Solución de Examen de Comunicación de Datos

22 de Julio de 1999 (Ref.: scdt9907.doc)

Problema 1

Se dispone de un enlace bidireccional libre de errores que une dos nodos, por el cual se envían y reciben frames de 32 bytes cada uno. Estos frames se utilizan para multiplexar en el tiempo hasta 30 canales de datos, enviando 1 byte por frame de cada canal y 2 de control, de forma de que en apariencia se dispone de 30 canales cuya velocidad de transmisión es 1/32 de la real.

Se desea establecer conexiones bidireccionales entre ambos nodos utilizando para ello los canales antes descriptos. Al establecer la conexión cada lado elegirá, para su sentido, una posición en cada frame (de entre los que estén libres de los 30 existentes) tomando un criterio circular. En general las posiciones elegidas no serán las mismas en uno y otro sentido. La información de apertura y cierre de conexión, con indicación de las posiciones utilizadas, se enviará utilizando exclusivamente los 2 bytes que quedan libres por frame (bytes de control).

Se pide:

Especificar en un lenguaje de alto nivel un administrador del enlace que permita: abrir y cerrar conexiones, indicar solicitudes de conexión y desconexión, enviar y recibir bytes para las conexiones abiertas.

Los procesos que pidan conexión al administrador, la identificarán por un *número de conexión* entre 0 y 255 (independiente del canal asignado por el administrador). En el envío y recepción de bytes se indicará el *número de conexión*.

Para esto se cuenta con las siguientes primitivas para enviar y recibir frames de 32 bytes:

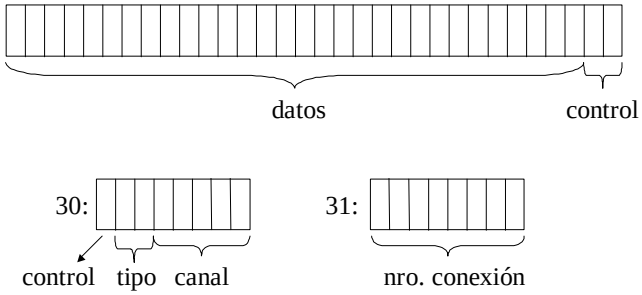
- enviar_frame (in frame)
- recibir_frame (out hay_frame, out frame) (no bloqueante)

Se asumirá que:

- Las conexiones siempre tienen dato para enviar.
- Las aperturas y cierres de conexiones son esporádicas (menos de una por frame enviado).
- No se solicitarán a la vez más de 15 conexiones.

Idea:

Se utilizarán los 30 primeros bytes del frame para datos (de los 30 canales), y los 2 últimos para control. El byte 30 tendrá un bit que indica si el frame contiene información de control, 2 bits que indican el tipo de operación (solicitud de conexión, indicación de conexión o solicitud de desconexión) y los restantes 5 bits indican el canal al que se aplica la operación. El byte 31 se utilizará en los casos de solicitud o indicación de conexión para el número de conexión.



Podemos ver el frame como conteniendo 5 campos:

- datos: array [0..29] of byte → bytes 0 al 29
- control: boolean → bit 0, byte 30
- tipo: (00, 01, 10) → bits 1 y 2, byte 30
- canal: 0..29 → bits 3 a 7, byte 30
- conexión: 0..255 → byte 31

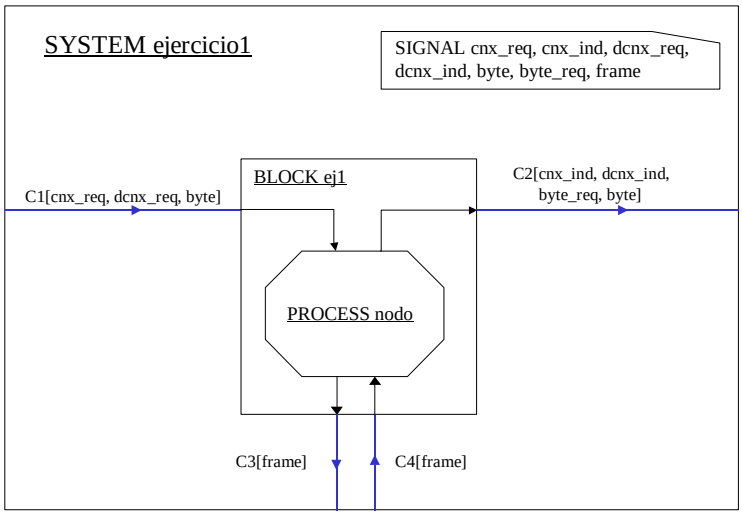
Para el campo tipo se usarán las siguientes constantes:

```
solcnx   → valor 00   // solicitud de conexión
indcnx   → valor 01   // indicación de conexión
dcnx     → valor 10   // solicitud de desconexión
```

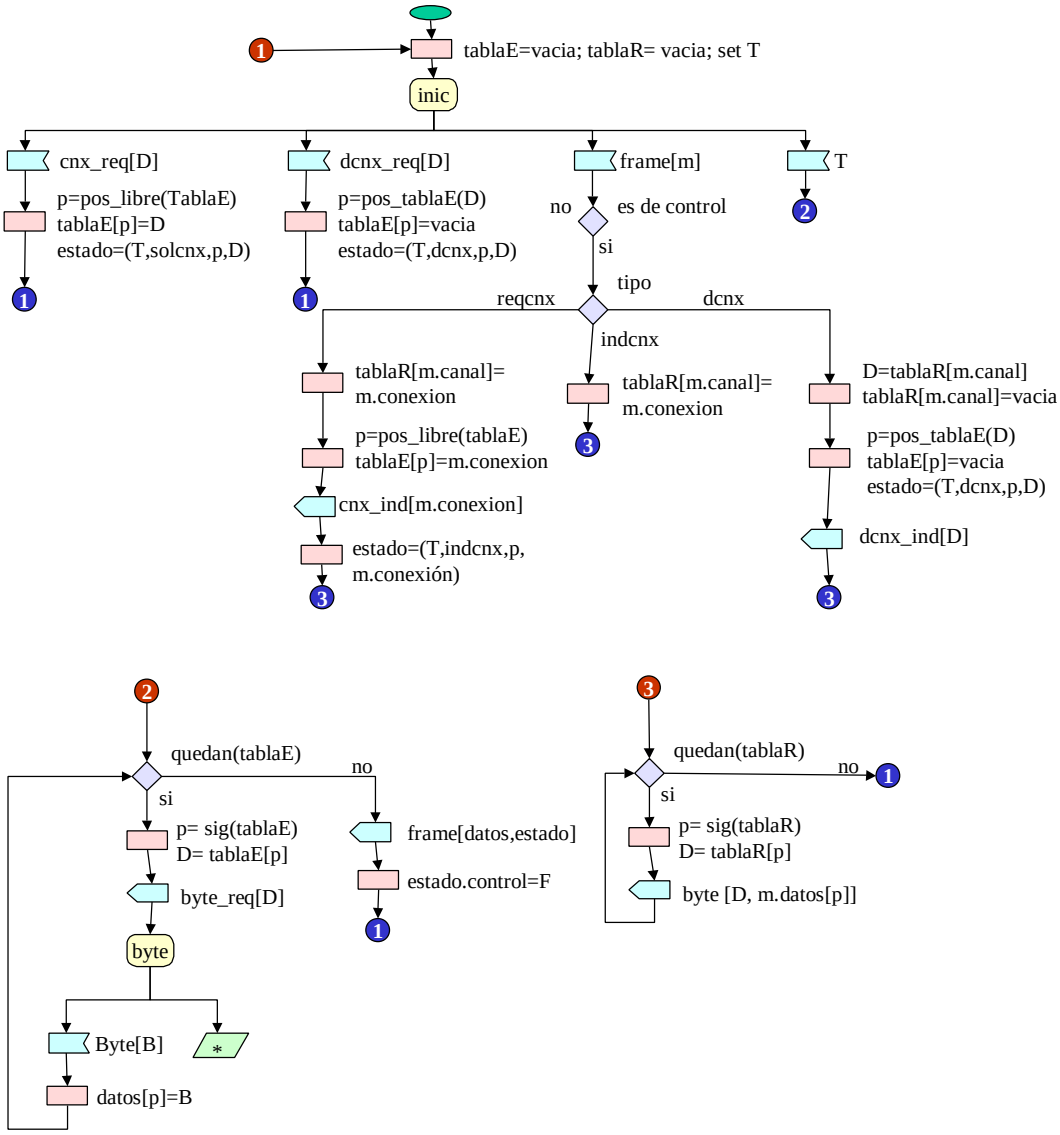
Se usarán además las constantes booleanas: T y F.

Se utilizarán las siguientes variables:

```
tablaE:  array [0..29] of 0..255;      // tabla de conexiones de los canales para enviar
tablaR:  array [0..29] of 0..255;      // tabla de conexiones de los canales para recibir
estado:  record
             control:   boolean;
             tipo:      (solcnx, indcnx, dcnx);
             canal:     0..29;
             conexión: 0..255,
         end;
```



Process nodo



Problema 2.

Un nodo intermedio *I* (en el que no se originan ni son destinados paquetes) posee una línea de entrada y cuatro líneas de salida. Las cuatro líneas de salida conducen directamente a un mismo nodo *D*.

El nodo *I* cuenta con las siguientes funciones:

- recibo_paquete (out paquete) (bloqueante)
  // Me devuelve un paquete a enviar al nodo *D*.
- envio_paquete (in paquete; in linea; out costo_paquete)
  // Envía el paquete a la línea indicada y devuelve el costo de envío.

El costo de envío es proporcional al largo del paquete enviado y va variando con el tiempo.

Se pide:

Programar un algoritmo adaptativo que distribuya él trafico en forma inversamente proporcional al costo por byte de las distintas líneas. Se dispone de una función que devuelve el largo del paquete a enviar.

Idea:
Enviar primero 4 paquetes (1 por cada línea) para determinar los costos de cada línea. De acuerdo a esos costos determinar las proporciones de paquetes que deben salir por cada una. Para cada paquete recibido ver por que línea conviene enviar, de manera de mantenerse lo más cerca posible de las proporciones (se enviará por la línea que más se aparte de las proporciones indicadas para ella). Para esto es necesario llevar la cuenta de la cantidad de paquetes enviados por cada línea.
Luego de N paquetes enviados, se calcularán nuevamente los costos de cada línea, de manera de adaptarse a cambios en los costos. El valor de N dependerá del grado de evolución de los costos, teniendo un valor más grande si los costos cambian poco, y un valor pequeño si varían rápidamente. De esta manera no se sobrecarga al nodo con procesamiento innecesario.

Se utilizarán las siguientes variables:

```
costos:        array [1..4] of real;     // costo por byte de cada línea
proporciones:  array [1..4] of real;     // proporción de paquetes a enviar por cada línea
envios:        array [1..4] of integer;  // cantidad de paquetes enviados por cada línea
tot:           integer;                  // cantidad total de paquetes enviados

PROCEDURE Nodo_D
BEGIN
    while true do
        for L = 1 to 4 do
            recibo_paquete (p);
            c = envio_paquete (p,L);
            costos[L] = c / largo_paquete (p);
            envios[L] = 1;
        endfor
        obtengo_proporciones ( );
        for tot = 5 to N do
            recibo_paquete (p);
            L = elijo_linea ( );
            envio_paquete (p,L);          // ignoro el valor de retorno
            envios[L] ++;
        endfor
    endwhile
END;
```

```
FUNCTION elijo_linea ( ) : integer
VAR A: array [1..4] of real;
BEGIN
    for L= 1 to 4 do
        A[L] = envios[L] / tot;
    endfor
    return maximo (proporciones – A);      // máximo elemento a elemento.
END;

PROCEDURE obtengo_proporciones ( )
VAR     A: array [1..4] of real;
        s: real
BEGIN
    for L= 1 to 4 do
        A[L] = 1 / costos[L];
    endfor
    s = suma (A);      // suma elemento a elemento.
    for L= 1 to 4 do
        proporciones[L] = A[L] / s;
    endfor
END;
```

Observaciones:

- Los costos devueltos por la función envio_paquete se considerán sólo en el primer loop. Una implementación no bloqueante de dicha función (no espero por el resultado) obtiene muy buenos resultados ya que se puede paralelizar los envíos. La manera más fácil de lograr esa implementación no bloqueante, es teniendo 2 funciones: una para enviar el paquete (no bloqueante) y otra que solicite el costo (bloqueante, pero se utilizará sólo en el primer loop).
- Se cuenta la cantidad de paquetes enviados, y no la cantidad de bytes. Esto es correcto, ya que al enviar muchos paquetes, la media de los largos será similar en cada línea (es poco probable que los paquetes largos toquen todos para la misma línea). Podría implementarse que envios cuente los bytes enviados por cada línea y tot la cantidad total de bytes.
- La función elijo_linea hace que el tráfico se distribuya en forma bastante parejo entre las líneas. Podrían haberse enviado una cierta cantidad de paquetes por la primer línea, luego la segunda, ... hasta la cuarta. Esto generaría las mismas proporciones en los grandes números (luego de muchos paquetes), pero utilizaría las líneas en forma despareja (sobrecargadas durante un tiempo, y ociosas otro tanto). Además no podría aprovechares el paralelismo descrito anteriormente.

Problema 3.

a) Diseñar un método de encriptado por substitución polialfabética que tome como base tres permutaciones fijas (conocidas por el receptor), que se aplicarán sucesivamente a cada carácter a encriptar, y que se irán rotando circularmente con cada carácter codificado (/decodificado).

Utilizo la siguiente estructura para tener las tres permutaciones:

```
Permutaciones: array [1..3][A..Z] of A..Z;
```

Por ejemplo:

```
                  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

1a. fila :        K Q D H F G E Z J B L M N O W C R S T U V P X Y I A
2a. fila :        N D E S Z H I J K V M C B P Q R F T U L W X Y G A O
3a. fila :        G E F D Q I J Z L N M O A H R S T Y V W X U K P B C
```

Para criptografiar, aplico sucesivamente la primera, la segunda y la tercera permutación.

```
PROCEDURE cifrador(permut: permutaciones, msg: string, var result: string);
VAR
    ind: integer;

BEGIN
    ind:=0;
    while (not fin(msg)) do
        result[ind]:=permut[3,permut[2,permut[1,msg[ind]]]];
        permut:=Roto(permut);
        ind:=ind+1;
    end while;
END.


FUNCTION roto(permut: permutaciones):permutaciones;
VAR
    result: permutaciones;
    i: integer;

BEGIN
    result[1,'Z']:=permut[1,'A'];
    result[2,'Z']:=permut[2,'A'];
    result[3,'Z']:=permut[3,'A'];
    for i='A' to 'Y' do
        result[1,i]:=permut[1,i+1];
        result[2,i]:=permut[1,i+1];
        result[3,i]:=permut[1,i+1];
    end for;
    return (result);
END.
```

```
FUNCTION  dncrpt(ind:int, encod_char: char):char
// esta función desencripta el carácter codificado, según la permutación permut (tipo permutaciones).

VAR
    carácter: char;
    encontre: boolean;

BEGIN
    caracter:= A;
    encontre:= false;
    while (carácter <> ultimo_carácter  and not encontre) do
        if encod_char==permut[ind,carácter]
            encontre=true;
    end while
    return (carácter)
END.


PROCEDURE decifrador(permut: permutaciones, codif_msg: string, var result: string);
VAR
    ind: integer;

BEGIN
    ind:=0;
    while (not fin(codif_msg)) do
        result[ind]:= dncrpt(1,dncrpt(2,dncrpt(3,msg[ind])));
        permut:=roto(permut);
        ind:=ind+1;
    end while
END;
```

b) Indicar si la solución diseñada puede verse afectada por la pérdida de mensajes.

La pérdida de parte del mensaje, rompe la secuencia de encriptado, ya que no se sabe, cuantas veces se roto la permutación.

Esto hace imposible la desencripción. Para poder desencriptarlo debería conocer la cantidad de caracteres perdidos. El único caso que se puede continuar la desencripción correctamente es si se pierden un múltiplo de 26 caracteres.