CM

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL

INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

IMPLEMENTACION DEL NIVEL DE PAQUETES DE LA RECOMENDACION X.25

Tesis que presenta el Ing. Luis Angel Trejo Rodríguez para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERIA ELECTRICA con opción en COMPUTACION.

Trabajo dirigido por el Dr. Jan Janeček Hyan.

Becario de CONACYT.

México, D.F., a 18 de agosto de 1989.

G R A C I A S.

Al Centro de Investigación y de Estudios Avanzados del I.P.N.

A mi asesor:
Dr. Jan Janeček Hyan, por sus valiosas orientaciones.

Al Dr. Manuel Guzmán Rentería, con especial respeto por haberme iniciado en este campo y por su continuo apoyo.

Al Dr. Guillermo Morales Luna, ejemplo a seguir en este centro de estudios.

A mis profesores, y en especial a un compañero y amigo, Andrés, sin cuya preciada ayuda muchas cosas no hubieran sido posibles.

A CONACYT.

Con profundo cariño
a mis padres, quienes
siempre han sido mis
guías preferidos.

A mis hermanos,
Daniel y Pedro.

A Cris.

I N D I C E G E N E R A L:

INTRODUCCION.

La Organización Internacional de Normalización (ISO) estableció en 1977 un subcomité para desarrollar estándares para la interconexión de grupos de computadoras heterogéneas, conocido como el subcomité 16 o SC16. Su objetivo principal fue la creación de un conjunto de protocolos de comunicación que permitiría a los diferentes fabricantes de sistemas interconectarse entre sí, siempre y cuando se apegaran a un conjunto de protocolos estandarizados. El "Modelo de Referencia de la Interconexión de Sistemas Abiertos" (OSI) fue terminado en 1979, también conocido como el "Modelo de los 7 niveles".

Por otro lado, el Comité Consultivo Internacional de Telegrafía y Telefonía (CCITT), uno de los principales organismos normalizadores en transmisión de datos, aprobó en 1976 la Recomendación X.25, la cual define la interfaz entre una computadora o terminal y una red pública de datos que trabaje en modo de paquetes.

Recomendación X.25: "INTERFAZ ENTRE EL EQUIPO TERMINAL DE DATOS (ETD) Y EL EQUIPO DE TERMINACION DE CIRCUITO DE DATOS (ETCD) PARA EQUIPOS TERMINALES QUE FUNCIONAN EN EL MODO PAQUETE Y CONECTADOS A REDES PUBLICAS DE DATOS POR CIRCUITOS ESPECIALIZADOS". (Ginebra, 1976; modificada en Ginebra, 1980 y Málaga-Torremolinos, 1984).

El establecimiento en diversos países de redes públicas de datos que proporcionan servicios de transmisión de datos con conmutación de paquetes, hace necesaria la normalización para facilitar el interfuncionamiento internacional.

La Rec. X.25 del CCITT comprende tres niveles de protocolos (Nivel 1 al 3) del modelo OSI: Nivel físico, Nivel enlace y Nivel paquetes. (Fig. 1).

NIVEL 1: Nivel físico, concierne a la transmisión de bits a través de un medio físico.

NIVEL 2: Nivel enlace, incluye el formato de tramas, manejo de errores, garantiza una línea de transmisión confiable.

NIVEL 3: Nivel paquetes o nivel red, controla el tráfico de las diferentes llamadas virtuales y las multiplexa para que pasen por el canal físico.

Estos tres niveles son completamente independientes entre sí, lo que permite se realicen cambios sin afectar la operación de algún otro. Un nivel adyacente es afectado solamente si los cambios afectan la interfase de ese nivel.

Cada nivel desempeña un conjunto de funciones bien definidas usando únicamente un conjunto de servicios bien definidos provistos por el nivel inferior. Estas funciones implementan un conjunto de servicios que serán accesados y utilizados sólo por el nivel superior.

En base a este modelo, se ha realizado la implementación de los tres niveles que comprenden la Rec. X.25. El nivel físico ha sido desarrollado en el Laboratorio de Programación de Sistemas de esta sección por el Ing. Rodolfo Rosado y su equipo de trabajo; el nivel enlace fue objeto de estudio

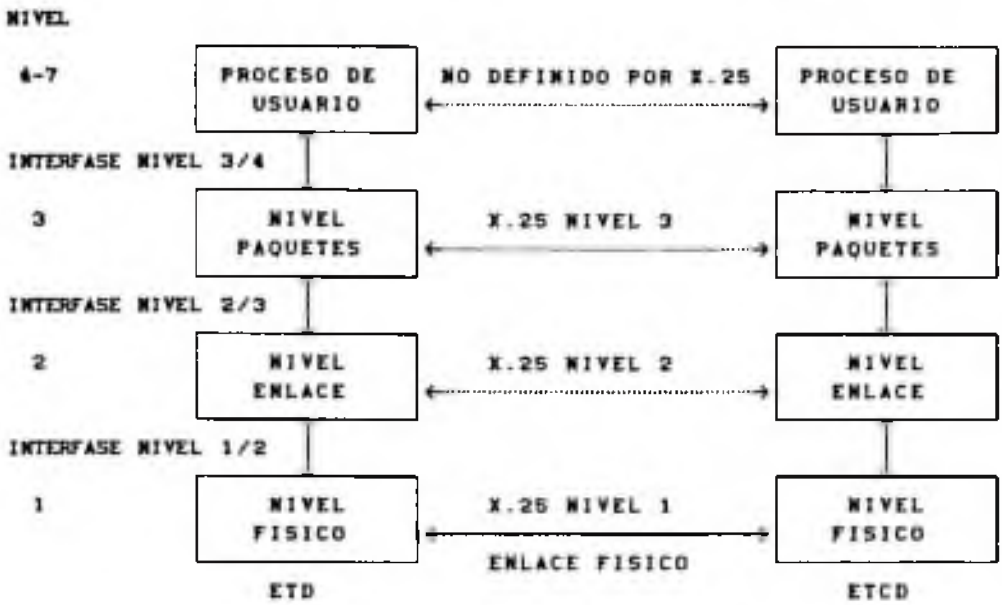


FIG.1 LUGAR DE X.25 EN LA ARQUITECTURA DE NIVELES. LA UNICA CONEXION FISICA REAL ENTRE LAS DOS ESTACIONES (ETD/ETCD) ES EL ENLACE FISICO ENTRE LOS DOS NIVELES FISICOS. LAS OTRAS CONEXIONES MOSTRADAS ENTRE DOS DE LOS MISMOS NIVELES NO SON FISICAS, SINO MAS BIEN CONEXIONES LOGICAS HECHAS POR EL PROTOCOLO DE COMUNICACIONES CORRESPONDIENTE A ESE NIVEL.

en los Laboratorios I y II, habiéndose integrado un equipo para su desarrollo, en el cual participé junto con el Ing. Octavio Juárez y el Ing. Daniel Gómez Galindo, siendo nuestro lider de proyecto el M.en C. César A. Galindo Legaria; finalmente, la implementación del nivel de paquetes es el objetivo de la presente tesis, cuya parte práctica se finalizó en el mes de mayo del año en curso.

El contenido de la tesis está dividido de la siguiente manera:

Parte I.- Descripción general de los Niveles 2 y 3 (Nivel Enlace y Nivel paquetes) de la Rec. X.25 del CCITT.

Permite adquirir una visión general del concepto fundamental de estos dos niveles. Una vez asimilado el funcionamiento y los servicios que ellos proporcionan, es muy sencillo interpretar y comprender la implementación realizada.

La documentación de la implementación del nivel enlace se puede consultar en el reporte final del proyecto de Laboratorios I y II. Este reporte incluye una guía muy sencilla para la comprensión de las funciones principales del chip WD2511 (dispositivo LSI que soporta en su totalidad el nivel 2 de X.25). Recordemos que este chip fue utilizado para la implementación del nivel enlace.

En cuanto al nivel paquetes, se expresa un poco más a detalle su funcionamiento en base a la Rec. X.25 del fascículo VIII.3 del CCITT.

Parte II.- Implementación del Nivel Paquetes de la Rec. X.25.

Se detalla la manera de trasladar la recomendación a programas. Se muestran las estructuras utilizadas, funciones principales, servicios que ofrece el nivel, modo de utilización, características del control de flujo (tamaño de paquetes, tamaño de ventana, numeración de paquetes, etc.) , interfase con el nivel enlace, utilización de los servicios del nivel enlace, etc.

Parte III.- Se presentan dos módulos de prueba; el primero de ellos funciona de una manera interactiva usuarios-terminales, mientras que el segundo crea procesos transmisores y procesos receptores que generan paquetes secuenciales a manera de ráfagas.

Apéndice A.- La implementación de los niveles se llevó a cabo utilizando el lenguaje de programación C junto con un núcleo de concurrencia desarrollado en el mismo lenguaje. Estas primitivas fueron diseñadas por el Dr. Manuel Guzmán Rentería y el Ing. Andrés Vega e implementadas por este último, todo esto en la sección de computación del CINVESTAV. Este apéndice nos da una descripción global del núcleo introduciendo el concepto de programación en tiempo real y finalizando con la explicación de las principales primitivas de que consta el kernel. No contempla la sintaxis de las mismas.

Apéndice B.- Configuración global del PAD. Unicamente lo que concierne al programador del WD2511 y lo referente al manejo de interrupciones para transmisión, recepción, interrupciones generadas por el WD2511 e interrupciones del núcleo de concurrencia o kernel. Direcciones de los puertos de control para la programación de contadores 8253A, controlador de interrupciones 8259A, Usart 8251A y WD2511.

Esperamos que el presente trabajo de tesis constituya un estímulo más para los estudiantes de este centro que enfocan su maestría hacia el área de redes de computadoras y protocolos de comunicación; asimismo que sirva como un punto de referencia para futuras investigaciones en este campo, o más específicamente, para la continuación de este proyecto en lo que se refiere al siguiente nivel, nivel 4, basado en la Recs. X.28 y X.29.

I.A.- NIVEL DE ENLACE DE X.25.

I.A.1.- FUNCIONES DEL NIVEL DE ENLACE.

Los procedimientos especificados por el nivel de enlace de X.25 proveen un mecanismo para poder intercambiar mensajes a través de un enlace de datos. Cada mensaje se envía dentro de una trama. Estos procedimientos definen mecanismos mediante los cuales estas tramas pueden ser transmitidas y recibidas detectando errores originados en el nivel físico, corrigiéndolos por retransmisión; en fin, llevando un control general de los datos transmitidos por el enlace.

El nivel enlace de X.25 está basado en un subconjunto de HDLC(High - Level Data Link Control Procedure) llamado LAPB (Link AccessProcedure Balanced).

I.A.2.- DELIMITADORES DE TRAMA Y TRANSPARENCIA

Todas las tramas de información se delimitan por una secuencia "bandera" que consiste de un bit cero seguido de seis bits uno y un bit cero (01111110). Para evitar la transmisión de esta secuencia dentro de la información se utiliza la técnica del "relleno de bits", que brinda transparencia. Esta técnica consiste en que cada vez que se envían cinco unos seguidos, el transmisor inserta un bit cero que se elimina en la recepción cuando el receptor detecta cinco unos seguidos excepto, por supuesto, cuando se trata de una "bandera".

Estas funciones están generalmente implementadas en el hardware del transmisor/receptor.

Cuando el enlace está ocioso (no se están transmitiendo datos) se envían continuamente "banderas" con el objeto de obtener sincronización.

Los dos últimos bytes antes de la bandera final de cualquier trama se definen como una Secuencia de Verificación de Trama (SVT). La función que genera este número generalmente está implementada en el hardware del transmisor y el receptor (en este caso el WD2511).

I.A.3.- ESTRUCTURA DE LA TRAMA.

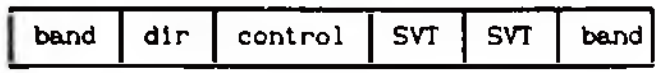
HDCL define los dos primeros bytes que siguen a la bandera de inicio como campos de Dirección y Control, a los que seguirá el campo de Información cuya longitud será variable hasta un límite determinado por las características de los servicios de la red. El número de bytes del campo de información siempre es un número entero.

El campo de dirección puede tomar dos valores:

"A" = 3 (i.e. 00000011 en binario) en tramas que contienen una instrucción del ETCD al ETD y para respuestas a estas instrucciones del ETD al ETCD.

"B" = 1 (i.e. 00000001 en binario) en tramas que contienen una instrucción del ETD al ETCD y para respuestas a estas instrucciones del ETCD al ETD.

Las tramas, entonces, tendrán la estructura siguiente:



o, para tramas con campo de información:

El byte de control define la función de la trama.

Existen tres tipos de tramas: de información, supervisorias y no-numeradas (I-trama, S-trama, U-trama), cuyos formatos son los siguientes:

bits	8	7	6	5	4	3	2	1
I-trama	N(R)			P	N(S)			0
S-trama	N(R)			PF	S		0	1
U-trama	M			PF	M		1	1

Donde N(R) = número secuencial de recepción.
N(S) = número secuencial de transmisión.
S = bits para funciones supervisorias.
M = bits para funciones modificadoras.
PF = bit de Poll/Final.

El bit PF se usa para asegurar integridad en algunas respuestas, evitando la confusión debida a la pérdida de algún "acuse de recibo" (Acknowledgement). Cuando el bit P se activa al enviar un comando, la respuesta deberá activar el bit F. El bit P es pues activado en comandos que requieren respuesta. En caso de no recibirse la respuesta adecuada dentro de un tiempo dado, la trama de comando se retransmitirá.

Las I-tramas se utilizan para el intercambio de información a través del enlace.

Las S-tramas se utilizan para funciones de supervisión del enlace como para el reconocimiento de I-tramas o para solicitar retransmisión o suspensión de envíos.

Las U-tramas realizan el control del enlace sin usar números de secuencia, en particular se utilizan para establecer el enlace a partir de un estado en el que no se conocen dichos números de secuencia o en la recuperación de un estado de error.

CAMPO DE CONTROL

Tipo de trama	Comando	BITS							
		8	7	6	5	4	3	2	1
I	I	N(R)			P	N(S)			0
S	RR	N(R)			PF	0	0	0	1
S	RNR	N(R)			PF	0	1	0	1
S	REJ	N(R)			PF	1	0	0	1
U	SABM	0	0	1	P	1	1	1	1
U	UA	0	1	1	F	0	0	1	1
U	FRMR	1	0	0	F	0	1	1	1
U	DISC	0	1	0	P	0	0	1	1

Las funciones para los diferentes tipos de comandos son:

I (Information).- Para la transmisión de datos a través del enlace a una velocidad determinada por el receptor, con detección y corrección de errores.

RR (Receiver Ready).- Indica al otro extremo del enlace que el receptor está listo para recibir I-tramas, envía el acuse de recibo para las tramas con número de secuencia menor o igual a N(R)-1. También se utiliza para limpiar una condición de ocupado establecida con la transmisión de un RNR.

RNR (Receiver Not Ready).- Se envía para informar al otro extremo que no es posible recibir más I-tramas ya que el buffer de recepción se encuentra lleno. Esta condición puede limpiarse mediante un UA, RR, REJ o SABM. Normalmente se usa un REJ ya que éste señala la trama a partir de la cual debe comenzar la retransmisión.

REJ (REJect).- Solicita la retransmisión de I-tramas comenzando con el indicado en N(R). Se envía esta instrucción cuando el receptor recibe una trama fuera de secuencia. También se usa para limpiar una condición de ocupado (RNR).

SABM (Set Asynchronous Balanced Mode).- Cuando va precedido por una instrucción UA durante el estado de desconexión, indica al ETCD que el ETD desea enlazarse. También se usa para reinicializar el enlace en ambas direcciones cuando éste se encuentra establecido, inicializando las variables de recepción y transmisión (V(R) y V(S)), y perdiendo las I-tramas que no hayan recibido acuse de recibo.

UA (Unnumbered Acknowledgement).- Lo utiliza cualquier extremo para mandar acuse de recibo de una instrucción no-numerada SABM o DISC.

FRMR (FRaMe Reject).- Lo usa cualquier extremo para indicar una condición de error que no puede ser recuperada mediante retransmisiones. La trama tiene un campo de información de tres bytes que sigue al campo de control. El primer byte contiene una copia del campo de control de la trama que se está rechazando; el segundo los valores de V(R) y V(S) del transmisor; el tercero contiene un código que indica el tipo de error que causó la transmisión del FRMR.

Bytes del campo de información:

	8	7	6	5	4	3	2	1
primero	Campo control de la trama rech.							
segundo	V(R)			CR	V(S)			0
tercero	0	0	0	0	Razón			

Donde CR toma el valor de cero si la trama rechazada era una instrucción, en caso contrario (si era una respuesta) toma el valor de uno.

Razón tendrá los valores:

Bits :	4	3	2	1
Campo de control no válido	0	0	0	1
Campo de info no válido	0	0	1	1
Campo de info demasiado largo	0	1	0	0
N(R) no válido	1	0	0	0

DISC (DISConnect).- Indica al receptor que el transmisor cesa su operación, el receptor transmitirá una instrucción UA y se desconectará después de un intervalo T1. Cuando no se encuentra establecido el enlace el ETCD enviará un DISC a intervalos de T1 para indicar al ETD que está listo para aceptar una instrucción para reestablecer el enlace.

La siguiente tabla muestra las respuestas requeridas a cada uno de los comandos de HDLC. Un comando seguido de una 'P' indica que el comando lleva el bit de Poll encendido. Una respuesta seguida de una 'F' indica que la respuesta lleva el bit de Final encendido.

Comando	Respuesta
I	I, RR, RNR, REJ
I(P)	RR(F), RNR(F), REJ(F)
SABM	UA
DISC	UA
SABM(P)	UA(F)
DISC(P)	UA(F)
RR(P)	RR(F), RNR(F), REJ(F)
RNR(P)	RR(F), RNR(F), REJ(F)
REJ(P)	RR(F), RNR(F), REJ(F)

I.A.4.- VARIABLES DE SECUENCIA.

N(R) (Número de secuencia de Recepción).- Es el número de secuencia esperado de la siguiente I-trama a recibir. Le indica al extremo del enlace que se han recibido correctamente todas las I-tramas hasta N(R)-1 (mod 8).

N(S) (Número de Secuencia de Transmisión).- Es el número de secuencia de cada I-trama que se envía.

V(S) (Variable de estado de Transmisión).- Contiene el número de secuencia de la siguiente I-trama por transmitirse. Se incrementa en uno en módulo ocho después de que se transmite una I-trama.

V(R) (Variable de estado de Recepción).- Contiene el valor de la siguiente I-trama a recibir. Cada vez que se recibe una I-trama válida con N(S) igual a V(R), el valor de V(R) se incrementa en uno módulo ocho.

I.A.5.- ALGUNOS PARAMETROS DEL SISTEMA DE ENLACE.

Temporizador T1.- Cuando este tiempo expira se genera la retransmisión de una trama. Su valor debe ser mayor al tiempo que toma transmitir una trama de tamaño máximo y recibir dos tramas similares más el tiempo de propagación de la señal a través del enlace físico.

Temporizador T2.- Es el tiempo máximo que puede transcurrir antes de enviar el acuse de recibo de una trama. Reconocimiento aislado.

Temporizador T3.- Es el tiempo que el ETCD / ETD espera por una instrucción de establecimiento de enlace antes de entrar o regresar al estado de desconexión (link-down). T3 = T1 x N2, donde N2 es el número máximo de retransmisiones de una trama.

N1.- Número máximo de bits en una trama.

N2.- Número máximo de transmisiones de una trama. Incluye la transmisión inicial, normalmente es igual a 20.

I.A.6.- DESCRIPCION DE LA TRANSFERENCIA DE INFORMACION.

Suponiendo que el enlace ya ha sido establecido, cualquier extremo le puede indicar al otro su deseo de recibir datos transmitiendo una trama RR (Receiver Ready) o, todo lo contrario, transmitiendo una trama RNR (Receiver Not Ready).

Cuando la última trama recibida fue un RNR, todas las futuras transmisiones deben suspenderse hasta que se reciba un RR. Suponiendo que ya se ha recibido un RR, es decir, el otro extremo está listo para recibir y el transmisor tiene una trama que enviar, la mandará con N(S) igual al valor actual de V(S), después incrementará V(S) y pondrá en N(R) el valor actual de V(R). Si el temporizador de T1 no está corriendo, se activará en ese momento. Si el valor de V(S) es igual al valor del último N(R) recibido mas siete (número máximo de tramas transmitidas y sin reconocimiento) entonces

las siguientes transmisiones son suspendidas hasta que se reciba una trama que contenga un N(R) mayor (en módulo 8), pudiendo realizar únicamente retransmisiones.

Para que sea aceptada una I-trama y sus datos sean pasados al nivel de paquetes, su SVT debe ser correcta y su N(S) igual al V(R) del receptor; en este caso, el valor de este V(R) se incrementará en uno.

Una I-trama próxima a enviarse llevará en N(R) el valor de V(R); en caso de no haber ninguna I-trama que enviar, se transmitirá ya sea un RR si existe espacio en el buffer de recepción, o un RNR; de cualquier forma, el valor de N(R) se tomará del valor corriente de V(R).

Cualquier I-trama que se reciba con SVT no válida será ignorada.

Cuando se recibe una I-trama con N(S) diferente al V(R) del receptor se descarta y se envía un REJ con N(R) igual a uno más el valor de N(S) de la última I-trama recibida en secuencia.

Un N(R) de cualquier I-trama o S-trama recibido, se considera como el acuse de recibo de todas las tramas hasta el valor de N(R) -1 (mod 8).

El temporizador T1 es reactivado cada vez que se recibe un N(R) (que reconoce hasta N(R)-1 tramas),y queda por lo menos una en espera de reconocimiento. Si este temporizador T1 expira, el procedimiento de retransmisión de tramas comienza.

Cuando se recibe un REJ, el transmisor actualiza su variable V(S) al valor de N(R) contenido en el REJ y comienza a retransmitir desde la trama indicada.

El temporizador T1 se activa cuando se recibe un RNR. Si expira antes que se reciba un RR, un comando supervisorio, ya sea RR, RNR o REJ será enviado con el bit P puesto en 1. Si dentro del periodo de temporización no se recibe una respuesta, la S-trama será retransmitida un total de N2 veces. Después de N2 retransmisiones sin respuesta se reiniciará el enlace transmitiendo un SABM(Set Asynchronous Balanced Mode). Este comando se retransmite a intervalos T1 hasta N2 veces, si no existe aún respuesta, el enlace entra a un estado de desconexión(link-down).

I.A.7.- CONEXION.

Siempre que el ETCD esté activo y sea capaz de llevar a cabo una conexión del nivel de enlace, enviará un DISC al ETD. Este DISC se retransmitirá cada T1 segundos con el bit de Poll encendido, ya que es una retransmisión del DISC original. Cuando el ETD desea reconectarse al ETCD, espera recibir un DISC al que contestará con un UA dentro de un intervalo T1, el cual llevará encendido el bit Final; de lo contrario, el ETCD continuará en el estado de desconexión enviando el comando DISC. Al recibir la instrucción UA, el ETCD esperará por una instrucción SABM durante un periodo T3; en caso de no recibir este SABM volverá a la transmisión de comandos DISC, el primero de los cuales no llevará encendido el bit de Poll. Cuando el ETCD recibe una instrucción SABM del ETD, sin el bit de Poll encendido, contestará con un UA sin encender el bit Final. En este momento se habrá realizado la conexión pudiendo ser enviadas I-tramas en cualquier dirección, con las restricciones de control de flujo ya descritas.

I.A.8.- DESCONEXION.

Para desconectar el enlace el ETD envia un DISC sin activar el bit de Poll. El ETCD responderá transmitiendo un UA, sin activar el bit Final; y esperará por un periodo T3.

Si el ETD desea reestablecer el enlace inmediatamente, transmitirá un SABM tan pronto reciba el UA de respuesta a su DISC. Este SABM deberá ser recibido por el ETCD antes que su temporizador T3 ya activado expire.

I.B.- NIVEL DE PAQUETES DE X.25.

I.B.1.- FUNCIONES DEL NIVEL DE PAQUETES.

Recordemos que X.25 define el formato y el significado de la información intercambiada a través del interfaz ETD/ETCD para los protocolos de los niveles 1, 2 y 3.

El nivel 1 de X.25 trata sobre la parte eléctrica, la mecánica y la funcional de la interfaz ETD/ETCD. En realidad, X.25 no define lo anterior sino que más bien hace referencia a otras dos normas, X.21 y X.21 bis, las cuales definen las interfases digital y analógica, respectivamente.

La función del nivel 2 es la de asegurar una comunicación confiable entre el ETD y el ETCD aún cuando estén conectados por una línea física ruidosa.

El nivel 3 maneja y administra conexiones entre un par de ETDs. Provee dos formas de conexión, llamadas virtuales y circuitos virtuales permanentes. Una llamada virtual es como una llamada telefónica ordinaria: se establece una conexión, se transfiere información y finalmente la conexión es terminada. En contraste, un circuito virtual permanente es como una línea fija, aunque no existe un canal físico exclusivo asignado. Siempre está presente, y el ETD en cualquiera de los dos extremos puede enviar información en el momento que desee sin pasar por la fase de conexión. Los circuitos virtuales permanentes son generalmente usados cuando se maneja un volumen muy grande de información.

Esta información es transferida en forma de paquetes, la norma nos dice:

"Cada paquete que deba transferirse a través del interfaz ETD/ETCD estará contenido dentro del campo de información del nivel de enlace, que delimitará su longitud; el campo de información contendrá un sólo paquete".

MANIPULACION DEL CAMPO DE INFORMACION
PARA LA FORMACION DE PAQUETES

FIG I.1.LOCALIZACION DE LOS PAQUETES DENTRO DE LA ESTRUCTURA DE LA TRAMA DEL NIVEL DE ENLACE.

Cada paquete transferido a través del interfaz ETD/ETCD comprende por lo menos tres octetos. Estos octetos contienen un identificador general de formato, un identificador de canal lógico y un identificador de tipo de paquete. Se añaden otros campos según sea necesario. En el cuadro I.1se indican los tipos de paquetes y su utilización en diversos servicios.

La conexión se realiza de la siguiente manera: cuando un ETD quiere conectarse con otro ETD, construye un paquete de *PETICION DE LLAMADA* y lo pasa al ETCD. La subred se encarga entonces de hacerlo llegar hasta el ETCD destino, el cual lo entrega al ETD destino. Si el ETD destino desea aceptar

TIPO DE PAQUETE		OCTETO 3. BITS
DEL ETCD AL ETD	DEL ETD AL ETCD	8 7 6 5 4 3 2 1
ESTABLECIMIENTO Y LIBERACION DE LA COMUNICACION		
LLAMADA ENTRANTE	PETICION DE LLAMADA	0 0 0 0 1 0 1 1
COMUNICACION ESTABLECIDA	LLAMADA ACEPTADA	0 0 0 0 1 1 1 1
INDICACION DE LIBERACION	PETICION DE LIBERACION	0 0 0 1 0 0 1 1
CONF DE LIBERACION POR EL ETCD	CONF DE LIBERACION POR EL ETD	0 0 0 1 0 1 1 1
DATOS E INTERRUPCION		
DATOS DEL ETCD	DATOS DEL ETD	X X X X X X X 0
INTERRUPCION POR EL ETCD	INTERRUPCION POR EL ETD	0 0 1 0 0 0 1 1
CONF DE INTERRUPCION POR EL ETCD	CONF DE INTERRUPCION POR EL ETD	0 0 1 0 0 1 1 1
CONTROL DE FLUJO Y REINICIO		
RR DEL ETCD(MODULO 8)	RR DEL ETD(MODULO 8)	X X X 0 0 0 0 1
RR DEL ETCD(MODULO 128)[a]	RR DEL ETD(MODULO 128)[a]	0 0 0 0 0 0 0 1
RNR DEL ETCD(MODULO 8)	RNR DEL ETD(MODULO 8)	X X X 0 0 1 0 1
RNR DEL ETCD(MODULO 128)[a]	RNR DEL ETD(MODULO 128)[a]	0 0 0 0 0 1 0 1
	REJ DEL ETD(MODULO 8)[a]	X X X 0 1 0 0 1
	REJ DEL ETD(MODULO 128)[a]	0 0 0 0 1 0 0 1
INDICACION DE REINICIO	PETICION DE REINICIO	0 0 0 1 1 0 1 1
CONF DE REINICIO POR EL ETCD	CONF DE REINICIO POR EL ETD	0 0 0 1 1 1 1 1
REARRANQUE		
INDICACION DE REARRANQUE	PETICION DE REARRANQUE	1 1 1 1 1 0 1 1
CONF DE REARRANQUE POR EL ETCD	CONF DE REARRANQUE POR EL ETD	1 1 1 1 1 1 1 1
DIAGNOSTICO		
DIAGNOSTICO[a]		1 1 1 1 0 0 0 1
REGISTRO[a]		
	PETICION DE REGISTRO	1 1 1 1 0 0 1 1
CONF DE REGISTRO		1 1 1 1 0 1 1 1

CUADRO I.1.IDENTIFICADOR DEL TIPO DE PAQUETE.

a) NO ESTA DISPONIBLE NECESARIAMENTE EN TODAS LAS REDES.

la llamada manda de regreso un paquete de *LLAMADA ACEPTADA*. Cuando el ETD origen recibe el paquete de *LLAMADA ACEPTADA*, el circuito virtual ha sido establecido. (En realidad, este paquete de *LLAMADA ACEPTADA* se conoce como paquete de *COMUNICACION ESTABLECIDA* por tener dirección ETCD-ETD pero es prácticamente el mismo).

En este momento ambos ETDs pueden utilizar la conexión full-duplex para intercambiar paquetes de datos. Cuando cualquiera de los dos desee terminar la comunicación, manda un paquete de *PETICION DE LIBERACION* al otro extremo, el cual contesta con un paquete de *CONFIRMACION DE LIBERACION*. La Fig.I.2 muestra las tres fases en una conexión de X.25.

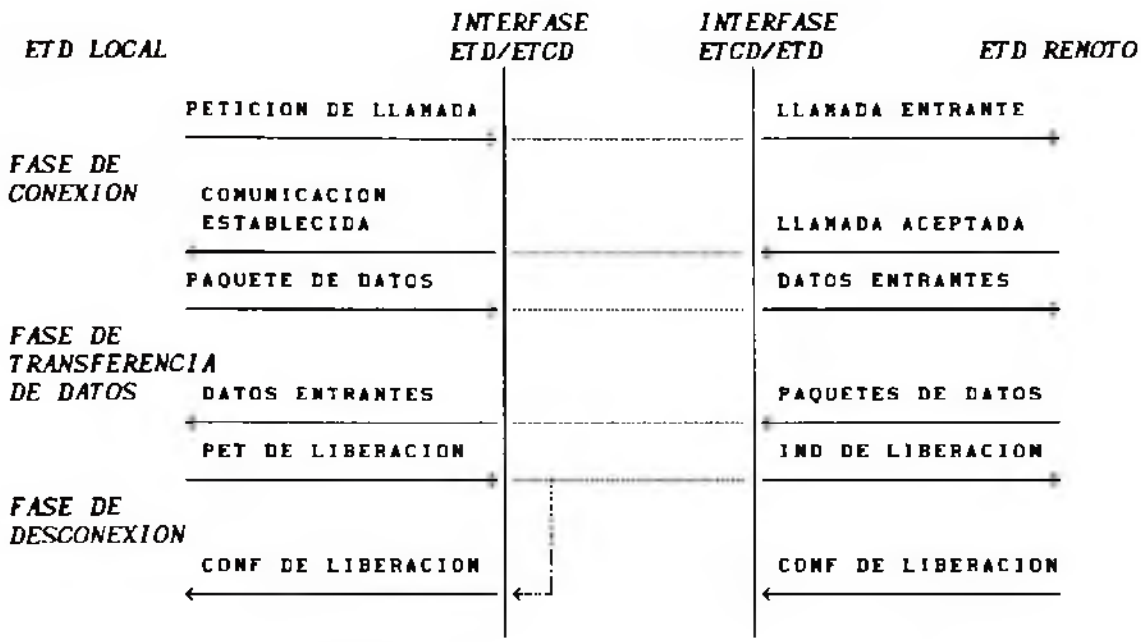


FIG I.2.LAS TRES FASES EN UNA CONEXION DE X.25.

El ETD origen puede escoger cualquier número de circuito virtual que no se esté utilizando en ese momento para identificar la conexión. Si este número de circuito virtual está siendo usado en el ETD, el ETCD destino debe reemplazarlo antes de entregar el paquete. Así, la selección del número de circuito virtual para llamadas salientes es hecha por el ETD y para llamadas entrantes por el ETCD.

Cuando un ETD y un ETCD transfieren simultáneamente un paquete de *PETICION DE LLAMADA* y un paquete de *LLAMADA ENTRANTE*, que especifican el mismo canal lógico, se produce lo que se conoce como colisión de llamadas. El ETCD dará curso a la *PETICION DE LLAMADA* y anulará la *LLAMADA ENTRANTE*.

Para disminuir al mínimo el riesgo de colisión de llamadas, el algoritmo de búsqueda del ETD deberá comenzar por el canal lógico en el estado preparado con el número más alto, mientras que el algoritmo de búsqueda por el ETCD de un canal lógico para una nueva llamada entrante consistirá en utilizar el canal lógico más bajo en el estado preparado.

Cabe señalar que cuando no hay ninguna llamada en curso, se dice que el canal lógico está en el estado preparado.

La Fig.I.3a) muestra el formato del paquete de *PETICION DE LLAMADA*. Este paquete, como todos los demás paquetes de X.25, comienzan por un encabezado de tres byes.

Los campos de GRUPO y CANAL juntos forman un número de circuito virtual de 12 bits. El circuito virtual 0 está reservado para futuras aplicaciones, así que en principio, un ETD tiene hasta 4095 circuitos virtuales abiertos simultáneamente.

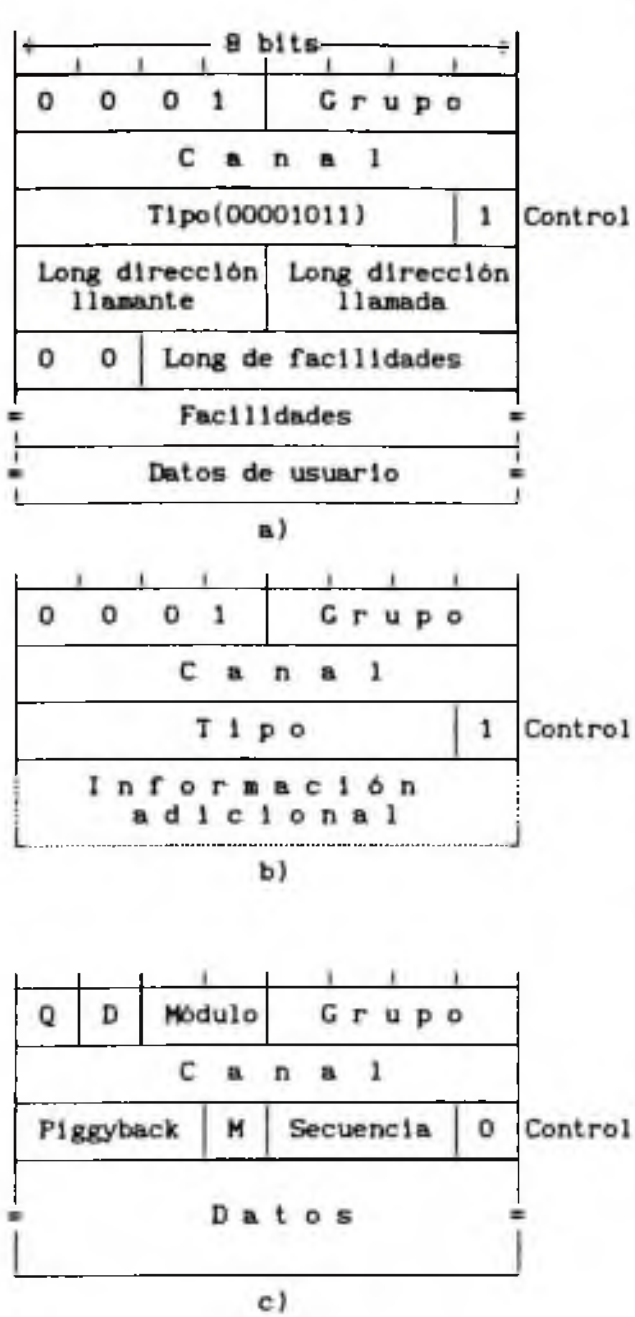
El campo de TIPO en todos los paquetes indica el tipo de paquete de que se trata. El bit de CONTROL es 1 en todos los paquetes de control y 0 en todos los paquetes de datos.

Los siguientes campos de la Fig.I.3a)son exclusivos del paquete de *PETICION DE LLAMADA*. Los siguientes dos campos indican la longitud de las direcciones del ETD que llama y del ETD llamado respectivamente. Ambas direcciones están codificadas en decimal, utilizando 4 bits por dígito.

El campo de LONGITUD DE FACILIDADES indica el número de campos de facilidades que siguen. Estos campos de facilidades son usados para requerir a la red ciertas funciones específicas. Estas funciones varían entre red y red. Un ejemplo de facilidad es el cobro revertido (collect calls). Esta facilidad es de suma importancia para organizaciones que poseen una gran cantidad de terminales remotas que inicializan llamadas hacia un computador central. Si todas estas terminales siempre solicitan la facilidad de cobro revertido, la organización recibirá únicamente un sólo recibo en lugar de cientos de ellos. Una facilidad más es la petición de un circuito virtual en modo simplex, en lugar de full-duplex, por ejemplo.

El ETD que llama puede especificar también una longitud máxima de los paquetes y un tamaño de ventana determinado en lugar de los valores utilizados por omisión, 128 bytes por paquete y tamaño de ventana igual a 2. Si el ETD llamado no acepta la propuesta puede mandar una contrapropuesta en el campo de facilidades del paquete de *LLAMADA ACEPTADA*. Esta contrapropuesta únicamente puede cambiar los valores inicialmente propuestos acercándolos a los valores por omisión, nunca los aleja. Algunos otros ejemplos de facilidades son: numeración secuencial extendida (7 bits), tamaño de ventana, longitud de paquetes, petición de cobro revertido, aceptación de cobro revertido, selección rápida, etc.

El campo de DATOS DE USUARIO permite al ETD enviar hasta 16 bytes de datos dentro del paquete de *PETICION DE LLAMADA*. Los ETDs deciden por sí solos que hacer con esta información. Puede ser usado, por ejemplo, para indicar con que proceso en el ETD llamado se quiere conectar el ETD llamante. Alternativamente puede contener un password.



Tipo	3er. byte
DATOS	PPPMSSS0
PETICION DE LLAMADA	00001011
LLAMADA ACEPTADA	00001111
PET DE LIBERACION	00010011
CONF DE LIBERACION	00010111
INTERRUPCION	00100011
CONF DE INTERRUPCION	00100111
RECEIVE READY	PPP00001
RECEIVE NOT READY	PPP00101
REJECT	PPP01001
PET DE REINICIO	00011011
CONF DE REINICIO	00011111
PET DE REARRANQUE	11111011
CONF DE REARRANQUE	11111111
DIAGNOSTICO	11110001

d)

FIG I.3. FORMATO DE LOS PAQUETES DE X.25. a)PAQUETE DE PETICION DE LLAMADA. b)PAQUETE DE CONTROL. c)PAQUETE DE DATOS. d)CAMPO DE TIPO (P = PIGGY-BACK S = SEQUENCE, M = MORE).

La Fig.I.3b) muestra el formato de los otros paquetes de control. Algunos están formados únicamente por el encabezado; algunos otros tienen uno o dos bytes adicionales. Por ejemplo, el cuarto byte del paquete de *PETICION DE LIBERACION* indica la razón por la cual se está liberando.

El formato del paquete de datos se muestra en la Fig.I.3c). El bit Q indica que se trata de información calificada (Qualified). La norma no indica nada acerca de la diferencia entre información calificada y la no calificada, pero la intención es la de permitir al nivel de transporte y niveles superiores distinguir entre paquetes de control y paquetes de información. El campo de CONTROL siempre es 0 en los paquetes de datos. Los campos SECUENCIA y PIGGYBACK son usados para control de flujo que utiliza un protocolo de ventana deslizante. El número de secuencia es módulo 8 si MODULO es igual a 0 1, y módulo 128 si MODULO es 1 0. (0 0 y 1 1 no son válidos). Si se utiliza el número de secuencia con módulo 128, el encabezado es expandido con un byte extra para acomodar los campos de nueva longitud SECUENCIA y PIGGYBACK. El significado del campo PIGGYBACK es determinado por el valor del bit D. Si D = 0, los reconocimientos subsecuentes significan que únicamente el ETCD local recibió correctamente el paquete, no implica que el ETD remoto lo haya recibido.

El campo de MORE permite al ETD indicar que un grupo de paquetes pertenecen a un mismo mensaje. En un mensaje largo, todos los paquetes tendrán el bit de MORE puesto a 1, excepto el último. Sólamente un paquete lleno podrá tener este bit prendido. La subred es la encargada de agrupar diferentes paquetes y entregarlos como uno sólo y así mismo, del proceso inverso, separar un mensaje en divesos paquetes. Lo anterior es de suma importancia, pues supongamos que una terminal tiene una capacidad de almacenamiento muy pequeña, la subred le entregará únicamente paquetes pequeños acorde a esta capacidad; es decir, el paquete original será fraccionado en diferentes paquetes completos con el bit MORE puesto en 1, menos el último. Por el contrario, esta terminal transmite paquetes pequeños que son agrupados por la subred en uno sólo, siempre que éstos tengan el bit MORE encendido, menos el último.

La Fig.I.3d) muestra los otros paquetes de control.

El paquete de *INTERRUPCION* permite mandar información al igual que el paquete de *DATOS* con la diferencia de que el paquete de *INTERRUPCION* no está sujeto al control de flujo, por lo que no posee un número secuencial de transmisión ni un número secuencial de recepción (reconocimiento). Este paquete es reconocido mediante un paquete de *CONFIRMACION DE INTERRUPCION*.

El paquete RR (Receive Ready) es usado para mandar reconocimientos aislados; es decir, cuando no existe tráfico de regreso donde realizar el "piggy-back". El campo PPP indica el número de paquete esperado. Si el número de secuencia es módulo 128, un byte extra es requerido en su formato.

El paquete RNR (Receive Not Ready) le sirve al ETD para indicarle al ETD remoto que cese temporalmente de enviarle mensajes. RR puede ser usado para indicarle que proceda con la transmisión.

El paquete de *REARRANQUE* y el de *REINICIO* son usados para la recuperación de ciertos errores. Una petición de reinicio se aplica a un circuito virtual específico y tiene el efecto de reinicializar las variables relacionadas con el control de flujo: número de secuencia de transmisión = 0, número del próximo paquete a ser reconocido = 0, número de paquete esperado = 0, valor de las ventanas de transmisión y recepción = 0. Un uso común de la petición de reinicio es para que el ETCD le indique al ETD que la subred se encuentra fuera de servicio. Después de haber recibido el paquete de petición de reinicio, el ETD no tiene forma de saber si quedaron paquetes, previamente transmitidos, sin llegar a su destino. La recuperación en este caso debe ser hecha por el nivel de transporte. Dos bytes extra en el paquete de *PETICION DE REINICIO*, permiten dar una explicación de las causas del reinicio. Esta petición también puede ser hecha por el ETD, por supuesto.

La petición de rearranque es bastante más seria. Es usada cuando un ETD o un ETCD se encuentra caído, por lo tanto se necesitan liberar todos los circuitos virtuales establecidos. Una petición de rearranque equivale a realizar peticiones de reinicio en cada uno de los circuitos virtuales.

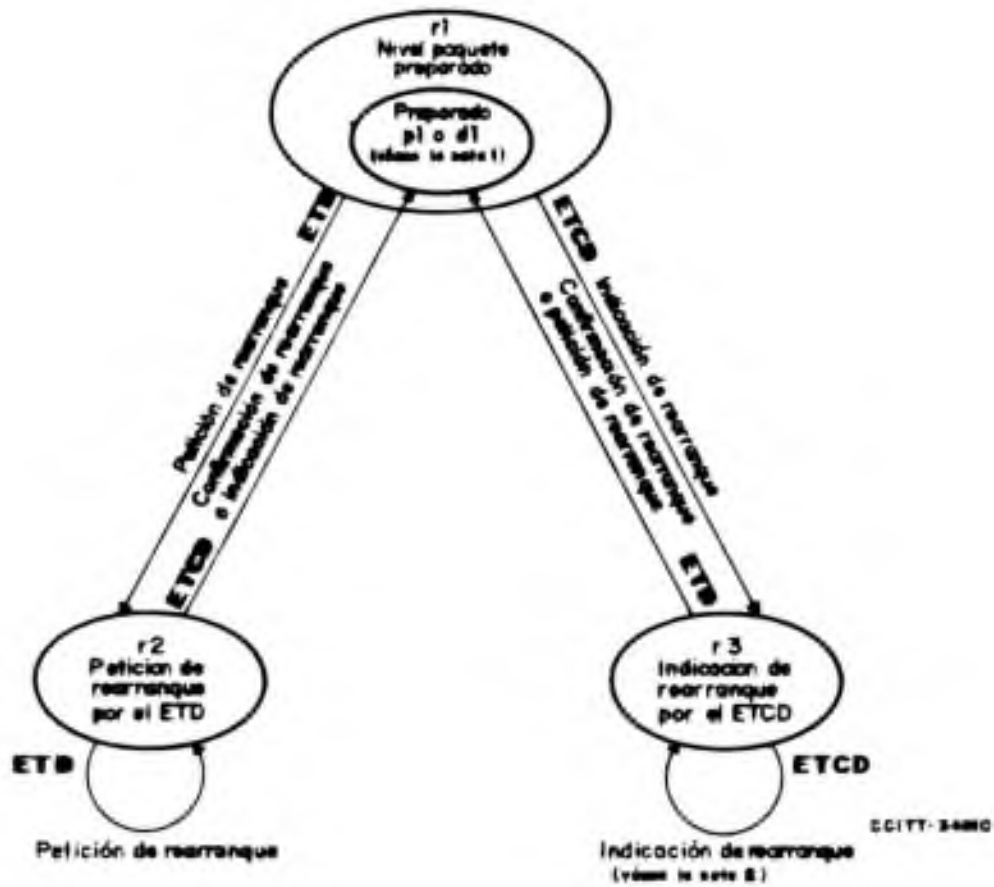
La norma explica los procedimientos anteriores como:

Procedimiento de rearranque: "El procedimiento de rearranque se utiliza para iniciar o reiniciar el interfaz ETD/ETCD en el nivel de paquetes. El procedimiento de rearranque libera simultáneamente todas las llamadas virtuales y reinicia todos los circuitos virtuales permanentes en el interfaz ETD/ETCD".

Procedimiento de reinicio: "El procedimiento se usa para reiniciar la llamada virtual o el circuito virtual permanente, y tiene por efecto la supresión, en cada sentido, de todos los paquetes de *DATOS* y de *INTERRUPCION* que puedan hallarse en la red. Cuando una llamada virtual o un circuito virtual permanente acaba de ser objeto de un reinicio en el interfaz ETD/ETCD, la ventana asociada a cada sentido de transmisión de datos tiene un borde inferior igual a 0, y la numeración de los paquetes de *DATOS* que atraviesen seguidamente el interfaz ETD/ETCD para un sentido de transmisión de datos partirá de 0".

El paquete de *DIAGNOSTICO* permite a la red informarle al usuario de problemas, incluyendo errores en los paquetes transmitidos por el mismo usuario, por ejemplo, campo de TIPO del paquete no definido.

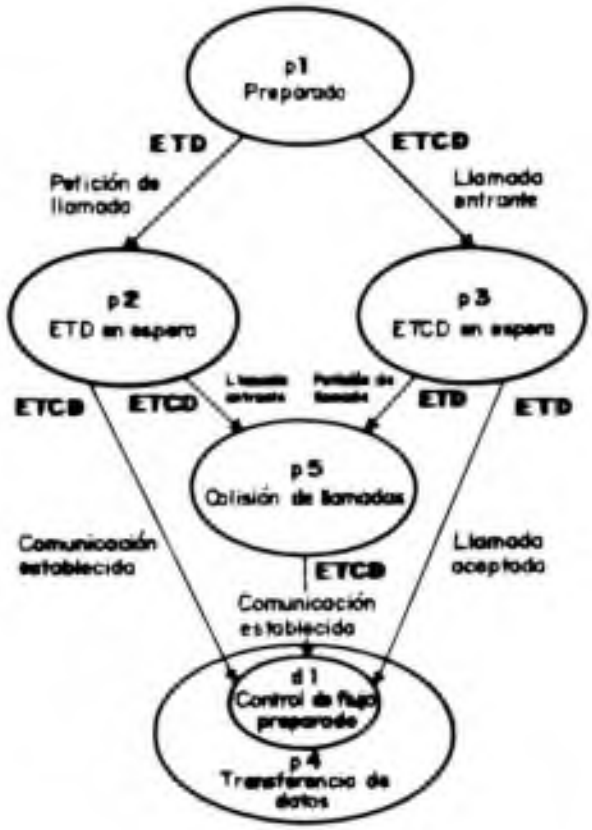
La norma X.25 provee diagramas de estado para describir los procedimientos principales: procedimiento de rearranque, establecimiento de la comunicación, liberación de la comunicación y procedimiento de reinicio. Las Figs. I.4, I.5 y I.6 muestran estos diagramas.



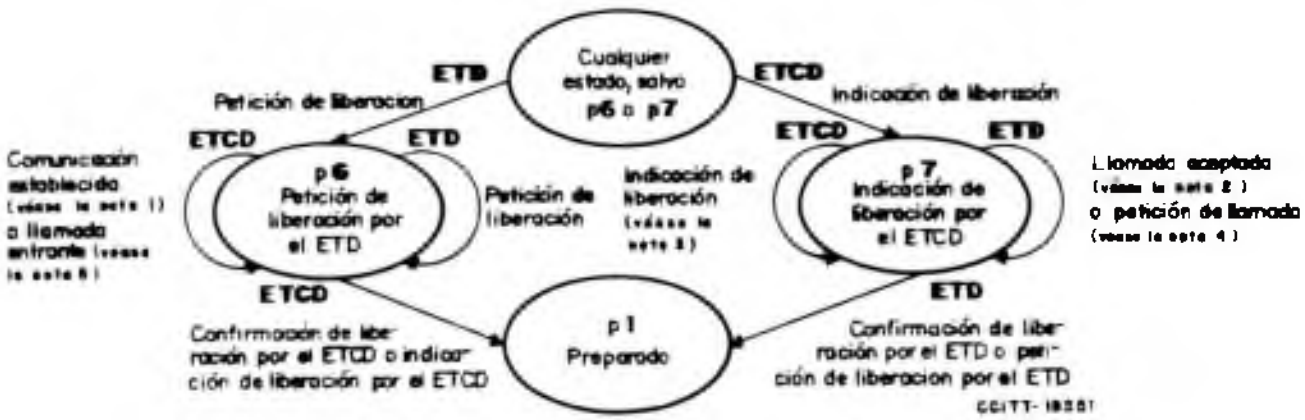
Nota 1 — Estado p1 para llamadas virtuales y estado d1 para circuitos virtuales permanentes.

Nota 2 — Esta transición puede tener lugar después del período de temporización T10.

FIG. I.4. DIAGRAMA DE ESTADOS PARA LA TRANSFERENCIA DE PAQUETES DE REARRANQUE.



a) Fase de establecimiento de la comunicacion



b) Fase de liberación de la comunicación

Nota 1 — Esta transición sólo es posible si el estado anterior era *ETD en espera (p2)*.

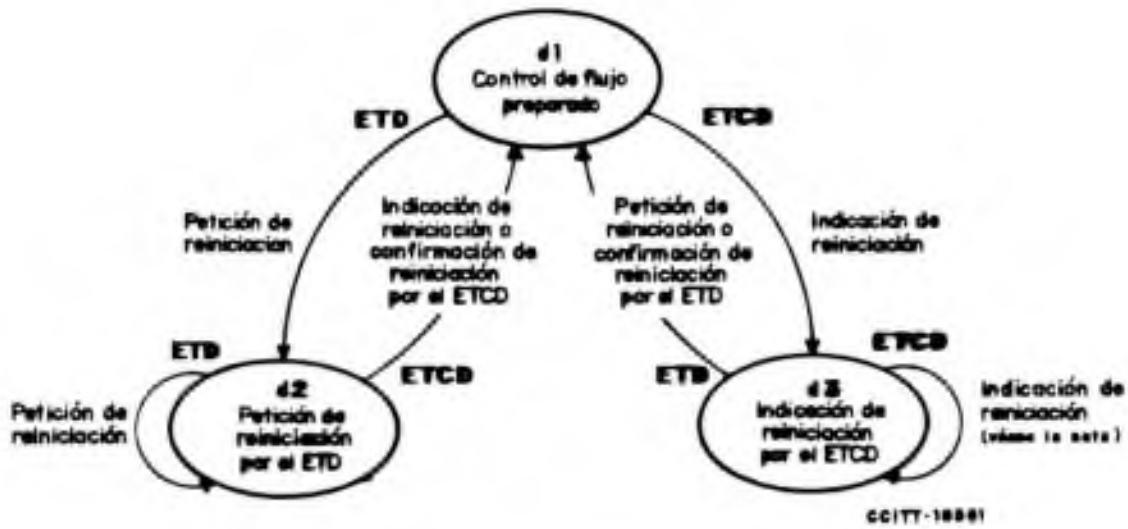
Nota 2 — Esta transición sólo es posible si el estado anterior era *ETCD en espera (p3)*.

Nota 3 — Esta transición puede tener lugar después del periodo de temporización T15.

Nota 4 — Esta transición sólo es posible si el estado anterior era *preparado (p1)* o *ETCD en espera (p3)*.

Nota 5 — Esta transición sólo es posible si el estado anterior era *preparado (p1)* o *ETD en espera (p2)*.

FIG.I.5.DIAGRAMA DE ESTADOS PARA LA TRANSFERENCIA DE PAQ. DE ESTABLECIMIENTO Y LIBERACION DE LA COMUNICACION DENTRO DEL ESTADO DE NIVEL PAQUETE PREPARADO(r1)



Nota — Este temporizador puede tener lugar después del periodo de temporización T12.

FIG I.6. DIAGRAMA DE ESTADOS PARA LA TRANSFERENCIA DE PAQUETES DE REINICIACION DENTRO DEL ESTADO DE TRANSFERENCIA DE DATOS (p4).

I.B.2.- PROCEDIMIENTO DE REARRANQUE.

REARRANQUE POR EL ETD.

El ETD puede pedir en cualquier momento un rearranque transfiriendo por la interfaz ETD/ETCD un paquete de *PETICION DE REARRANQUE*. El interfaz para cada canal lógico se halla entonces en el estado de PETICION DE REARRANQUE POR EL ETD (r2).

El ETCD confirmará el rearranque transfiriendo un paquete de *CONFIRMACION DE REARRANQUE POR EL ETCD*, y haciendo pasar los canales lógicos usados para llamadas virtuales al estado de PREPARADO (p1), y los canales lógicos usados para circuitos virtuales permanentes al estado de CONTROL DE FLUJO PREPARADO (d1).

El tiempo transcurrido en el estado de PETICION DE REARRANQUE POR EL ETD (r2) no excederá del tiempo límite T20.

REARRANQUE POR EL ETCD.

El ETCD puede indicar un rearranque transfiriendo por el interfaz ETD/ETCD un paquete de *INDICACION DE REARRANQUE*. El interfaz para cada canal lógico se halla entonces en el estado de INDICACION DE REARRANQUE POR EL ETCD (r3). En este estado del interfaz ETD/ETCD, el ETCD hará caso omiso de todos los paquetes, excepto de los de *PETICION DE REARRANQUE* Y *CONFIRMACION DE REARRANQUE POR EL ETD*.

El ETD confirmará el rearranque transfiriendo un paquete de *CONFIRMACION DE REARRANQUE POR EL ETD*, y pasando los canales lógicos usados para llamadas virtuales al estado de PREPARADO (p1), y los canales lógicos usados para circuitos virtuales permanentes al estado de CONTROL DE FLUJO PREPARADO (d1).

Cuando el ETD no confirma el rearranque dentro del periodo de temporización T10, el ETCD realiza las acciones necesarias.

I.B.3.- ESTABLECIMIENTO DE LA COMUNICACION.

ESTABLECIMIENTO DE LA COMUNICACION POR EL ETD.

El ETD llamante indicará una petición de llamada transfiriendo un paquete de *PETICION DE LLAMADA* por el interfaz ETD/ETCD. El canal lógico seleccionado por el ETD está en tal caso en el estado ETD EN ESPERA (p2).

El paquete de *PETICION DE LLAMADA* utilizará el canal lógico que se encuentre en el estado PREPARADO (p1) y tenga el número mayor dentro de la gama convenida con la Administración, reduciendo al mínimo el riesgo de colisión de llamadas.

La recepción de un paquete de *COMUNICACION ESTABLECIDA* en el ETD llamante, con el mismo canal lógico que el especificado en el paquete de *PETICION DE LLAMADA*, indica que la llamada ha sido aceptada por el ETD llamado mediante un paquete de *LLAMADA ACEPTADA*. Esto hace pasar el canal lógico especificado al estado TRASFERENCIA DE DATOS (p4).

El tiempo transcurrido en el estado ETD EN ESPERA (p2) no excederá del tiempo límite T21.

ESTABLECIMIENTO DE LA COMUNICACION POR EL ETCD.

El ETCD indicará que hay una llamada entrante transfiriendo por el interfaz ETD/ETCD un paquete de *LLAMADA ENTRANTE*. Esto hace pasar el canal lógico al estado ETCD EN ESPERA (p3).

El paquete de *LLAMADA ENTRANTE* utilizará el canal lógico de número menor que presente el estado PREPARADO (p1).

El ETD llamado indicará su aceptación de la llamada transfiriendo por el interfaz ETD/ETCD un paquete de *LLAMADA ACEPTADA* que especifique el mismo canal lógico que el del paquete de *LLAMADA ENTRANTE*. Esto hace pasar el canal lógico especificado al estado de TRASFERENCIA DE DATOS (p4).

Si el ETD llamado no acepta la llamada mediante un paquete de *LLAMADA ACEPTADA* o no la rechaza mediante un paquete de *PETICION DE LIBERACION* dentro del periodo de temporización T11, el ETCD lo considerará como un error de procedimiento del ETD llamado y liberará la llamada virtual.

I.B.4.- LIBERACION DE LA COMUNICACION

LIBERACION POR EL ETD.

En cualquier momento, el ETD puede indicar la liberación transfiriendo por el interfaz ETD/ETCD un paquete de *PETICION DE LIBERACION*. El canal lógico está en tal caso en el estado PETICION DE LIBERACION POR EL ETD (p6). Cuando el ETCD esté preparado para liberar el canal lógico, transferirá por el interfaz ETD/ETCD un paquete de *CONFIRMACION DE LIBERACION POR EL ETCD* que especifique el canal lógico. El canal lógico queda así en el estado PREPARADO (p1).

El tiempo transcurrido en el estado PETICION DE LIBERACION POR EL ETD (p6) no deberá exceder del tiempo límite T23.

LIBERACION POR EL ETCD.

El ETCD indicará la liberación transfiriendo por el interfaz ETD/ETCD un paquete de *INDICACION DE LIBERACION*. El canal lógico está entonces en el estado INDICACION DE LIBERACION POR EL ETCD (p7). El ETD responderá transfiriendo por el interfaz ETD/ETCD un paquete de *CONFIRMACION DE LIBERACION POR EL ETD*. El canal lógico queda así en el estado PREPARADO (p1).

Cuando el ETD no confirma la liberación dentro del periodo de temporización T13, el ETCD realiza las acciones necesarias.

I.B.5.- PROCEDIMIENTO DE REINICIO.

El procedimiento de reiniciación sólo puede aplicarse en el estado de TRANSFERENCIA DE DATOS (p4) del interfaz ETD/ETCD.

REINICIO POR EL ETD.

El ETD indicará una petición de reinicio transmitiendo un paquete de *PETICION DE REINICIO* que especifique el canal lógico que ha de ser reiniciado. Esto hace pasar el canal lógico al estado de PETICION DE REINICIO POR EL ETD (d2).

El ETCD confirmará el reinicio transmitiendo al ETD un paquete de *CONFIRMACION DE REINICIO POR EL ETCD.* Esto hace pasar el canal lógico al estado de CONTROL DE FLUJO PREPARADO (d1).

El tiempo transcurrido en el estado de PETICION DE REINICIO POR EL ETD (d2) no excederá del tiempo límite T22.

REINICIO POR EL ETCD.

El ETCD indicará un reinicio transmitiendo al ETD un paquete de *INDICACION DE REINICIO* que especifique el canal lógico que se reinicia y el motivo del reinicio. Esto hace pasar el canal lógico al estado de INDICACION DE REINICIO POR EL ETCD (d3). En este estado, el ETCD hará caso omiso de los paquetes de *DATOS*, de *INTERRUPCION*, *RR* y *RNR*.

El ETD confirmará el reinicio transmitiendo al ETCD un paquete de *CONFIRMACION DE REINICIO POR EL ETD.* Esto hace pasar el canal lógico al estado de CONTROL DE FLUJO PREPARADO (d1).

Cuando el ETD no confirma el reinicio dentro del periodo de temporización T12, el ETCD realiza las acciones necesarias.

I.B.6.- EFECTOS DEL NIVEL FISICO Y DEL NIVEL ENLACE SOBRE EL NIVEL PAQUETE.

Un fallo en el nivel físico y/o en el nivel enlace se define como una condición en la cual el ETCD no puede transmitir o no puede recibir ninguna trama debido a condiciones anormales causadas, por ejemplo, por un fallo de la línea situada entre el ETD y el ETCD. Cuando se detecta un fallo en el nivel físico y/o en el nivel de enlace, el ETCD liberará las llamadas virtuales y reiniciará los circuitos virtuales permanentes. Lo hará también cuando el nivel físico y/o el nivel enlace se encuentran fuera de servicio por otras causas. Una condición de fuera de servicio en el nivel enlace, por ejemplo, incluye la recepción por el ETCD de una instrucción DISC.

Cuando desaparece el fallo en el nivel físico y en el nivel enlace, se activará el procedimiento de rearranque y se transmitirá al extremo distante de cada circuito virtual permanente una reiniciación con la causa "ETD distante operacional".

La norma X.25 original (1976) es muy parecida a la actualmente descrita. No incluía el bit D, el paquete de *DIAGNOSTICO*, negociación de la longitud de paquetes y tamaño de la ventana, ni algunos otros detalles. Sin embargo, hubo una gran demanda por la existencia del servicio de datagramas además de circuitos virtuales. Los Estados Unidos y el Japón, presentaron propuestas (bastante conflictivas) de la arquitectura para el servicio de datagramas. El CCITT aceptó ambas propuestas en 1980, haciendo la norma aún más complicada de lo que ya era. Algunos años después se encontró que nadie estaba utlizando datagramas por lo que este concepto fue separado de la norma de 1984.

A pesar de esto, la demanda de este servicio continuó. Aplicaciones como puntos de venta, autorización de tarjetas de crédito, etc., tienen la característica de que se inicia una llamada para hacer alguna petición y la parte llamada únicamente otorga su respuesta, sin requerir de la conexión después de esto. Para satisfacer esta demanda, el CCITT agregó la facilidad de "Selección Rápida" en 1984, la cual es usada en el paquete de *PETICION DE LLAMADA*, que se ha expandido para incluir 128 bytes de datos de usuario. Para la red, este paquete sigue siendo únicamente un intento para establecer un circuito virtual. El ETD puede rechazar la petición de llamada mediante un paquete de *PETICION DE LIBERACION*, el cual también ha sido expandido con 128 bytes para incluir la información de respuesta. Sin embargo, la llamada también puede ser aceptada, en cuyo caso, el circuito virtual es establecido de manera normal.

II.- IMPLEMENTACION DEL NIVEL DE PAQUETES DE X.25.

II.1.- ESTADOS DEL NIVEL DE PAQUETES.

A partir de los diagramas de las figs. I.4, I.5 y I.6 (pags. I.15, I.16 y I.17), se definieron los diferentes estados probables en los que puede estar una interfaz ETD/ETCD correspondiente a un circuito virtual determinado.

Fue necesario especificar un poco más, y en cierto grado cambiar el significado de cada uno de los estados. Por ejemplo, para el diagrama de transferencia de paquetes de *REARRANQUE*, se tienen los estados r1(nivel paquete preparado), r2(petición de rearranque por el ETD) y r3(indicación de rearranque por el ETCD). El estado r1 queda como tal, pero r2 y r3 se funden para formar dos estados diferentes:

r23E: Petición/Indicación de rearranque por el ETD/ETCD extremo.

r23L: Indicación/Petición de rearranque por el ETCD/ETD local.

Este tipo de estados posee dos valores. Por ejemplo, r23E puede significar petición de rearranque por el ETD extremo o bien indicación de rearranque por el ETCD extremo.

De la misma manera, r23L puede tomar los valores indicación de rearranque por el ETCD local o petición de rearranque por el ETD local.

El valor que toma cada estado depende de si es el ETD o el ETCD el que está viendo la interfaz.

La fig. II.1 muestra el lugar de la interfaz ETD/ETCD local y el de la interfaz ETCD/ETD remota en una red pública de datos.

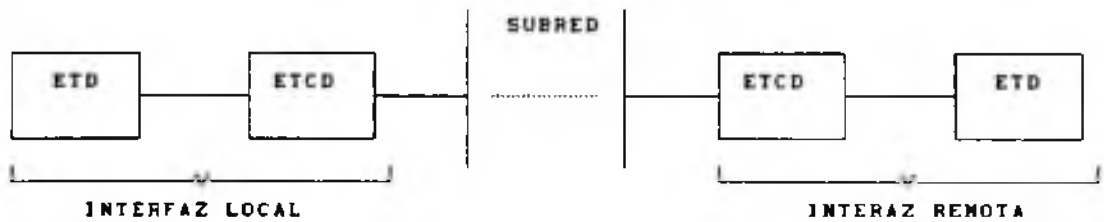
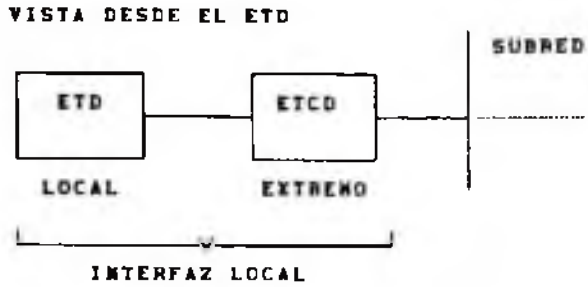
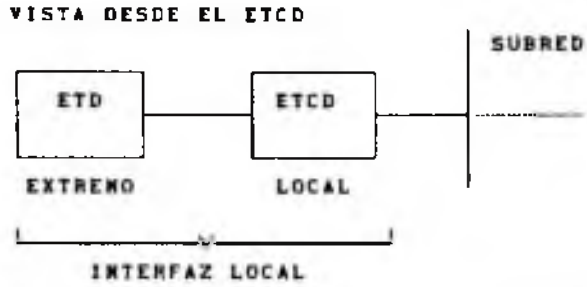


FIG.II.1. INTERFAZ ETD/ETCD LOCAL E INTERFAZ ETCD/ETD REMOTA EN UNA RED PUBLICA DE DATOS.

Cuando se dice ETD o ETCD "extremo" se refiere al ETD o ETCD del otro lado de la interfaz ETD/ETCD local, y cuando se dice ETD o ETCD "local" se refiere al lado desde el cual se está viendo la intefaz ETD/ETCD local (fig. II.2).



a) INTERFAZ LOCAL VISTA DESDE EL ETD. EL ETD ES LA PARTE "LOCAL" Y EL ETCD LA PARTE "EXTREMO".



b) INTERFAZ LOCAL VISTA DESDE EL ETCD. EL ETCD ES LA PARTE "LOCAL" Y EL ETD LA PARTE "EXTREMO".

FIG. 11.2. SIGNIFICADO DE "EXTREMO" Y "LOCAL" EN LA INTERFAZ ETD/ETCD LOCAL.

Para los subsecuentes dibujos, el sentido de la flecha representa la dirección con la cual se transmite un paquete. La parte de arriba de la flecha se divide en dos: estado actual de la interfaz y el tipo de paquete transmitido. La parte de abajo de la flecha representa el estado al cual pasa la interfaz después de haberse transmitido el paquete. Arriba del cuadro a doble línea se indica el temporizador activado por el ETD o ETCD en espera de respuesta al paquete transmitido.

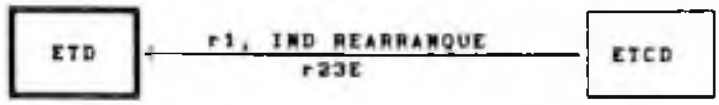
Ejemplo para el procedimiento de rearranque:

Supongamos que la interfaz local ETD/ETCD se encuentra en el estado de r1(nivel paquete preparado) para todos los circuitos virtuales.

* Desde el punto de vista del ETD en la interfaz local:

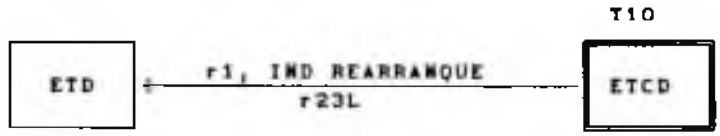
Si el ETD inicia un rearranque, lo hace transmitiendo por la interfaz ETD/ETCD un paquete de *PETICION DE REARRANQUE*, pasando la interfaz al estado r23L(petición de rearranque por el ETD local).

Si es el ETCD el que inicia el rearranque, lo hace transmitiendo por la interfaz ETD/ETCD un paquete de *INDICACION DE REARRANQUE*, el cual al llegar al ETD hace que este último entre al estado r23E(indicación de rearranque por el ETCD extremo).

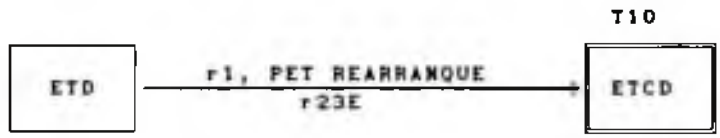


* Desde el punto de vista del ETCD en la interfaz local:

Si el ETCD inicia un rearranque, lo hace transmitiendo por la interfaz ETD/ETCD un paquete de *INDICACION DE REARRANQUE*, pasando la interfaz al estado r23L(indicación de rearranque por el ETCD local).



Si es el ETD el que inicia el rearranque, lo hace transmitiendo por la interfaz ETD/ETCD un paquete de *PETICION DE REARRANQUE*, el cual al llegar al ETCD hace que este último entre al estado r23E(petición de rearranque por el ETD extremo).



De esta manera se manejan todos los demás estados en los que una interfaz ETD/ETCD puede estar, para un circuito virtual determinado.

Estos estados se muestran en el cuadro II.1. Este cuadro representa el autómata de recepción para el nivel de paquetes de X.25. A partir de él, la programación es casi directa, pues indica las acciones a ejecutar cuando se recibe algún tipo de paquete correspondiente a un circuito virtual específico, dado el estado de dicho circuito virtual.

El significado de los estados es:

```
p1        :Preparado.
p23E      :ETD/ETCD extremo en espera.
p23L      :ETCD/ETD local en espera.
d1        :Control de flujo preparado.
d23E      :Petición/Indicación de reinicio por el ETD/ETCD extremo.
d23L      :Indicación/Petición de reinicio por el ETCD/ETD local.
p5        :Colisión de llamadas.
p67E      :Petición/Indicación de liberación por el ETD/ETCD extremo.
p67L      :Indicación/Petición de liberación por el ETCD/ETD local.
r23E      :Petición/Indicación de rearranque por el ETD/ETCD extremo.
r23L      :Indicación/Petición de rearranque por el ETCD/ETD local.
```

Para los paquetes:

```
1: Petición/Indicación de rearranque.
2: Confirmación de rearranque.
3: Llamada. (Llamada entrante / Petición de llamada)
4: Confirmación de llamada.
5: Petición/Indicación de liberación.
6: Confirmación de liberación.
7: Petición/Indicación de reinicio.
8: Confirmación de reinicio.
9: Datos, Interrupción o Control de flujo.
PAQ_ERR_1: Paquete de longitud menor a 3 bytes.
PAQ_ERR_2: Paquete con identificador general de formato no válido.
PAQ_ERR_3: Paquete con canal lógico no asignado.
PAQ_ERR_4: Paquete distinto al de rearranque, confirmación de rearran -
           que y diagnóstico con canal lógico igual a 0.
PAQ_ERR_5: Paquete de rearranque, confirmación de rearranque o diagnós-
           tico con canal lógico diferente de 0.
PAQ_ERR_6: Paquete con identificador del tipo de paquete no válido.
```

II.2.- DESCRIPCION DE LOS ESTADOS.

El nombre de los estados corresponde exactamente a los estados representados en los diagramas de las figs.I.4, I.5 y I.6, a excepción de aquellos compuestos de dos números y una letra.

Para poder hacer una correspondencia entre estos estados y los diagramas, es necesario considerar lo siguiente:

ESTADO XXXL: Significa que en la interfaz ETD/ETCD:

- Vista desde el ETD, este fue el que inició el procedimiento. El ETD es el lado local en la interfaz ETD/ETCD.

- Vista desde el ETCD, este fue el que inició el procedimiento. El ETCD es el lado local en la interfaz ETD/ETCD.

ESTADO XXXE: Significa que en la interfaz ETD/ETCD:

* Vista desde el ETD, el ETCD fue el que inició el procedimiento. El ETCD es el lado extremo en la interfaz ETD/ETCD.

* Vista desde el ETCD, el ETD fue el que inició el procedimiento. El ETD es el lado extremo en la interfaz ETD/ETCD.

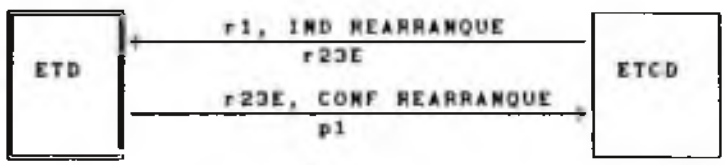
Estado r23L (Indicación/Petición de rearranque por el ETCD/ETD local).

* Vista desde el ETD: Del estado r1, el ETD transmitió un paquete de *PETICION DE REARRANQUE* y activó el temporizador T20 en espera de respuesta a su petición. La interfaz pasa al estado r2(petición de rearranque por el ETD) para el diagrama de la fig.I.4 y al estado r23L para la implementación.

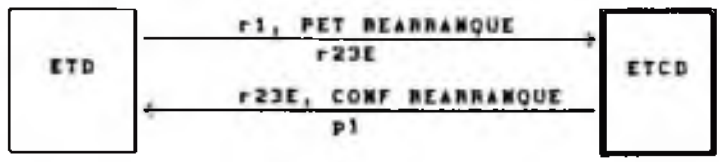
* Vista desde el ETCD: Del estado r1, el ETCD transmitió un paquete de *INDICACION DE REARRANQUE* y activó el temporizador T10 en espera de respuesta a su indicación. La interfaz pasa al estado r3(indicación de rearranque por el ETCD) para el diagrama de la fig.I.4 y al estado r23L para la implementación.

Estado r23E (Petición/Indicación de rearranque por el ETD/ETCD extremo).

* Vista desde el ETD: Del estado r1, el ETD recibió del ETCD un paquete de *INDICACION DE REARRANQUE*. La interfaz pasa momentáneamente al estado r3(indicación de rearranque por el ETCD) para el diagrama de la fig.I.4 y al estado r23E para la implementación. El ETD transmite un paquete de *CONFIRMACION DE REARRANQUE* en respuesta a la indicación de rearranque del ETCD. La interfaz pasa al estado p1(preparado).

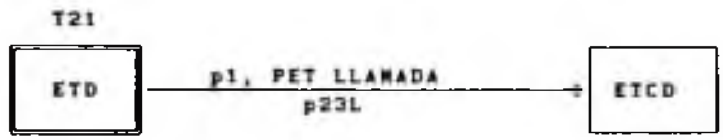


* Vista desde el ETCD: Del estado r1, el ETCD recibió del ETD un paquete de *PETICION DE REARRANQUE*. La interfaz pasa momentáneamente al estado r2(petición de rearranque por el ETD) para el diagrama de la fig.I.4 y al estado r23E para la implementación. El ETCD transmite un paquete de *CONFIRMACION DE REARRANQUE* en respuesta a la petición de rearranque del ETD. La interfaz pasa al estado p1(preparado).



Estado p23L (ETCD/ETD local en espera).

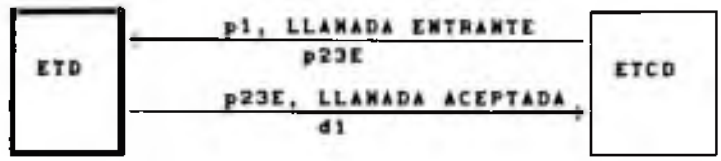
* Vista desde el ETD: Del estado p1, el ETD transmitió un paquete de *PETICION DE LLAMADA* y activó el temporizador T21 en espera de respuesta a su petición. La interfaz pasa al estado p2(ETD en espera) para el diagrama de la fig.I.5 y al estado p23L para la implementación.



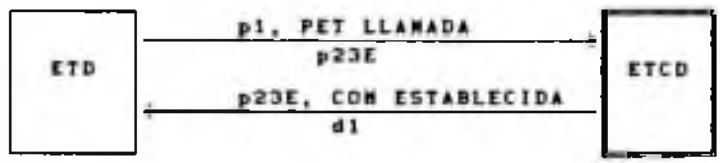
* Vista desde el ETCD: Del estado p1, el ETCD transmitió un paquete de *LLAMADA ENTRANTE* y activó el temporizador T11 en espera de respuesta a su indicación. La interfaz pasa al estado p3(ETCD en espera) para el diagrama de la fig.I.5 y al estado p23L para la implementación.

Estado p23E (ETD/ETCD extremo en espera).

* Vista desde el ETD: Del estado p1, el ETD recibió del ETCD un paquete de *LLAMADA ENTRANTE*. La interfaz pasa momentáneamente al estado p3(ETCD en espera) para el diagrama de la fig.I.5 y al estado p23E para la implementación. El ETD transmite un paquete de *LLAMADA ACEPTADA* en respuesta al paquete de *LLAMADA ENTRANTE* del ETCD. La interfaz pasa al estado d1(control de flujo preparado).

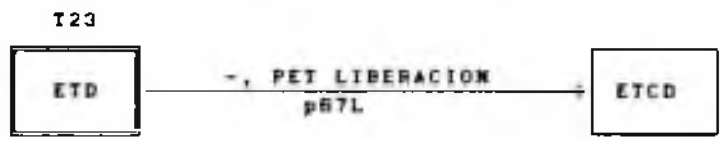


* Vista desde el ETCD: Del estado p1, el ETCD recibió del ETD un paquete de *PETICION DE LLAMADA*. La interfaz pasa momentáneamente al estado p2(ETCD en espera) para el diagrama de la fig.I.5 y al estado p23E para la implementación. El ETCD transmite un paquete de *COMUNICACION ESTABLECIDA* en respuesta al paquete de *PETICION DE LLAMADA* del ETD. La interfaz pasa al estado d1(control de flujo preparado).

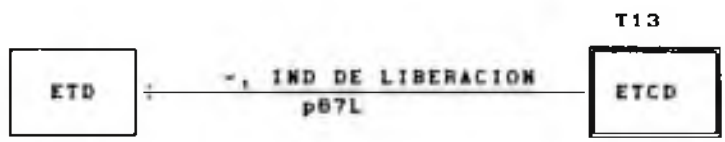


Estado p67L (Indicación/Petición de liberación por el ETCD/ETD local).

* Vista desde el ETD: De cualquier estado, salvo p6 o p7, el ETD transmitió un paquete de *PETICION DE LIBERACION* y activó el temporizador T23 en espera de respuesta a su petición. La interfaz pasa al estado p6(petición de liberación por el ETD) para el diagrama de la fig.I.5 y al estado p67L para la implementación.

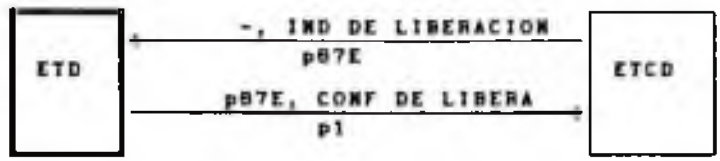


* Vista desde el ETCD: De cualquier estado, salvo p6 o p7, el ETCD transmitió un paquete de *INDICACION DE LIBERACION* y activó el temporizador T13 en espera de respuesta a su indicación. La interfaz pasa al estado p7(indicación de liberación por el ETCD) para el diagrama de la fig.I.5 y al estado p67L para la implementación.

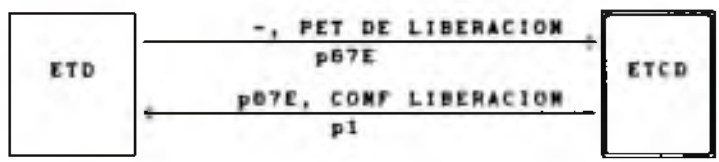


Estado p67E (Petición/Indicación de liberación por el ETD/ETCD extremo).

* Vista desde el ETD: De cualquier estado, salvo p6 o p7, el ETD recibió del ETCD un paquete de *INDICACION DE LIBERACION*. La interfaz pasa momentáneamente al estado p7(indicación de liberación por el ETCD) para el diagrama de la fig.I.5 y al estado p67E para la implementación. El ETD transmite un paquete de *CONFIRMACION DE LIBERACION* en respuesta al paquete de *INDICACION DE LIBERACION* del ETCD. La interfaz pasa al estado p1(preparado).



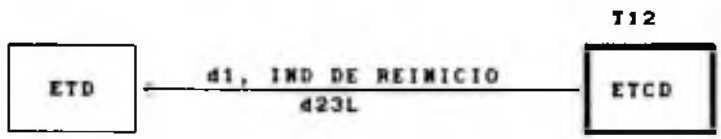
* Vista desde el ETCD: De cualquier estado, salvo p6 o p7, el ETCD recibió del ETD un paquete de *PETICION DE LIBERACION*. La interfaz pasa momentáneamente al estado p6(petición de liberación por el ETD) para el diagrama de la fig.I.5 y al estado p67E para la implementación. El ETCD transmite un paquete de *CONFIRMACION DE LIBERACION* en respuesta al paquete de *PETICION DE LIBERACION* del ETD. La interfaz pasa el estado p1(preparado).



Estado d23L (Indicación/Petición de reinicio por el ETCD/ETD local).

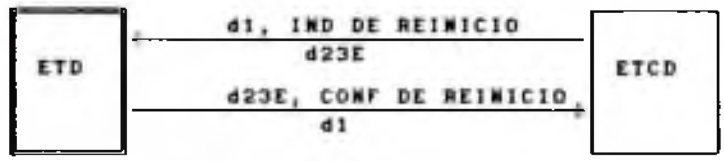
* Vista desde el ETD: Del estado d1, el ETD transmitió un paquete de *PETICION DE REINICIO* y activó el temporizador T22 en espera de respuesta a su petición. La interfaz pasa al estado d2(petición de reinicio por el ETD) para el diagrama de la fig.I.6 y al estado d23L para la implementación.

* Vista desde el ETCD: Del estado d1, el ETCD transmite un paquete de *INDICACION DE REINICIO* y activó el temporizador T12 en espera de respuesta a su indicación. La interfaz pasa al estado d3(indicación de reinicio por el ETCD) para el diagrama de la fig.I.6 y al estado d23L para la implementación.

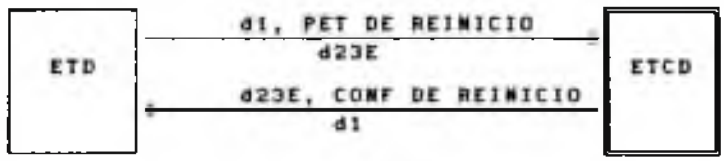


Estado d23E (Petición/Indicación de reinicio por el ETD/ETCD extremo).

* Vista desde el ETD: Del estado d1, el ETD recibió del ETCD un paquete de *INDICACION DE REINICIO*. La interfaz pasa momentáneamente al estado d3(indicación de reinicio por el ETCD) para el diagrama de la fig.I.6 y al estado d23E para la implementación. El ETD transmite un paquete de *CONFIRMACION DE REINICIO* en respuesta al paquete de *INDICACION DE REINICIO* del ETCD. La interfaz pasa al estado d1(control de flujo preparado).



* Vista desde el ETCD: Del estado d1, el ETCD recibió del ETD un paquete de *PETICION DE REINICIO*. La interfaz pasa momentáneamente al estado d2(petición de reinicio por el ETD) para el diagrama de la fig.I.6 y al estado d23E para la implementación. El ETCD transmite un paquete de *CONFIRMACION DE REINICIO* en respuesta al paquete de *PETICION DE REINICIO* del ETD. La interfaz pasa al estado d1(control de flujo preparado).



ESTADO / PAQUETE	r1								r23E	r23L
	p1	p23E	p23L	p4			p5	p67E	p67L		
				d1	d23E	d23L					
Rearranque	N O R M A L (r23E)									DESCARTAR	NORM (r1)
Conf Rearr	E R R O R (r23L) # 17									E R R O R (r23L) #18	D E S C A R T A R
Llamada	NOR(p23E)	ERR (p67L) #21	NOR(p5)	ERROR(P67L) #23			ERR (p67L) #24	ERR (p67L) #25	DESC		
Conf llamada	ERR(p67L)		NOR(d1)								
Liberacion	N O R M A L (p67E)							DESC	NORM (p1)		
Conf Libera	E R R O R (p67L) #20	E R R (p67L) #21	E R R (p67L) #22	E R R O R (p67L) #23			ERR (p67L) #24	ERR (p67L) #25	DESC		
Reinicio				NOR(d23E)	DESC	NORM (d1)					
Conf Reini				ERR(d23L)#27	ERR (d23L) #28						
Datos,Int,CF				NOR(d1)		DESC					
PAQ_ERR 5	ERR(p67L) #41			ERR(d23L)#41			ERR(p67L) #41			ERR(r23L)#41	
PAQ_ERR 6	ERR(p67L) #33			ERR(d23L)#33			ERR(p67L) #33			ERR(r23L)#33	
PAQ_ERR 4	D I A G N O S T I C O #36										

CUADRO 11.1. REPRESENTA EL AUTOMATA DE RECEPCION. EL NOMBRE DE LAS RUTINAS EN EL CODIGO FUENTE CORRESPONDE A LOS NOMBRES EN ESTE AUTOMATA.

II.3.- ESTADO DEL NUCLEO DESPUES DE INICIALIZAR EL MODULO PAQUETES.

Es importante conocer el funcionamiento del núcleo en C utilizado en esta implementación. Para ello, el apéndice A contiene una descripción general de este núcleo, por lo que se recomienda ampliamente consultarlo. De esta manera, las operaciones básicas para el manejo de concurrencia presentadas en esta implementación, no significarán ningún problema.

En la fig.II.3 se presentan los descriptores de procesos en el arreglo PRIORITY. Todos estos procesos son procesos base.

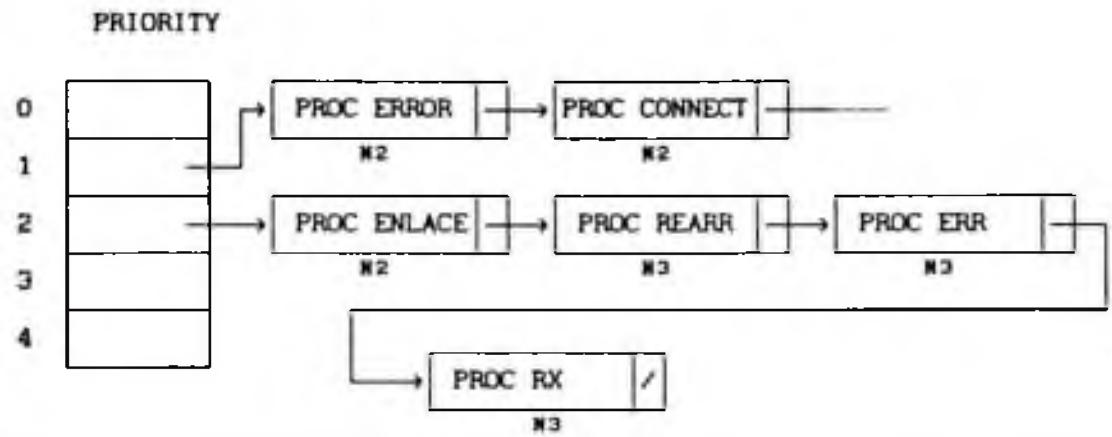


FIG.II.3. SE PRESENTAN LOS DESCRIPTORES DE PROCESOS EN EL ARREGLO PRIORITY. TODOS ELLOS SON PROCESOS BASE.

N2 son los procesos correspondientes al nivel enlace y N3 los del nivel paquetes.

Recordemos que PROC ENLACE es el proceso que inicializa el módulo de enlace y muere cuando la conexión queda establecida, lo cual despierta el proceso PROC CONNECT y comunica al nivel superior el evento de conexión mediante la instrucción Signal2(sem_rearranque).

Podemos inferir que la primera instrucción del proceso de rearranque es un Wait(sem_rearranque) por lo que entra en ejecución e invoca el procedimiento de rearranque con la causa «Red operacional» y un mensaje a cada una de las terminales activas «Nivel de paquetes inicializado».

De esta manera, el procedimiento de rearranque se invoca cada vez que se establece la conexión en la interfaz ETD/ETCD, cumpliendo así con la norma (fig.II.4).

De una manera similar, el proceso de error en el nivel paquete, PROC ERROR, es despertado por el proceso de error del nivel de enlace mediante un Signal2(sem_paq_error), siempre que suceda un error en el nivel de enlace. Este proceso de error libera todos los canales lógicos con la causa «Red fuera de servicio», cumpliendo así con la norma (fig. II.5).

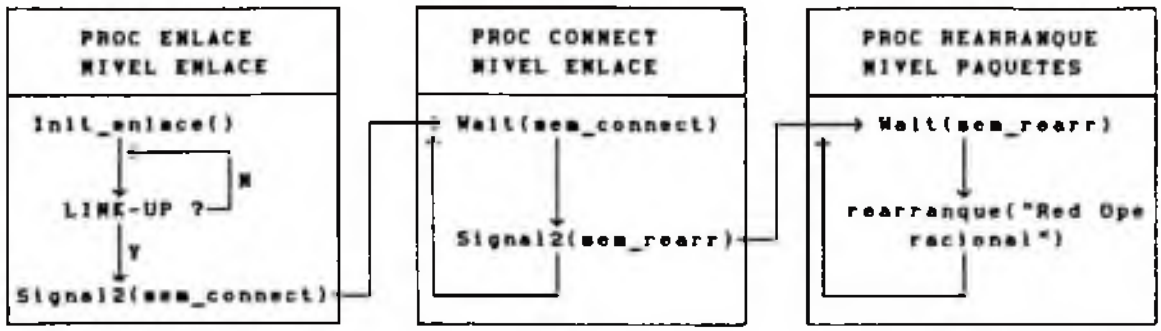


FIG.II.4. COMUNICACION ENTRE EL NIVEL DE ENLACE Y EL NIVEL DE PAQUETES PARA EL PROCEDIMIENTO DE REARRANQUE.

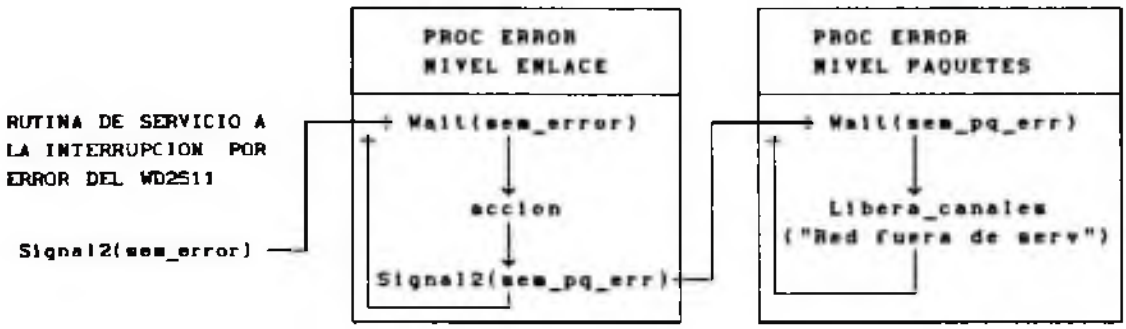


FIG.II.5. COMUNICACION ENTRE EL NIVEL DE ENLACE Y EL NIVEL DE PAQUETES PARA EL MANEJO DE ERRORES.

Finalmente, el tercer proceso del nivel paquetes es el que corresponde al autómata de recepción PROC RX. Este proceso utiliza la primitiva del nivel de enlace rx_paq = RxTrama(), donde rx_paq apunta a la trama que acaba de ser recibida por el nivel de enlace (fig. II.6).

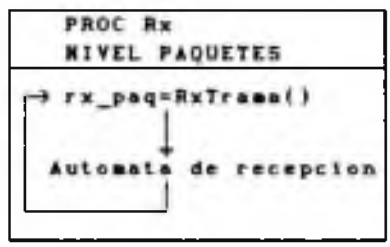


FIG.II.6.PROCESO DE RECEPCION DE PAQUETES. UTILIZA LA PRIMITIVA DEL NIVEL DE ENLACE RX_TRAMA(). REPRESENTA EL CUERPO INICIAL DEL AUTOMATA DE RECEPCION.

Lo que corresponde a este autómata de recepción es el analizar el contenido de rx_paq y efectuar las acciones necesarias.

Primeramente se analiza si el paquete es un paquete de error, tomándose las acciones correspondientes según el caso.

II.4. PAQUETES DE ERROR.

PAQ_ERR_1: Paquete de longitud menor a 3 bytes. El paquete se ignora y se transmite DIAG #38 (paquete de *DIAGNOSTICO* con la causa #38).

PAQ_ERR_2: Paquete con identificador general de formato no válido. El paquete se ignora y se transmite DIAG #40.

PAQ_ERR_3: Paquete con canal lógico no asignado. Razones:

- Sale del rango permitido.
- Va dirigido a un canal lógico no activo; es decir, que no tiene una terminal activa asignada y el paquete es diferente al de *LLAMADA ENTRANTE/PETICION DE LLAMADA*. Si el canal lógico está inactivo y el paquete es de *LLAMADA ENTRANTE/PETICION DE LLAMADA*, se hace una búsqueda de alguna terminal activa sin canal lógico asignado. Si se tiene éxito, se asocia el canal lógico con la terminal, estableciendo así un circuito virtual. De ser así, el paquete no es del tipo **PAQ_ERR_3**; en caso contrario, si la búsqueda no tiene éxito, el paquete es del tipo **PAQ_ERR_3**. El paquete es entonces ignorado y se transmite DIAG #36.

PAQ_ERR_4: Paquete distinto al de *REARRANQUE, CONFIRMACION DE REARRANQUE Y DIAGNOSTICO*, con canal lógico igual a 0. El paquete se ignora y se transmite DIAG # 36.

PAQ_ERR_5: Paquete de *REARRANQUE, CONFIRMACION DE REARRANQUE O DIAGNOSTICO* con canal lógico diferente de 0. El paquete se filtra al canal lógico correspondiente para que el autómata de recepción efectúe la acción indicada.

PAQ_ERR_6: Paquete con identificador de tipo de paquete indefinido. El paquete se filtra al canal lógico correspondiente para que el autómata de recepción efectúe la acción indicada.

Los cuatro primeros paquetes de error son atendidos por el nivel superior del proceso de recepción; no afectan el estado de la interfaz, pues únicamente se transmite un paquete de *DIAGNOSTICO* #X. Los dos últimos paquetes de error se dejan pasar hasta el autómata de recepción para que éste tome la acción correspondiente (según el estado del canal para el cual va dirigido).

Si el paquete recibido es una paquete de *DIAGNOSTICO*, únicamente es decodificado y desplegado en la terminal, no afecta el estado de la interfaz.

Si el paquete recibido no es ninguno de los anteriores, se considera como un paquete correcto a partir del cual junto con el estado de la interfaz se ejecutará la rutina correspondiente:

```
switch(ESTADO) {
     case r1   : edo_r1();
                 break;
     case r23E : edo_r23E();
                 break;
     case r23L : edo_r23L();
                 break;
}
```

II.5.- AUTOMATA DE RECEPCION.

A partir del cuadro II.1 se pueden inferir las acciones que efectuarán las rutinas anteriores según el tipo de paquete recibido y el estado del canal lógico hacia el cual va dirigido. Existe una correspondencia casi directa entre el cuadro II.1 y las rutinas implementadas.

edo_r1(). Nivel paquete preparado.

Cuando el estado de la interfaz es r1, ESTADO = r1, y el paquete recibido es un paquete de *PETICION/INDICACION DE REARRANQUE* entonces se ejecuta la rutina NOR_r23E(). Si el paquete recibido es un paquete de CONFIRMACION DE REARRANQUE, entonces se ejecuta la rutina ERR_r23L(17). Para cualquier otro paquete recibido en este estado de la interfaz se ejecuta la rutina autómata_rx().

edo_r23E(). (Petición/Indicación de rearranque por el ETD extremo).

Cuando el estado de la interfaz es r23E, ESTADO = r23E, y el paquete recibido es un paquete de *PETICION/INDICACION DE REARRANQUE*, entonces el paquete es simplemente descartado. Si el paquete recibido es del tipo *PAQ_ERR_5*, se invoca la rutina de error ERR_r23L(41). Si el paquete recibido es del tipo *PAQ_ERR_6*, se invoca la rutina de error ERR_r23L(33). Para cualquier otro paquete recibido en este estado de la interfaz, se invoca la rutina de error ERR_r23L(18).

edo_r23L(). Indicación/Petición de rearranque por el ETCD/ETD local).

Cuando el estado de la interfaz es r23L, ESTADO = r23L, se hará caso omiso a cualquier paquete recibido diferente al de *PETICION/INDICACION DE REARRANQUE* y al de *CONFIRMACION DE REARRANQUE*. Si alguno de estos dos paquetes llega, se ejecutará la rutina NOR_r1().

Cuando se invoca la rutina autómata_rx() es porque el paquete recibido contiene un canal lógico diferente de 0 y es necesario tomar ciertas acciones a partir de dos elementos: el estado actual del canal lógico para el cual va dirigido y el tipo de paquete recibido.

```
autómata_rx()                    <-- EXTRAIDO DEL PAQUETE RECIBIDO
{
     switch(CANAL[canal_lógico].estado) {
        case p1   : edo_p1();        break;
        case p23E : edo_p23E();      break;
        case p23L : edo_p23L();      break;
EDO DEL case d1   : edo_d1();        break;
CANAL   case d23E : edo_d23E();      break;
LOGICO  case d23L : edo_d23L();      break;
        case p5   : edo_p5();        break;
        case p67E : edo_p67E();      break;
        case p67L : edo_p67L();      break;
     }
}
```

Una vez teniendo el estado del canal lógico, se conoce la columna a utilizar en el cuadro II.1. Unicamente hace falta el renglón, el cual está determinado por el tipo de paquete que acaba de llegar.

Por ejemplo, si el estado del canal lógico es p1 y el paquete que llegó es un paquete de *PETICION DE LLAMADA/LLAMADA ENTRANTE*, la acción a ejecutar es NOR_p23E().

Todas las llamadas a subrutinas a partir de los diferentes estados del canal lógico, se llevan a cabo de esta manera a partir del cuadro II.1, a excepción de algunos detalles como los siguientes:

-En el estado edo_p23L() (ETD/ETCD local en espera), la primera instrucción es desactivar el temporizador TX1 que se activó en el momento de haber transmitido un paquete de *PETICION DE LLAMADA/LLAMADA ENTRANTE*. Cualquier paquete que llegue es considerado como respuesta a esta petición de llamada o llamada entrante.

-En el estado edo_d1() (control de flujo preparado) los paquetes de *DATOS, RR, RNR, INTERRUPCION Y CONFIRMACION DE INTERRUPCION* afectan el control de flujo, del cual se hablará más adelante con detalle. El resto de los paquetes que lleguen en este estado del canal lógico, son tratados según el cuadro II.1.

-En el estado edo_d23L() (Indicación/Petición de reinicio por el ETCD/ETD), la primera instrucción es desactivar el temporizador TX2 que se activó en el momento de haber transmitido un paquete de *REINICIO*. Cualquier paquete que llegue es considerado como respuesta a esta indicación/petición de reinicio.

-En el estado edo_p5() (colisión de llamadas), si es el ETD el que recibe un paquete de *COMUNICACION ESTABLECIDA*, se desactiva el temporizador TX1 y se ejecuta NOR_d1() (control de flujo preparado). Si es el ETCD el que recibe un paquete de *LLAMADA ACEPTADA* se invoca la rutina de error ERR_p67L(24).

Recordemos que en una colisión de llamadas, es el ETCD el que dá paso al paquete de *PETICION DE LLAMADA*, y olvida el paquete de *LLAMADA ENTRANTE* previamente transmitido.

-En el estado edo_p67L() (petición/indicación de liberación por el ETD/ETCD local) para el paquete de *PETICION/INDICACION DE LIBERACION* y para el de *CONFIRMACION DE LIBERACION*, el temporizador TX3 es desactivado y se ejecuta el procedimiento NOR_p1() (preparado).

II.6.- RUTINAS DEL AUTOMATA DE RECEPCION.

A continuación se explicarán las rutinas ejecutadas a partir del autómata de recepción. La ejecución de estas rutinas depende del estado del canal lógico para el cual va dirigido el paquete recién llegado y del tipo de este paquete; es decir, se explicarán los procedimientos de que consta el cuadro II.1. Para el significado de los temporizadores, consultar el punto II.9.

NOR_r1()

- La interfaz ETD/ETCD pasa el estado r1 (Nivel paquete preparado).
- Se desactiva el temporizador TX0.

NOR_r23E()

•La interfaz pasa momentáneamente al estado r23E (Petición/Indicación de rearranque por el ETD/ETCD extremo), pues se recibió un paquete de *PETICION/INDICACION DE REARRANQUE.*

•Se transmite un paquete de *CONFIRMACION DE REARRANQUE.*

•Se decodifica el campo de causa de rearranque y diagnóstico del paquete de *PETICION/INDICACION DE REARRANQUE.*

•La interfaz pasa al estado r1 (Nivel paquete preparado).

ERR_r23L(DIAG)

•Se activa el procedimiento de rearranque con la causa «Error de procedimiento local» y el diagnóstico DIAG.

NOR_p1()

•Se desactiva el canal lógico.

•Se disasocia la terminal del canal lógico.

•El estado del canal lógico pasa al estado p1 (preparado).

NOR_p23E()

•El canal lógico pasa momentáneamente al estado p23E (ETD/ETCD extremo en espera) pues acaba de recibir un paquete de *PETICION DE LLAMADA/LLAMADA ENTRANTE.*

•Se transmite un paquete de *LLAMADA ACEPTADA/COMUNICACION ESTABLECIDA.*

•El canal lógico pasa al estado d1 (control de flujo preparado).

NOR_p67E()

•El canal lógico pasa momentáneamente la estado p67E (Petición / Indicación de liberación por el ETD/ETCD extremo) pues se acaba de recibir un paquete de *PETICION/INDICACION DE LIBERACION.*

•Se transmite un paquete de *CONFIRMACION DE LIBERACION.*

•Se hacen las acciones necesarias para liberar el canal lógico mediante la rutina libera_cl(#canal a liberar, razón).

•Se decodifica el campo de liberación y el de diagnóstico del paquete de *PETICION/INDICACION DE LIBERACION* recibido.

•Se desactiva el canal lógico.

•Se disasocia la terminal del canal lógico.

•El estado del canal lógico pasa al estado p1(preparado).

ERR_p67L(DIAG)

•Se realizan las acciones necesarias para liberar el canal lógico mediante la rutina libera_cl(#canal, razón).

•Se transmite un paquete de *PETICION/INDICACION DE LIBERACION* con la causa «Error de procedimiento local» y el diagnóstico DIAG.

•Se activa el temporizador TX3 del canal lógico.

•El estado del canal lógico pasa al estado p67L (Indicación/Petición de liberación por el ETCD/ETD local).

NOR_p5()

•El estado del canal lógico pasa al estado p5(colisión de llamadas).

•Si se trata de un ETCD, se transmite un paquete de *COMUNICACION ESTABLECIDA* y el estado del canal lógico pasa al estado d1(control de flujo preparado).

•En caso de ser ETD, se activa el temporizador TX1.

NOR_d1()

·El estado del canal lógico pasa al estado d1(control de flujo preparado).

·Se informa al nivel superior que se ha alcanzado el estado de transferencia de datos para ese canal lógico, por lo que puede comenzar a transmitir información. Lo anterior se logra mediante la instrucción Signal2(sem_conexión).

NOR_d23E()

·El estado del canal lógico pasa momentáneamente al estado d23E(petición/indicación de reinicio por el ETD/ETCD extremo).

·El canal lógico es reiniciado con ayuda de la rutina reinicia_cl(#canal a reiniciar, razón).

·Se transmite un paquete de *CONFIRMACION DE REINICIO*.

·Se decodifica el campo de causa de reinicio y el de diagnóstico del paquete de *PETICION/INDICACION DE REINICIO* recibido.

·El estado del canal lógico pasa al estado d1(control de flujo preparado).

ERR_d23L(DIAG)

·El canal lógico es reiniciado con ayuda de la rutina reinicia_cl (#canal a reiniciar, razón).

·Se transmite un paquete de *PETICION/INDICACION DE REINICIO* con la causa «Error de procedimiento local» y el diagnóstico #DIAG.

·Se activa el temporizador TX2.

·El estado del canal lógico pasa al estado d23L(indicación/petición de reinicio por el ETCD/ETD local).

II.7.- ESTRUCTURAS UTILIZADAS EN LA IMPLEMENTACION.

CANAL LOGICO:

```
Canal {
        activo     : El canal está activo si forma parte de un circuito --
                     virtual.
        terminal   : # de terminal asociada al canal lógico.
        estado     : Estado del canal.
        p_s        : # secuencial del siguiente paquete a ser transmitido.
        p_a        : # secuencial del siguiente paquete a ser reconocido.
        p_r        : # secuencial del siguiente paquete a recibir.
        TX1        : Temporizador activado al transmitir un paquete de PE -
                     TICION DE LLAMADA/LLAMADA ENTRANTE.
        TX2        : Temporizador activado al transmitir un paquete de PE -
                     TICION/INDICACION DE REINICIO.
        TX3        : Temporizador activado al transmitir un paquete de PE -
                     TICION/INDICACION DE LIBERACION.
        no_temp    : Número de veces seguidas que TX3 ha expirado sin re --
                     cibir respuesta al paquete de PETICION/INDICACION DE -
                     LIBERACION.
        TRec       : Temporizador que representa el tiempo que hay que es -
                     perar antes de mandar un reconocimiento aislado; es  -
                     decir, no hay "piggy-back".
        TAck       : Temporizador que representa el tiempo máximo de espera
                     para que un paquete o grupo de paquetes previamente  -
                     transmitidos sean reconocidos por la estación remota.
        PAQ_Tlook  : Ventana de transmisión.
        tx_index   : Indice para el manejo de la ventana de transmisión que
                     apunta al siguiente paquete a ser transmitido.
        ack_index  : Indice para el manejo de la ventana de transmisión que
                     apunta al siguiente paquete a ser reconocido.
        Tx_habilit: Bandera que indica si la transmisión está habilitada.
        sem_paq_tx: Semáforo para la transmisión. Indica el número máximo
                     de paquetes que pueden ser transmitidos sin reconoci-
                     miento. Es igual al valor del tamaño de la ventana de
                     transmisión.
        PAQ_Rlook  : Ventana de recepción.
        rx_index   : Indice para el manejo de la ventana de recepción que
                     apunta al siguiente paquete a ser recibido.
        n4_rx_index: Indice para el manejo de la ventana de recepción que
                     apunta al siguiente paquete previamente recibido que -
                     ha de ser procesado por el nivel superior.
        sem_paq_rx: Semáforo que indica el número de paquetes recibidos no
                     procesados por el nivel superior.
        sem_conexión: Semáforo para indicarle al nivel superior que el es-
                     tado del canal se encuentra en el estado de transferen
                     cia de datos.
}
```

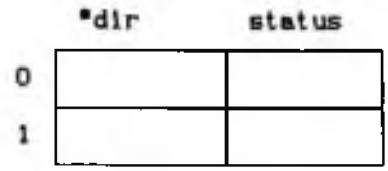
Existe un arreglo de canales de tamaño igual al número de terminales + 1.

```
Canal    CANAL[NO_TERM + 1];
```

VENTANA DE TRANSMISION:

La ventana de transmisión forma parte de la estructura del canal.

PAQ_Tlook[W]
W es el tamaño de la ventana de transmisión y recepción y tiene un valor actual igual a 2.

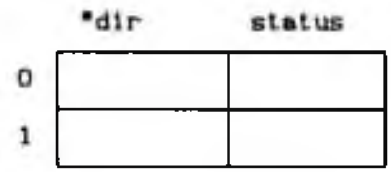


*dir apunta al buffer de datos de transmisión del nivel superior.
status puede tomar los siguientes valores:
0 = Buffer de transmisión listo (vacio) o paquete reconocido.
1 = Listo para transmitir o esperando reconocimiento.

VENTANA DE RECEPCION:

La ventana de recepción forma parte de la estructura del canal.

PAQ_Rlook[W]
W es el tamaño de la ventana de transmisión y recepción y tiene un valor actual igual a 2.



*dir apunta al buffer de datos de recepción del nivel superior.
status puede tomar los siguientes valores:
1 = Listo para recibir.
0 = Recibido y no procesado por el nivel superior.

Se debe notar que tanto la ventana de transmisión como la de recepción forman parte de la estructura de cada canal lógico. Esto es, cada canal lógico o circuito virtual establecido, en el momento de entrar al estado de transferencia de datos (d1), tiene su propio control de flujo y por lo tanto, sus propias ventanas de transmisión y de recepción.

TERMINALES:

```
Tabla {
        activa     : Indica si la terminal está activa o no (apagada/pren--
                     dida).
        dir_dat    : Dirección del puerto de datos del 8251A correspondien-
                     te a la terminal.
        dir_ctl    : Dirección del puerto de control del 8251A correspon  -
                     diente a la terminal.
        cl_asoc    : Número de canal lógico al cual la terminal se encuen--
                     tra asociado.
        com_buffer[BUFFER_SIZE]: Buffer para comandos o datos de  terminal
                     hasta 128 bytes.
        com_index  : Indice para el manejo del buffer com_buffer.
        com_in     : Bandera que sirve para indicar que la terminal está  -
                     introduciendo un comando.
        *ptr_dat   : Apuntador al buffer de datos de transmisión del  nivel
                     superior; es decir, toma el valor correspondiente al -
                     de la ventana de transmisión del canal lógico con el -
                     que esta asociado.
        dat_index  : Junto con *ptr_dat (ptr_dat+dat_index) se introduce la
                     información a donde apunta la ventana de transmisión.
        dat_in     : Bandera que sirve para indicar que la terminal está  -
                     introduciendo datos.
        sem_com    : Semáforo que sirve para indicar que hay algún  comando
                     listo para procesarse.
}
```

El número de terminales se encuentra en la constante NO_TERM, quedando la definición de esta estructura como sigue:

```
Tabla    TERMINAL[NO_TERM + 1];
```

Se debe tomar en cuenta la terminal utilizada para diagnóstico.

II.8.- SIGNIFICADO DE LOS TEMPORIZADORES DEL NIVEL DE PAQUETES.

T10: Es activado cuando el ETCD transmite un paquete de *INDICACION DE REARRANQUE.*

T20: Es activado cuando el ETD transmite un paquete de *PETICION DE REARRANQUE.*

T11: Es activado cuando el ETCD transmite un paquete de *LLAMADA ENTRANTE.*

T21: Es activado cuando el ETD transmite un paquete de *PETICION DE LLAMADA.*

T12: Es activado cuando el ETCD transmite un paquete de *INDICACION DE REINICIO.*

T22: Es activado cuando el ETD transmite un paquete de *PETICION DE REINICIO.*

T13: Es activado cuando el ETCD transmite un paquete de *INDICACION DE LIBERACION.*

T23: Es activado cuando el ETD transmite un paquete de *PETICION DE LIBERACION.*

La implementación no hace diferencia si se trata de un temporizador en el ETCD o en el ETD, por lo que:

T10, T20 se resume a TX0;

T11, T21 se resume a TX1;

T12, T22 se resume a TX2;

T13, T23 se resume a TX3.

II.9.- ACCIONES REALIZADAS AL EXPIRAR ALGUN TERMPORIZADOR.

TX0 (REARRANQUE): Transmite nuevamente el paquete de *PETICION/INDICACION DE REARRANQUE* con la causa «Expiración del temporizador en la indicación de rearranque».

TX1 (LLAMADA): No es posible hacer conexión. Disasocia la relación canal lógico-terminal preestablecida cuando se transmitió el paquete de *PETICION DE LLAMADA/LLAMADA ENTRANTE.* El estado del canal pasa al estado p1 (preparado).

TX2 (REINICIO): Se procede a liberar el canal lógico correspondiente por lo que se activa TX3.

TX3 (LIBERACION): Se retransmite el paquete de *PETICION/INDICACION DE LIBERACION* hasta cuatro veces. Si TX3 expira por quinta vez, se invoca el procedimiento de rearranque.

Existen dos temporizadores más por canal correspondientes al control de flujo de dicho canal. Estos son:

TAck: Temporizador para el reconocimiento de paquetes transmitidos. Cuando expira se procede a la liberación del canal lógico.

TRec: Temporizador para la transmisión de reconocimientos aislados. Al expirar, es decir, cuando no existen paquetes de datos a transmitir, se transmite un paquete RR.

II.10.- CONTROL DE FLUJO.

Los paquetes relacionados con el control de flujo son: *DATOS, RR, RNR, INTERRUPCION Y CONFIRMACION DE INTERRUPCION.* Estos paquetes sólo son aceptados si el estado del canal es d1(control de flujo preparado).

La estructura de la ventana de transmisión y la de recepción han sido previamente descritas. Dentro de la estructura de cada canal lógico, existen ciertas variables que están asociadas al manejo de dichas ventanas. Estas son:

Para la ventana de transmisión:

CANAL[i].PAQ_Tlook[W]

	*dir	status
0		
1		

CANAL[i].PAQ_Tlook[j].dir: Apunta al buffer de datos de transmisión del nivel superior.

CANAL[i].PAQ_Tlook[j].status: Puede tomar los siguientes valores:
0 = Buffer de transmisión listo (vacío) o paquete reconcido.
1 = Listo para transmitir o esperando reconocimiento.

CANAL[i].tx_index: Indice para el manejo de la ventana de transmisión que apunta al siguiente paquete a ser transmitido.

CANAL[i].ack_index: Indice para el manejo de la ventana de transmisión que apunta al siguiente paquete a ser reconocido.

CANAL[i].sem_paq_tx: Semáforo que indica el número máximo de paquetes que pueden ser transmitidos sin reconocimiento. Inicialmente es igual al valor del tamaño de la ventana de transmisión (W).

CANAL[i].Tx_habilit: Bandera que indica si la transmisión está habilitada o deshabilitada.

Para la ventana de recepción:

CANAL[i].PAQ_Rlook[W]

	*dir	status
0		
1		

CANAL[i].PAQ_Rlook[j].dir: Apunta al buffer de datos de recepción del nivel superior.

CANAL[i].PAQ_Rlook[j].status: Puede tomar los siguientes valores:
1 = Listo para recibir.
0 = Recibido y no procesado por el nivel superior.

CANAL[i].n4_rx_index: Indice para el manejo de la ventana de recepción que apunta al siguiente paquete a recibir por el nivel superior.

CANAL[i].rx_index: Indice para el manejo de la ventana de recepción que apunta al siguiente paquete a recibir.

CANAL[i].sem_paq_rx: Semáforo que indica el número de paquetes recibidos no procesados por el nivel superior.

El control de flujo toma ciertas acciones dependiendo del paquete recibido.

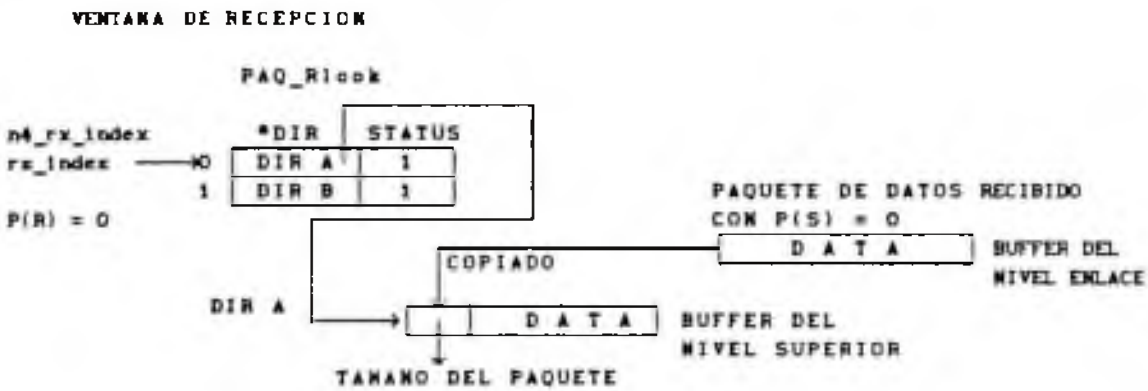
PAQUETE DE DATOS RECIBIDO:

Si el número secuencial del paquete recibido es diferente al número de paquete esperado, o si el buffer de recepción correspondiente no tiene su variable STATUS = 1 (listo para recibir), entonces el paquete recibido está fuera de secuencia o fuera de la ventana de recepción, por lo que se procede a reiniciar el canal lógico con la causa «Error de procedimiento local» y el diagnóstico «P(S) no válido».

El paquete de *DATOS* contiene un "piggy-back" que reconoce uno o varios paquetes. Si este número está fuera de la ventana de transmisión se activa el procedimiento de reinicio con la causa «Error de procedimiento local» y el diagnóstico «P(R) no válido».

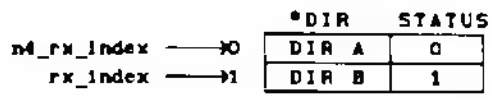
Si no se presenta ninguno de los casos anteriores el paquete de *DATOS* recibido es correcto y se pasa al buffer de recepción correspondiente.

Supongamos el siguiente estado inicial del CANAL[i] para el cual va dirigido el paquete recién llegado:



n4_rx_index y rx_index tienen el valor de 0, apuntan al borde inferior de la ventana de recepción. El status para este buffer de recepción es 1 (listo para recibir) y DIR A apunta al buffer de datos de recepción del nivel superior. Además el número de paquete esperado es igual a 0.

Cuando el paquete de datos llega y se ha demostrado que es correcto, su información es copiada hacia el buffer de datos de recepción del nivel superior, poniendo en el primer byte el valor del tamaño del paquete recibido. Finalmente:



n4_rx_index sigue apuntando a 0, pues el paquete puesto en el buffer del nivel superior no ha sido procesado. La variable status es puesta en 0 (paquete recibido y no procesado por el nivel superior).

El proceso del nivel superior deberá incrementar n4_rx_index módulo W y regresar el valor de la variable status a 1 (listo para recibir) cuando haya liberado el buffer de recepción.

La variable rx_index se incrementa módulo W (rx_index = 1).

P(R) se incrementa módulo 8 (siguiente paquete a recibir = 1).

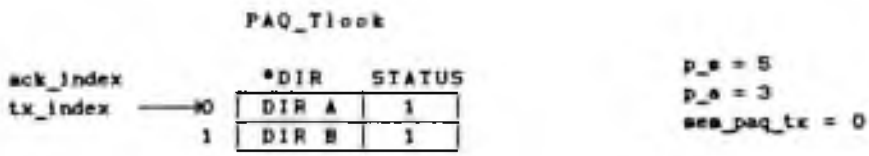
Si la ventana de recepción se llena al recibir el paquete de *DATOS*, inmediatamente se transmite un paquete *RNR*.

Se activa TRec (temporizador para el reconocimiento de paquetes).

Signal2(sem_paq_rx). Avisa al nivel superior que se ha recibido un paquete.

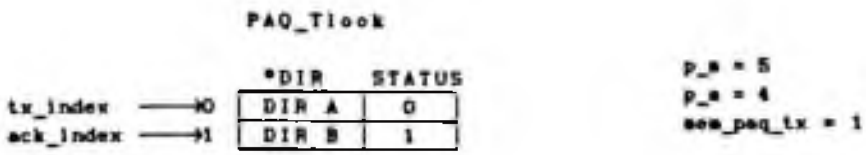
El paquete de *DATOS* recibido contiene un número que reconoce uno o más paquetes (piggy-back). El reconocimiento de paquetes se hace en la rutina valido_pr().

La rutina inicialmente extrae este número del paquete de *DATOS* y lo pone en la variable Paq_pr. Este número (siguiente paquete a recibir en la estación remota) reconoce hasta Paq_pr-1. Supongamos el siguiente estado de Paq_Tlook:



Esto significa que se han transmitido dos paquetes de *DATOS* que aún no han sido reconocidos, estos paquetes son el #3 y el #4, por lo que p_s (#sec del próximo paquete a transmitir) = 5. El próximo paquete a ser reconocido (p_a) es entonces igual a 3. Por otro lado, los índices auxiliares tx_index y ack_index son igual a 0. STATUS = 1 para las dos entradas de la ventana de transmisión porque ambos paquetes están esperando reconocimiento. Por lo anterior, el semáforo de transmisión sem_paq_tx = 0, lo cual deshabilita la transmisión; es decir, el proceso transmisor queda bloqueado en el momento de ejecutar un Wait(sem_paq_tx).

Si la variable Paq_pr (número extraído del paquete de *DATOS* que indica el reconocimiento hasta Paq_pr-1) es correcta y supongamos que su valor es igual a 4, entonces el estado de Paq_Tlook quedaría:



Lo anterior reconoce únicamente un paquete, al paquete #3, lo cual permite la transmisión de un paquete de *DATOS* más debido al valor del semáforo de transmisión.

PAQUETE RR RECIBIDO: Recordemos que en su formato incluye un número secuencial para reconocimiento de paquetes. Este número es validado mediante la rutina valido_pr() que efectúa el reconocimiento correspondiente. En caso de no ser válido, se reinicia el canal lógico con la causa «Error de procedimiento local» y el diagnóstico «P(R) no válido». En caso contrario, habilita la transmisión (Tx_habilit = 1) pues este paquete indica que la estación remota está lista para seguir recibiendo paquetes.

PAQUETE RNR RECIBIDO: Es sujeto al mismo procedimiento que el paquete *RR* sólo que deshabilita la transmisión (Tx_habilit = 0) en lugar de habilitarla, pues este paquete indica que la estación remota no está en condiciones para seguir recibiendo paquetes.

PAQUETE DE INTERRUPCION RECIBIDO: Este paquete no está sujeto a control de flujo. Su información es desplegada en la terminal en el momento de recibirse y se transmite inmediatamente un paquete de *CONFIRMACION DE INTERRUPCION.*

PAQUETE DE CONFIRMACION DE INTERRUPCION RECIBIDO: Se indica en la pantalla de la terminal que su paquete de *INTERRUPCION* previamente transmitido ha sido confirmado.

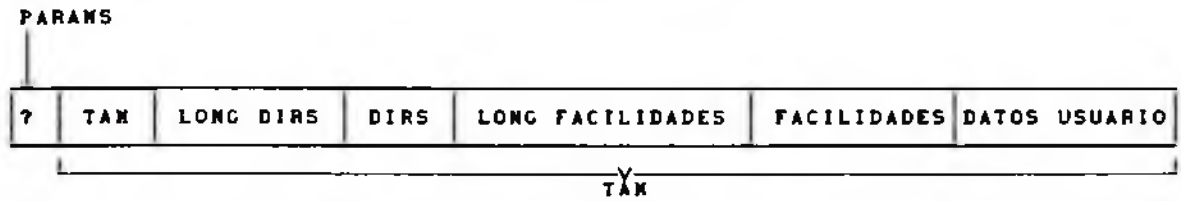
II.11.- TRANSMISION DE PAQUETES.

Se tienen rutinas auxiliares para la transmisión de los diferentes paquetes del nivel paquetes. Estas son:

Tx_rearranque(causa,DIAG) : *PETICION/INDICACION DE REARRANQUE.*

Tx_c_rearranque() : *CONFIRMACION DE REARRANQUE.*

Tx_llamada(canal,params) : *PETICION DE LLAMADA/LLAMADA ENTRANTE.*



NOTA: EL BYTE ? PUEDE SER USADO PARA INDICAR DESDE EL NIVEL SUPERIOR EL TIPO DE PAQUETE O COMANDO DE QUE SE TRATA.

Tx_c_llamada(canal) : *LLAMADA ACEPTADA/COMUNICACION ESTABLECIDA.*

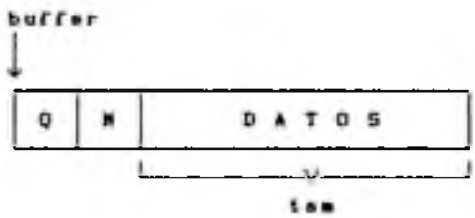
Tx_liberación(canal,causa,DIAG): *PETICION/INDICACION DE LIBERACION.*

Tx_c_liberación(canal) : *CONFIRMACION DE LIBERACION.*

Tx_reinicio(canal,causa,DIAG) : *PETICION/INDICACION DE REINICIO.*

Tx_c_reinicio(canal) : *CONFIRMACION DE REINICIO.*

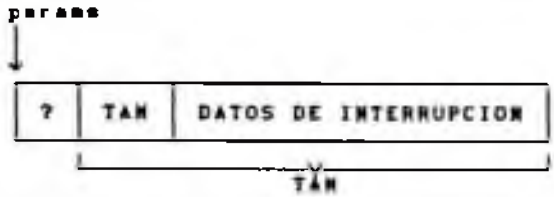
Tx_datos(canal,buffer,tam) : *DATOS.*



Si el byte Q en el buffer de parámetros es '0', entonces el valor del bit Q en el paquete de *DATOS* es igual a 0; cualquier otro valor, el bit Q es igual a 1.

Si el byte M en el buffer de parametros es '0', entonces el valor del bit M en el paquete de *DATOS* es igual a 0; cualquier otro valor, el bit M es igual a 1.

Tx_interrupción(canal,params) : *INTERRUPCION.*



NOTA: EL BYTE ? PUEDE SER USADO PARA INDICAR DESDE EL NIVEL SUPERIOR EL TIPO DE PAQUETE O COMANDO DE QUE SE TRATA.

Tx_c_interrupción(canal) : *CONFIRMACION DE INTERRUPCION.*

Tx_diagnóstico(DIAG) : *DIAGNOSTICO.*

Tx_RR(canal) : *RR.*

Tx_RNR(canal) : *RNR.*

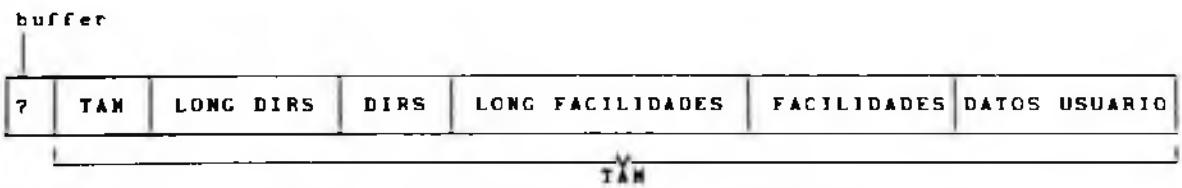
Todas las rutinas anteriores utilizan una rutina auxiliar llamada encabezado(canal,ident) que forma los tres primeros bytes del paquete a transmitir a partir de los parámetros canal e identificador del tipo de paquete. Para la transmisión de paquetes se utiliza la primitiva de transmisión del nivel enlace TxTrama(tx_paq).

II.12.- PRIMITIVAS DEL NIVEL PAQUETES.

- InitPaquete().
- int CONEXION(word terminal, char *buffer).
- int Tx_PaqDat(word canal, char *buffer, word tam).
- Tx_RR(word canal).
- Tx_RNR(word canal).
- Tx_interrupción(word canal, char *buffer).
- NOR_REINICIA(word canal, byte causa, byte diag).
- NOR_LIBERA(word canal).

Init_Paquete(). Primitiva utilizada para iniciar el nivel de paquetes.

int CONEXION(word terminal, char *buffer). Cuando una terminal quiere transmitir un paquete de *PETICION DE LLAMADA*, invoca este procedimiento con su número de terminal y los parámetros de conexión en la variable buffer, con el siguiente formato:



NOTA: EL BYTE ? PUEDE SER USADO PARA INDICAR DESDE EL NIVEL SUPERIOR EL TIPO DE PAQUETE O COMANDO DE QUE SE TRATA.

CONEXION regresa un valor. Si el valor regresado es 1, significa que se ha asignado un número de canal lógico y se ha transmitido un paquete de *PETICION DE LLAMADA/LLAMADA ENTRANTE*, por lo que el canal se encuentra en el estado p23L(ETD/ETCD local en espera) y TX1 se encuentra activado. Si el valor regresado es 0, entonces no es posible hacer conexión debido a que no existe algún canal lógico disponible.

int Tx_PaqDat(word canal, char *buffer, word tam).

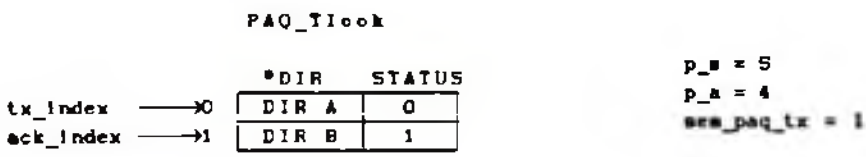
Al igual que la primitiva de CONEXION, esta rutina regresa un valor para indicar el éxito o fracaso en la operación.

El valor regresado es 0, de error, si la transmisión no está habilitada o si el estado del canal es diferente de d1(control de flujo preparado).

Esta rutina hace un Wait(sem_paq_tx), el cual representa el número de buffers libres en la ventana de transmisión PAQ_Tlook. Por esta razón, el que utiliza esta rutina desde el nivel superior debe ser un proceso. Al salir del Wait(), actualiza el campo de status de PAQ_Tlook (status = 1), incrementa en 1 módulo W (W = tamaño de la ventana de transmisión) tx_index e invoca la rutina auxiliar Tx_datos(canal, buffer, tam). Finalmente activa el temporizador TAck (tiempo de espera para que se reciba el reconocimiento del paquete recién enviado) y detiene el temporizador TRec (el paquete de DATOS recién enviado contiene un "piggy-back").

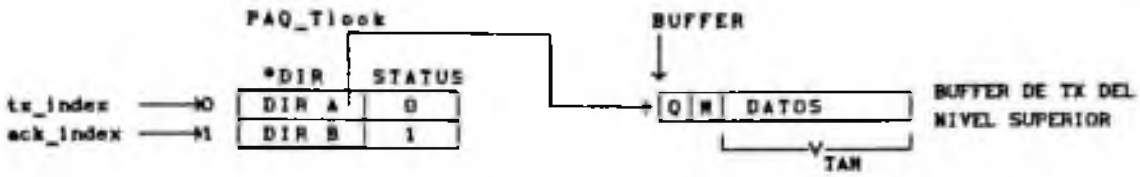
El formato de buffer debe ser el siguiente:

Supongamos el siguiente estado de la ventana de transmisión del CANAL[1] y de las variables asociadas. El CANAL[i] se encuentra en el estado d1 (control de flujo preparado):

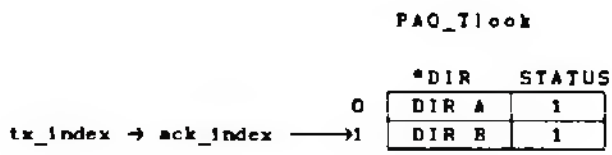


Se ha transmitido ya un paquete de datos (p_s = 4) aún no reconocido por lo que p_s = 5 (número secuencial del siguiente paquete a ser transmitido) y p_a = 4 (número secuencial del siguiente paquete a ser reconocido). La variable ack_index es el índice de la ventana de transmisión que contiene la dirección (DIR B) del paquete transmitido. La variable status es igual a 1 (esperando reconocimiento).

La dirección del buffer de transmisión del nivel superior del siguiente paquete a transmitir la podemos encontrar con el índice tx_index. Se observa que esta dirección es DIR A y que status es igual a 0 (buffer de transmisión listo (vacio)).



La primitiva Tx_PaqDat(canal,*buffer,tam) hará en una de sus instrucciones un Wait(sem_paq_tx). Como el valor del semáforo es 1, el proceso no se bloquea y continúa, dejando el estado de la ventana de transmisión y de sus variables asociadas como sigue:

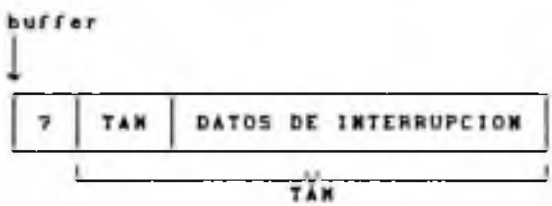


Inmediatamente después invoca la rutina auxiliar Tx_datos(canal, buffer, tam), la cual copia la información del buffer de transmisión del nivel superior al buffer del nivel de enlace para su transmisión mediante la primitiva Tx_Trama(tx_paq), donde tx_paq es un apuntador a la trama del nivel de enlace.

Tx_RR(word canal). Primitiva para transmitir un paquete *RR*.

Tx_RNR(word canal). Primitiva para transmitir un paquete *RNR*.

Tx_interrupcion(word canal, char *buffer). Primitiva para transmitir un paquete de *INTERRUPCION*. El formato de buffer debe ser:



NOTA: EL BYTE ? PUEDE SER USADO PARA INDICAR DESDE EL NIVEL SUPERIOR EL TIPO DE PAQUETE O COMANDO DE QUE SE TRATA.

NOR_REINICIA(word canal, byte causa, byte diag). Primitiva utilizada para reiniciar el canal lógico deseado con causa y diagnóstico. Transmite un paquete de *PETICION/INDICACION DE REINICIO* y pone el estado del canal lógico en d23L(indicación/petición de reinicio por el ETCD/ETD local). Activa el temporizador TX2.

NOR_LIBERA(word canal). Primitiva utilizada para liberar el canal lógico con la causa «Originada en el ETD»; es decir, es una liberación normal, no forzada por algún error, sino mas bien cuando se desea de manera voluntaria finalizar una conexión. Transmite un paquete de *PETICION/INDICACION DE LIBERACION* y pone el estado del canal lógico en p67L(indicación/petición de liberación por el ETCD/ETD local). Activa el temporizador TX3.

Con ésto quedan detalladas las principales rutinas y las primitivas de que consta la implementación en C del nivel de paquetes de X.25. A continuación se procede a describir los dos módulos de prueba desarrollados para este nivel. Estos módulos utilizan las primitivas y estructuras del nivel, lo que hará mas claro lo recién expuesto en esta sección.

III.- MODULOS DE PRUEBA.

Se han preparado dos módulos de prueba que pueden servir de base para la programación del nivel superior. El primero de ellos está dividido en dos, uno transmisor, y el otro receptor. Se forman dos canales lógicos a través de los cuales se realiza la transferencia de paquetes en un solo sentido a manera de ráfagas. El lado receptor únicamente los procesa y envía reconocimientos.

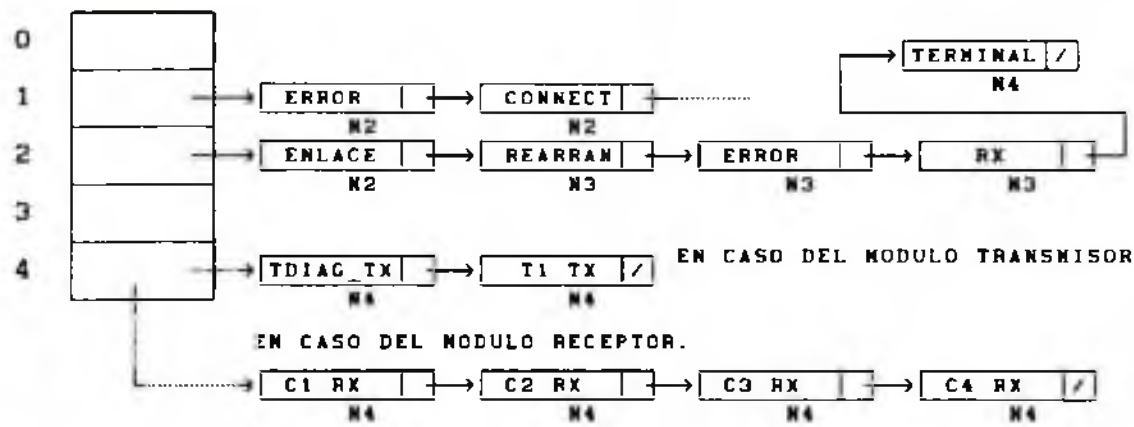
El segundo de ellos es un módulo de prueba que permite la interacción usuario-terminal a partir de un grupo de comandos. Este módulo permite la transferencia de paquetes en ambos sentidos del canal lógico.

III.A.- MODULO 1.

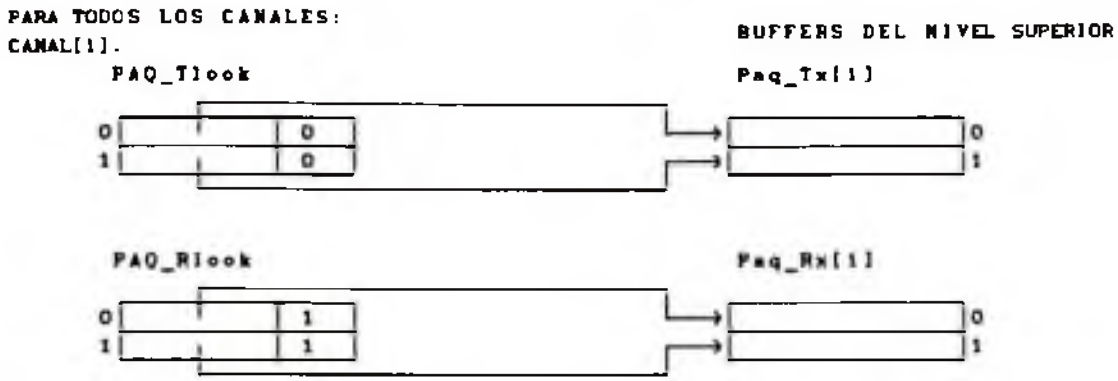
```
void_modf main()
{
    InitKernel();
    InitEnlace();
    InitPaquete();
    InitN4();
    ExecuteKernel();
}
```

MUY IMPORTANTE LA SECUENCIA DE INSTRUCCIONES.

Init_N4() crea procesos del nivel 4 quedando el estado del núcleo como sigue:



El proceso TERMINAL es un proceso que inicializa las terminales y los buffers para transmisión y para recepción. Utiliza la terminal #0 y la terminal #1, dejándolas activas. Los buffers de transmisión/recepción quedan como sigue:



III.A.1.- MODULO 1. TRANSMISOR.

El proceso T_DIAG_TX es el proceso transmisor correspondiente a la terminal de diagnóstico o terminal #0.

T1_TX es el proceso transmisor correspondiente a la terminal #1.

Ambos procesos funcionan de la misma manera. Para la terminal #X, se utiliza el buffer asignado a su estructura para indicar los parámetros de llamada e invoca la primitiva de CONEXION.

```
status = CONEXION(X, TERMINAL[X].com_buffer);
```

Donde X es el número de terminal y com_buffer es el apuntador al buffer asignado a la terminal X; com_buffer contiene los parámetros de llamada y debe cumplir con el formato previamente descrito.

Si status es correcto, el proceso transmisor hace un Wait(sem_conexión) que es el semáforo que indica que el estado del canal lógico se encuentra en d1(control de flujo preparado; o lo que es lo mismo, transferencia de datos). A partir de aquí, llena de información el buffer del nivel superior con el formato del paquete de *DATOS* previamente descrito, e invoca la primitiva status = TxPaqDat(canal, ptr, tam); donde ptr es un apuntador al inicio del buffer del nivel superior. El resultado de la transferencia se refleja en el valor de la variable status.

III.A.2.- MODULO 1. RECEPTOR.

CX_RX es el proceso receptor correspondiente al canal #X.

Los cuatro procesos funcionan de una manera similar.

Para el canal lógico #X:

Se realiza un Wait(sem_paq_rx) que es el semáforo que indica que un paquete de *DATOS* ha sido recibido en el canal lógico correspondiente.

A partir del valor de n4_rx_index (índice para el manejo de la ventana de recepción que apunta al siguiente paquete previamente recibido y que ha de ser procesado por el nivel superior) se conoce la dirección del siguiente paquete de *DATOS* a procesar por el nivel superior. El paquete de *DATOS* es procesado y la ventana de recepción actualizada (el status del buffer correspondiente a la ventana de recepción es puesto en 1 y el índice n4_rx_index se incrementa en 1 módulo W).

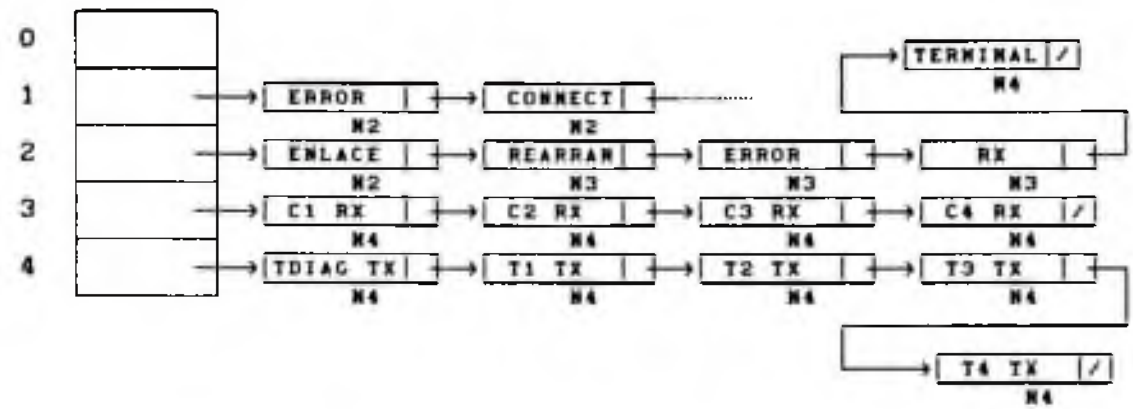
Este módulo permite principalmente observar el comportamiento de las ventanas de transmisión y de recepción. Se aprecia principalmente el control de flujo implementado en el nivel de paquetes.

III.B.- MODULO 2.

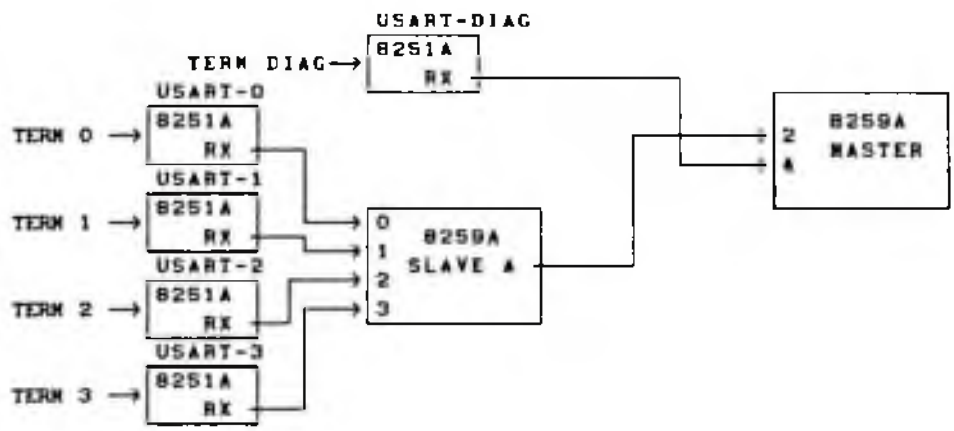
```
void _modf main()
{
     InitKernel();
     InitEnlace();
     InitPaquete();
     InitEstación();
     Init_TR_Paq();
     ExecuteKernel();
}
```

MUY IMPORTANTE LA SECUENCIA DE INSTRUCCIONES.

InitEstación() crea los procesos correspondientes al nivel 4 quedando el núcleo como sigue:



El proceso TERMINAL es el encargado de habilitar las interrupciones por recepción de las terminales y modificar el vector de interrupciones colocando las rutinas de servicio correspondientes.



Los procesos C1_RX, C2_RX, C3_RX y C4_RX funcionan exactamente de la misma manera que los procesos de recepción del módulo 1 de prueba (receptor).

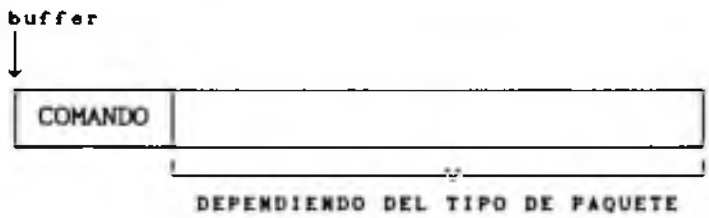
Los procesos TDIAG_TX, T1_TX, T2_TX, T3_TX y T4_TX son los procesos utilizados para la transmisión desde la terminal correspondiente; todos ellos funcionan de manera similar.

Para la terminal X, primeramente realiza un Wait(sem_com), que es el semáforo que indica que existe un comando listo para procesarse proveniente de la terminal X. La preparación de este comando se explicará más adelante.

```
proce_tX_tx()
{
     while(1) {
          Wait(TERMINAL[X].sem_com);
          transmisión_paq(#terminal, #canal lógico, buffer);
     }
}
```

Inmediatamente después se invoca una rutina común a todos los procesos transmisores llamada transmisión_paq().

La rutina transmisión_paq() espera recibir como parámetros el # de terminal del cual proviene el comando, el # de canal lógico asociado a esa terminal y un apuntador al buffer donde se encuentra el comando. Este buffer de comando debe tener el siguiente formato:



PARA EL FORMATO DE PAQUETES, VER TRANSMISION DE PAQUETES EN LA IMPLEMENTACION DEL NIVEL DE PAQUETES.

Los comandos permitidos son los siguientes:

'&' : Para desplegar la información de los canales lógicos.
'*' : Para desplegar la información de las terminales.
'C' : Para establecer una conexión.
'L' : Para liberar una llamada.
'S' : Para reiniciar un canal lógico.
'R' : Para la transmisión de *RR*.
'N' : Para la transmisión de *RNR*.
'D' : Para la transmisión de *DATOS*.
'I' : Para la transmisión de *INTERRUPCION*.

Las rutinas de servicio a las interrupciones por recepción provenientes de las terminales son manejadas de la siguiente manera:

Inicialmente las terminales se encuentran en estado inactivo. El nivel de paquetes no reconoce la terminal hasta que ésta se encuentra prendida y se haya oprimido la tecla <Return>.

En este momento, la terminal se encuentra activa y la rutina de servicio a posteriores interrupciones por recepción es redireccionada a una nueva rutina, que es la que va a ayudar en la preparación del buffer de comando que la terminal desea ejecutar. El proceso es el siguiente:

Desde la terminal se teclea un caracter que representa un comando. Este caracter es analizado por la rutina de servicio, permitiendo únicamente los caracteres en la lista de comandos previamente citada. Si el comando es diferente de 'D', entonces la terminal entra en un estado de com_in = 1, que significa que un comando está siendo introducido en el buffer de la terminal. A partir de este momento, los caracteres subsecuentes serán introducidos al buffer de comando de la terminal hasta que un caracter de <Return> sea encontrado. En este instante se efectúa un Signal2(TERMINAL[X].sem_com), que es el semáforo por el cual el proceso transmisor correspondiente a la terminal #X estaba esperando.

Si el comando es igual a 'D', ocasiona que la terminal entre en un estado de dat_in = 1, que significa que un paquete de *DATOS* se está introduciendo al buffer del nivel superior al cual apunta la ventana de transmisión del canal lógico correspondiente. A partir de este estado, los caracteres subsecuentes serán introducidos al buffer del nivel superior hasta que un caracter <Return> sea encontrado. Entonces, la rutina de servicio efectúa un Signal2(TERMINAL[X].sem_com), que es el semáforo por el cual el proceso transmisor correspondiente a la terminal #X estaba esperando.

Con lo anterior, podemos concluir que los procesos transmisores despiertan por dos causas: cuando existe un comando listo en el buffer de la terminal correspondiente o cuando existe un paquete de *DATOS* listo en el buffer del nivel superior.

Se encontró que los dos módulos de prueba anteriores son suficientes para comprobar en cierto grado el buen funcionamiento del nivel de paquetes; además ilustran el manejo de las primitivas que este nivel proporciona. Sirven también, como modelo base para la implementación del nivel superior o nivel 4.

CONCLUSIONES.

Esperamos que este trabajo de tesis dé las bases para la continuación del proyecto del PAD (Packet Assembler/Disassembler) en lo que se refiere a la implementación de las recomendaciones X.28 y X.29.

Actualmente en la Sección de Computación, se tienen implementadas ya las recomendaciones X.25, X.28 y X.29 en PL/M, utilizando el mismo nivel físico (PAD). Se espera una mejoría en la nueva implementación debido a la utilización del lenguaje C y a las experiencias recabadas en la realización del proyecto en PL/M.

La documentación del nivel de enlace está disponible con el Ing. Andrés Vega, responsable del trabajo de Laboratorios I y II, cuatrimestre septiembre-diciembre, 1988.

A partir de estos dos documentos es posible continuar con la implementación de las Recs. X.28 y X.29, como probable tema de tesis, y teniendo además conocimientos básicos sobre Redes de Computadoras y Protocolos de Comunicación, así como de programación en tiempo real.

Al final de este trabajo de tesis se encuentra la bibliografía referente a estos temas. El apéndice A, que habla sobre la descripción general del núcleo de concurrencia en C, es de suma importancia pues toca el tema de programación en tiempo real de una manera práctica, enfocada a la utilización y comprensión inmediata de las primitivas implementadas para el manejo de concurrencia en C.

APENDICE A.- DESCRIPCION GENERAL DEL NUCLEO DE CONCURRENCIA EN C.

El núcleo en C permite que el lenguaje C pueda ser utilizado como un lenguaje concurrente. En aplicaciones de tiempo real es necesario contar con esta herramienta para poder crear procesos o tareas que puedan ejecutarse "al mismo tiempo"; es decir, como si existiera un procesador para cada una de las tareas o procesos.

Generalmente los procesos que intervienen en un ambiente de tiempo real no son del todo independientes, sino que generalmente existe una intercomunicación entre ellos ocasionando lo que se conoce como sincronización de procesos. El concepto fundamental de concurrencia se basa en que van a existir diferentes procesos que van a competir por el uso de recursos.

Por ejemplo, puede darse el caso de que más de un proceso compita por el uso de una impresora; es decir, que todos ellos deseen al mismo tiempo mandar un archivo a imprimir. Es necesario que uno y solamente uno de ellos pueda utilizarla y no libere la impresora sino hasta que haya terminado de imprimir su archivo. Mientras tanto, los demás procesos deberán de esperar en una cola hasta que la impresora sea liberada; entonces, nuevamente sólo un proceso podrá utilizarla, en este caso el primero de la cola de espera.

Como recursos, aparte de impresoras, pueden ser también graficadores, discos, archivos, en fin, cualquier elemento del sistema que pueda ser compartido por los diferentes usuarios(análogamente un usuario equivale a un proceso).

El objetivo principal del sistema en tiempo real es que cada proceso "sienta" que el procesador le pertenece y no se de cuenta que existen otros procesos ejecutándose "al mismo tiempo".

Generalmente el núcleo consigue esto asignándole a un proceso el procesador como máximo un tiempo predeterminado(llamado "quantum") o hasta que el proceso libere el procesador voluntariamente; es decir, antes de que expire el "quantum".

Cuando alguna de las dos situaciones anteriores ocurre, el núcleo procede a realizar un cambio de contexto; esto es, guarda el medio ambiente del proceso actual y regresa el medio ambiente del nuevo proceso que va a ejecutarse.

Con medio ambiente nos referimos al valor de las variables de un proceso y valores de los registros de la máquina en un momento determinado.

Existe una tercera razón por la cual un proceso puede liberar el procesador. Por ejemplo, si un proceso requiere del uso de algún recurso, y este recurso no está disponible, tiene que entrar en estado de espera o bloqueado y liberar el procesador, con el fin de que otros procesos puedan ejecutarse.

En el caso del núcleo implementado en C, no se maneja el concepto de "quantum" para liberar el procesador, sino que cada proceso puede liberarlo de dos maneras: voluntariamente o cuando al competir por algún recurso, éste no esté disponible (entra en estado de bloqueado).

Antes de comenzar de lleno con el manejo del núcleo en C, es necesario introducir un concepto mas, los semáforos. Estos semáforos son manipulados en base a dos primitivas: Wait() y Signal().

Cuando un proceso solicita un recurso, tiene que hacer una petición por medio de la primitiva Wait(recurso). Supongamos que el proceso requiere del uso de la impresora. Tendrá que hacer entonces una petición de la siguiente manera:

```
Wait(impresora)
```

El valor de impresora corresponderá al número de impresoras existentes en el sistema. La primitiva Wait() realiza el siguiente análisis:

* Checa el valor de impresora.
* Si su valor es > 0 significa que por lo menos existe una impresora disponible para ese proceso.
  - Asigna la impresora al proceso.
  - El valor de impresora se decrementa en 1 (significa que va a haber una impresora menos disponible, pues se acaba de asignar una).
* Si su valor <= 0 significa que no hay impresoras disponibles.
  - El proceso entra en estado de bloqueado.
  - Decrementa en 1 el valor de impresora (significa que hay un proceso más en espera por la impresora).
  - Libera el procesador para que el núcleo lo asigne a un nuevo proceso.

La primitiva Signal() realiza lo contrario; es decir, libera el recurso para que otro proceso pueda utilizarlo. Por ejemplo, al liberar el uso de la impresora, un proceso tendrá que realizar la siguiente primitiva:

```
Signal(impresora).
```

Esta primitiva incrementa en 1 el valor de impresora (significa que existe una impresora mas disponible). Lo anterior ocasiona que sólo un proceso de los que estaban esperando por este recurso pueda salir del estado BLOQUEADO y pueda ser considerado para competir por el uso del procesador.

La variable impresora se conoce como semáforo, pues dependiendo de su valor puede hacer que un proceso pare o siga.

En conclusión, el valor inicial de un semáforo, corresponde al número de recursos de ese tipo que existen en el sistema (Ej. número de impresoras).

Con lo anterior se procederá a dar una visión global del funcionamiento del cuerpo principal del núcleo en C.

FUNCIONAMIENTO DEL NUCLEO EN C -

Conceptos:

Comenzaremos exponiendo diferentes conceptos que utiliza el núcleo para su ejecución. Hasta el momento hemos hablado de procesos en general que pueden existir en un sistema en un momento determinado. Más especificamente pueden existir cuatro tipos de procesos, que por su naturaleza, sólo uno de ellos es verdaderamente proceso (pues se genera un cambio de contexto cuando entra en ejecución) mientras que los otros tres son mas bien procedimientos (pues se invocan únicamente con un "call" sin cambio de contexto; es decir, como si fueran procedimientos o funciones).

Existen pues:

1> PROCESOS BASE

2> PROCEDIMIENTOS BASE

3> PROCEDIMIENTOS PERIODICOS

4> PROCEDIMIENTOS ACTIVADOS POR TEMPORIZADOR

En breve se explicará a detalle las caracteristicas de cada uno de ellos, pero primero es necesario esclarecer algunos conceptos más, como son el DP, TIMER, PRIORITY, PERIODIC, por lo que sólo se dará una descripción global.

1> PROCESO BASE

Cuando entran a ejecución generan un cambio de contexto en el medio ambiente del sistema. El núcleo decide su activación.

2> PROCEDIMIENTO BASE

Cuando entran en ejecución no generan un cambio de contexto. Son únicamente llamados. El núcleo decide su activación.

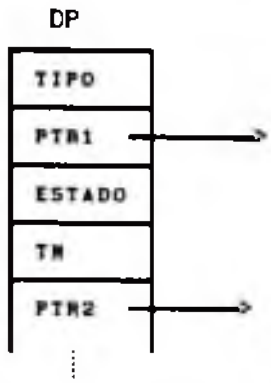
3> PROCEDIMIENTOS PERIODICOS

No generan un cambio de contexto en el medio ambiente. Son activados periódicamente después de un número predeterminado de ticks de reloj.

4> PROCEDIMIENTOS ACTIVADOS POR TEMPORIZADOR

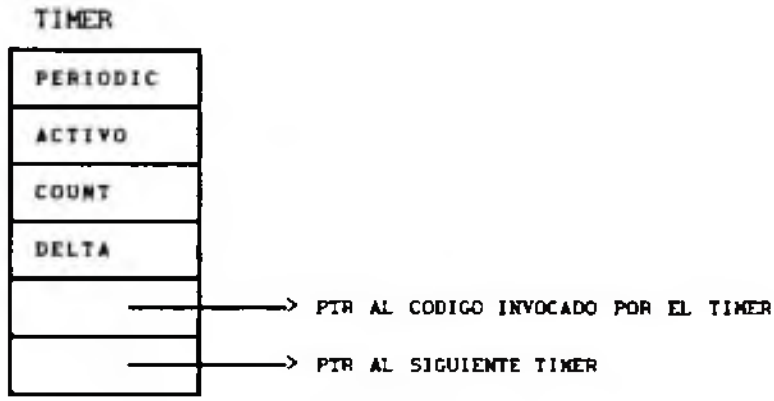
No generan un cambio de contexto en el medio ambiente y son activados cuando un contador para cada proceso alcanza 0. Cada tick de reloj decrementa el contador de todos los procesos en 1. El contador de cada proceso puede tomar cualquier valor entero positivo.

Para entender un poco más lo que en realidad es un proceso, diremos que se simplemente el código que representa la realización de una tarea. Cada proceso tiene asociado a él un Descriptor de Proceso (DP), en el cual se tiene información relevante como la que sigue:



- TIPO :PUEDE SER PROCESO BASE, PROCEDIMIENTO BASE O PROCEDIMIENTO PERIODICO.
- PTR1 :APUNTADOR AL CODIGO ASOCIADO AL PROCESO.
- ESTADO :INACTIVO, LISTO, CORRIENDO O BLOQUEADO.
- TM :TAMANO DEL STACK ASOCIADO AL PROCESO BASE.
- PTR2 :APUNTADOR AL INICIO DEL STACK DONDE EL PROCESO BASE GUARDA SU MEDIO AMBIENTE (VARIABLES LOCALES, VALORES DE LOS REGISTROS DE LA MAQUINA, DIRECCION DE REGRESO, ETC.).

En el caso de procedimientos activados por temporizador, la estructura de sus DP's, llamados TIMERS, es diferente:



- PERIODIC : NOS DICE SI EL TIMER ES PERIODICO O NO.
  0 = NO PERIODICO.
  1 = PERIODICO.
- ACTIVO : REPRESENTA EL ESTADO DEL TIMER.
  0 = INACTIVO.
  1 = ACTIVO.
- COUNT : REPRESENTA EL TIEMPO INICIAL EN EL QUE EL TIMER EXPIRARA E INVOCARA EL PROCEDIMIENTO CORRESPONDIENTE.
- DELTA : EL TIEMPO QUE QUEDA PARA LA EXPIRACION DEL TIMER (DESPUES DE QUE TODOS LOS TIMERS DELANTE DE EL HAYAN EXPIRADO).

Ahora bién, la parte del núcleo que se encarga de decidir qué proceso obtendrá el procesador se le llama Despachador, que de aquí en adelante llamaremos Dispatcher(), el cual es la parte principal del Kernel o Núcleo. Existen dos maneras de activar a los procesos:

1> Dispatcher(). Decidirá solamente entre procesos base y procedi -- mientos base.

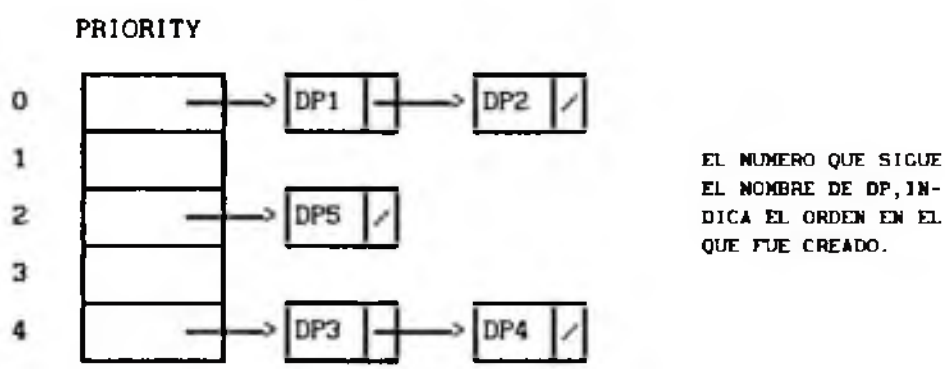
2> Interrupción de reloj. Activará procedimientos periódicos y pro - cedimientos activados por temporizador.

PRIORIDAD: Los procesos son manejados por prioridades, utilizándose un rango de 0 a 4, donde 0 es la prioridad más alta. Los procesos con mayor prioridad son los que obtendrán primero el procesador antes que los de menor prioridad. Cuando dos procesos tienen la misma prioridad, se forman en una cola conforme fueron llegando.

Estructuras con las que trabaja el Dispatcher() y la rutina de servicio del Reloj. Tipos de procesos que pertenecen a cada uno de ellos.

El Dispatcher() atiende únicamente a dos tipos de procesos: procesos base y procedimientos base. Trabaja con un arreglo de apuntadores a DP's llamado PRIORITY. Este arreglo va de 0 a 4, donde el índice 0 representa los procesos de mayor prioridad y el índice 4 los de más baja.

Ejemplo:



DP1 : Descriptor del Proceso 1
Prioridad = 0
Tipo = Proceso Base

DP2 : Descriptor del Proceso 2
Prioridad = 0
Tipo = Proced. Base

DP3 : Descriptor del Proceso 3
Prioridad = 4
Tipo = Proced. Base

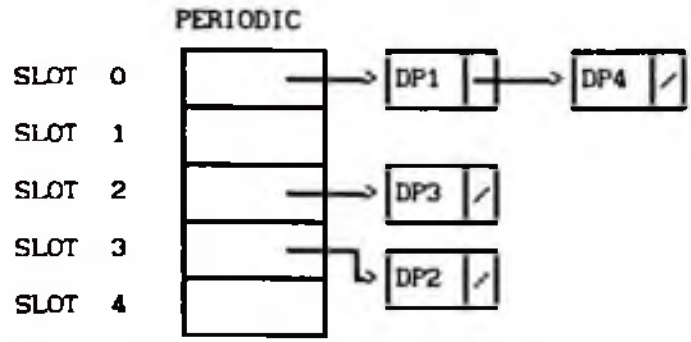
DP4 : Descriptor del Proceso 4
Prioridad = 4
Tipo = Proceso Base

DP5 : Descriptor del Proceso 5
Prioridad = 2
Tipo = Proced. Base

SOLO PROCESOS BASE Y PROCEDIMIENTOS BASE
El orden de ejecución del esquema anterior sería:
DP1 - DP2 - DP5 - DP3 - DP4.

La interrupcipn de Reloj atiende a los otros tipos de procesos y utiliza las siguientes estructuras:

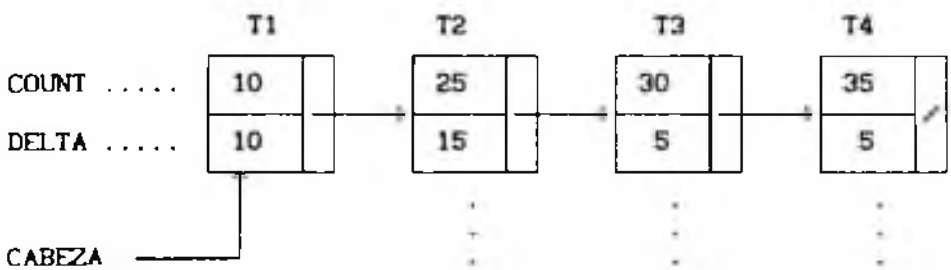
* Para Procedimientos Periódicos utiliza un arreglo de apuntadores a DP's llamado PERIODIC con un rango de prioridad de 0 a 4 (Ranuras de tiempo o Slots). Ejemplo:



DP1 : Descriptor del Proceso 1
Slot = 0
DP2 : Descriptor del Proceso 2
Slot = 3
DP3 : Descriptor del Proceso 3
Slot = 2
DP4 : Descriptor del Proceso 4
Slot = 0

SOLO PROCEDIMIENTOS PERIODICOS

* Utiliza una cola de TIMERS para procedimientos activados por temporizador. Ejemplo:



T1: Timer 1 que activa su procedimiento asociado después de 10 ticks de reloj (Delta ticks).

T2: Timer 2 que activa su procedimiento asociado después de 15 ticks de reloj (Después de los 10 de T1 + 15 de T2 = 25 de T2).

T3: Timer 3.

T4: Timer 4.

Estados en los que se pueden encontrar cada tipo de procesos.

La tabla A.1 muestra los estados posibles en los que pueden estar cada proceso en un momento determinado:

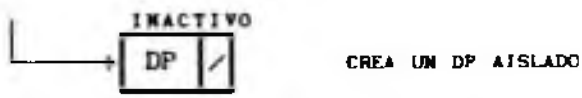
TIPO DE PROCESO	INACTIVO	LISTO	CORRIENDO	BLOQUEADO	ATENDIDO POR
PROCESO BASE	■	■	■	■	Dispatcher()
PROCED. BASE	■	■	■	■	Dispatcher()
PROCED. PERIODICO	■	■	■		Int Reloj
PROCED. ACT.X TEMP	ACTIVO/INACTIVO				Int Reloj

TABLA A.1.

PROCESOS BASE, PROCEDIMIENTOS PERIODICOS

INACTIVO:- El proceso ha sido creado, existe físicamente su DP en algún lugar de la memoria pero no es tomado en cuenta por el Dispatcher().

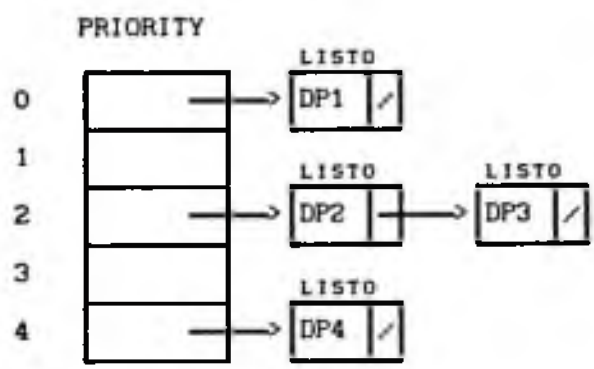
El proceso es creado mediante una primitiva llamada CreateProcess()

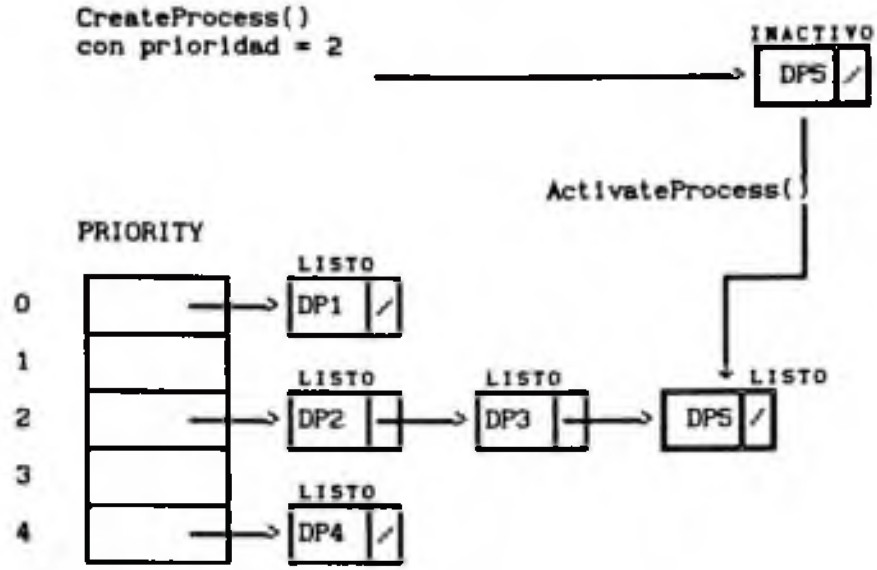


LISTO:- El DP es insertado en el arreglo PRIORITY dependiendo de la prioridad con la que haya sido creado. Es entonces cuando el Dispatcher() ya lo toma en cuenta para competir por el uso del procesador.

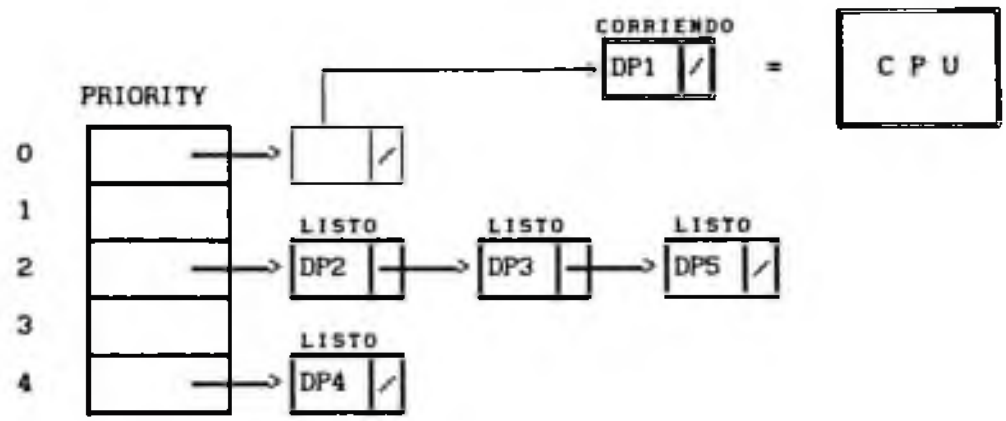
Lo anterior se consigue con la primitiva ActivateProcess().

Ejemplo, inicialmente:





CORRIENDO:- Cuando el Dispatcher() decide que el proceso asociado al DP de mayor prioridad entra a ejecución y por lo tanto es dueño temporal del procesador.



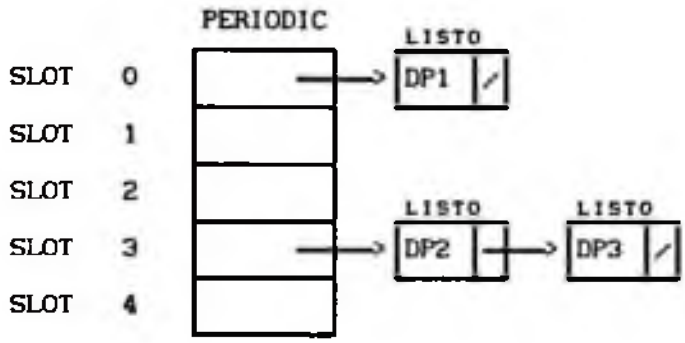
Nótese que el DP seleccionado es quitado físicamente del arreglo de PRIORITY.

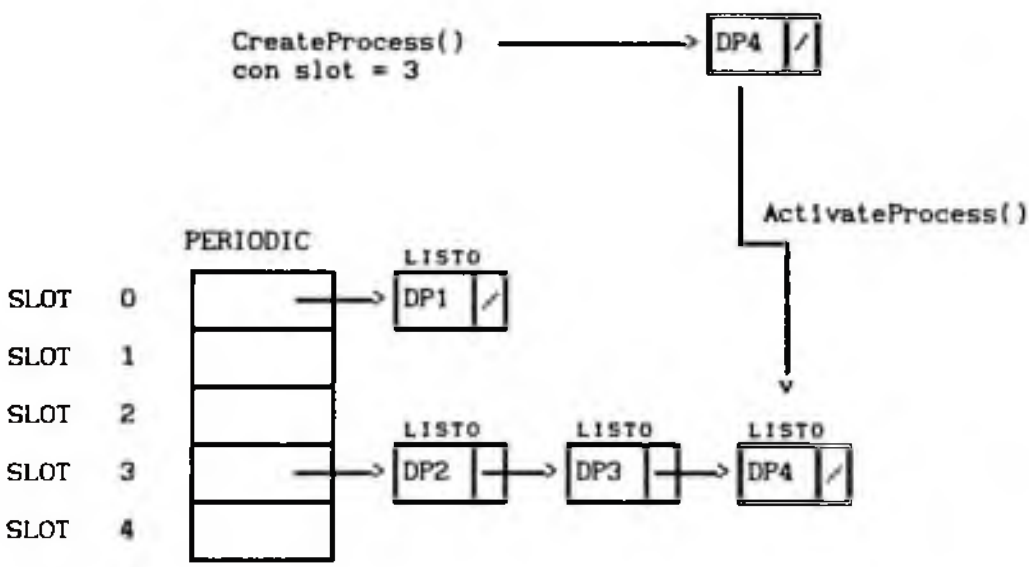
BLOQUEADO:- Cuando el proceso que está ejecutándose (Estado = CORRIENDO) realiza un Wait() a un semáforo cuyo valor <= 0 (No existen disponibles recursos de ese tipo). El proceso actual abandona el procesador para que alguien más pueda utilizarlo mientras el recurso se libera.

PROCESOS PERIODICOS

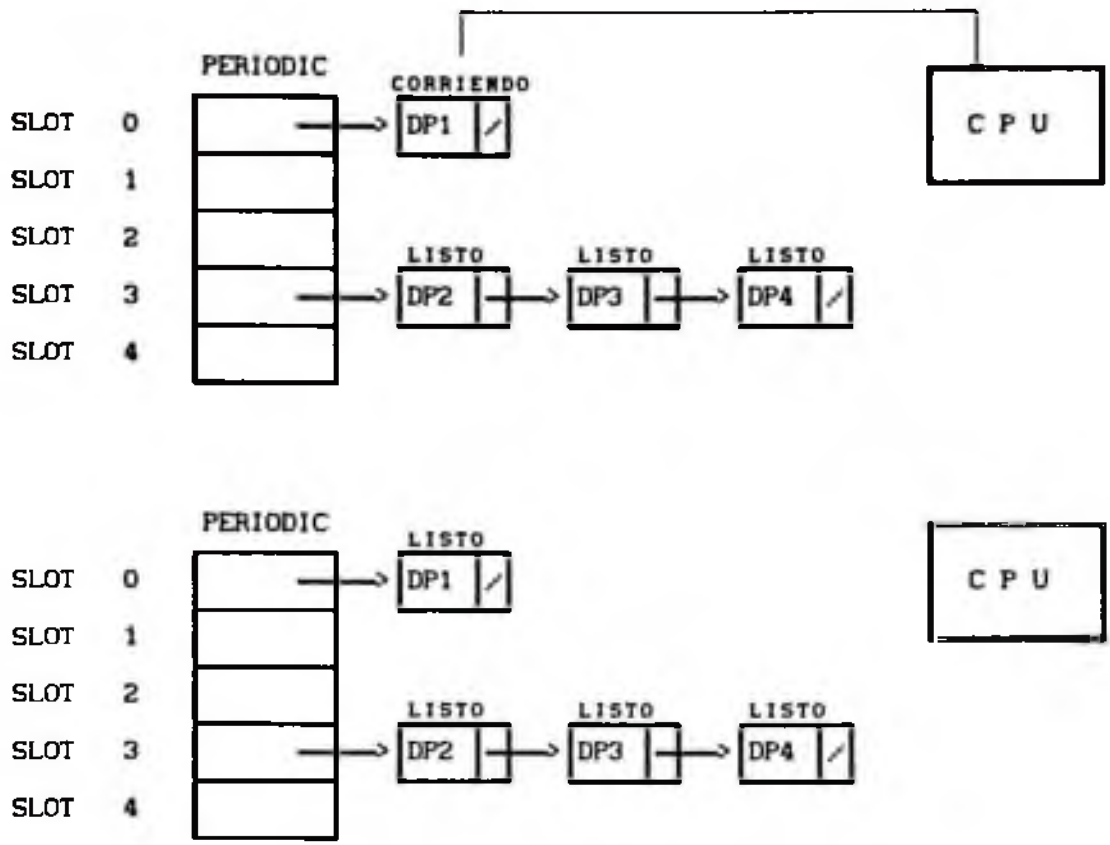
INACTIVO:- El DP asociado a un proceso es creado mediante la primitiva CreateProcess(). El DP existe fisicamente en algún lugar de la memoria pero no es tomado en cuenta por la interrupción de reloj.

LISTO:- El DP es insertado en el arreglo PERIODIC dependiendo del valor SLOT con el que fue creado. Es entonces cuando el DP ya es tomado en cuenta por la Int de Reloj. Lo anterior se consigue mediante la primitiva ActivateProcess().





CORRIENDO:- Cuando la Int de Reloj encuentra que el proceso asociado al DP en el SLOT siguiente es el próximo a entrar en ejecución.



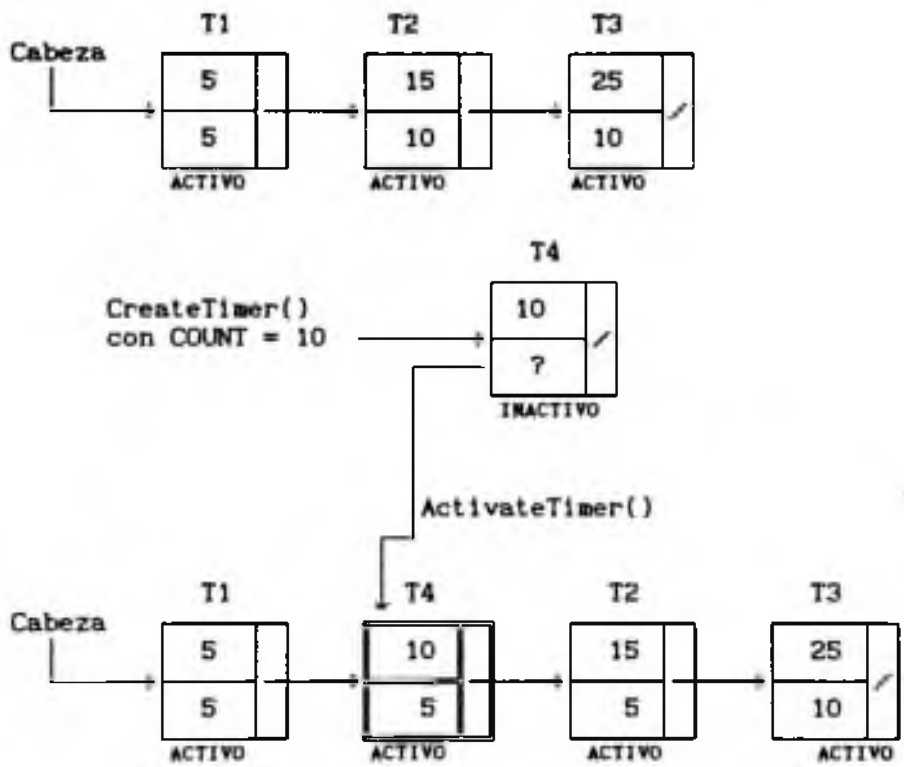
Nótese que el DP seleccionado no es retirado de PERIODIC. Característica fundamental de los procedimientos periódicos. Cuando termine de ejecutarse entrará al estado = LISTO. El estado = BLOQUEADO no es permitido.

PROCEDIMIENTOS ACTIVADOS POR TEMPORIZADOR.

INACTIVO:- El TIMER asociado a un proceso es creado mediante la primitiva CreateTimer(). El TIMER existe fisicamente en algún lugar de la memoria pero no es tomado en cuenta por la Int de Reloj.

ACTIVO:- El TIMER es insertado en la cola de timers en el lugar que le corresponda dependiendo de su valor "count". El TIMER es entonces tomado en cuenta por la rutina de interrupción. Lo anterior se consigue mediante la primitiva StartTimer().

Ejemplo:



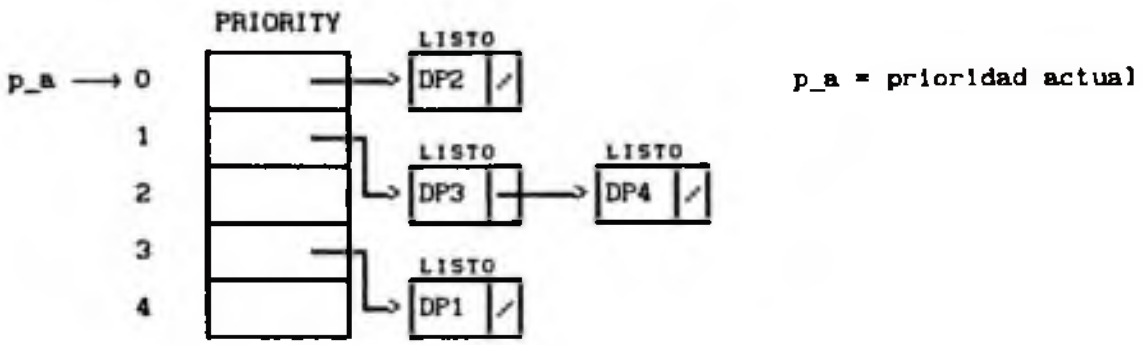
Funcionamiento General del Dispatcher()

El Disptacher() es simplemente un ciclo en el cual barre de arriba hacia abajo el arreglo PRIORITY, ejecutándo los procesos asociados a los DP's que va encontrando en cada índice de PRIORITY.

Utiliza una variable de control llamada prioridad_actual, la cual representa el índice de PRIORITY.

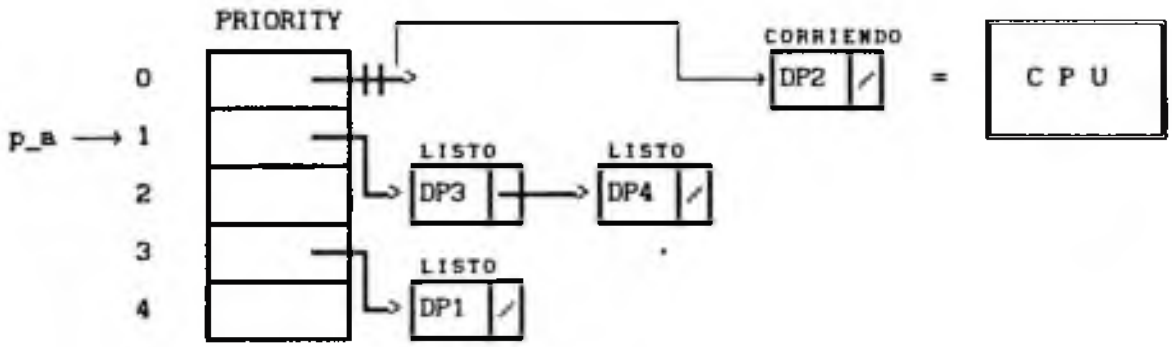
El Dispatcher() sigue la siguiente regla: Atender al DP que tenga la prioridad más alta. Ejemplo:

Se tienen los siguientes procesos base y procedimientos base en PRIORIRITY (Fueron previamente creados con CreateProcess() y activados con ActivateProcess()).



Al entrar el Dispatcher() en ejecución lo que hace es:

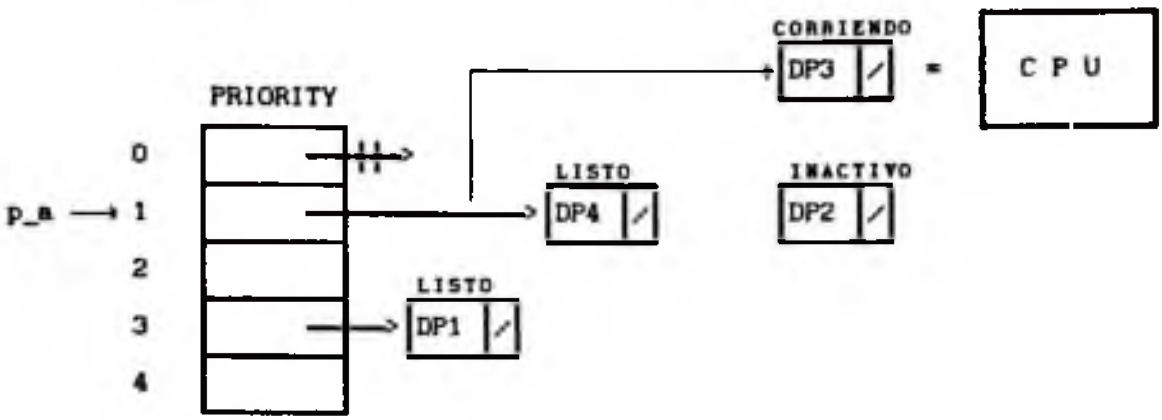
```
proceso_actual = PRIORITY[prioridad_actual];
Ejecuta proceso_actual => DP2;
```



Como ya no hay mas DP's en PRIORITY[0], prioridad_actual incrementa su valor hasta el siguiente índice de PRIORITY donde por lo menos exista un DP, por lo tanto prioridad_actual = 1.

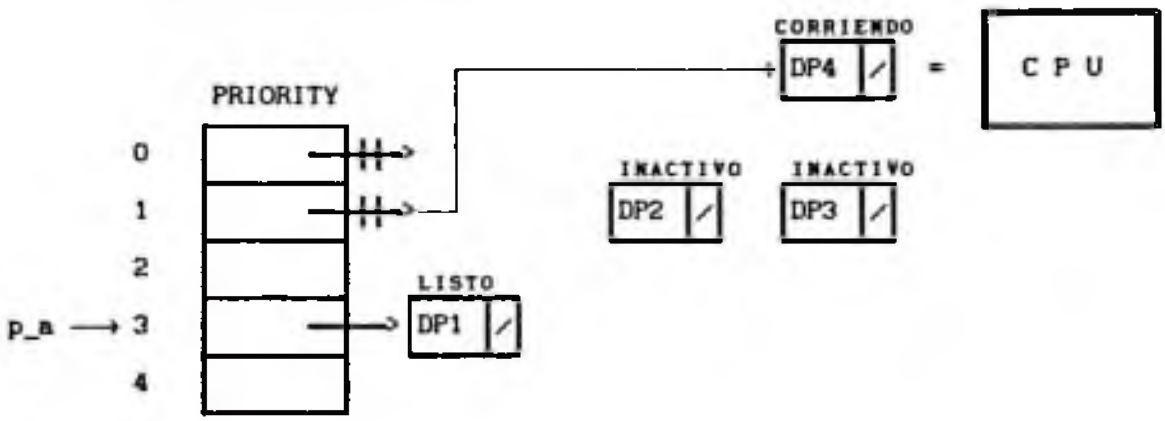
Ahora, DP2 termina de correr normalmente, entonces libera el procesador, pasa a estado = INACTIVO y regresa el control al Dispatcher().

```
proceso_actual = PRIORITY[p_a] = DP3
Ejecuta proceso_actual => DP3
prioridad_actual = 1
```



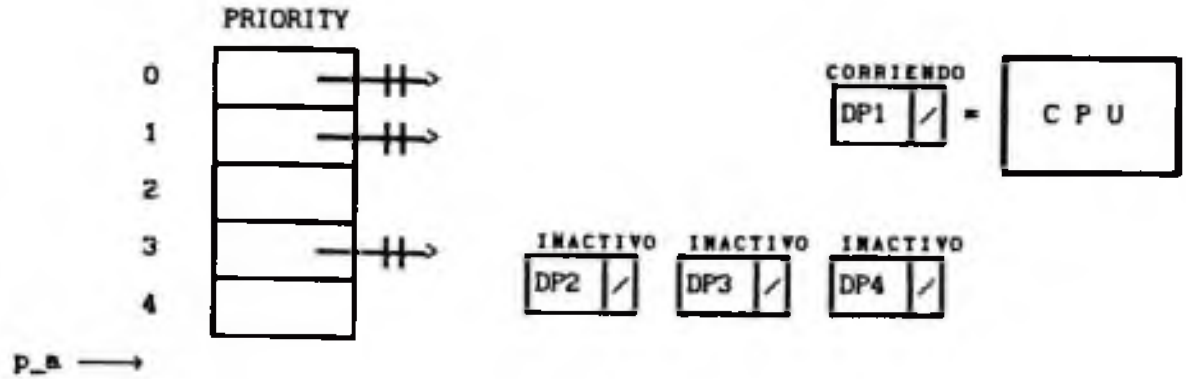
DP3 termina de correr normalmente, libera el procesador y regresa el control al Dispatcher().

```
proceso_actual = PRIORITY[p_a] = DP4
Ejecuta proceso_actual => DP4
prioridad_actual = 3
```



DP4 termina de correr normalmente, libera el procesador y regresa el control al Dispatcher().

```
proceso_actual = PRIORITY[p_a] = DP1
Ejecuta proceso_actual => DP1
prioridad_actual = 5
```



Cuando DP1 termina de correr, libera el procesador y regresa el control al Dispatcher(). Como prioridad_actual está fuera del rango de prioridades el Dispatcher() termina su ejecución.

NOTAS:

* Para evitar que el Dispatcher() termine su ejecución, se utiliza un proceso NULO que no va a permitir que prioridad_actual salga del rango.

* Los procesos que terminaron de correr pasan al estado = INACTIVO, y sólo serán tomados en cuenta nuevamente cuando sean reactivados (ActivateProcess()).

* Cada vez que se activa un proceso (ActivateProcess()) si la prioridad -en valor numérico- que contiene el proceso activado es menor que prioridad_actual, entonces prioridad_actual se actualizará al nuevo valor. Lo anterior ocasionará que el Dispatcher() atienda primero a los DP's de mayor prioridad (menor valor numérico).

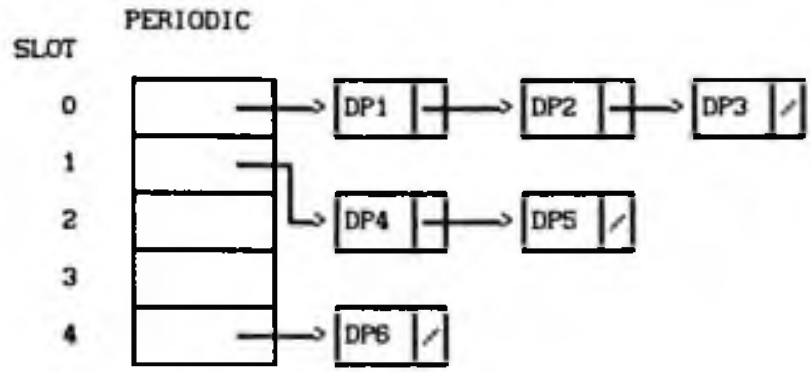
IMPORTANTE: En el funcionamiento anterior se asume que los procesos no ejecutan operaciones de Wait() y Signal(), pues aún no han sido explicados a detalle. La política de funcionamiento del Dispatcher() sigue siendo la misma.

Funcionamiento de la rutina activada por la Int de Reloj.

Utilizada para ejecutar procedimientos periódicos y procedimientos activados por TIMERS.

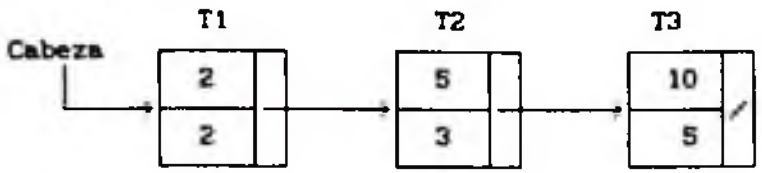
Utiliza un tiempo base = 18 donde cada tick de reloj decrementa en 1 este tiempo base (18.2 ticks de reloj en PC-XT equivale a 1 seg.).

* Cuando el tiempo base llega a 0 (cada 18 ticks) se atienden los DP's que están en el arreglo PERIODIC. El índice usado para escoger la ranura de PERIODIC es controlada por una variable llamada ranura_actual, que inicialmente vale 0. Ejemplo:

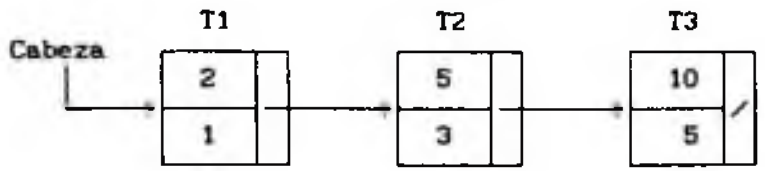



```
18 Ticks !!!
     Ejecución secuencial
     DP1 -> DP2 -> DP3
     ranura_actual se incrementa en módulo 5
     entonces ranura_actual = 1
18 Ticks !!!
     Ejecución secuencial
     DP4 -> DP5
     ranura_actual = 2
18 Ticks !!!
     No hay ejecución
     ranura_actual = 3
18 Ticks !!!
     No hay ejecución
     ranura_actual = 4
18 Ticks !!!
     Ejecución secuencial
     DP6
     ranura_actual = 0
18 Ticks !!!
     Ejecución secuencial
     DP1 -> DP2 -> DP3
     ranura_actual = 1
          .
          .
          .
       etc
```

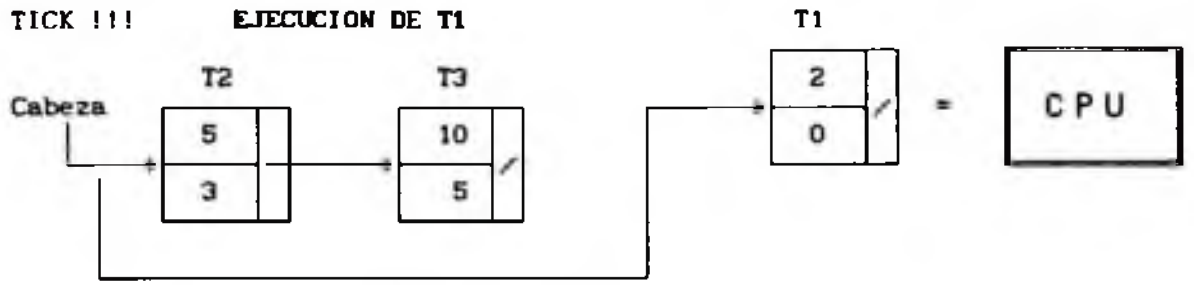
• Por otro lado, cada tick de reloj decrementa el valor de DELTA del primer TIMER en la cola de timers. Ejemplo:



TICK !!!



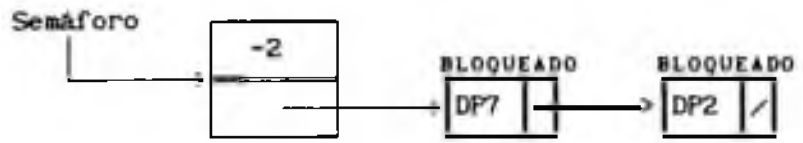
TICK !!! **EJECUCION DE T1**



Estructuras de los Semáforos.

Los semáforos tienen una estructura compuesta por dos elementos, el primero es el valor del semáforo y el segundo un apuntador a una cola de DP's que están en estado = BLOQUEADO, en espera a que el recurso representado por el semáforo sea liberado.

Ejemplo:



Acciones del Wait()

Cuando un proceso en ejecución realiza un Wait() sobre algún semáforo, el valor del semáforo es decrementado en 1.

Si el valor del semáforo después de esta operación es negativo, (significa que el recurso no está disponible) el proceso actual pasa del estado = CORRIENDO al estado = BLOQUEADO y se forma en la cola del semáforo. El procesador es liberado y se transfiere el control al Dispatcher(). Recordemos que la primitiva Wait() sólo puede ser usada por procesos base y por procedimientos base.

Acciones del Signal()

Cuando un proceso en ejecución realiza un Signal() sobre algún semáforo, el valor del semáforo es incrementado en 1.

Si existe algún DP en la cola de espera del semáforo, este es quitado de aquí e insertado en el índice de PRIORITY que le corresponde. Es decir, pasa del estado = BLOQUEADO al estado = LISTO. Si el proceso recién insertado es de mayor prioridad que prioridad_actual, prioridad_actual es actualizada.

El proceso que invocó el Signal() puede perder o no el procesador (Transfiriendo el control al Dispatcher()) dependiendo del tipo de Signal() (Signal() Signal2()). Lo anterior se explicará a detalle en la siguiente sección.

MODULOS PRINCIPALES DEL NUCLEO EN C =

Los módulos que forman el cuerpo principal del núcleo son:

1> <NUCLEO>
2> <CORRUTINAS>
3> <SEMAFOROS>
4> <TEMPORIZADORES>
5> <ABORT>

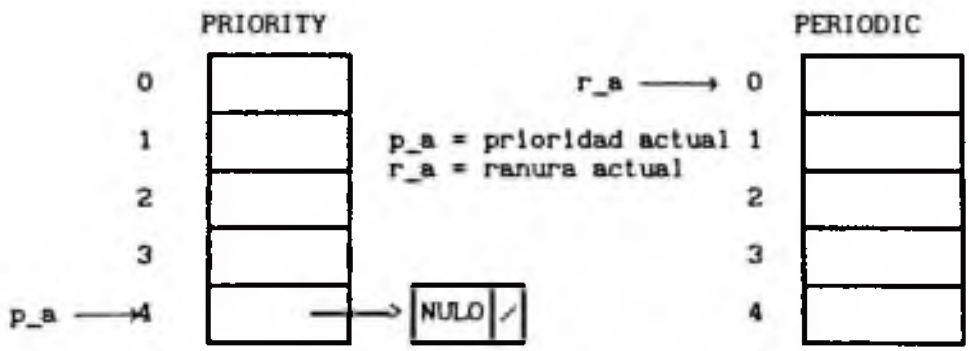
Se explicarán las funciones de cada módulo relacionados con el funcionamiento global del núcleo, a excepción del módulo de corrutinas, cuyas funciones son específicamente relacionadas al cambio de contexto entre procesos y creación del Stack para cada proceso base.

1.- *<NUCLEO>*

InitKernel()

- Inicializa módulos externos como: Atomic(), Storage(), Corrutin(), Lowio(), Inout(), Abort().

- Crea las siguientes estructuras de trabajo:



- Crea y activa un procedimiento NULO con prioridad = 4, cuyo código es la reactivación de este mismo proceso. Esto con el fin de que el Dispatcher() no termine sino hasta la ejecución del proceso abort (contenido en el módulo abort).

- Pone en el vector de interrupciones la dirección de las nuevas rutinas de interrupción de reloj y teclado.

ExecuteKernel()

- Manda llamar al Dispatcher().

CreateProcess()

- Crea el DP del proceso con información como prioridad, tipo de proceso, dirección del código del proceso, tamaño del Stack, etc. El estado = INACTIVO.

ActivateProcess()

- Inserta los procesos base y procedimientos base en PRIORITY y los procedimientos periódicos en PERIODIC.

- Si la prioridad del proceso activado es menor numéricamente (mayor prioridad) que prioridad_actual, entonces se actualiza prioridad_actual a la prioridad del proceso recién activado.

- Procesos base y procedimientos base sólo se insertan si su estado = INACTIVO, pasando a estado = LISTO.

- Procedimientos periódicos se insertan sin importar su estado, pasando a estado = LISTO.

Dispatcher()

- Es un ciclo controlado por la variable prioridad_actual.
  While prioridad_actual < MAXPRIORITY, donde
  MAXPRIORITY actualmente = 5

- El cuerpo del ciclo es el siguiente:

  - Utiliza GetProcess() para sacar el DP del proceso al que le corresponde la utilización del procesador.

  - Si el proceso extraído con GetProcess() es proceso base, entonces realiza un Transfer(), esto es, un cambio de contexto y pasa del estado = LISTO al estado = CORRIENDO.

  - Si es procedimiento base, entonces hace únicamente un llamado al procedimiento correspondiente. Cuando regresa, el estado del procedimiento = INACTIVO.

GetProcess()

- Saca el 1er. DP de la cola apuntada por PRIORITY[prioridad_actual] y mientras la cola apuntada por PRIORITY[prioridad_actual] esté vacía, incrementa prioridad_actual en 1. El proceso NULO con prioridad = 4 evita que prioridad_actual alcance el valor de MAXPRIORITY = 5.

GetPrio()

- Para saber el valor de prioridad de un proceso base o procedimiento base.

SetPrio()

- Para poner el valor de la prioridad de un proceso a un valor dado. Si la prioridad actual del proceso es diferente a la nueva prioridad y el estado = LISTO entonces se saca de la cola de PRIORITY en la que esté y se mete al final de la cola de PRIORITY correspondiente al nuevo valor.

2.- <SEMAFOROS>

CreateSem()

WaitSem()

- Si el valor del semáforo sobre el cual se hace el Wait() es <= 0 entonces:

  - Estado del proceso actual = BLOQUEADO.
  - Pone el DP del proceso actual al final de la cola del semáforo.
  - Transfiere el control al Dispatcher().

- Decrementa el valor del semáforo.

Signal()

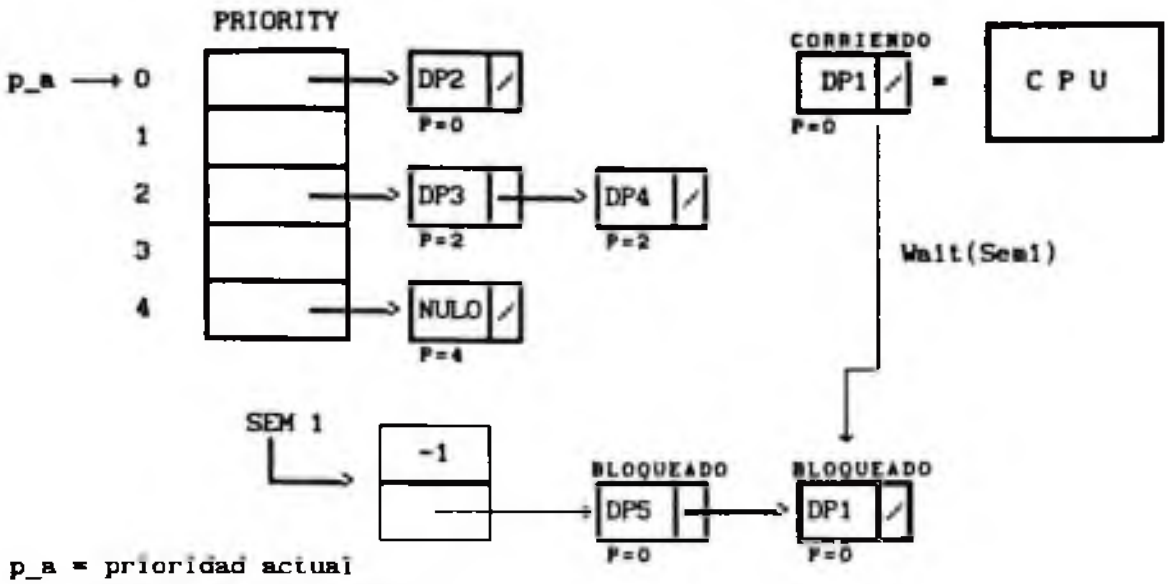
- Incrementa el valor del semáforo.

- Si existe un DP como primer elemento de la cola del semáforo, realiza las siguientes acciones:

  - Quita el primer DP de la cola del semáforo y lo inserta en la cabeza de PRIORITY que le corresponde, lo cual actualiza prioridad actual.
  - Inserta el DP del proceso actual al final de la cola de PRIORITY que le corresponda.
  - Transfiere el control al Dispatcher().

NOTA: Wait() y Signal() sólo pueden ser usados por procesos base y por procedimientos base.

Ejemplo de Wait() y Signal(). Supongamos el siguiente estado del núcleo:

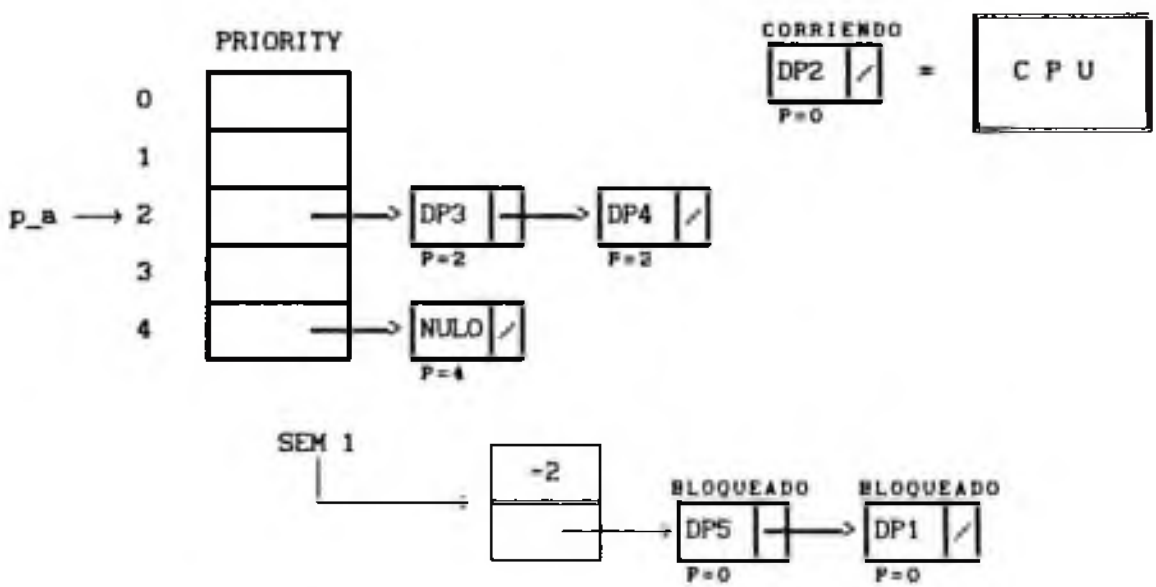


p_a = prioridad actual
p = prioridad del proceso

DP1 se encuentra corriendo y ejecuta un Wait(Sem1), por lo tanto se realizan las siguientes acciones:

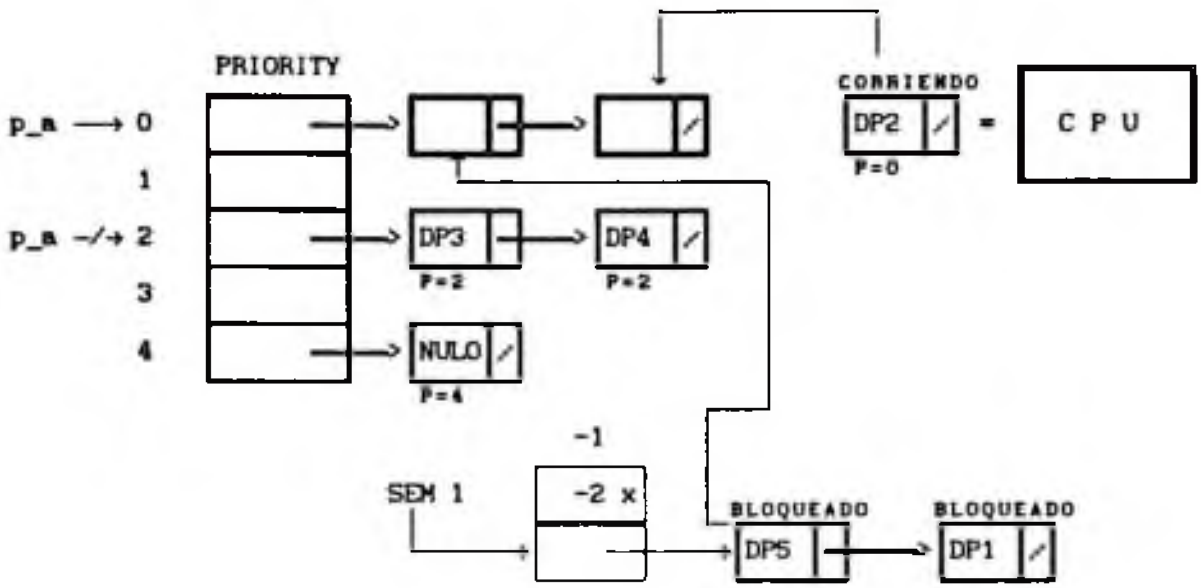
- Estado del proceso actual = BLOQUEADO.
- Inserta DP del proceso actual al final de la cola del semáforo.
- Transfiere el control al Dispatcher().

Quedando el estado del núcleo de la siguiente manera:

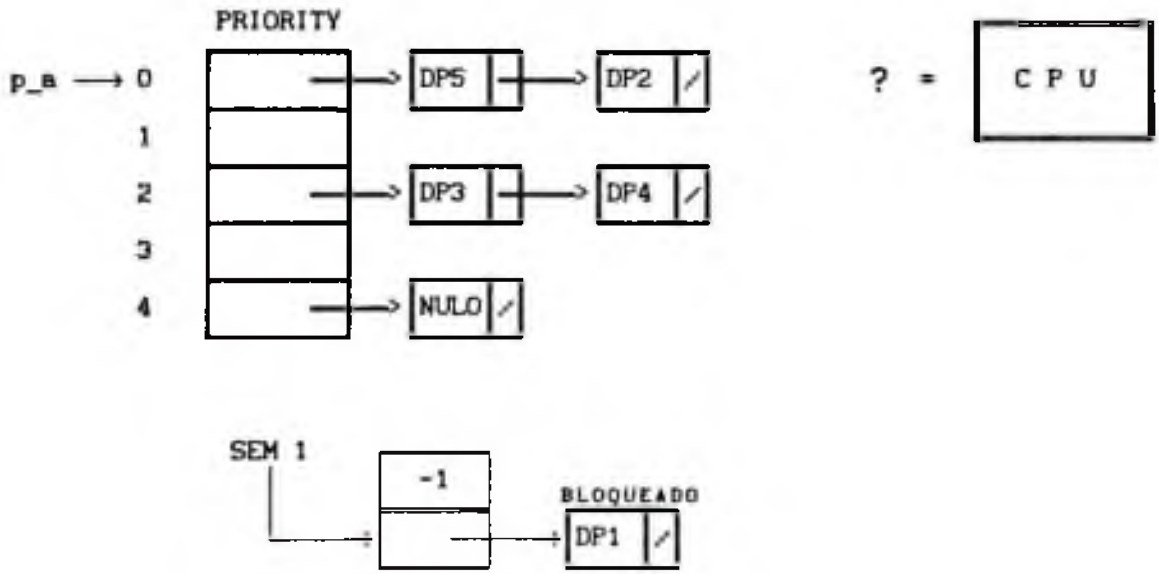


Si al efectuar el Wait(), el valor del semáforo es > 0, únicamente se decrementa y el proceso actual continúa con su ejecución normal.

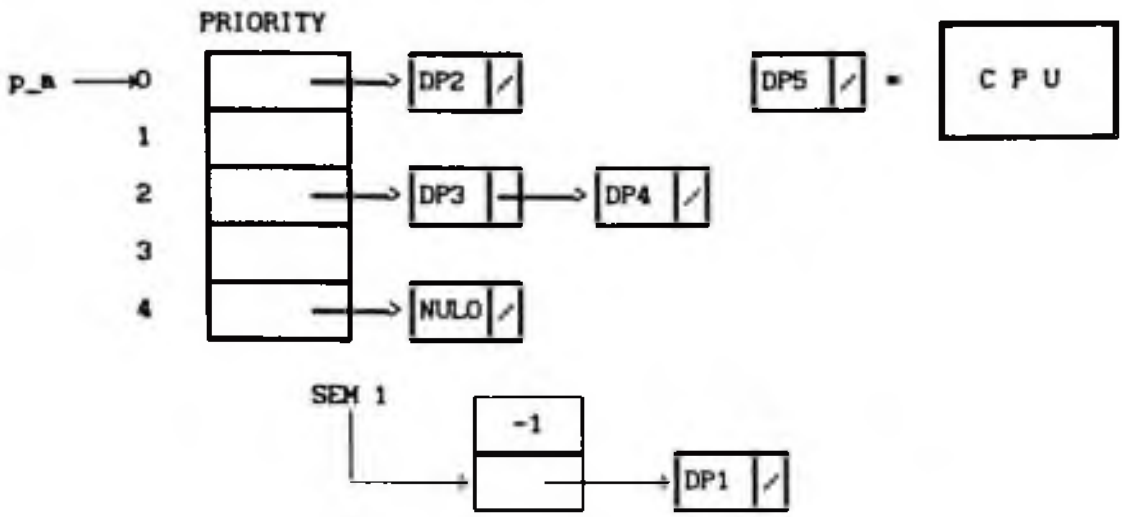
Tomando como inicio el estado del núcleo anterior, supongamos que el proceso actualmente en ejecución -DP2- realiza un Signal(Sem1), entonces se realizarán las siguientes acciones:



Llegando al siguiente estado del núcleo:



Inmediatamente después se le transfiere el control al Dispatcher(), quedando el estado final del núcleo como sigue:



Si al efectuar la operación Signal(), el semáforo no tiene ningún DP en su cola de espera, el valor del semáforo simplemente se incrementa y el proceso actual continúa con su ejecución normal.

Signal2()

- Incrementa el valor del semáforo.
- Si existe un DP como primer elemento de la cola del semáforo, lo quita de ahí y lo inserta en la cabeza de PRIORITY que le corresponde, lo cual actualiza prioridad_actual.
- El proceso actual no pierde el procesador a diferencia de Signal().
- Esta primitiva la pueden usar aparte de los procedimientos base y procesos base, los procedimientos periódicos y procedimientos activados por temporizador.

MulSignal()

- Funciona de igual manera que Signal(), con la diferencia de que el valor del semáforo no es incrementado en 1, sino en un valor INC entero positivo. De esta manera, son despertados de 0 a INC procesos que estaban formados en la cola del semáforo.

MulSignal2()

- Funciona de igual manera que Signal2(), sólo que el semáforo es incrementado en un valor INC, por lo tanto salen INC (como máximo) procesos de la cola de espera del semáforo y entran en la cabeza de PRIORITY que les corresponde.

GetSemVal()

- Regresa el valor actual del semáforo.

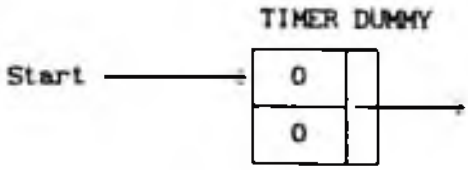
SetSemVal()

- El valor del semáforo es puesto a un valor específico.

3.- <TEMPORIZADORES>

InitTimer()

- Utiliza CreateTimer() para crear la siguiente estructura de trabajo:



CreateTimer()

- Para crear el TIMER con su información respectiva:

  Periódico: Si es periódico o no.

  Cuenta: Cuenta inicial para la expiración del TIMER.

  Procedimiento: Apuntador al código que pertenece al procedimiento que se activará cuando Cuenta = 0.

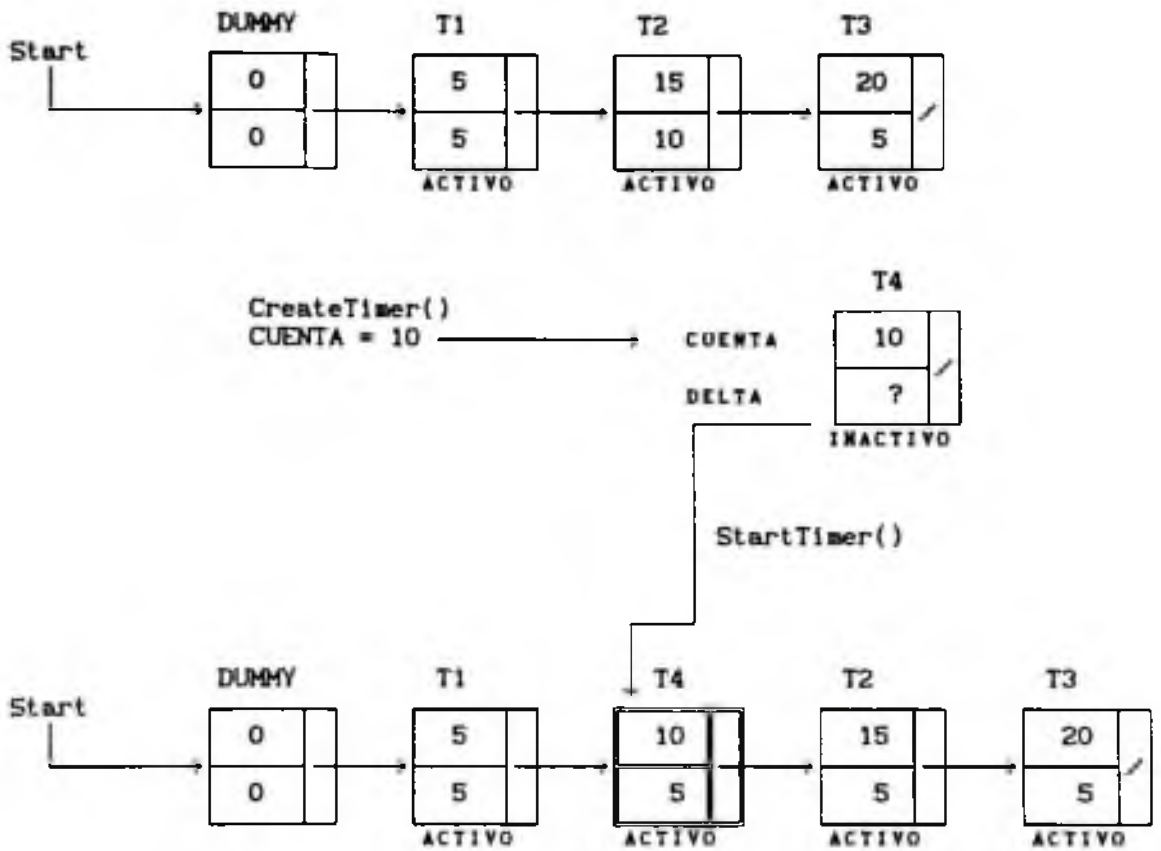
- Cuando un TIMER es periódico significa que al expirar llamará el procedimiento asociado e inmediatamente después se reinsertará en la cola de timers.

- Estado del TIMER = INACTIVO.

- El TIMER existe fisicamente en algún lugar de la memoria pero no es tomado en cuenta por la rutina de interrupción.

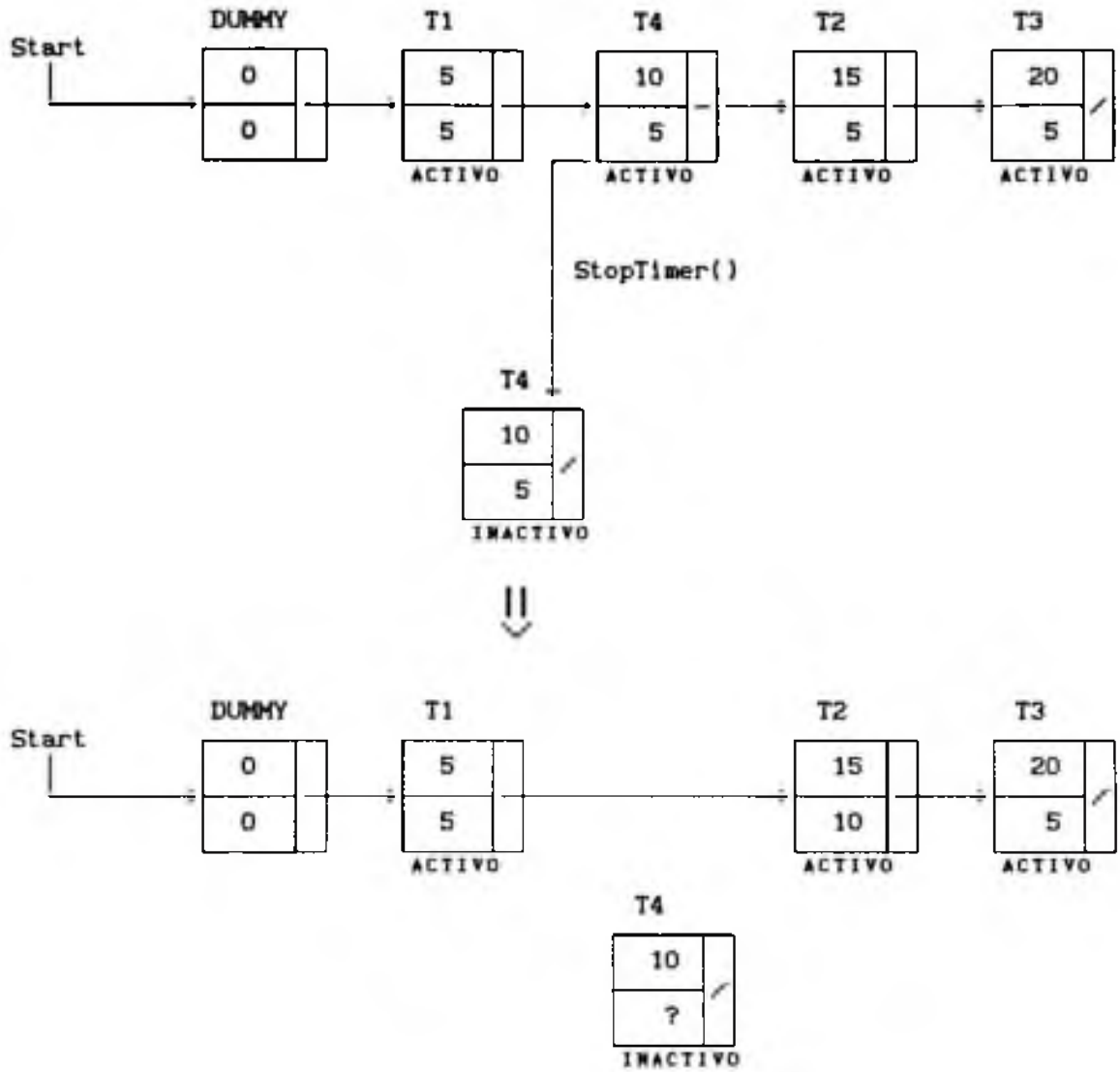
StartTimer()

- Para activar un TIMER.
- Lo inserta en la cola de timers, donde ya será tomado en cuenta por la rutina de interrupción.
- Si el TIMER estaba ACTIVO, primero lo DESACTIVA



StopTimer()

- Para desactivar un TIMER.
- Ya no será tomado en cuenta por la rutina de interrupción pero existe físicamente con estado = INACTIVO.

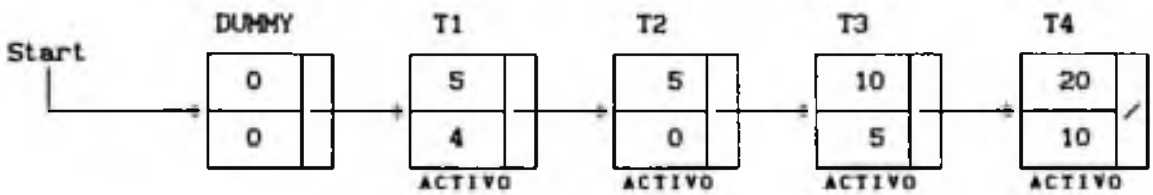


KillTimer()

- Para liberar la memoria de un TIMER.
- Si su estado = ACTIVO, lo desactiva (StopTimer()).
- Libera la memoria.

TimerHandler()

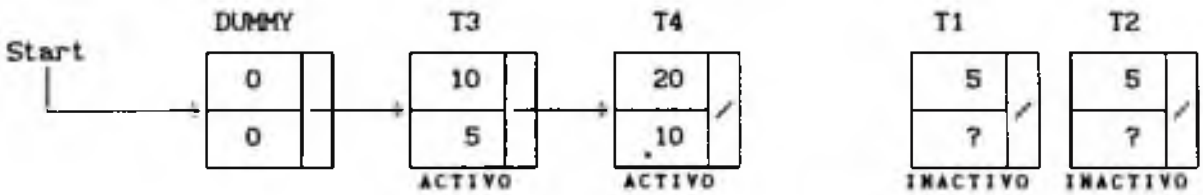
- Es la rutina de servicio del reloj.
- Cada tick del reloj decrementa el valor de Delta del primer TIMER. Si su valor = 0 ejecuta el procedimiento asociado y el TIMER queda con estado = INACTIVO, a menos de que sea periódico.



DESPUES DE 4 TICKS DE RELOJ EJECUCION SECUENCIAL-
DE LOS PROCEDIMIENTOS ASOCIADOS A:

T1 -> T2

QUEDANDO EL ESTADO FINAL DE LA COLA DE TIMERS CO-
MO SIGUE:



4.- *<ABORT>*

InitAbort()

- Crea un semáforo con valor inicial = 0.

- Crea y activa un proceso base con máxima prioridad.

- La primera instrucción del proceso es un Wait() al semáforo abort, por lo que entra en estado = BLOQUEADO.

- La combinación de las teclas CTRL ALT DEL generan un Signal() al semáforo abort, lo cual despierta al proceso ocasionando el final del NUCLEO con la siguiente instrucción:

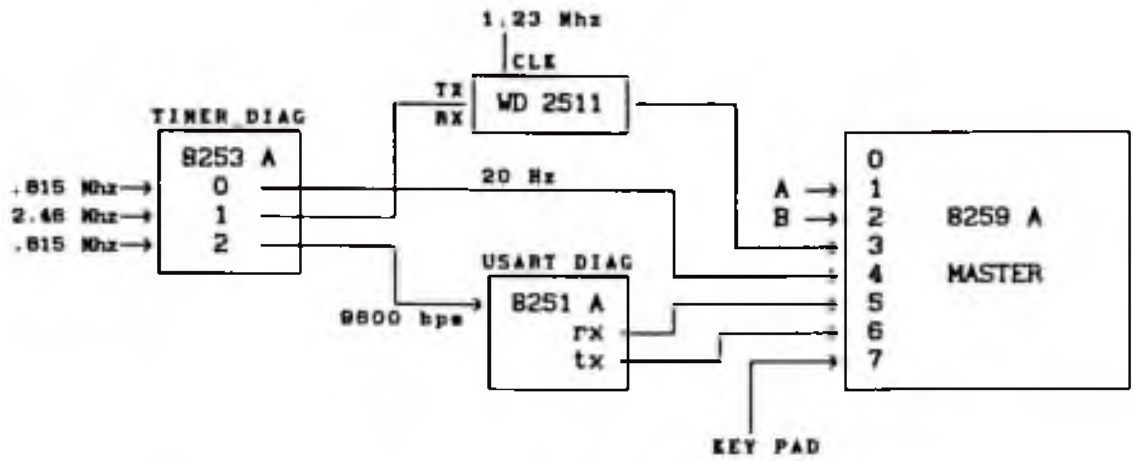
  prioridad_actual = MAXPRIORITY = 5.

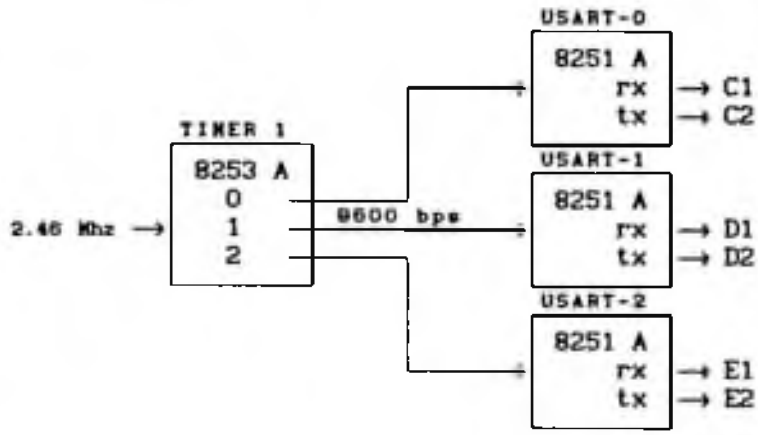
  Recordemos que esta condición ocasiona que el Dispatcher() termine su ciclo.

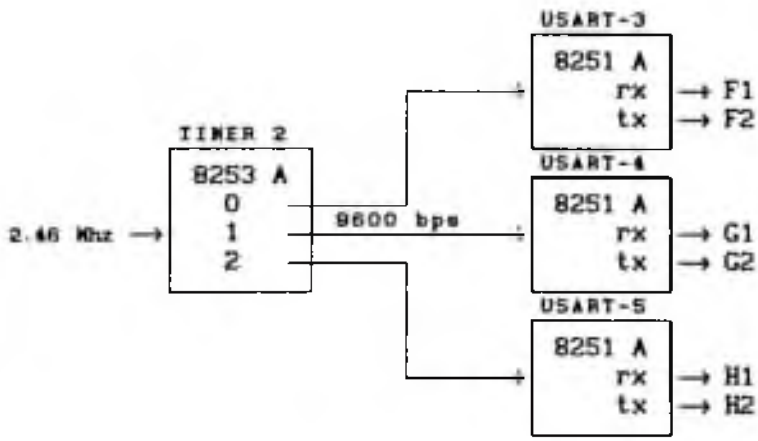
- Lo anterior regresa el control a la función ExecuteKernel() del módulo <NUCLEO>, cuyas instrucciones eran:

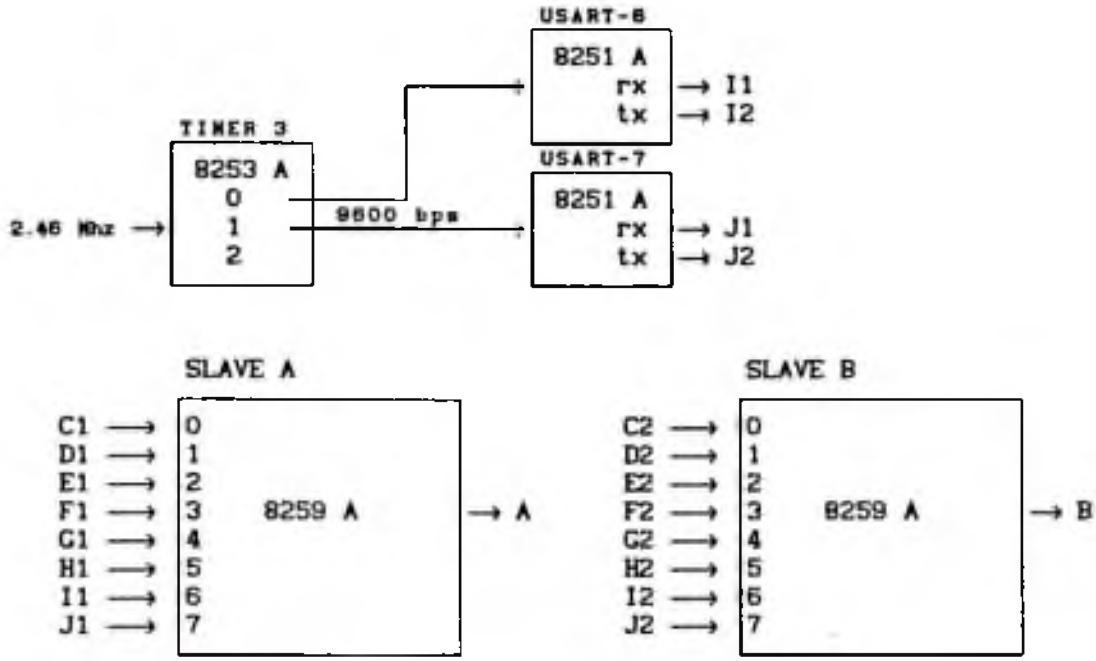
```
Dispatcher()        ACABA DE TERMINAR SU  EJECUCION.
RestoreInts()       SE REESTABLECEN LAS  INTERRUPCIONES
DisableInts()       DE RELOJ  Y  TECLADO  A  SU  ESTADO
Enable()            ORIGINAL Y EL NUCLEO TERMINA.
```

APENDICE B.- CONFIGURACION GLOBAL DEL PAD.








USART-6
8251 A
rx → I1
tx → I2
TIMER 3
8253 A
0
1
2
2.46 Mhz →
9600 bps
USART-7
8251 A
rx → J1
tx → J2
SLAVE A
C1 → 0
D1 → 1
E1 → 2
F1 → 3
G1 → 4
H1 → 5
I1 → 6
J1 → 7
8259 A
→ A
SLAVE B
C2 → 0
D2 → 1
E2 → 2
F2 → 3
G2 → 4
H2 → 5
I2 → 6
J2 → 7
8259 A
→ B

DECODIFICACION DE PUERTOS DEL PAD

PERIFERICO	DECODIFICACION	NO. DE REGS.
WD2511	0xA0...0xBE	16
Usart_Diag	0xD8, 0xDA	2
Timer_Diag	0xD0, 0xD2, 0xD4, 0xD6	4
Master	0xDC, 0xDE	2
Slave A	0x120, 0x121	2
Slave B	0x122, 0x123	2
Timer_1	0x110, 0x111, 0x112, 0x113	4
Timer_2	0x114, 0x115, 0x116, 0x117	4
Timer_3	0x118, 0x119, 0x11A, 0x11B	4
Usart_0	0x100, 0x101	2
Usart_1	0x102, 0x103	2
Usart_2	0x104, 0x105	2
Usart_3	0x106, 0x107	2
Usart_4	0x108, 0x109	2
Usart_5	0x10A, 0x10B	2
Usart_6	0x10C, 0x10D	2
Usart_7	0x10E, 0x10F	2

BIBLIOGRAFIA.

Andrew A. Tanenbaum.
COMPUTER NETWORKS.
Second Edition.
Prentice-Hall, Inc. 1988.

R.J. Deasington.
X.25 EXPLAINED:
Protocols for Packet Switching Networks.
Second Edition.
Ellis Horwood Limited, 1987.

WESTERN DIGITAL CORPORATION.
WD2511 X.25 Packet Network Interface (LAPB), 1979.

CCITT.
Libro Rojo.
Tomo VIII, Fascículo VIII.3, 1984.

S.T. Allworth.
INTRODUCTION TO REAL TIME SOFTWARE DESIGN.
Mc.Millan Publishers LTD, London 1981.

M. Ben-Ari.
PRINCIPLES OF CONCURRENT PROGRAMMING.
Prentice-Hall Inc. 1982.

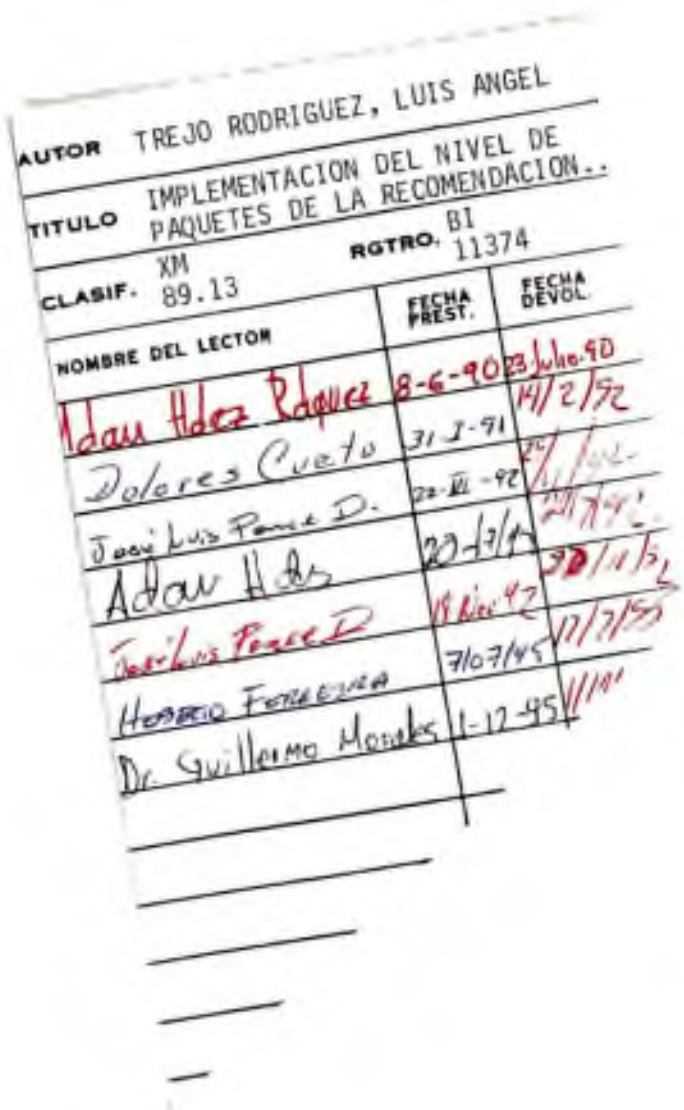
A. Iván Pech.
DISEÑO DE LA ARQUITECTURA DE UN PROGRAMA PARA LA IMPLEMENTACION DEL PROTOCOLO X.25.
Trabajo de tesis. Departamento de Ingeniería Eléctrica.
CINVESTAV del IPN, 1988.

El jurado designado por la Sección de Computación del Departamento de Ingeniería Eléctrica del Centro de Investigación y de Estudios Avanzados del I.P.N. aprobó esta tesis el 22 de agosto de 1989.

Dr. Jan Janeček Hyan

M.en C. César A. Galindo Legaria

M.en C. Carlos Hirsch Ganievich



AUTOR TREJO RODRIGUEZ, LUIS ANGEL

TITULO IMPLEMENTACION DEL NIVEL DE PAQUETES DE LA RECOMENDACION..

CLASIF. XM 89.13 RGTRO. BI 11374

NOMBRE DEL LECTOR	FECHA PREST.	FECHA DEVOL.
Adan Hdez Blanco	8-6-90	23/Ago.90
Dolores Cueto	31-I-91	14/2/92
Jesús Luis Ponce D.	22-XI-92	27/Nov.
Adan Hdez	20-I-93	20/1/93
Jesús Luis Ponce D	19 Nov. 93	17/2/93
Heberto Fonseca	7/07/95	11/10/
Dr. Guillermo Morales	1-12-95	