SOLUCION EXAMEN COMUNICACION DE DATOS
14 de Febrero de 2000
(ref: scdt0002.doc)

Problema 1:

Diseñar y programar un protocolo de transferencia de archivos que pueda dejarse corriendo inatendido (por ejemplo durante la noche) y que realice los máximos esfuerzos por tener completa la transferencia la mañana siguiente, incluyendo (repetidos reintentos de) reconexión si la conexión se pierde, y aprovechamiento de lo ya transferido si lo hay, tomando las precauciones adecuadas en caso de que el archivo a transferir haya sido actualizado en el origen en el ínterin.

El proceso cliente responderá a una solicitud de transferencia de archivos, para la cual se conectará al servidor. La solicitud tiene como parámetros la dirección del servidor, el nombre del archivo a transferir y el sentido de la transferencia (GET o PUT).
Una vez transferido o no el archivo (fin del proceso), se devolverá el resultado de la transferencia: 0 si fue exitosa, -1 si la transferencia no se pudo realizar.

Se establecerá una conexión sobre una capa 4 con conexión, más concretamente se trabajará sobre TCP/IP, por tal motivo, no nos preocuparemos por la pérdida de mensajes.
Utilizaremos las primitivas de TCP/IP:

- `int socket ( int type);`
  Crea y devuelve un socket file descriptor (sockfd).
- `int bind (int sockfd, struct sockaddr *my_addr);`
  Asigna un direccionamiento al socket *sockfd*. Retorna 0 (éxito) o –1 (error).
- `int connect(int sockfd, struct sockaddr *serv_addr);`
  Conecta con un servidor en la conexión identificada con *sockfd*, y los datos aportados en *sockaddr*. Retorna 0 (éxito) o –1 (error).
- `int accept(int sockfd, struct sockaddr *addr);`
  Acepta la conexión para el socket indicado por *sockfd*, devolviendo la información de la conexión en la estructura *sockaddr*. Retorna 0 (éxito) o –1 (error).
- `int close(int sockfd);`
  Cierra la conexión identificada como *sockfd*. Retorna 0 (éxito) o –1 (error).
- `int send(int sockfd, const void *msg, int len );`
  Envía por *sockfd* el mensaje contenido en *msg*, de largo *len*. Retorna la cantidad de bytes enviados o –1 (error).
- `int recv(int sockfd, void *buf, int len);`
  Recibe por *sockfd* un mensaje que almacena en *buf*, con un largo máximo de *len*. Retorna la cantidad de bytes recibidos o –1 (error).

Una vez establecida la conexión se enviarán (vía send y recv) distintos tipos de paquetes, algunos de ellos de control, y otros conteniendo los distintos bloques de transferencia (porciones de tamaño fijo) del archivo. Para ello consideraremos la siguiente estructura de paquetes:

```
typedef Tpaquete = struct {
                        int tipo,
                        char *data
                    }
```

Los distintos tipos de paquetes son:

- GET: Para solicitar la transferencia de archivos del cliente al servidor. Como parámetro se pasa la identificación del archivo (nombre).
- PUT: Para solicitar la transferencia de archivos del servidor al cliente. Como parámetro se pasa la identificación del archivo (nombre).
- FECHA: Para pasar la fecha de modificación del archivo. El parámetro es dicha fecha.
- POS: Para indicar la posición (en bloques de transferencia) desde la cual se debe transferir. El parámetro es la posición (en una situación normal la posición es 0).
- DATA: Para transferir bloques del archivo.
- EOF: Para indicar que se terminó de transferir el archivo.

Un bosquejo de las acciones del protocolo es el siguiente:

Solicitud de GET:

- Se establece la conexión.
- El cliente envía un paquete de tipo GET con el nombre del archivo.
- El servidor envía un paquete de tipo FECHA con la fecha de modificación del archivo.
- El cliente envía un paquete de tipo POS con la posición (en bloques de transferencia) desde la cual se debe transferir. (0 si es el primer intento, la posición siguiente a la última recibida si es un reintento).
- El servidor envía paquetes de tipo DATA con los bloques del archivo.
- El servidor envía un paquete de tipo EOF.
- Se cierra la conexión.

Solicitud de PUT:

- Se establece la conexión.
- El cliente envía un paquete de tipo PUT con el nombre del archivo.
- El cliente envía un paquete de tipo POS con la posición (en bloques de transferencia) desde la cual se va a transferir. (0 si es el primer intento, la posición siguiente a la última recibida si es un reintento).
- El cliente envía paquetes de tipo DATA con los bloques del archivo.
- El cliente envía un paquete de tipo EOF.
- Se cierra la conexión.

Si la conexión se pierde, las primitivas de TCP/IP devolverán un error, por lo que se reiniciará la conexión. Se harán hasta N reintentos. De no poder transferir completamente el archivo se avisará del error.

El cliente tendrá 2 variables de control: *leidos* que lleva la cuenta de cuantos bloques se han transferido, *last_update* que es la fecha de última actualización del archivo, determinada al comienzo de la transmisión, e *intentos* que es la cantidad de veces que se retransmitió el archivo.

Ante una caída de la conexión se comparará la fecha, con la fecha actual del archivo. Asumimos que el sistema operativo no deja modificar el archivo mientras se realiza la transferencia (ya que lo tengo abierto para lectura); si el sistema no lo controla, no lo controlaremos nosotros.
Si las fechas coinciden se continuará la transmisión del archivo, sino se borrará la parte transferida y se comenzará desde el principio.

```
#define ERROR –1
#define OK 0


int ProcesoCliente (Address server, string arch, int sentido) {
   int sockfd;
   struct sockaddr server_addr;
   int status, intentos=0;
   global int leidos=0;
   global date lastUpdate;

   sockfd= AbroCliente (server);
   if (sockfd < 0) return ERROR;

   while (intentos < N) {
      // me conecto
      status= connect (sockfd, &server_addr);

      if (status > 0) {
         if (sentido == GET)
            status= GetEnCliente (sockfd,arch);
         else
            status= PutEnCliente (sockfd,arch);
      }

      // si termino o dio error cierro todo
      fclose (arch);
      close (sockfd);

      if (status == ERROR) intentos ++;
      else return OK;
   }

   return ERROR;
}


int ProcesoServidor () {
   int sockfd;
   struct sockaddr client_addr;
   int status, sentido;
   string arch;

   sockfd= AbroServidor ();
   if (sockfd < 0) return ERROR;


   while (TRUE) {
      // me conecto
      status= accept (sockfd, &client_addr);

      // recibo pedido
      if (status >= 0)
         status= recibo_datos (sockfd, &sentido, &arch);

      if (status > 0) {
         if (sentido == GET)
            status= GetEnServidor (sockfd,arch);
         else
            status= PutEnServidor (sockfd,arch);
      }

      // si termino o dio error, vuelvo a escuchar pedidos
      fclose (arch);
      close (sockfd);
   }
}
```

```
int GetEnCliente (int sockfd, string arch) {
   int status, tipo;
   string data;
   date f;

   // envio un GET
   status= envio_datos (sockfd,GET,arch);
   if (status == ERROR) return ERROR;

   // espero la fecha
   status= recibo_datos (sockfd,&tipo,&f);
   if (status == ERROR or tipo != FECHA) return ERROR;

   // comparo lo transferido (si lo hay)
   if (leidos > 0 and f == lastUpdate) fopen (arch, "A"); // append
   else {
      leidos= 0;
      fopen (arch, "W");  // sobreescribir
      lastUpdate= f;
   }

   // envio la posicion
   status= envio_datos (sockfd,POS,leidos);
   if (status == ERROR) return ERROR;

   // recibo los bloques del archivo
   status= recibo_datos (sockfd,&tipo,&data);

   while (status == OK and tipo==DATA) {
      fwrite (arch,data);
      leidos ++;
      status= recibo_datos (sockfd,&tipo,&data);
   }

   if (status == ERROR) return ERROR;
   if (tipo == EOF) return OK;
   else return ERROR;
}


int PutEnCliente (int sockfd, string arch) {
   int status, tipo;
   string data;
   date f;

   // envio un PUT
   status= envio_datos (sockfd,PUT,arch);
   if (status == ERROR) return ERROR;

   // comparo lo transferido (si lo hay)
   f= fdate (arch);
   if (leidos > 0) {
      if (f != lastUpdate) {
         leidos= 0;
         lastUpdate= f;
      }
   else lastUpdate= f;

   // avanzo la posicion en el archivo
   fseek (arch, leidos);

   // envio la posicion
   status= envio_datos (sockfd,POS,leidos);
   if (status == ERROR) return ERROR;

   // envio los bloques del archivo
   fopen (arch, "R");    // lectura
```

```
   while (!feof(arch) and status == OK) {
      data= fread(arch);
      leidos ++;
      status= envio_datos (sockfd,DATA,data);
   }
   if (status == ERROR) return ERROR;

   // termino de transferir, envio EOF
   status= envio_datos (sockfd,EOF,NULL);
   return status;
}


int GetEnServidor (int sockfd, string arch) {
   int status, tipo, leidos;
   string data;
   date f;

   // envio la fecha
   f= fdate (arch);
   status= envio_datos (sockfd,FECHA,f);
   if (status == ERROR) return ERROR;

   // espero la posicion
   status= recibo_datos (sockfd,&tipo,&leidos);
   if (status == ERROR or tipo != POS) return ERROR;

   // envio los bloques del archivo
   fopen (arch, "R");    // lectura
   fseek (arch, leidos); // avanzo posiciones

   while (!feof(arch) and status == OK) {
      data= fread(arch);
      status= envio_datos (sockfd,DATA,data);
   }
   if (status == ERROR) return ERROR;

   // termino de transferir, envio EOF
   status= envio_datos (sockfd,EOF,NULL);
   return status;
}


int PutEnServidor (int sockfd, string arch) {
   int status, tipo, leidos;
   string data;
   date f;

   // espero la posicion
   status= recibo_datos (sockfd,&tipo,&leidos);
   if (status == ERROR or tipo != POS) return ERROR;

   // tengo en cuenta lo transferido (si lo hay)
   if (leidos > 0) fopen (arch, "A"); // append
   else fopen (arch, "W");     // sobreescribir

   // recibo los bloques del archivo
   status= recibo_datos (sockfd,&tipo,&data);

   while (status == OK and tipo==DATA) {
      fwrite (arch,data);
      status= recibo_datos (sockfd,&tipo,&data);
   }

   if (status == ERROR) return ERROR;
   if (tipo == EOF) return OK;
   else return ERROR;
}
```

```
int envio_datos (int sockfd, int tipo, string data) {
   Tpaquete m;
   int status;

   m.tipo= tipo;
   m.data= data;
   largo= length(m);
   status= send (sockfd, m, largo);
   return (status == largo);
}


int recibo_datos (int sockfd, int *tipo, string *data) {
   Tpaquete m;
   int status;

   status= recv (sockfd, m, MAXLARGO);
   if (status < 0) return ERROR;
   *tipo= m.tipo;
   *data= m.data;
   return OK;
}


int AbroCliente (Address server, struct sockaddr *server_addr) {
   int sockfd= socket (SOCK_STREAM);
   server_addr→port= TRANSF_PORT;
   server_addr→address= server;
   return sockfd;
}


int AbroServidor () {
   struct sockaddr server_addr;
   int sockfd= socket (SOCK_STREAM);
   if (sockfd < 0) return ERROR;
   server_addr.port= TRANSF_PORT;
   server_addr.address= obtengo_mi_dirección_IP();
   status= bind (sockfd, &server_addr);
   if (status < 0) return ERROR;
   return sockfd;
}
```

Problema 2:

Sea una red multiprotocolo, que está compuesta de dos subredes. Ambas subredes se encuentran conectadas a una red de acceso general tipo TCP/IP, a través de la cual se comunican.

Para los distintos protocolos se deben tener las siguientes consideraciones:

- Se cuenta con dos funciones, para enviar y recibir paquetes desde la subred a la red TCP/IP:
  - *enviar(protocolo, largo, paquete)*
  - *recibir(protocolo, paquete)*

  donde *protocolo* es un número entre 0 y 127 que indica el protocolo del paquete, *paquete* es el paquete a enviar o recibir, en el formato del protocolo indicado, y *largo* indica la longitud en bytes del paquete a enviar.
- Los enlaces con la red de tipo TCP/IP admiten paquetes de máximo 1500 bytes, mientras que los paquetes generados en ambas subredes, pueden ser de cualquier largo.
- Los paquetes deben ser entregados para los distintos protocolos en el mismo orden que se reciben.

Se pide:

a) Especificar en un lenguaje de alto nivel, el proceso que atiende la conexión de la subred con la red general TCP/IP.

Se establecerá una conexión TCP/IP entre los dos procesos que intercomunican las subredes. TCP/IP se encargará de controlar que lleguen todos los paquetes, y en orden, por lo que no será necesario que este protocolo lo controle nuevamente.

Se deberán fragmentar los mensajes largos, por lo que se deberá indicar al otro lado si se trata de un fragmento o de un paquete completo. Para ello, nos alcanza con un solo bit. Este bit estará prendido (en 1) cuando el paquete original continúa en el siguiente paquete, esto le indicará al proceso del otro lado, que deberá concatenarlo con el siguiente. Cuando se envíe el último fragmento (se aplica también a paquetes chicos, que se envían enteros) el bit estará apagado (en 0).

También se debe indicar al otro lado el protocolo del que provenía el paquete, para que pueda ser entregado al protocolo correcto. Entonces, de los 1500 bytes de datos que pueden enviarse por la conexión TCP/IP, destinaremos 7 bits para indicar el protocolo, 1 bit para indicar la fragmentación y 1499 bytes para datos (del paquete original).

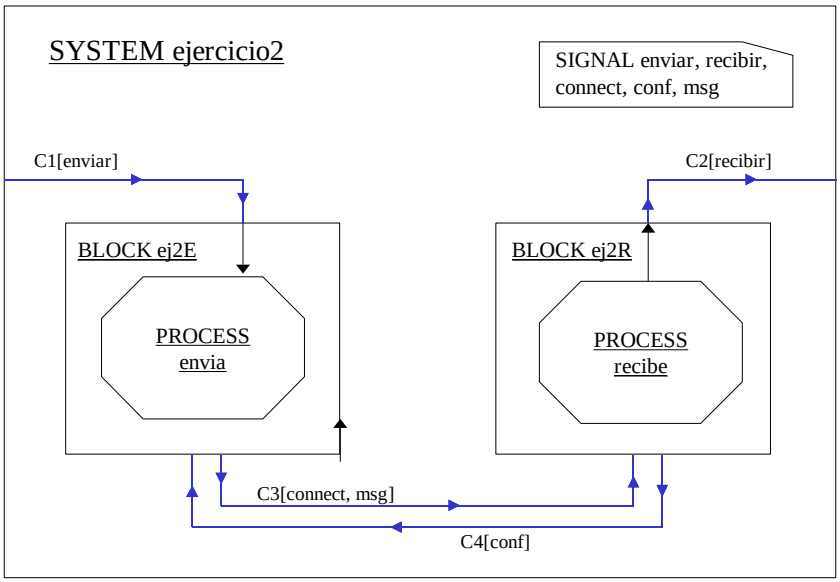
```
typedef paquete_tcp = struct {
                        bit protocolo[7],
                        bit fragmento,
                        char *mensaje
                      }
```

Se utilizarán las primitivas de sockets para establecer la conexión e intercambiar paquetes. De cada lado se tendrán 2 procesos, uno para enviar y otro para recibir paquetes. El proceso que recibe paquetes realiza un accept (open pasivo), y el proceso que envia paquetes realiza un connect (open activo). Esto se hace una única vez al comienzo de los procesos, luego se utiliza la conexión para enviar todo el flujo de paquetes por ella. No se cerrará la conexión en ningún momento. De perderse la conexión por causas externas, esta se restaurará al levantar nuevamente los procesos. Esto podría manejarse también como una excepción.

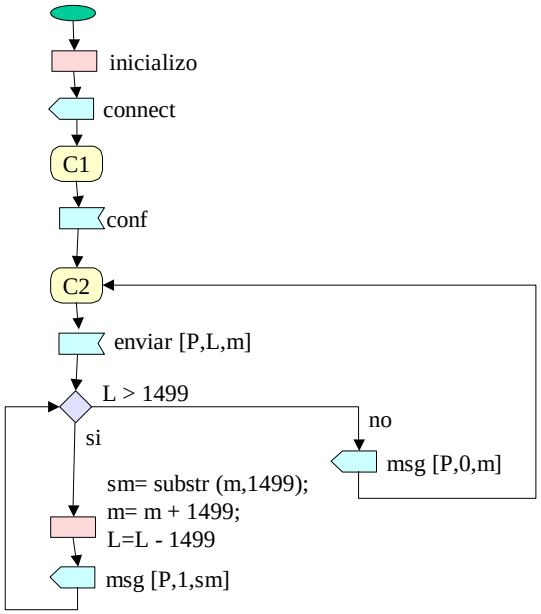
Traduciremos las funciones de sockets a señales de SDL.

- accept:
  La invocación bloqueante del **accept** se simbolizará esperando la señal *connect* y el establecimiento de la conexión (retorno del **accept**) se simbolizará enviando la señal *conf*.
- connect
  La invocación bloqueante del **connect** se simbolizará enviando la señal *connect* y el establecimiento de la conexión (retorno del **connect**) se simbolizará esperando la señal *conf*.

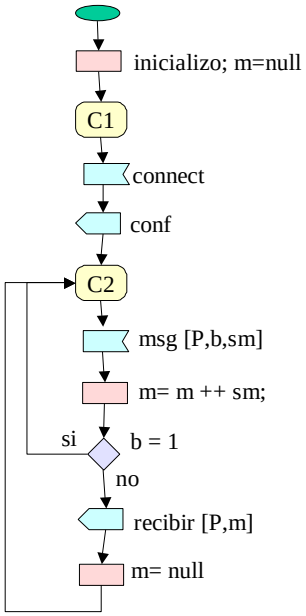
- send
  El **send** se simbolizará enviando la señal *msg*. No se esperará respuesta ya que TCP/IP asegura el envío del paquete.
- recv
  La invocación bloqueante del **recv** se simbolizará esperando la señal *msg*.



Process envia



Process recibe



b) ¿Si se deseara tener más seguridad en el sistema compuesto por las dos subredes, qué se puede mejorar?

Se debería encriptar los paquetes enviados. Los responsables de encriptar y desencriptar los paquetes serían los procesos *envia y recibe*. Como sería una tarea interna al protocolo podría utilizarse cualquier método de encriptación.

Problema 3:

Se desea diseñar los mecanismos de una capa 3 con conexión, permitiendo conexión, envío y recepción de paquetes y desconexión, de forma de que el camino sea predeterminado por el nodo de origen.

Para esto, existe un nodo, del que se conoce la dirección de capa 3, que es capaz de determinar el camino óptimo entre un origen y un destino. La red considerada es perfectamente estable.

Se pide:

a) Especificar las primitivas a implementar, indicando brevemente su función, y formato de paquetes utilizado.

Se tendrán dos tipos de tablas:

- Las tablas de ruteo, que tendrán los nodos y circuitos de entrada-salida, y un indicador de estado:

```
VAR tabla: record
    host1: Nodo,
    cv1: CircuitoVirtual,
    host2: Nodo,
    cv2: CircuitoVirtual,
    estado: (requerido, establecido)
end;
```

- Las tablas de mapeo a la capa superior que tendrán para cada host remoto, el nodo de salida y el circuito virtual.:

```
VAR tabla_circuitos: record
    host: Nodo,
    cv: CircuitoVirtual;
end;
```

No se tendrán TimeOuts para las conexiones a medio establecer, el indicador de estado ayudará a que un proceso que corra periódicamente borre las conexiones no establecidas.

Primitivas propuestas:

- Entre los Nodos y su capa superior:
  - → conectar (H: Nodo)
    Consulta el camino al nodo buscador, agrega una nueva entrada en la tabla y envía un paquete de tipo *conexion_req* al nodo siguiente del camino. Una vez finalizada la conexión llegará un *conexion_resp*.
  - → desconectar (H: Nodo)
    Busca el nodo siguiente en la tabla, envía un paquete de tipo *desconexion_req* al mismo, y borra la entrada en la tabla.
  - ←→mensaje (H: Nodo, msg: Tmensaje)
    Busca el nodo siguiente en la tabla y envía un paquete de tipo *datos* al mismo.
  - ← conectado (H: Nodo)
    Informa a la capa superior que se estableció una conexión con el nodo *H*.
  - ← desconectado (H: Nodo)
    Informa a la capa superior que se cerró la conexión con el nodo *H*.

- Entre los Nodos entre si (nivel capa 3)
  - ←→conexion_req (Nenv, Nrec: Nodo, cv:CircuitoVirtual, Horigen: Nodo, camino: List of Nodo)
    Si soy el nodo destino envío la señal *conectado* a la capa superior, agrega una nueva entrada en la tabla y envía un paquete de tipo *conexion_resp* al nodo que envió el paquete.
    Si no soy el nodo destino asigna un circuito virtual, agrega a una nueva entrada en la tabla y envía un paquete de tipo *conexion_req* al siguiente nodo del camino.
  - ←→conexion_resp (Nenv, Nrec: Nodo, cv:CircuitoVirtual, Horigen: Nodo)
    Si soy el nodo destino envío la señal *conectado* a la capa superior.
    Si no soy el nodo destino busca el nodo siguiente en la tabla y envía un paquete de tipo *conexion_resp* al mismo.

←→desconexion_req (Nenv, Nrec: Nodo, cv:CircuitoVirtual)

Si soy el nodo destino envío la señal *desconectado* a la capa superior y borra la entrada en la tabla.

Si no soy el nodo destino busca el nodo siguiente en la tabla, borra la entrada y envía un paquete de tipo *desconexion_req* al mismo.

←→datos (Nenv, Nrec: Nodo, cv:CircuitoVirtual, tipo: Integer, msg: String)

Si soy el nodo destino envío la señal *mensaje* a la capa superior.

Si no soy el nodo destino busca el nodo siguiente en la tabla y envía un paquete de tipo *datos* al mismo.

En el caso de los mensajes de *datos* pueden ser de 3 tipos:

- BÚSQUEDA: Se utiliza para preguntarle al nodo de búsqueda, cual es el mejor camino a otro nodo. En el campo *msg* se colocarán las direcciones de los nodos origen y destino del camino.
- RESPUESTA: Lo utiliza el nodo de búsqueda para responder a un paquete de búsqueda. En el campo *msg* se colocará el camino entre los nodos.
- DATOS: Se utiliza para mandar datos.

Se tendrán varias variables globales:

YO: Nodo	indica mi nodo
NODOBUSCADOR: Nodo	dirección del nodo especial de búsqueda.
CAMINOBUSCADOR: ListOfNodo	mejor camino al nodo de búsqueda.

Al comienzo cada nodo se conectará al nodo de búsqueda, estableciendo un circuito virtual permanente: cvbus. Luego, cada vez que necesite mejores caminos utilizará ese circuito virtual para las consultas. La variable NodoBuscador es el nodo por el que salen solicitudes al nodo buscador (primer nodo del camino).

b) Especificar en un lenguaje de alto nivel las primitivas expresadas en el punto a)

Funciones auxiliares:

- Para el manejo de tabla_circuitos:
  - AgregarTablaHost (H:Nodo, S: Nodo, cv:CircuitoVirtual) → Agrega la pareja a la tabla.
  - BorrarTablaHost (S: Nodo, cv:CircuitoVirtual) → Borra la entrada de cv en la tabla.
  - BuscoTablaHost (H: Nodo, var S: Nodo, var cv:CircuitoVirtual): boolean → Devuelve verdadero si H está en la tabla, y en las variables S y cv el nodo de salida y el circuito virtual con el que está conectado.
  - BuscoTablaHost (S: Nodo, cv:CircuitoVirtual, var H: Nodo): boolean → Devuelve verdadero si cv está en la tabla para el nodo de salida S, y en la variable H el host con el que está conectado.

- Para el manejo de tabla_circuitos:
  - AgregarTabla (N1: Nodo, cv1:CircuitoVirtual, N2: Nodo, cv2:CircuitoVirtual) → Agrega la tupla a la tabla (inserta una sóla tupla), con estado requerido. Puede suponerse un evento que periódicamente, borre las entradas requeridas que no fueron establecidas.
  - ConfirmarTabla (**N**: Nodo, cv:CircuitoVirtual) → Cambia a establecido el estado de la entrada de <N,cv> en la tabla (busca de ambos lados).
  - BorrarTabla (**N**: Nodo, cv:CircuitoVirtual) → Borra la entrada de <N,cv> en la tabla (busca de ambos lados).
  - BuscoTabla (N1: Nodo, cv1:CircuitoVirtual, var N2: Nodo, var cv2:CircuitoVirtual): boolean → Busca la pareja <N1,cv1> en la tabla (busca de ambos lados), y devuelve verdadero si la encontró, y en N2 y cv2 la pareja conectada.
  - CircuitoVirtualDisponible (N: Nodo): integer → Devuelve el menor número de circuito no utilizado en la tabla para el nodo N (busca de ambos lados).

```
PROCESS IMP
BEGIN

    /* Al comienzo establezco un circuito virtual con el nodo buscador */
    if (YO != NODOBUSCADOR) then
        NodoBuscador = first (CAMINOBUSCADOR)
        camino= resto (CAMINOBUSCADOR)
        cvbus= circuito_virtual_disponible(NodoBuscador);
        Conexion_req (YO, NodoBuscador, cvbus, camino, NODOBUSCADOR);
        Set TimeOut
```

```
    while not conectado
        wait (evento);
        case evento of
            Conexion_resp (NodoEnvia, NodoRecibe, cv, HostOrigen):
                if (HostOrigen = YO) then
                    conectado = TRUE
                endif
            TimeOut
                Conexion_req (YO, NodoBuscador, cvbus, camino, NODOBUSCADOR);
        end case
    end while
endif

/* Recibo solicitudes */
while true loop
    wait (evento);
    case evento of

        Conectar (OtroHost):
            Datos (YO, NodoBuscador, cvbus, BÚSQUEDA, YO, OtroHost);

        Desconectar (OtroHost):
            esta= BuscoTablaHost (OtroHost, NodoSiguiente, cv);
            if (esta) then
                Desconexion_req (YO, NodoSiguiente, cv);
                BorrarTablaHost (NodoSiguiente, cv);
            endif

        Mensaje (OtroHost, msg):
            esta= BuscoTablaHost (OtroHost, NodoSiguiente, cv);
            if (esta) then
                Datos (YO, NodoSiguiente, cv, DATOS, msg);
            endif

        Conexion_req (NodoEnvia, NodoRecibe, cv, HostOrigen, camino):
            if (vacio(camino))     /* viene para mi */
                AgregarTablaHost (HostOrigen, NodoEnvia, cv);
                AgregarTabla (NodoEnvia, cv, YO, -1);
                Conectado (HostOrigen);
                Conexion_resp (YO, NodoEnvia, cv, HostOrigen);
            else
                NodoSig = first (camino)
                camino= resto (camino)
                cvd= circuito_virtual_disponible(NodoSig);
                AgregarTabla (NodoEnvia, cv, NodoSig, cvd);
                Conexion_req (YO, NodoSig, cvd, camino, HostOrigen);
            endif

        Conexion_resp (NodoEnvia, NodoRecibe, cv, HostOrigen):
            esta= BuscoTabla (NodoEnvia, cv, NodoSiguiente, cvs);
            if (esta) then
                if (NodoSiguiente = YO) then
                    Conectado (HostOrigen);
                    InsertarTablaHost (HostOrigen, NodoEnvia, cv);
                else
                    Conexión_resp (YO, NodoSiguiente, cvs, HostOrigen);
                endif
                ConfirmarTabla (NodoEnvia, cv);
            endif
```

```
            Desconexion_req (NodoEnvia, NodoRecibe, cv):
                esta= BuscoTabla (NodoEnvia, cv, NodoSiguiente, cvs);
                if (esta) then
                    if (NodoSiguiente = YO) then
                        esta= BuscoTablaHost (NodoEnvia, cv, OtroHost);
                        if (esta) then
                            Desconectado (OtroHost);
                            BorrarTablaHost (NodoEnvia, cv);
                        endif
                    else
                        desconexion_req (YO,NodoSiguiente, cvs);
                    endif
                    BorrarTabla (NodoEnvia, cv);
                endif

            Datos (NodoEnvia, NodoRecibe, cv, tipo, msg):
                esta= BuscoTabla (NodoEnvia, cv, NodoSiguiente, cvs);
                if (esta) then
                    if (NodoSiguiente = YO) then
                        if (tipo = BÚSQUEDA)          /* solo entrará acá el nodo de búsqueda */
                            HostOrigen= first(msg)
                            HostDestino= resto(msg)
                            camino= MejorCamino (HostOrigen, HostDestino)
                            Datos (YO, NodoEnvia, cv, RESPUESTA, camino);
                        else if (tipo = RESPUESTA) then                    /* continuar la conexión */
                            camino= msg
                            OtroHost= last (camino)
                            NodoSig= first (camino)
                            camino= resto (camino)
                            cvd= circuito_virtual_disponible(NodoSig);
                            AgregarTablaHost (OtroHost, camino, cvd)
                            Conexión_req (YO, NodoSig, cvd, camino, YO);
                        else
                            esta= BuscoTabla_host (cv, OtroHost);
                            if (esta) then
                                Mensaje (OtroHost, msg);
                            endif
                        endif
                    else
                        Datos (YO, NodoSiguiente, cvs, tipo, msg);
                    endif
                endif

        endcase
    endloop
END;
```