Solución de Examen de Comunicación de Datos

15 de diciembre de 1999 (Ref.: scdt9912.doc)

Problema 1:

Diseñar y programar un protocolo de comunicaciones que acepte mensajes de cuatro fuentes en un equipo cliente y los envíe a un servidor multiplexándolos por un único canal. En el equipo servidor cada mensaje debe ser procesado, respectivamente, por una de cuatro aplicaciones, la que originará un mensaje de respuesta que se enviará por el canal a la fuente que originó el pedido.

Cada fuente puede enviar mensajes sin necesidad de esperar la respuesta del anterior. Las respuestas no necesariamente llegan en orden a la fuente, problema que es resuelto por ésta.

Se requiere que en el equipo cliente se restrinja la aceptación de nuevos mensajes de una fuente a *N-1* por segundo, si en el equipo servidor se detecta en la aplicación correspondiente un tiempo de respuesta de *N* transacciones por segundo, en el minuto anterior. Asimismo, se requiere que se incremente en uno la cantidad de mensajes que se acepta de una fuente toda vez que la aplicación parezca capaz de procesar (porque lo hace ágilmente o porque no las hay) mas de *N-1* transacciones por segundo.

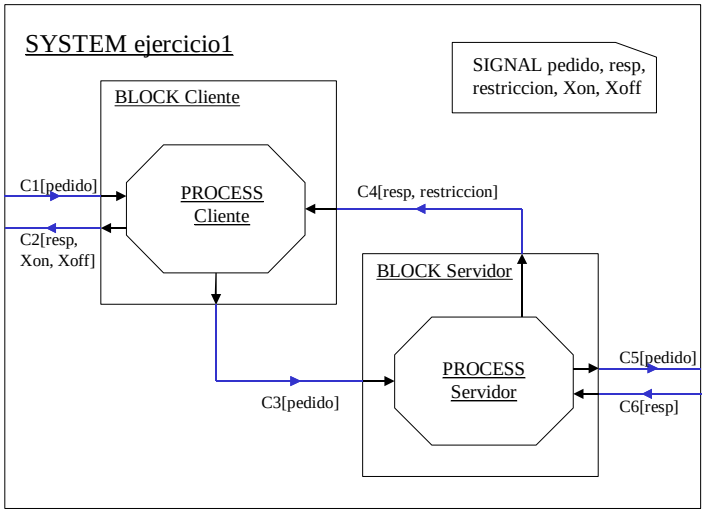
Se pide:
Diseñar y programar el protocolo descripto.

Se utilizará el siguiente formato de paquetes:

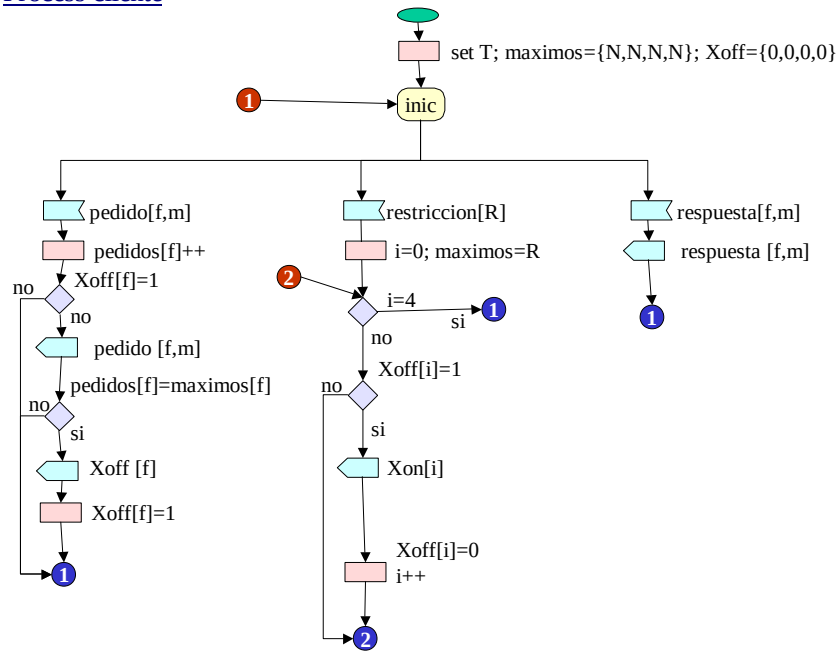
```
typedef paquete=  struct {
                    int tipo,
                    int fuente,
                    string data
                  }
```

El campo tipo puede ser *pedido, resp, restriccion, Xon* y *Xoff*. El campo fuente indicará el número de fuente (0 a 3). Los paquetes de datos *pedido* y *resp* se usarán para enviar una solicitud de servicio a una aplicación para responder al servicio respectivamente, en el campo *data* enviarán los datos de/para la aplicación. El paquete de control *restricción* se utilizará para indicarle al cliente como controlar el flujo (se le da la cantidad máxima por minuto), en el campo *data* se enviarán los topes máximos, y se ignorará el campo *fuente*. Los paquetes de control *Xon* y *Xoff* se utilizarán para habilitar y deshabilitar el flujo procedente de las fuentes, no contendrán campo *data*.

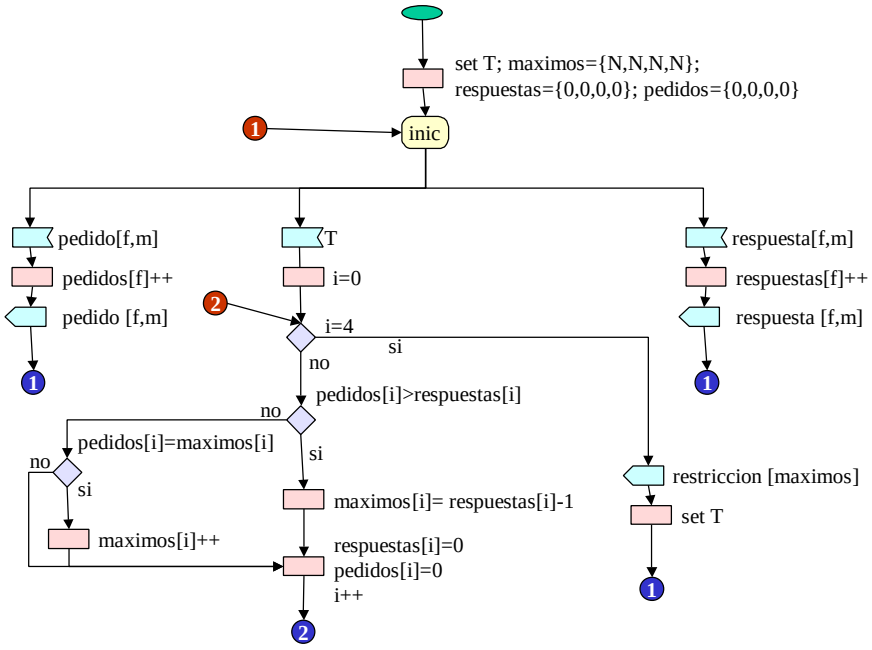
La estrategia del protocolo es comparar la cantidad de pedidos con la cantidad de respuestas. Si hay más pedidos que respuestas, se restringirá al cliente, en caso contrario si se está enviando el máximo (y se están atendiendo) se le aumentará su tope en uno. Inicialmente los máximos se setean en N. El cliente sabiendo la cantidad de pedidos que puede enviar, habilitará y deshabilitará a las fuentes con los paquetes de Xon y Xoff respectivamente. Si las fuentes deshabilitadas envían paquetes, éstos se ignorarán.



Process cliente



Process servidor



Problema 2:

a) Dada una línea por la que se transmite un frame, el cual se ha codificado con un polinomio CRC conocido (donde se envía el mensaje y al final el resto), se desea implementar una función que recibe de a un bit y chequea el polinomio hasta esa posición.
Es decir, que por cada bit recibido se controlará si el CRC chequea hasta esa posición, devolviendo un booleano como resultado.
Notar que si el frame se recibe correctamente, para el último bit el CRC se chequeará bien, por lo que devolverá un *TRUE* como resultado de la invocación del último bit.

Se pide:
Implementar la función en alto nivel e indicar cómo deben inicializarse las distintas variables globales utilizadas por el sistema.

Se utilizará una variable global: *resto*, de tipo *array de bits*, que se comporta *shift register*. La variable será inicializada con cero. (Por abuso del lenguaje usaremos el entero 0 representando al array con todas las posiciones en 0).
El tamaño del array será N+1, siendo N el grado del polinomio. Notar que el bit más significativo siempre estará en 0 (sino podría seguirse dividiendo).

Cada nuevo bit (entrada de la función) será introducido al array por la derecha. El nuevo valor del array se dividirá entre el polinomio obteniendo el nuevo resto. El bit más significativo se utilizará para aplicar la operación: cuando el bit está en 1 se aplicará la función XOR con el polinomio, cuando está en 0 no se hace nada.

0	1	1	…	1	1	0
N	N-1	N-2		2	1	0

```
#define TRUE 1
#define FALSE 0

public int resto [N+1]= 0;

int calculo_resto (int bit) {

   shitf (resto,bit);
   if (resto[N] == 1)
      resto= XOR(resto,polinomio);

   return resto==0;
}
```

b) Sea un sistema que transmite frames, compuestos por celdas de 53 bytes. Las celdas están compuestas de 5 bytes de header y 48 bytes de datos. El header contiene 4 bytes de datos y un byte (el último) de CRC de los 4 anteriores, donde el CRC es de 8 bits (conocido).
El framming no contiene ninguna bandera conocida, mas que cumplir lo indicado anteriormente (no se indica comienzo ni final del frame, ni de cada celda).

Se pide:
Utilizando la parte a) generar un proceso que sincronice los frames después de haber recibido 10 celdas correctas, y chequee el sincronismo de las mismas, entregando las celdas completas recibidas.

En una primera fase debe encontrarse el comienzo de una celda. Para ello por cada bit leído se invocará a la función *calculo_resto* de la parte a), con los últimos 40 bits, buscando si corresponden al header.
Luego de encontrarse un posible header, deben leerse los 48 bytes de datos (384 bits), y 9 celdas más, controlando que al final de los headers la función calculo_resto devuelva TRUE.

Para la invocación de la función *calculo_resto*, se definirá N=8 (grado del polinomio), y se definirá la variable *polinomio* con el polinomio dado. Al comienzo de cada celda se inicializará la variable *resto*=0, que es el shift register utilizado por la función *calculo_resto*.

Para la fase de encontrar un posible comienzo de celda debe chequearse con cada bit leído. Por tanto se utilizará un shift register del largo del header (40 bits).

```
public int header [40]= 0;

void sincronizo_celdas () {

   int celda [424];
   int ok;
   int bit;

   while (TRUE) do {

      // encuentro comienzo
      do {
         leo_bit (bit);
         shitf (header,bit);

         resto =0;
         // chequeo que el CRC del header (ultimos 40 bits) verifique
         for (int i=0; i<40; i++) {
            ok= calculo_resto (header[i]);
         }
      }while (ok == FALSE);

      // ya tengo un posible header; ahora leo los restantes 48 bytes
      for (int i=0; i<384; i++) {
         leo_bit (bit);
      }

      // controlo 9 celdas mas
      for (int vueltas=0; vueltas<9 && ok; vueltas++) {

         // header
         for (int i=0; i<40; i++) {
            leo_bit (bit);
            ok= calculo_resto (bit);
         }

         // restantes 48 bytes
         if (ok) {
            for (int i=0; i<384; i++) {
               leo_bit (bit);
            }
         }
      }

      // ya estoy sincronizado, ahora leo mientras verifiquen los headers
      while (ok) {
         // header
         resto= 0;
         for (int i=0; i<40; i++) {
            leo_bit (bit);
            ok= calculo_resto (bit);
            shift(celda,bit);
         }

         // restantes 48 bytes
         if (ok) {
            for (int i=0; i<384; i++) {
               leo_bit (bit);
               shift(celda,bit);
            }
         }

         // entrego la celda
         if (ok)
            entrego_celda (celda);
      }

   }
}
```

Problema 3:

Se desea implementar un protocolo (de nivel 3) para propagar rutas entre routers. El objetivo es que cada router si lo desea, indique a otros routers cuáles son las sub-redes a las que tiene acceso.

El protocolo deberá cumplir los siguientes esquemas básicos:

- Para nombrar una sub-red se indicará el conjunto de direcciones que la componen de la siguiente forma:
  <*dirección_base*> - <*largo_subnet_mask*>.
  El *largo_subnet_mask* indica cuántos bits de la *dirección base* deben tomarse comenzando desde el bit más significativo.
  Por ejemplo: 206.99.44.0 - 24, indica que los primeros 24 bits son fijos, y los restantes 8 bits varían formando el conjunto de direcciones de la subred (de la forma 206.99.44.X, con X=0..255).
- Los routers almacenan las rutas en una tabla con el siguiente formato de registro:
  <*dirección_base*> - <*largo_subnet_mask*> - <*hops*> - <*dirección_salida*>.
  La *dirección_base* y el *largo_subnet_mask* indican las direcciones de la subred, *hops* indica la cantidad de saltos que transita para llegar a la misma y *dirección_salida* identifica el punto de entrada de otro router a través de una interfase conectada (dirección).
  Por ejemplo un registro del tipo: *206.99.44.0- 24- 3- 200.40.20.4* implica que todos los paquetes con dirección destino 206.99.44.X, con X=0..255, pueden direccionarse a través de la interfase identificada con la dirección 200.40.20.4, con un camino de 3 saltos.
  Otro ejemplo: *206.99.44.0- 23- 0- 206.99.44.5* indica que los paquetes con dirección destino 206.99.44.X y 206.99.45.X se pueden enviar por la interfase identificada con la dirección 206.99.44.5, y que el segmento de red que los contiene esta directamente conectado al router (0 hops).
- Los routers se comunican con otros similares, a través de sus interfases activas (identificadas en ambos extremos por una dirección), y deben transmitir y recibir la información de las subredes a las que tiene acceso. Si se tiene acceso a una subred por más de una interfase, se publicará la de menor cantidad de hops. En cada una de las interfases se ejecuta una instancia de este protocolo (si hay más de una interfase se ejecutan más de una instancia), y modifica la tabla de ruteo en caso de caída de la interfase que se está controlando.
- El protocolo funciona en un puerto conocido de TCP: *router_port*. Se intercambiarán mensajes de 4 tipos:
  ABRIR - Establece o restablece una conexión en un sentido con otro router. El otro router puede si lo desea, abrir la conexión en el otro sentido.
  ESTOY_ACTIVO - Indica que el router está funcionando. Este mensaje debe manejarse cada un tiempo fijo, y de no ser recibido se dará de baja la conexión.
  ACTUALIZAR - Permite el envío de información al otro router.
  NOTIFICACION - Indica el resultado (respuesta) de una operación de *ACTUALIZAR* o *ABRIR*.
- En caso de caída de una conexión o un router, se deben *ACTUALIZAR* las tablas indicando que la ruta no se encuentra activa.

Se pide:
Especificar el protocolo en un lenguaje de alto nivel, indicando además el formato de los mensajes utilizados y los procesos utilizados para la actualización de las tablas de ruteo.

Se implementarán 2 procesos: *router_envia* y *router_recibe*. El primero se encargará de abrir una conexión y periodicamente enviar las tablas al segundo, el cual se encargará sólo de notificar y actualizar sus tablas. En un mismo equipo podrán correr estos 2 procesos.

La tabla del router tendrá una entrada por cada subred para cada salida, pero se publicará solamente la entrada con menor cantidad de hops. Cuando se cae una ruta no se borrarán las entradas en la tabla, sino que se marcarán como borradas poniendo hops= -1.
Periódicamente se enviará toda la tabla al otro router, siempre y cuando hayan habido cambios. La variable global booleana *hubo_cambios* servirá a tales efectos.
Esta variable debe ser compartida entre todos los procesos de envíar y recibir del router (al igual que la tabla), ya que al caer alguna ruta, o abrirse una nueva conexión, o recibirse cambios se deben propagar a todas las interfases.

Se enviará toda la tabla en un mensaje de *actualizar*. Los procedimientos de *empaquetar* y *desempaquetar* se encargarán de poner y sacar la tabla en el mensaje.

La tabla se implementará como un diccionario de clave *direccion_base*, *largo_subnet_mask*, *direccion_salida*. Las funciones de manejo de la misma son las siguientes:

```
int esta_tabla (direccion base, int mask, direccion salida)
   → devuelve 1 si la subred está en la tabla

void busco_tabla (direccion base, int mask, direccion salida, int *hops)
   → devuelve los datos asociados a la subred

void agrego_tabla (direccion base, int mask, direccion salida, int hops)
   → agrega la subred a la tabla

void cambio_tabla (direccion base, int mask, direccion salida, int hops)
   → cambia los datos asociados a la subred

int es_minimo_tabla (direccion base, int mask, direccion salida)
   → devuelve 1 si la entrada es la mínima para la subred. Para el mínimo no
     se tienen en cuenta los hops en –1 (borrados).
```

Las siguientes funciones se utilizan para iterar en la tabla:

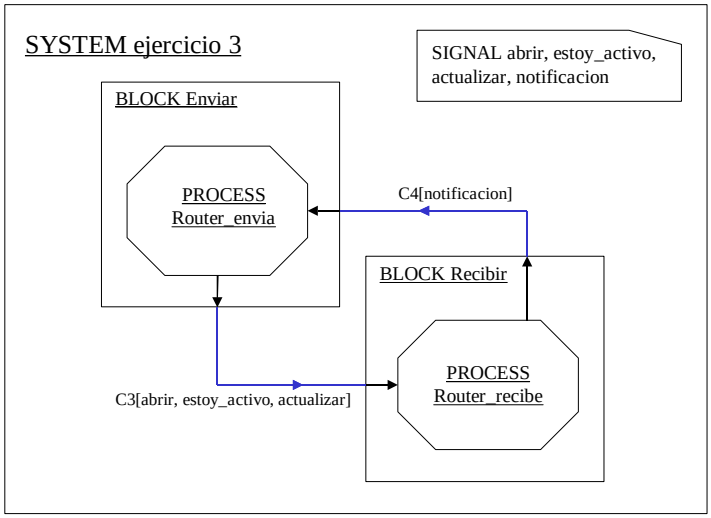
```
int quedan_en_tabla ()
   → devuelve 1 si no ha terminado de iterar

void proximo_tabla (direccion *base, int *mask, direccion *salida, int *hops)
   → devuelve los datos del siguiente en la iteracion
```

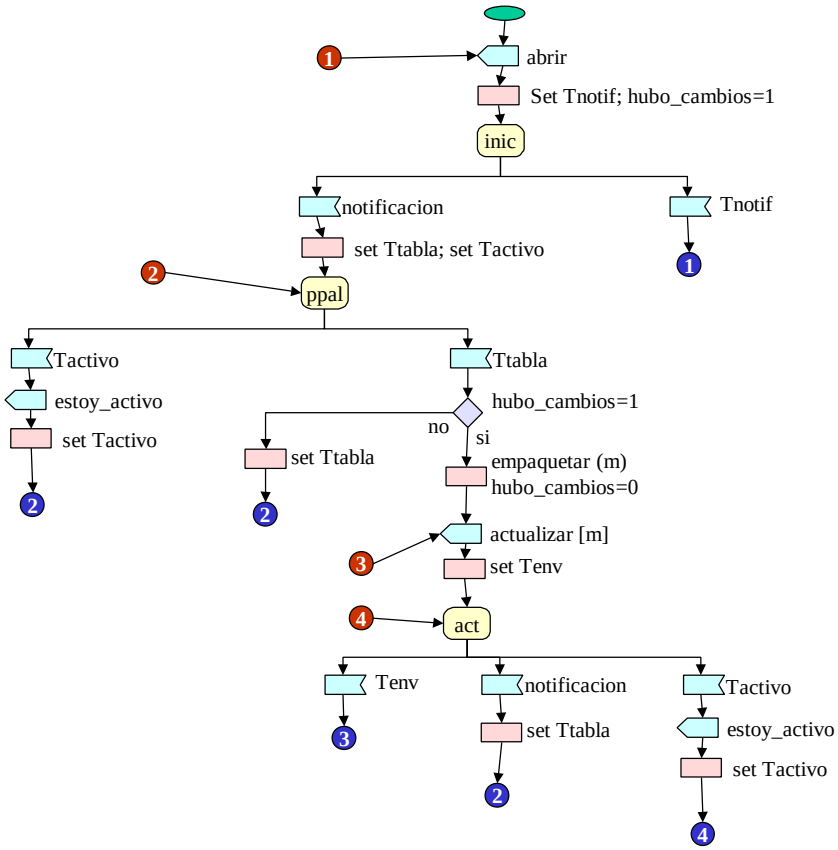
Se utilizará el siguiente formato de paquetes:

```
typedef paquete=  struct {
                    int tipo,
                    string data
                }
```

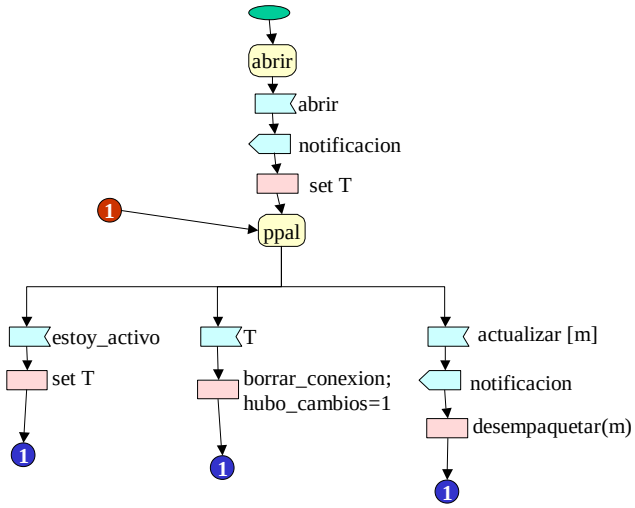
El campo *tipo* puede ser *abrir*, *estoy_activo*, *notificación* o *actualizar*. Los mensajes de *abrir*, *estoy_activo* y *notificación* no tienen nada en el campo *data*, mientras que el mensaje de *actualizar* tiene en *data* los datos de la tabla.



Process Router_envia



Process Router_recibe




```
string empaquetar () {
   string mensaje= NULL;
   while (quedan_en_tabla) {
      proximo_tabla (&base,&mask,&salida,&hops);
      if (es_minimo_tabla (base,mask,salida))
         agregar_string (mensaje,base,mask,salida,hops);
   }
   return mensaje;
}


void desempaquetar (string mensaje) {
   while (mensaje != NULL) {
      quitar_string (mensaje,base,mask,salida,hops);
      proceso (base,mask,hops,salida);
   }
   return mensaje;
}


void proceso (direccion base, int mask, int hops, direccion salida) {
   if (hops == -1) {     // esta borrado
      if (esta_tabla (base,mask)) {
         busco_tabla (base,mask,&h,&s);
         if (s == OTRO_ROUTER)
            cambio_tabla (base,mask,-1,OTRO_ROUTER);
      }
   } else  {  // esta activo
      if (esta_tabla (base,mask)) {
         busco_tabla (base,mask,&h,&s);
         if ((s == OTRO_ROUTER) || (h > hops+1))
            cambio_tabla (base,mask,hops+1,OTRO_ROUTER);
      } else
         agrego_tabla (base,mask,hops+1,OTRO_ROUTER);
   }
}


void borrar_conexion () {
   while (quedan_en_tabla) {
      proximo_tabla (&base,&mask,&hops,&salida);
      if (salida == OTRO_ROUTER)
         cambio_tabla (base,mask,-1,OTRO_ROUTER);
   }
}
```