de ejecución de ControladorTasa es:
$ java ControladorTasa <puerto escucha> <puerto servidor> <Bps entrante>

Se pide que usted desarrolle en Java el programa ControladorTasa.java

Ayuda: Se sugiere crear tres flujos de ejecución y usar el equivalente a una tarjeta de crédito de bytes para la aceptación de bytes. Un flujo de ejecución se dedica a traspasar -sin límite- el tráfico saliente, otro flujo traspasa datos entrantes a ServidorControlado en la medida que tenga crédito disponible, y otro flujo a intervalos de aproximadamente 1 segundo paga el crédito consumido para que el segundo flujo pueda enviar datos.

```
import java.io.*;
import java.net.*;
/*
After a Client arrives, it creates the following objects:
              -->fromServer -> UpStreamThread -> toListen -->
serverSocket_/                                               \__clientSock
             \                                               /
              <-toServer <- DownStreamThread <- fromListen <-

*/
public class ControladorTasa {
  static int listeningPort = 1234;
  static int serverPort = 4321;
  static int tasa = 100000;
  public static void main (String args[]) {
```

```
    listeningPort = Integer.parseInt (args[0]);
    serverPort = Integer.parseInt (args[1]);
    tasa = Integer.parseInt (args[2]);

    try {
      ServerSocket throttingSocket = new ServerSocket(listeningPort);
      while(true) {
        Socket clientSock = throttingSocket.accept();
        Socket serverSock = new Socket("localhost", serverPort);
        new UpStreamThread(serverSock.getInputStream(),
                           clientSock.getOutputStream()).start();
        new DownStreamThread(clientSock.getInputStream(),
                             serverSock.getOutputStream(), tasa).start();
      }
    } catch (IOException e){e.printStackTrace(); }
    System.out.println("finishing main");
  }
}

class UpStreamThread extends Thread {
  private InputStream fromServer;  /* input stream from server socket */
  private OutputStream toListen; /* output stream to listening socket */
  UpStreamThread(InputStream is, OutputStream os) {
    fromServer = is;
    toListen = os;
  }
  public void run() {  /* we use this thread to send traffic from Server
                          to Client without control */
    int rc;
    byte buf[] = new byte[32];
    try {
      while ((rc=fromServer.read(buf)) > 0) {
        toListen.write(buf, 0 , rc);
      }
      fromServer.close();
      toListen.close();
    } catch (IOException e){ }
    System.out.println("finishing UpStreamThread");
  }
}
class DownStreamThread extends Thread {
  private InputStream fromListen;
  private OutputStream toServer;
  private int credit;
  DownStreamThread (InputStream is, OutputStream os, int tasa){
    fromListen = is;
    toServer = os;
    credit = tasa;
    new PaymentThread (this, tasa).start();
  }
  public void run() {
    int sc,rc; /* send chars and receive chars */
    boolean DONE=false;
    byte buf[] = new byte[32];
    while(!DONE) {
      try {
        synchronized(this) { /*needed to call wait*/
          while(credit == 0)
```

```
            try {
              wait();
            } catch (InterruptedException e){ }
          sc=buf.length>credit?credit:buf.length;
        }  /* synchronized */
        rc=fromListen.read(buf,0,sc);
        toServer.write(buf, 0, rc);
        decrementCredit(rc);
      } catch (IOException e){
        DONE=true;
      }
    }
    System.out.println("finishing DownStreamThread");
  }
  synchronized void incrementCredit(int increment, int limit) {
    credit += increment;
    if (credit > limit) credit=limit;
    notify();
  }
  synchronized void decrementCredit(int decrement) {
    credit -= decrement;
  }
}

class PaymentThread extends Thread  {
  private DownStreamThread downStream;
  private int tasa;
  PaymentThread(DownStreamThread ds, int t){
    downStream = ds;
    tasa = t;
  }
  public void run() {
    while(downStream.isAlive()) {
      try {
        sleep(500);
      } catch (InterruptedException e){ }
      downStream.incrementCredit((tasa+1)/2, tasa); /* +1 para redondear
                                                       al entero superior*/
    }
    System.out.println("finishing PaymentThread");
  }
}
```