Segundo Certamen
Tiempo 15:40 hrs - 17:10 hrs. Responder un problema por página

1.- 35 pts. Para medir la tasa de bajada obtenida por TCP en transferencias desde un servidor, se pide hacer un programa servidor generador de tráfico de bajada para cada cliente que se conecte a él. El servidor se ejecuta usando:
$ generador <tamaño_paquete_en_byte> <periodo_en_ms>
… servidor esperando en puerto 34332
La aplicación ocupa un puerto libre y lo muestra según se indica. La aplicación envía regularmente un paquete del tamaño indicado y a intervalo dado por el periodo indicado.
Desarrolle un programa en C para este servidor. Este programa atiende a sólo un cliente por vez. Se pide que la tasa generada sea lo más cercana posible a la deseado (tamaño_paquete_en_byte/ periodo_en_ms) .

```
#include <stdio.h>
#include <sys/time.h> /*gettimeofday*/
#include <sys/socket.h>  /*socket */
#include <sys/types.h>  /*socket */
#include <stdlib.h>  /* malloc */
#include <arpa/inet.h> /* htonl */

main( int argc, char * argv[] ){
  int welcomeSocket, clientSocket;
  struct sockaddr_in server, from;
  int fromlen, length, period, i;
  char * buffer;
  struct timeval tf, t;

  server.sin_family = AF_INET;
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  server.sin_port = htons(0);

  welcomeSocket = socket (AF_INET,SOCK_STREAM,0);
  bind(welcomeSocket, (struct sockaddr *) &server, sizeof(server));
  length = sizeof(server);
  getsockname (welcomeSocket, (struct sockaddr *) &server,&length);
  printf("Server Port is: %d\n", ntohs(server.sin_port));

  length = atoi(argv[1]);
  buffer = (char *) malloc(length);
  for (i=0; i<length; i++)   buffer[i]='1';  /* no necesario */
  period = atoi(argv[2])*1000;/*period in usec*/

  listen(welcomeSocket,4);
  fromlen = sizeof(from);
  clientSocket = accept(welcomeSocket, (struct sockaddr *) &from, &fromlen);
  gettimeofday(&tf,NULL);
  for(;;){
    tf.tv_sec = tf.tv_sec + (tf.tv_usec+period)/1000000;
    tf.tv_usec = (tf.tv_usec+period)%1000000;
    if (write(clientSocket, buffer, length) <0 )
        exit(0);
    gettimeofday(&t, NULL);
    usleep(1000000*(tf.tv_sec-t.tv_sec)+(tf.tv_usec-t.tv_usec));
  }
```

```
}
```

2.- 65 pts. Regulador de tasa. Se pide desarrollar un programa Java que regula la tasa de bytes transferidos desde su entrada estándar hacia su salida estándar. El programa se ejecuta según:
$ java **regulador** <bucket_size_en_bytes> <bytes_autorizados_en_bytes> <periodo_en_ms>
regulador usa el algoritmo "leaky bucket", el cual incrementa regularmente un entero (nivel del balde -bucket-) en la cantidad de bytes autorizados y según el periodo indicado. Este nivel parte en cero y su valor se satura al llegar a bucket_size_en_bytes (es decir el nivel del balde tiene un límite). Cuando el nivel es mayor que cero y hay datos en la entrada estándar, se pasan tantos datos a la salida estándar según el menor valor entre el nivel y la cantidad de bytes de datos, y se decrementa en nivel en la cantidad de datos efectivamente pasados.

El programa termina con Control-C. Puede usar el método sleep(periodo_en_ms) de la clase Thread para los incrementos del entero. Notar que cuando el nivel llega a cero, lo deseable es que ante la presencia de datos de la entrada estándar no tener una espera ocupando CPU. Lo mismo cuando el nivel llega a su máximo, lo deseable es no intentar aumentarlo mientras éste no se haya decrementado.

```
import java.io.*;
public class Regulador  {
  public static void main (String args[]) throws IOException {
    int size = Integer.parseInt(args[0]);
    int autorizados = Integer.parseInt(args[1]);
    int periodo = Integer.parseInt(args[2]);
    Bucket bucket = new Bucket(size, autorizados, periodo);
    bucket.start();

    byte[] buffer = new byte[100];
    int i, nAsked, nGranted;
    while((nAsked = System.in.read(buffer)) > 0 ) {
      i=0;
      do {
         nGranted = bucket.reduceLevel(nAsked-i);
         System.out.write(buffer, i, nGranted);
         i+= nGranted;
      } while (i < nAsked);
    }
    System.exit(0);
  }
}

class Bucket extends Thread {
  int level, size, increment, period;

  Bucket (int size, int autorizado, int periodo ) {
    this.size = size;
    increment = autorizado;
    period = periodo;
    level=0;
  }
  public void run() {
    try {
      while (true) {
        synchronized (this) {
           if (level+increment < size ){
             level += increment;
```

```
            notify();
          } else if (level < size)
             level = size;
          else
             wait(); // bucket full case
        }
        sleep (period);
      }
    } catch (InterruptedException e) {
    }
  }
  public synchronized int reduceLevel(int byte2send) {
    try {
       while (true) {
         if ((level-byte2send)>=0) {
           level-=byte2send;
           notify();
           return byte2send;
         } else if (level > 0) {
           byte2send = level;
           level=0;
           return byte2send;
         } else wait();  // bucket empty case
       }
    } catch (InterruptedException e) {
    }
    return 0;
  }
}
```