Solución de Examen de Comunicación de Datos

13 de Agosto de 1999 (Ref.: scdt9908.doc)

Problema 1

Se cuenta con las siguientes primitivas de sockets, disponibles en un sistema:

- int socket(int type);
  Crea y devuelve *sockfd* (socket file descriptor) que se podrá utilizar en llamadas posteriores.
  - *type*: SOCK_STREAM o SOCK_DGRAM (sockets del TCP o del UDP)
- int bind(int sockfd, struct sockaddr *my_addr);
  Asigna un direccionamiento al socket *sockfd*. Retorna 0 (éxito) o –1 (error).
  - *sockfd*: especifica el descriptor file descriptor del socket que se asociará.
  - *sockaddr* :es la estructura que contiene el direccionamiento. Este tiene la siguiente estructura:
    - u_short port = puerto;
    - char *address = INADDR_ANY(cualquier dirección) o dirección de la otra punta de la conexión.
- int connect(int sockfd, struct sockaddr *serv_addr);
  Conecta con un servidor en la conexión identificada con *sockfd*, y los datos aportados en *sockaddr*.
  (ver sockaddr en bind). 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*. (ver sockaddr en bind). 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).
- int close(int sockfd);
  Cierra la conexión identificada como *sockfd*. Retorna 0 (éxito) o –1 (error).

Se pide:
Implementar las siguientes primitivas en función de las primitivas de sockets presentadas anteriormente.

- int p_open(int socket_type, u_short port,);
  Esta función implementa un open pasivo, en el puerto *port*, y devuelve el identificador de conexión.
  socket_type: SOCK_STREAM o SOCK_DGRAM (sockets del TCP o del UDP)
  port: El puerto a escuchar.
- int a_open (u_short port, int type, char *netaddress);
  Esta función implementa un open activo, para hacer una conexión al servidor/puerto *port*, y devuelve identificador de conexión.
  type: es SOCK_STREAM o SOCK_DGRAM,
  netaddress: dirección del servidor a conectar.

- int send(int sockfd, char *buf, int count);
  Esto es un write() en la conexión *sockfd*. Envía lo depositado en *buf* , hasta *count* bytes. Se cerciorará de que todos los datos estén transmitidos. Retorna 0 (éxito) o –1 (error).
- int receive(int sockfd, const char *buf, int count);
  Esto es un read(), a la conexión *sockfd* . Lo leído lo deposita en *buf*, y se reciben hasta *count* bytes. Retorna 0 (éxito) o –1 (error).
- int close(int sockfd);
  Cierra la conexión identificada como *sockfd*. Retorna 0 (éxito) o –1 (error).

Las primitivas propuestas para enviar y recibir datos por el socket, no diferencian entre una comunicación mediante streams o datagramas. Las mismas no tienen como parámetro la dirección del otro extremo, por lo que dicha dirección, aún en el caso de datagramas, debe fijarse en el momento de creación del socket. Es decir, que es necesario hacer un *connect* aún cuando el tipo de conexión sea de datagramas.

Para hacer el *bind* en el servidor, se asume la existencia de la función *obtengo_mi_direccion* que devuelve la dirección IP del equipo. Puede utilizarse también la constante INADDR_ANY. En el caso del cliente no es necesario hacer un *bind*, ya que el protocolo puede ejecutar en cualquier puerto, y el servidor se enterará del mismo como resultado del *accept*. No está mal hacer el *bind* en el cliente, y en tal caso, debería pedirse un puerto libre (hay funciones para ello).

```
#define ERROR -1
#define OK 0


int P_open (int socket_type, u_short port,) {

   int sockfd;
   struct sockaddr server_addr, client_addr;
   int status;

   sockfd= socket (socket_type);
   if (sockfd < 0) return ERROR;

   server_addr.port= port;
   server_addr.address= obtengo_mi_direccion();
   status= bind (sockfd, &server_addr);
   if (status < 0) return ERROR;

   status= accept (sockfd, &client_addr);
   if (status < 0) return ERROR;

   return sockfd;
}
```

```
int A_open (u_short port, int type, char *netaddress ) {

   int sockfd;
   struct sockaddr server_addr;
   int status;

   sockfd= socket (socket_type);
   if (sockfd < 0) return ERROR;

   server_addr.port= port;
   server_addr.address= netaddress;
   status= connect (sockfd, &server_addr);
   if (status < 0) return ERROR;

   return sockfd;
}


int Send (int sockfd, char *buf, int count ) {

   int status;

   status= send (sockfd, buf, count);
   if (status <> count) return ERROR;

   return OK;
}


int Receive (int sockfd, const char *buf, int count ) {

   int status;

   status= recv (sockfd, buf, count);
   if (recv < 0) return ERROR;

   return OK;
}


int Close (int sockfd) {

   return close (sockfd);
}
```

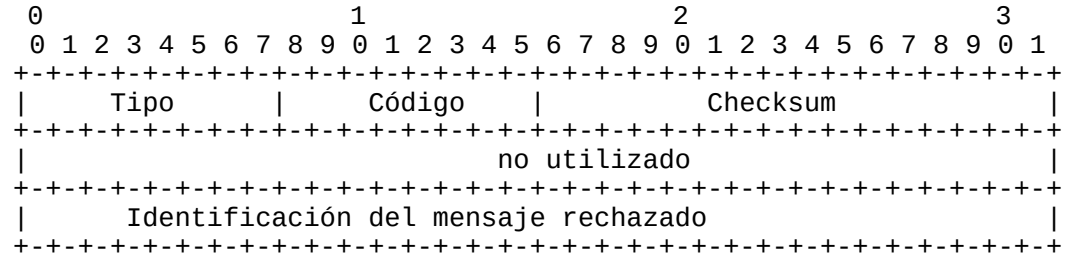
Problema 2:

Se cuenta con un servicio que permite enviar mensajes usando el cabezal básico del IP (header). En éste se cuenta con un campo Time to Live (TTL) que indica la cantidad de saltos (HOPS) que se le aceptará al mensaje. Cada vez que el mensaje atraviesa un router su valor es decrementado en 1 y si su valor es 0 entonces el datragrama es rechazado. Se cuenta con las siguientes primitivas:

envio_mens(header,data)
recibo_mens(header,data)

Al header se le debe proporcionar los campos: *Dirección origen*, *Dirección destino*, *TTL*. El campo *data* contiene los datos enviados.

Cuando un router rechaza un paquete (por haber vencido el TTL), lo hace enviando un mensaje al equipo que lo originó, encapsulado en un datagrama IP. El campo *data* del datagrama es el siguiente:



donde:

Tipo contiene el valor 11
Código contiene el valor 0 = TTL excedido
Identificación del mensaje rechazado contiene el Internet Header de dicho mensaje y sus primeros 64 bits de datos.

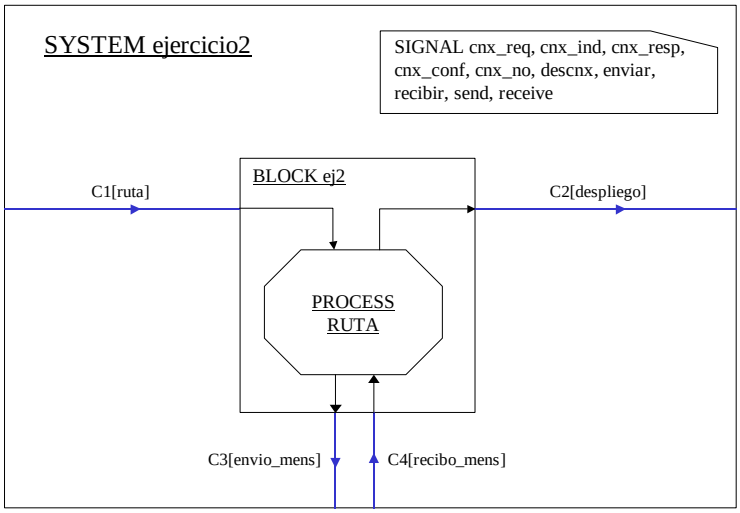
Se pide:

Implementar un protocolo que utilizando el servicio de mensajes anteriormente descripto, despliegue en pantalla la ruta que siguen los datagramas para ir de la máquina donde se originaron a otra destino.

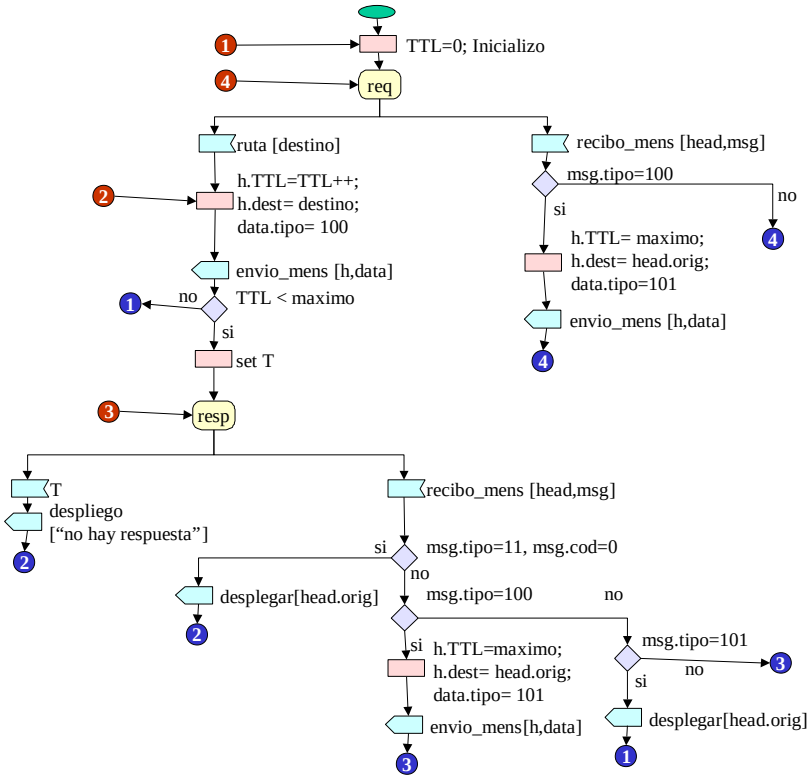
Se supone además que:

- el protocolo que implemente se debe ejecutar en todos los hosts y no en los routers.
- la función recibo_mens() permite acceder a todos los mensajes que llegan.
- se cuenta con una función que dado un mensaje devuelve su cabezal (header).
- se cuenta con una función que permite desplegar en pantalla un string.
- los valores del campo *Tipo* utilizados por los distintos protocolos son inferiores a 96.
- las rutas de los datagramas se mantienen estables.

Se utilizarán los siguientes valores para el campo *Tipo*: 100 – requerimiento, 101 – respuesta.



Process RUTA



Problema 3:

Escribir un programa que ejecuta a nivel de capa 7 utilizando una capa 4 sin conexión pero que requiere servicios con conexión.
Prestar especial atención a los siguientes aspectos:

- pérdida de mensajes,
- duplicación de mensajes,
- mensajes arribados fuera de orden.

Aclaración: Por fines didácticos presentamos una solución más completa que la exigida en el problema.

Esta implementación se centrará en simular el establecimiento de una conexión con el otro lado, y el control de los mensajes enviados y recibidos por esa conexión. No se implementará un servicio que acepte múltiples conexiones. Tampoco se utilizará un esquema cliente-servidor, sino que se tratará a ambos extremos de la supuesta conexión como iguales, pudiendo cualquiera de ellos iniciar una conexión o terminarla, y enviar o recibir mensajes.

Se utilizará un mecanismo tipo llamadas teléfonicas, es decir, si estoy conectado o tratando de iniciar una conexión, y me llega un pedido lo ignoró. Quien invoque al protocolo deberá encargarse de los reintentos.

Se utilizarán datagramas compuestos de un header y un área de datos. El header consiste de las direcciones de la capa 4. Se asume conocida mi dirección, la cual puede hallarse mediante alguna función al comenzar el protocolo. El campo de datos del datagrama los estructuraremos en 4 campos:

```
Type data=  record
                    tipo: (CREQ, CRESP, DREQ, DRESP, DATA, ACK),
                    secuencia: integer,
                    ack: integer,
                    datos: string
            end;
```

El campo tipo indica la operación que se quiere realizar: CREQ – requerimiento de conexión, CRESP – respuesta de conexión, DREQ – requerimiento de desconexión, DAT – envío de datos, ACK – confirmación de datos. El campo secuencia indica el número de secuencia del mensaje enviado, y el campo ack inidica el número de secuencia del último mensaje recibido.
Por un abuso de lenguaje, en el protocolo veremos al datagrama como compuesto de 6 campos: las 2 direcciones del header y los 4 campos de data. Cuando por el tipo de mensaje, no se necesite llenar los últimos campos, se enviarán rellenos, pero los omitiremos en la invocación de las primitivas.

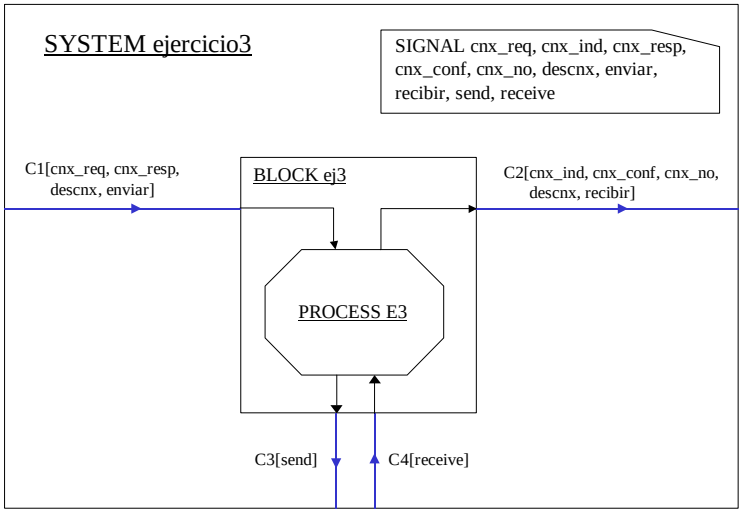
El protocolo de envío y recepción de mensajes tendrá ventanas de emisor y receptor de tamaño 8. Por un abuso de lenguaje se asumirá que las operaciones – y ++ aplicadas sobre los números de secuencia operan a módulo 16, es decir que 15++=0 y 0-1=15.
Las función *llena* indica si la ventana está llena, se utiliza para decidir si aceptar o no mensajes de la aplicación. La función *entra* evalúa si el número de secuencia entra dentro de la ventana, se utiliza para ver si el mensaje llegado del otro lado corresponde a la ventana de recepción. La función *vaciar* vacía la ventana hasta la posición indicada, se utiliza para liberar los buffers del emisor al recibir un ack.

Se utilizará un array para almacenar los tiempos de cada mensaje de la ventana de envío de mensajes. Se seteará un timer con el valor del menor de esos tiempos. El mantenimiento del timer se hará

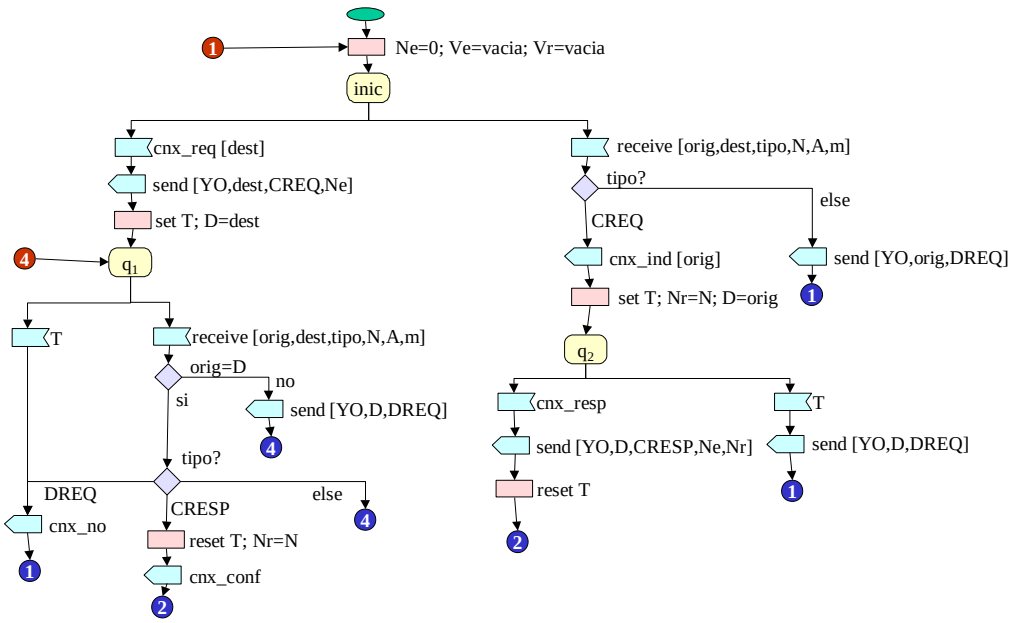
consultando el array. Cada vez que vence el timer, se lo seteará nuevamente con el tiempo correspondiente al siguiente mensaje.
La función *setear* setea el timer con el mínimo tiempo de los mensajes de la ventana. La función *mínimo* devuelve el número de secuencia del mensaje con mínimo tiempo asociado.

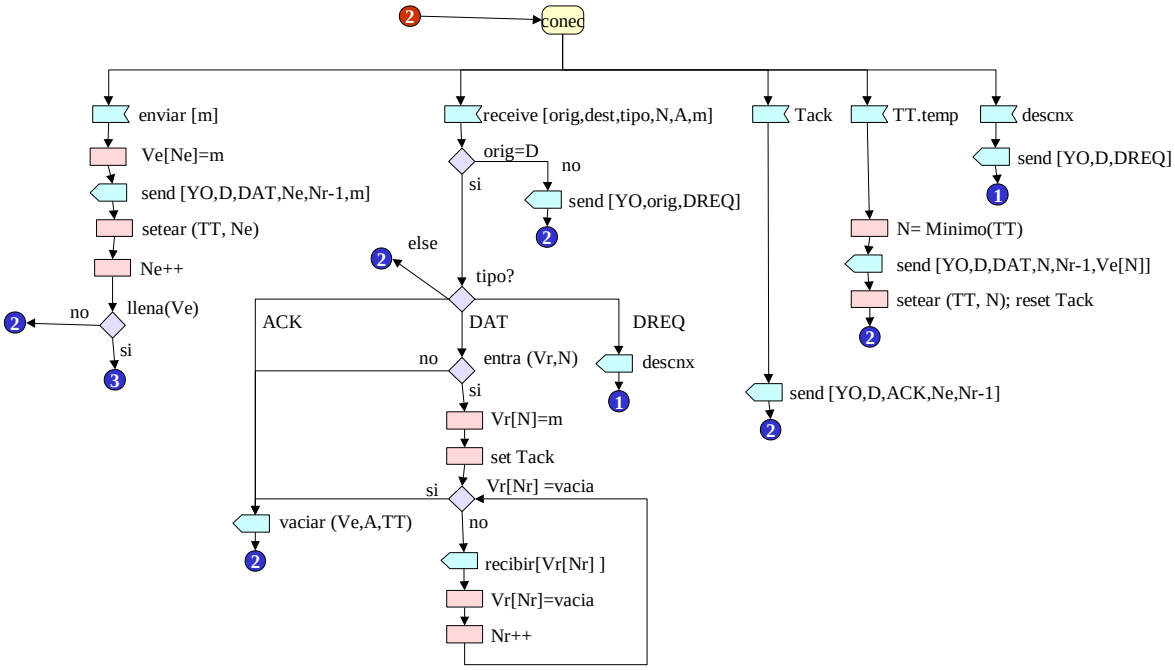
```
TYPE ventana= array[0..15] of string;
        tabla_timer= record
                            temp: timer,
                            val: array[0..15] of time
                     end;

VAR Ne, Nr: 0..15;        // números de secuencia de envío y recepción
       Ve, Vr: ventana;   // ventanas de envío y recepción
       T, Tack: timer;    // temporizadores
       TT: tabla_timer;   // tabla para control de la temporización de paquetes enviados

FUNCTION llena (vent: ventana) RETURN boolean
VAR ocup: 0..8
BEGIN
    ocup=0
    for i=0 to 15 do
        if (vent[a] <> vacia) then ocup++;
    return (ocup=8);
END

FUNCTION entra (vent: ventana, n: 0..15) RETURN boolean
VAR ocup: 0..8
BEGIN
    /*n debe estar entre Ne y Ne+7, cuidando el módulo*/
    if (Ne+7 <= 15) then
        return (n >= Ne and N <= Ne + 7)
    else if (n <= 15) then
        return (n >= Ne)
    else return (n < Ne – 9)
END

PROCEDURE vaciar (vent: ventana, a: 0..7, T:tabla_timer)
VAR Nmin: 0..15;
BEGIN
    while vent[a] <> vacia
        vent[a] = vacia;
        T.val[a] = vacia;
        a= a –1;
    endwhile
    if (T.val <> vacia) then
        Nmin= min (T.val);
        set T.temp, T.val[Nmin] + TIMEOUT;
    else
        reset T.temp;
    endif
END

PROCEDURE setear (T: tabla_timer, N: 0..15)
VAR Nmin: 0..15;
```

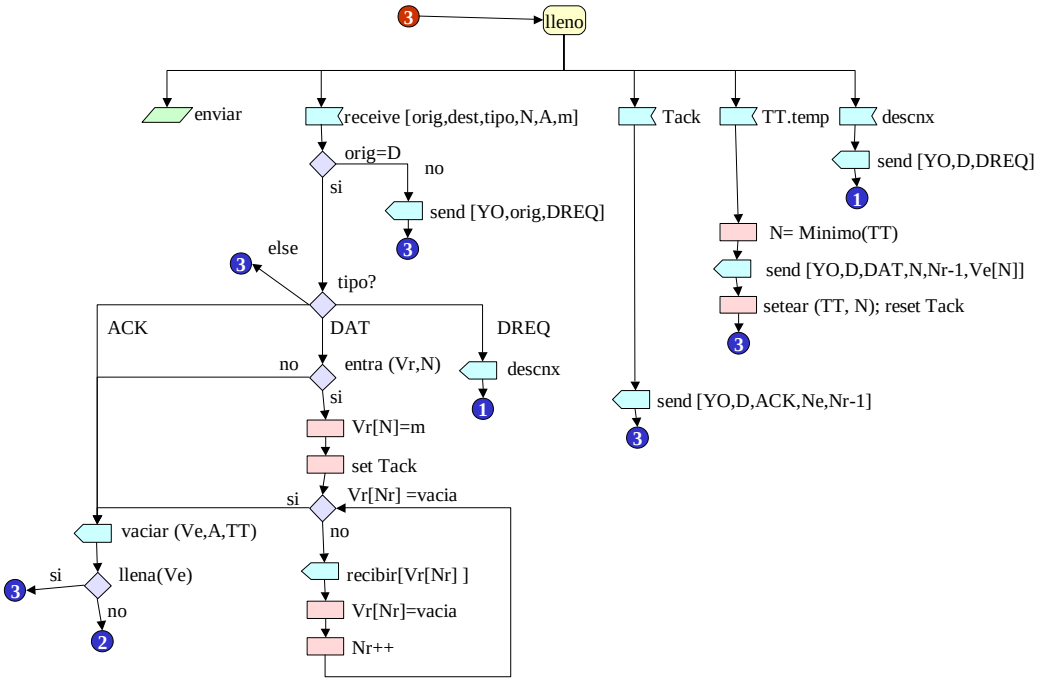
```
BEGIN
    if (T.val = vacia) then set T, TIMEOUT + NOW;
    T.val [N]= NOW;
END;

FUNCTION minimo (T: tabla_timer) return integer
BEGIN
    return min(T.val);
END;
```



Process E3




2
conec
enviar [m]
Ve[Ne]=m
send [YO,D,DAT,Ne,Nr-1,m]
setear (TT, Ne)
Ne++
llena(Ve)
no
si
receive [orig,dest,tipo,N,A,m]
orig=D
no
si
send [YO,orig,DREQ]
else
tipo?
ACK
DAT
DREQ
no
entra (Vr,N)
descnx
si
Vr[N]=m
set Tack
si
Vr[Nr] =vacia
vaciar (Ve,A,TT)
no
recibir[Vr[Nr]]
Vr[Nr]=vacia
Nr++
Tack
send [YO,D,ACK,Ne,Nr-1]
TT.temp
N= Minimo(TT)
send [YO,D,DAT,N,Nr-1,Ve[N]]
setear (TT, N); reset Tack
descnx
send [YO,D,DREQ]


3
lleno
enviar
receive [orig,dest,tipo,N,A,m]
orig=D
no
si
send [YO,orig,DREQ]
else
tipo?
ACK
DAT
DREQ
no
entra (Vr,N)
descnx
si
Vr[N]=m
set Tack
si
Vr[Nr] =vacia
vaciar (Ve,A,TT)
si
llena(Ve)
no
no
recibir[Vr[Nr]]
Vr[Nr]=vacia
Nr++
Tack
send [YO,D,ACK,Ne,Nr-1]
TT.temp
N= Minimo(TT)
send [YO,D,DAT,N,Nr-1,Ve[N]]
setear (TT, N); reset Tack
descnx
send [YO,D,DREQ]